

Python数据分析 与算法指南

(套装共8册)



清华大学出版社

总 目 录

[Python数据分析](#)

[Python大数据分析算法与实例](#)

[Python机器学习：数据分析与评分卡建模:微课版](#)

[Python数据可视化之Matplotlib与Pyecharts](#)

[Python数据预处理技术与实践](#)

[Python科学计算（第2版）](#)

[Python量化交易实战](#)

[Python王者归来](#)



人工
智能

科学与技术丛书

数据分析 — 数据清洗 — 数据可视化

系统论述Python数据分析与处理的理论、方法与关键技术
四位业内专家联袂推荐

Python

数据分析



江雪松 邹静◎编著
Jiang Xuesong Zou Jing

DATA ANALYSIS
USING PYTHON
PROGRAMMING LANGUAGE

程序代码
数据集

清华大学出版社

人工智能科学与技术丛书

Python 数据分析

江雪松 邹 静 编著

清华大学出版社
北 京

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

图书在版编目（CIP）数据

Python数据分析 / 江雪松，邹静编著. —北京：清华大学出版社，
2020.6

（人工智能科学与技术丛书）

ISBN 978-7-302-55517-9

I. ①P… II. ①江…②邹… III. ①软件工具-程序设计 IV.
①TP311.561

中国版本图书馆CIP数据核字（2020）第085999号

责任编辑：盛东亮 吴彤云

封面设计：李召霞

责任校对：李建庄

责任印制：杨 艳

出版发行：清华大学出版社

网 址： <http://www.tup.com.cn>, <http://www.wqbook.com>

地 址：北京清华大学学研大厦A座

邮 编：100084

社 总 机：010-62770175

邮 购： 010-62786544

投稿与读者服务： 010-62776969, c-service@tup.tsinghua.edu.cn

质量反馈： 010-62772015, zhiliang@tup.tsinghua.edu.cn

课件下载： <http://www.tup.com.cn>, 010-83470236

印 刷 者： 清华大学印刷厂

经 销： 全国新华书店

开 本： 185mm×260mm

印 张： 20

字 数： 487千字

版 次： 2020年7月第1版

印 次： 2020年7月第1次印刷

印 数： 1~2500

产品编号： 084953-01

作者简介

江雪松 2001年获工学硕士学位，曾服务于华为、诺基亚等公司，有丰富的产品团队与研发团队管理经验。先后担任项目经理、研发经理、产品研发负责人，管理多个产品全球支持与交付团队、软件维护业务。

邹静 2006年获理学硕士学位，现服务于四川省人民医院，担任副主任医师。拥有十多年的临床药物实验设计、数据统计与分析经验。先后参加了多个国家自然科学基金项目及四川省卫生厅科研项目。

内容简介

本书是一本系统讨论Python数据分析基础与案例实战的教程。全书共分为上下两篇：上篇为Pandas数据分析基础（第1章～第10章），首先介绍了数据分析及其基本流程、如何构建基于Python的数据科学开发环境，然后深入讨论了如何利用Python中的Pandas库进行基本的数据操作、数据清洗、数据整理，以及如何对数据进行可视化，最后用一个电商销售数据的分析案例对上篇的知识进行了总结；下篇为Python数据分析实战（第11章～第21章），尽可能多地为读者展示各种数据分析应用，目的是让读者体会到数据分析的作用。此外，下篇也对时间序列数据和大规模数据分析等进行了讨论。

为便于读者高效学习，快速掌握Python数据分析，本书提供全部的配套源代码以及数据。此外，作者单独录制了配套视频课程，针对该课程提供答疑。本书也额外提供了如何利用Basemap库和Folium库完成地理信息数据的可视化教程，供读者单独下载。

本书适合作为高校数据分析相关专业的课程教材，也可以作为从事数据分析的广大工作者的自学参考用书。

前言

根据Cummulus Media 2018年的数据，全球每分钟向谷歌发起370万次搜索，发送1800万条短信，YouTube上有430万条视频被观看，Facebook登录超过97万次，等等。人们刷微博、聊微信、用滴滴打车、用淘宝购物、用美团点外卖，每一条信息，每一次出行，每一次消费，每一次互动都成为其数字足迹！未来已来，我们进入了数据时代！

本人与数据正式结缘还要追溯到2003年担任软件项目经理时，当时分析项目团队的软件问题成为我的第一个数据分析任务。2009年，我开始管理多个产品的全球支持团队，与数据的关系更进了一步。作为通信设备商，我们的客户支持有本地技术支持、欧洲及美洲等区域技术支持以及全球技术支持。每层的技术人员解决的问题类型、难度、时间以及成本都不同，电信运营商的设备故障每分每秒都将给客户带来巨大影响。面对成千上万的客户问题，如何快速、高效、低成本地处理各种问题，需要用数据来帮助我们做出决策。2013年后，我开始转向负责软件维护业务，此时我关心的是：如何从软件维护中获取更多收入；如何更好地提高客户服务质量；针对全球的客户，如何对客户进行细分；哪些客户的维护业务对公司更有价值；面对客户时如何说服他们维护业务带来的价值远远超过了成本。这一切更需要数据来支持！

不仅工作中需要数据思维，我们的生活也需要数据思维，它可以帮我们找工作、买房、从投资中获利，等等。那么什么是数据思维？数据思维最核心的思想是利用数据解决问题，而利用数据解决问题则需要深度了解需求，了解真正要解决什么样的问题，解决问题背后的真实目的是什么。在解决问题的过程中，使用数据分析的方法，帮助

我们从庞杂的数据中提取有价值的信息，做出更好的决策，本书正是围绕此目标而编写。全书分上下两篇。上篇（第1章～第10章）介绍了什么是数据分析，如何利用Python中的Pandas库进行基本的数据操作、数据清洗、数据整理，如何对数据进行可视化。下篇（第11章～第21章）着重于数据分析实战，尽可能多地为读者展示各种数据分析应用，目的是让读者体会到数据分析的作用。其中既有数据分析在企业中的应用，如客户群组分析、客户细分、A/B测试等，又有数据分析在我们生活中的应用，如找工作、买房、投资等。通过这些案例，希望读者能够理解数据分析是没有边界的，只要能提出问题，就能找到它的用武之地。

随着可穿戴设备的兴起，物联网、人工智能伴随5G而来，用数据定义问题、用数据讲故事、用数据支持决策的能力也越来越重要，本书是作者对过去学习与工作的总结，希望能成为读者迈入数据世界的第一本书。

本书第7章、第14章和第15章由邹静完成，其余章节由江雪松完成。由于作者水平有限，书中难免有不足之处，还望读者不吝指正！

感谢清华大学出版社的盛东亮老师为出版本书提供的帮助！同时也要感谢本书编辑团队付出的辛勤劳动！将近一年时间的写作，意味着无法陪伴家人，在此也要感谢他们的理解与支持。

江雪松
2020年4月

说明

1. 写作风格

本书上篇为Pandas数据分析基础，因此作者对于代码的讲解相对详细。下篇为案例实战，经过上篇的学习，读者应该已经具备了自学Pandas中新函数的能力，因此在讲解时以案例为主，对部分代码不再逐行解读，希望读者能自己动手实践，加深理解。

2. 英文术语的翻译

本书中的英文术语在两种情况不进行翻译，采取保留英文原文的方式。第一种情况是专有术语，无中文翻译，如Pandas、Jupyter Notebook等。第二种情况是Pandas中的专有数据结构DataFrame和Series，采取不翻译的方式，另外对于DataFrame中的Index，如果可能在上下文产生歧义，我们会采取保留英文原文的方式。

3. 代码格式

本书中的代码均在Jupyter Notebook中编写完成，Jupyter Notebook的一大优势是提供了良好的用户交互输出。为了更好地区分代码与输出，全书中的代码采用如下格式。


```
>>> retail_data.dtypes
InvoiceNo      object
StockCode      object
Country        object
Total_price    float64
dtype: object
```

其中，前面带有“>>>”的代表Jupyter Notebook中的一段代码。例如，上面代码段中的“>>> retail_data.dtypes”代表的是一行代码，前面没有“>>>”的部分则代表了代码对应的输出。类似地，如下内容中“>>>”后面代表的是一段代码。

```
>>> def rad_to_degrees(x, pos):
    '角度幅度转换'
    # 两个参数分别是值与 tick 位置
    return round(x * 57.2985, 2)
```

4. 全书代码与数据下载

由于本书是一本数据分析实战的书，因此书中大量使用了各种不同真实场景的数据和代码，为了方便读者学习，我们也提供了数据和代码下载。读者可以关注微信公众号“见数知理”获得代码与数据的下载方式。

读者完成代码与数据下载后，如果按照本书第2章的方式设置了Jupyter Notebook，那么请将数据放到data目录，该目录应该与代码目录同级。

5. 勘误表

全书的勘误将通过公众号发布，读者阅读本书若发现任何问题，也欢迎通过公众号与我们交流。

学习说明：请关注“人工智能科学与技术”公众号，了解本书及后续可能更新的学习资源。

目 录

[前言](#)

[说明](#)

[上篇 Pandas数据分析基础](#)

[第1章 数据分析初探](#)

[1.1 “数据+”时代的到来](#)

[1.2 什么是数据分析](#)

[1.2.1 数据分析的目标](#)

[1.2.2 数据分析分类](#)

[1.2.3 典型的数据分析方法](#)

[1.3 数据分析的基本流程](#)

[1.3.1 问题定义](#)

[1.3.2 收集数据](#)

[1.3.3 数据处理](#)

[1.3.4 数据分析](#)

[1.3.5 结果解读与应用](#)

[1.4 硝烟中的数据分析](#)

[1.4.1 数据分析的产生](#)

[1.4.2 验证问题](#)

[1.4.3 寻找原因](#)

[1.4.4 数据怎么说](#)

[1.4.5 数据分析中应该避免的典型问题](#)

[第2章 搭建数据科学开发环境](#)

[2.1 为什么选择Python](#)

[2.1.1 人生苦短，我用Python](#)

[2.1.2 为何Python是数据科学家的最佳选择](#)

[2.2 Python数据科学开发栈](#)

[2.2.1 Cython](#)

[2.2.2 NumPy](#)

[2.2.3 IPython](#)

[2.2.4 Jupyter](#)

[2.2.5 SciPy](#)

[2.2.6 Matplotlib](#)

[2.2.7 Pandas](#)

[2.2.8 Scikit-learn](#)

[2.2.9 NetworkX](#)

[2.2.10 PyMC3](#)

[2.2.11 数据科学领域中最新的一些Python包](#)

[2.3 Anaconda的安装与使用](#)

[2.3.1 安装Anaconda](#)

[2.3.2 利用Conda管理Python环境](#)

[2.3.3 利用Conda管理Python包](#)

[2.3.4 安装本书所需的包](#)

[2.4 使用Jupyter Notebook进行可重复数据分析](#)

[2.4.1 Jupyter Notebook的配置](#)

[2.4.2 Jupyter Notebook中的单元格](#)

[2.4.3 Jupyter Notebook中的命令模式与编辑模式键](#)

[2.4.4 使用Jupyter Notebook进行数据分析](#)

[第3章 Pandas基础](#)

[3.1 什么是DataFrame](#)

[3.1.1 DataFrame的基本要素](#)

[3.1.2 数据类型](#)

[3.1.3 了解Series](#)

[3.1.4 链式方法](#)

[3.2 索引与列](#)

[3.2.1 修改索引与列](#)

[3.2.2 添加、修改或删除列](#)

[3.3 选择多列](#)

[第4章 数据筛选](#)

4.1 使用.loc和.iloc筛选行与列数据

4.1.1 选择Series和DataFrame中的行

4.1.2 同时选择行与列

4.2 布尔选择

4.2.1 计算布尔值

4.2.2 多条件筛选数据

第5章 开始利用Pandas进行数据分析

5.1 了解元数据

5.2 数据类型转换

5.3 缺失数据与异常数据处理

5.3.1 缺失值与重复值

5.3.2 处理缺失数据

5.3.3 NumPy与Pandas对缺失数据的不同处理方式

5.3.4 填充缺失值

5.4 处理重复数据

5.5 异常值

5.6 描述性统计

第6章 数据整理

6.1 什么是数据整理

6.1.1 数据的语义

6.1.2 整齐的数据

6.2 数据整理实战

6.2.1 列标题是值，而非变量名

6.2.2 多个变量存储在一列中

6.2.3 变量既在列中存储，又在行中存储

6.2.4 多个观测单元存储在同一表中

6.2.5 一个观测单元存储在多个表中

6.2.6 思考

第7章 分组统计

7.1 分组、应用和聚合

7.2 Pandas中的GroupBy操作

- [7.2.1 单列数据分组统计](#)
- [7.2.2 多列数据分组统计](#)
- [7.2.3 使用自定义函数进行分组统计](#)
- [7.2.4 数据过滤与变换](#)

[第8章 数据整合](#)

- [8.1 数据读入](#)
 - [8.1.1 基本数据读入方法](#)
 - [8.1.2 文件读取进阶](#)
 - [8.1.3 读取其他格式文件](#)
- [8.2 数据合并](#)
 - [8.2.1 认识merge操作](#)
 - [8.2.2 merge进阶](#)
 - [8.2.3 join与concat](#)

[第9章 数据可视化](#)

- [9.1 Matplotlib](#)
 - [9.1.1 绘制第一个散点图](#)
 - [9.1.2 理解figure与axes](#)
 - [9.1.3 Matplotlib中面向对象与类Matlab语法的区别](#)
 - [9.1.4 修改坐标轴属性](#)
 - [9.1.5 修改图形属性](#)
 - [9.1.6 定制图例，添加标注](#)
 - [9.1.7 子图](#)
 - [9.1.8 利用Matplotlib绘制各种图形](#)
- [9.2 Pandas绘图](#)
 - [9.2.1 Pandas基础绘图](#)
 - [9.2.2 整合Pandas绘图与Matplotlib绘图](#)
- [9.3 Seaborn](#)
 - [9.3.1 Seaborn中的样式](#)
 - [9.3.2 Seaborn绘制统计图形](#)
- [9.4 可视化进阶](#)
 - [9.4.1 其他可视化工具](#)

[9.4.2 推荐读物](#)

[第10章 探索性数据分析——某电商销售数据分析](#)

[10.1 数据清洗](#)

[10.1.1 分析准备](#)

[10.1.2 了解数据](#)

[10.2 数据清洗与整理](#)

[10.2.1 数据类型转换与错误数据删除](#)

[10.2.2 添加新数据](#)

[10.3 探索性数据分析](#)

[10.3.1 客户分析](#)

[10.3.2 订单趋势分析](#)

[10.3.3 客户国家分析](#)

[10.3.4 留给读者的问题](#)

[下篇 Python数据分析实战](#)

[第11章 群组分析](#)

[11.1 群组分析概述](#)

[11.1.1 从AARRR到RARRA的转变](#)

[11.1.2 什么是群组分析](#)

[11.2 群组分析实战](#)

[11.2.1 定义群组以及周期](#)

[11.2.2 群组分析具体过程](#)

[11.2.3 思考](#)

[第12章 利用RFM分析对用户进行分类](#)

[12.1 RFM分析简介](#)

[12.1.1 RFM模型概述](#)

[12.1.2 理解RFM](#)

[12.2 RFM实战](#)

[12.2.1 R、F、M值的计算](#)

[12.2.2 利用RFM模型对客户进行细分](#)

[12.2.3 思考](#)

[第13章 购物篮分析](#)

13.1 购物篮分析概述

13.1.1 什么是购物篮分析

13.1.2 购物篮分析在超市中的应用

13.1.3 购物篮分析实现

13.2 购物篮分析案例

13.2.1 Mlxtend库中Apriori算法使用介绍

13.2.2 在线销售数据购物篮分析

13.3 留给读者的思考

第14章 概率分布

14.1 随机数

14.2 常见的概率分布

14.2.1 均匀分布

14.2.2 正态分布

14.2.3 二项分布

14.2.4 泊松分布

14.2.5 几何分布与指数分布

14.3 点估计与置信区间

14.3.1 点估计

14.3.2 抽样分布与中心极限定理

14.3.3 置信区间

14.4 留给读者的思考

第15章 假设检验

15.1 假设检验概述

15.1.1 初识假设检验

15.1.2 假设检验的步骤

15.1.3 假设检验中的 I 类错误与 II 类错误

15.2 Python中的假设检验

15.2.1 单样本 t -test

15.2.2 双样本 t -test

15.2.3 配对 t -test

15.2.4 卡方检验

15.3 留给读者的思考

第16章 一名数据分析师的游戏上线之旅

16.1 游戏启动时间是否超过目标

16.1.1 启动时间是否超过3秒

16.1.2 构造启动时间监测图

16.2 次日留存率是否大于30%

16.3 应该在游戏第几关加入关联微信提示

16.3.1 A/B测试

16.3.2 贝叶斯解决方案

16.4 如何定价

16.5 留给读者的思考

第17章 利用数据分析找工作

17.1 设定分析目标

17.1.1 问题定义

17.1.2 获取数据

17.2 准备分析数据

17.2.1 数据准备

17.2.2 数据清洗

17.3 开始数据分析

17.3.1 职位来自哪里

17.3.2 职位薪酬如何

17.3.3 岗位要求

17.3.4 思考

第18章 用数据解读成都房价

18.1 设定分析目标

18.1.1 问题定义

18.1.2 获取数据

18.2 解读成都二手房

18.2.1 数据准备

18.2.2 列名调整

18.2.3 数据类型转换

[18.2.4 数据解读](#)

[18.2.5 思考](#)

[第19章 时间序列分析](#)

[19.1 认识时间序列数据](#)

[19.1.1 读入时间序列数据](#)

[19.1.2 时间序列数据的可视化](#)

[19.2 时间序列数据的分解](#)

[19.2.1 认识时间序列数据中的模式](#)

[19.2.2 Python中进行时间序列数据的分解](#)

[19.3 时间序列的平稳性](#)

[19.3.1 认识平稳与非平稳时间序列](#)

[19.3.2 如何让时间序列平稳](#)

[19.4 利用ARIMA模型分析家具销售](#)

[19.4.1 ARIMA模型简介](#)

[19.4.2 数据准备](#)

[19.4.3 ARIMA模型中的参数](#)

[19.5 留给读者的思考](#)

[第20章 股票数据分析](#)

[20.1 股票收益分析](#)

[20.1.1 获取股票数据](#)

[20.1.2 计算每日收益](#)

[20.1.3 多只股票收益比较](#)

[20.1.4 股价相关性分析](#)

[20.2 CAPM资产定价模型选股](#)

[20.2.1 CAPM公式](#)

[20.2.2 在Python中实现CAPM](#)

[20.3 留给读者的思考](#)

[第21章 大规模数据处理](#)

[21.1 不同规模数据处理工具的选择](#)

[21.2 利用Pandas处理大规模数据](#)

[21.2.1 文件分块读入](#)

[21.2.2 使用数据库](#)

[21.2.3 使用DASK](#)

[21.3 其他可选方法](#)

[21.4 留给读者的思考](#)

[返回总目录](#)

上篇 Pandas数据分析基础

第1章 数据分析初探

松下问童子，言师采药去。

只在此山中，云深不知处。

——贾岛《寻隐者不遇》

彭博社曾在报道中提到：“数据科学家将在职场受到万众瞩目，这些专业人士在市场上将供不应求，一些数据科学家的起薪将超过20万美元。”随着市场对数据分析技能的推崇，越来越多的人士也开始选择从事数据分析这一职业。然而，一些即使是从事数据分析多年的人士也简单地认为数据分析就是对数据进行清洗、整理后，要么把数据可视化，要么对数据进行建模或预测。殊不知数据分析是源于商业需求，唯有将其放入商业场景才能更好地理解数据分析，否则一切将成为无本之源。所谓纲举目张，作者希望通过本章的介绍，使读者更好地理解：

- 什么是数据分析以及数据分析的类型；
- 数据分析的基本流程；
- 现实中的数据分析；
- 数据分析中应避免的错误。

1.1 “数据+”时代的到来

根据Cummulus Media 2018年的数据，全球每分钟向谷歌发起370万次搜索，发送1800万条短信，YouTube上有430万条视频被观看，Facebook登录超过97万次。IBM预测，2020年人类产生的数据将会多达40万亿兆字节，这是一个天文数字。毋庸置疑，我们已经进入数据时

代，我们每天发布微博、聊微信，线上淘宝、京东的每一次浏览、购物，线下超市、加油站、餐馆的每次消费，使用滴滴打车，用微信或支付宝付款，这些都在产生数据。然而对企业、政府而言，我们面临的问题不是缺少数据，而是拥有太多信息，无法做出清晰决策。我们需要正确的数据来回答企业、政府遇到的问题；需要从这些数据中得出准确的结论；需要数据来帮助做出决策。简而言之，企业、政府需要进行更好的数据分析，需要正确的数据分析流程、方法、工具来处理信息。

1.2 什么是数据分析

那么什么是数据分析呢？顾名思义，数据分析就是数据（Data）加分析（Analysis）。“数据”就是数值，也就是我们通过观察、实验或计算得出的结果。数据有很多种，最简单的就是数字，数据也可以是文字、图像、声音等。数据可以用于科学研究、设计、验证等。

“分析”就是将研究对象的整体分为各个部分、方面、因素和层次，并分别加以考察的认识活动。分析的意义在于细致地寻找能够解决问题的主线，并以此解决问题。因此，数据分析就是：用适当的统计分析方法对收集来的大量数据进行分析，提取有用信息和形成结论，对数据加以详细研究和概括总结的过程。

这一过程也是质量管理体系的支持过程。在实际应用中，数据分析可帮助人们做出判断，以便采取适当行动。数据分析的数学基础在20世纪早期就已确立，但直到计算机的出现才使得实际操作成为可能，并使数据分析得以推广。数据分析是数学与计算机科学相结合的产物。如果用一句话来定义数据分析，那么可以认为数据分析就是利用数据来理性思考和决策的过程。

1.2.1 数据分析的目标

孤立的数据没有任何意义，唯有将其放到实际业务中才能产生价值，那么我们具体用数据分析来干什么呢？正如哲学有3个终极问题：我是谁？我从哪里来？我要到哪里去？与之对应，数据分析的目标也是回答这三大问题。

（1）我是谁：过去发生了什么。

如果你不知道自己在哪里，那么给你一张地图也没有任何意义。对企业而言，首要的任务就是了解过去发生了什么。以电商网站为例，企业需要了解新用户注册、用户复购、仓库备货、配送、营收等运营指标，提供这些指标来衡量公司的运营，用以说明当前业务是好还是坏，好的程度如何，坏的程度又如何。除了运营指标的监控，企业还需要了解各项业务的构成、业务的发展和变动情况等。针对“我是谁”的数据分析通常会以每天、每周、每月的报表形式来表现，有的时候企业还需要实时了解业务，如天猫“双十一”活动时，对销售额、订单、快递等的实时显示。

（2）我从哪里来：归因。

“我是谁”的问题，解决了现状问题，那么“我从哪里来”就需要解决问题的归因，即为什么会这样。通过现状分析，我们对企业的当前运营情况有了基本了解，但为什么用户最近流失，营收却增加了？为何配送最近总是延迟？客户满意度为什么最近在下降？这就是数据分析要解决的第二个问题，寻找问题的原因。

（3）我要到哪里去：预测。

“我要到哪里去”，简单来说就是告诉我们将来会发生什么。一方面，我们通过对企业运营现状的了解来帮助企业对未来发展趋势做出预测，为制定企业运营目标及策略提供有效的参考与决策依据，以保证企业的可持续健康发展；另一方面，我们需要实时预测客户的行为，针对客户进行精准营销，推断客户将商品加入购物车后的下一步

行为。类似的预测还有很多，例如，国外有的银行根据求职网站的岗位数量推断就业率；美国疾病控制和预防中心依据网民搜索分析全球范围内流感等疾病的传播状况，这些都是对未知的预测。

1.2.2 数据分析分类

典型的数据分析或商业分析（Business Analytics）分为以下3类：

- 描述性分析（Descriptive Analytics）：已经发生了什么？
- 预测性分析（Predictive Analytics）：将发生什么？
- 指导性分析（Prescriptive Analytics）：应该怎么办？

1. 描述性分析

描述性分析是传统数据分析的主要应用领域，使用的技术主要有基于数据仓库的报表、多维联机分析处理等，通过各种查询了解业务中发生了什么，寻找数据中的存在模式。例如，本月某类商品销售额是多少，客户平均订单价值是多少，客户留存率是多少。

2. 预测性分析

预测性分析主要是基于大数据（实际上也可以基于传统的数据仓库和数据库），采用各种统计方法以及数据挖掘技术预测业务中各个方面将要发生什么。例如，基于过去几年的时间序列销售数据预测明年的销售额；基于聚类分析、分类分析、逻辑回归等技术预测客户信用等级；基于关联分析预测不同商品组合可能产生的销售效果。目前各类热门的大数据方面的统计应用，包括数据挖掘技术等，都可归类到预测性分析。

3. 指导性分析

Prescriptive Analytics是一个比较难翻译的词，常规翻译为规范性分析，有些不明所以。此类分析的内在含义是它会告诉用户应该做什么以得到最优的结果，因此，翻译为指导性分析更加合适，也有翻译为决策分析的。它主要指采用运筹科学的方法，即运用数学模型或智能优化算法，对企业应该采取的最优行动给出建议。例如，采用数学模型确定最优的商品定价以实现利润最大化。再比如，应该怎样实现网页的最优广告位布局、生产企业最优的生产排程、最优的劳动力排班等。本书将重点讨论描述性分析与预测性分析。

1.2.3 典型的数据分析方法

数据分析与统计分析密不可分，从统计学角度，典型的数据分析方法可以分为以下几类。

- 描述性统计分析：应用统计特征、统计表、统计图等方法，对资料的数量特征及其分布规律进行测定和描述。
- 验证性统计分析：侧重于对已有的假设或模型进行验证。
- 探索性数据分析：主动在数据之中发现新的特征或有用的隐藏信息。

1. 描述性统计分析

描述性统计分析是用来概括、表述事物整体状况以及事物间关联、类属关系的统计方法。通过统计处理可以简单地用几个统计值来表示一组数据的集中趋势、离散程度以及分布形状，如图1.1所示。

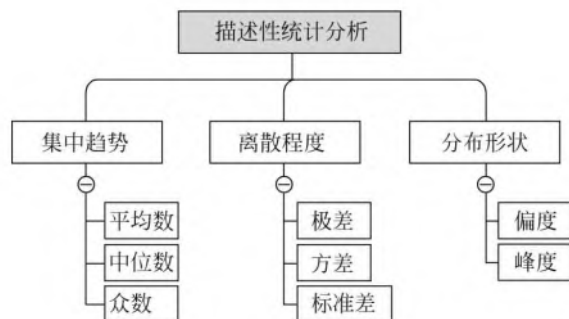


图1.1 描述性统计分析

2. 验证性统计分析

验证性统计分析是对数据模型和研究假设的验证，参数估计、假设检验以及方差分析是验证性统计分析中常用的方法。所谓参数估计就是用样本统计量去估计总体的参数。假设检验与参数估计类似，但角度不同，参数估计是利用样本信息推断未知的总体参数，而假设检验是对总体参数提出一个假设值，然后利用样本信息判断这一假设是否成立。假设检验可分为：

- 单样本假设检验；
- 双样本的均值比较假设检验；
- 成对样本的均值比较假设检验。

而方差分析则是通过比较总体各种估计间的差异来检验方差的正态总体是否具有相同的均值，是检验多因素之间差异显著性的重要统计分析方法，常用的方差分析方法有：

- 单因子方差分析；
- 双因子方差分析。

3. 探索性数据分析

探索性数据分析（Exploratory Data Analysis, EDA）是指对已有数据在尽量少的先验假设下通过作图、制表、方程拟合、计算特征量等手段探索数据的结构和规律的一种数据分析方法，该方法在20世纪70年代由美国统计学家J. K. Tukey提出。传统的统计分析方法常常先假设数据符合一种统计模型，然后依据数据样本来估计模型的一些参数及统计量，以此了解数据的特征，但实际中往往有很多数据并不符合假设的统计模型分布，导致数据分析结果不理想。探索性数据分析则是一种更加贴合实际情况的分析方法，它强调让数据自身“说话”，通过探索性数据分析可以真实、直接地观察到数据的结构和特征。探索性数据分析出现之后，数据分析的过程就分为两个阶段：探索阶段和验证阶段。探索阶段侧重于发现数据中包含的模式或模型，验证阶段侧重于评估所发现的模式或模型，很多机器学习算法（分为训练和测试两步）都遵循这种思想。当拿到一份数据时，如果做数据分析的目的不是非常明确、有针对性，可能会感到有些茫然，那么此刻就更加有必要进行探索性数据分析了，它能帮助我们初步了解数据的结构和特征，甚至发现一些模式或模型，再结合行业背景知识，也许就能直接得到一些有用的结论。

1.3 数据分析的基本流程

一个完整的数据分析项目可以分为5步，如图1.2所示。

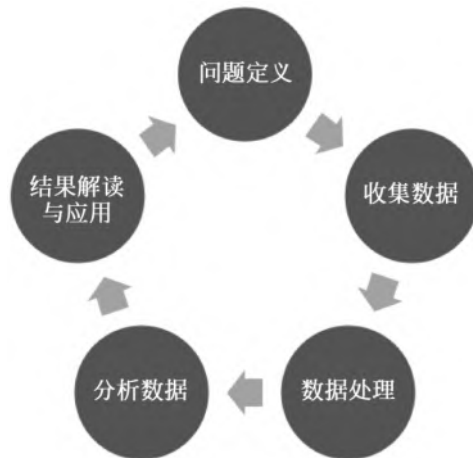


图1.2 数据分析流程

1.3.1 问题定义

企业或组织中的数据分析必须从正确的问题开始，而该问题必须清晰、简洁，同时要可度量。我们的目标是通过提出问题来帮助寻找新的解决方案，或者说解决特定问题。例如，公司通常会有用户数据、运营数据、销售数据等，我们需要利用这些数据来解决什么问题，得出什么结论。以下有一些例子：

- 某移动应用的新用户注册率趋势如何？
- 某游戏的玩家用户画像是如何的？
- 经常购买电商网站某品类的是哪类人群？
- 如何提高企业的销售额？
- 如何对用户画像，如何进行精准营销？

问题的定义通常需要分析人员对业务有深入了解，这也是经常提到的数据思维。例如，要提高企业销售，那么需要理解企业盈利模式是什么；收入可以通过增加用户来提高，还是提高价格；又或者是公司不应该专注于销售额，而应该关注利润。需要明确的是开始提出的问题只是出发点而非终点，很可能在针对问题进行了一系列研究后，

我们会修改最初的问题定义。而如何更好地定义问题，这就需要我们通过不断练习来寻找对数据的感觉。

1.3.2 收集数据

有了具体的问题，就需要准备获取相关的数据了。首先需要明确，问题对应的数据是什么，这些数据如何定义，如何度量。之后就需要考虑哪些数据是已经存在的，哪些数据需要通过对现有数据进行加工来获得，哪些数据还没有。典型的数据获取方式有以下几种。

(1) 企业数据库 / 数据仓库。大多数公司的销售、用户数据都可以直接从企业数据库获取。例如，可以根据需要提取某年所有的销售数据、提取今年销量最大的50件商品的数据、提取上海及广东地区用户的消费数据等。通过结构化查询语言（Structured Query Language, SQL）命令，我们可以快速完成这些工作。

(2) 外部公开数据集。一些科研机构、企业、政府都会开放一些数据。开放政府数据更是成为近年的热潮，典型的有2018年12月21日，美国众议院投票决定启用“H.R. 4174”（*Foundations for Evidence-Based Policymaking Act of 2017*）。首先，这项《公共、公开、电子与必要性政府数据法案》（又称《开放政府数据法案》）要求政府信息应以机器可读的格式，默认向公众开放数据，且此类出版物不会损害隐私或安全；其次，联邦机构在制定公共政策时，应循证使用。与之对应，我国国务院也制定了《促进大数据发展行动纲要》，要求“2018年底前建成国家政府数据统一开放平台，率先在信用、交通、医疗、卫生、就业、社保、地理、文化、教育、科技、资源、农业、环境、安监、金融、质量、统计、气象、海洋、企业登记监管等重要领域实现公共数据资源合理适度向社会开放”，截至2019年，已经有50多个地市开放了平台，开放了约15个领域的数据，包括教育科技、民生服务、道路交通、健康卫生、资源环境、文

化休闲、机构团体、公共安全、经济发展、农业农村、社会保障、劳动就业、企业服务、城市建设、地图服务。

（3）爬虫。利用爬虫去收集互联网上的数据是经常会采用的数据获取方式。例如，爬取淘宝上的商品信息；通过爬虫获取招聘网站某一职位的招聘信息；爬取租房网站上某城市的租房信息等。

（4）实验。如果想要判断新的应用界面是否会提高用户转化率，那么可以通过实验方式来获取，这其实就是我们熟悉的A/B测试。针对不同的问题可以设计各种不同的实验来获取相应的数据。

值得一提的是，我们有时并不能够获得所有需要的数据，不过这并不重要，因为我们的目标是通过有限的可获取的数据，提取更多有用的信息。

1.3.3 数据处理

数据处理是指对采集到的数据进行加工整理，形成适合数据分析的样式，保证数据的一致性和有效性。它是数据分析前必不可少的阶段。数据处理的基本目的是从大量的、可能杂乱无章的、难以理解的数据中抽取并推导出对解决问题有价值、有意义的数据。如果数据本身存在错误，那么即使采用最先进的数据分析方法，得到的结果也是错误的，不具备任何参考价值，甚至还会误导决策。

数据处理主要包括数据清洗、数据转化、数据抽取、数据合并、数据计算等处理方法。一般的数据都需要进行一定的处理才能用于后续的数据分析工作，即使再“干净”的原始数据也需要先进行一定的处理才能使用。现实世界中的数据大体上都是不完整、不一致的脏数据，无法直接进行数据分析，或分析结果不尽如人意。数据预处理有多种方法：数据清理、数据集成、数据变换、数据归约等。把这些影响分析的数据处理好，才能获得更加精确的分析结果。

以大众最近关心的空气质量数据为例，很可能其中有很多天的数据由于设备的原因是没有监测到的，有一些数据是记录重复的，还有一些数据是设备故障时监测无效的。那么需要用相应的方法去处理，如残缺数据，是直接去掉这条数据，还是用临近的值去补全，这些都是需要考虑的问题。当然在这里我们还可能会进行数据分组、基本描述统计量的计算、基本统计图形的绘制、数据取值的转换、数据的正态化处理等，通过这些操作掌握数据的分布特征，以帮助我们进一步深入分析和建模。

1.3.4 数据分析

进入数据分析阶段，需要了解不同方法适用的场景和问题。分析时应切忌滥用和误用统计分析方法。滥用和误用统计分析方法主要是由于对方法能解决哪类问题、方法适用的前提、方法对数据的要求等不清造成的。选择几种统计分析方法对数据进行探索性的反复分析也是极为重要的。每一种统计分析方法都有自己的特点和局限，因此，一般需要选择几种方法反复印证分析，仅依据一种分析方法的结果就断然下结论是不科学的。例如，在一定条件下，发现销量和价格成正比关系，那么可以据此建立一个线性回归模型，如果发现价格和广告是非线性关系，可以先建立一个逻辑回归模型来进行分析。一般情况下，回归分析的方法可以满足很大一部分的分析需求，当然也可以了解一些数据挖掘的算法和特征提取的方法来优化自己的模型，获得更好的结果。

通过数据分析，隐藏在数据内部的关系和规律就会逐渐浮现出来，那么通过什么方式展现出这些关系和规律，才能一目了然呢？一般情况下，数据是通过表格和图形的方式来呈现的，即用图表说话。常用的数据图表包括饼图、柱状图、条形图、折线图、散点图、雷达图等，当然可以对这些图表进一步整理加工，使之变为我们所需要的

图形，如金字塔图、矩阵图、瀑布图、漏斗图、帕累托图等。多数情况下，人们更愿意接受图形这种数据展现方式，因为它能更加有效、直观地传递出分析师所要表达的观点。

1.3.5 结果解读与应用

数据分析的结果需要以报告的形式展现，数据分析师如何把数据观点展示出来则至关重要。这一过程需要数据分析师的数据沟通能力、业务推动能力和项目工作能力。首先，深入浅出的数据报告、言简意赅的数据结论将更有利于业务理解和接受。其次，在理解业务数据的基础上，推动业务落地实现数据建议。通常，从业务最重要、最紧急、最能产生效果的环节开始是个好方法，与此同时需要考虑到业务落地的客观环境，即好的数据结论需要具备客观落地条件。最后，需要明确的是一个数据项目工作是循序渐进的过程，无论是数据分析项目还是数据产品项目，都需要数据分析师具备计划、领导、组织、控制的项目工作能力。

1.4 硝烟中的数据分析

数据分析源于具体的商业问题、商业需求，唯有将其放到实际的场景才能更好地理解数据，应用数据来创造价值。在本节中，我们将置身于一些真实的场景中，来讨论如何进行数据分析。

1.4.1 数据分析的产生

现实中的数据分析总是带着紧迫性。也许数据分析人员每天听到最多的就是：“事情紧急，我们需要现在就找到问题的答案！”以下场景相信大多数数据分析师都不陌生。

- 客户支持团队负责人发现最近客户问题响应时间严重滞后，需要知道这是什么原因导致的。
- 某家互联网公司发现最近新用户注册率下降，公司领导想知道这是什么原因导致的。
- 某移动应用激活率显著下降，需要找到问题根源。
- 某电子商务网站用户购物车弃购率增加，需要知道这是什么原因导致的。

无论是什么问题，找到原因并解决它是当务之急。公司领导者需要专注于从大量可用信息中获取可行动的知识，使用有效的工具以及流程来做出明智的决策，而数据分析则是背后的驱动工具。真实的数据分析无一例外都可能由上述的某一场景触发。

1.4.2 验证问题

开始任何数据分析之前，我们都需要快速验证前面定义的问题。为了验证问题，通常会提出更多问题。例如，思考这个问题可能导致更严重的问题吗？注册率的下降是否是网站故障导致？客户响应时间的增加是否是人员不足导致？如果考察更长的时间，能否确认这真的是一个异常吗？是否还有其他与此相关的问题发生？

另外，我们需要思考这个问题是否只是一个特例。例如，是否是报告错误（如选择了错误的日期或报告软件中的错误）？其他相关指标同样下降吗？如果下载量急剧下降但是激活却没有下降，那么是否可以得出“这是由于下载问题导致的”结论呢？如果公司同时使用多个系统的度量指标（如Google Analytics和公司自己的统计），两个系统是否都显示相同的下降？

这个快速的初步评估回答了两个问题：这实际上是一个问题吗？如果是的话，这里的核心问题是什么？对发生了什么的探究可以帮助

我们确认这是否是一个值得进一步研究的问题。

1.4.3 寻找原因

现在已经验证了问题，接下来是解决问题的时候了，通常会以如下步骤进行。

1. 寻找任何快速解决问题的可能性

需要明确的是，我们的目标总是尽可能用最小代价获取最大回报。因此，尽可能预先思考是否有明显的可能原因或问题的答案是开始数据分析前的第一步。我们需要思考是否仔细检查了问题的来源或报告，是否有任何异常或一次性原因（如已知的网站故障）。例如，电子商务网站的安全证书（Secure Socket Layer Certificates）可能已过期，浏览器弹出窗口警告，从而导致了购物车弃购率提高。如果能够快速解决问题，就无须开始更复杂的数据分析！如果不能，那可能需要进入下一步。

2. 询问其他团队

这个问题会影响或涉及其他团队吗？如果是，其他团队是否对可能的原因有了解？即使问题与其他团队之间没有明显的联系，也值得快速询问。例如，营销经理可能会询问后台部门：“我们注意到用户注册数量下降了，你们能想到在过去几周内部署的任何可能导致问题的软件变更吗？”这里获取到的回答将帮助我们进入第3步时更好地对问题原因进行假设。

3. 对可能原因进行假设

一个假设只是一个尚未得到证实的有根据的猜测。我们可以将其视为需要测试的问题的可能解释。如美国理论物理学家爱德华·泰勒所说：“事实是每个人都相信的简单陈述。除非被判有罪，否则就是无辜的。一个假设是一个没有人想要相信的新颖建议。在发现有效之前，它是无效的。”在这一步，最重要的是要考虑多个关于问题原因的假设，然后证明、反驳每一个问题。

回到前面提到的场景，我们可能会有不同假设。

(1) 客户支持问题的假设：问题单响应时间因何增加。

- 大量与服务相关的问题单，而与产品相关的问题单相比，服务问题响应时间更长。
- 只是某一个呼叫中心的问题。
- 客户支持团队成员不足，导致问题积压。
- 最近推出的新产品功能导致。

(2) 营销问题的假设：注册减少的原因。

- 某些地区的公众假期。
- 最近对营销网站（或网站中断）的更改。
- 星期一早上网站中断导致注册过程中出现错误。
- 转换率下降减少了注册量。
- 搜索排名（针对我们的产品页面）下降到搜索结果的第二页。

(3) 电子商务问题的假设：平均购物车弃购率因何增加。

- 加购物车的人绝对数值增加。
- 最近更改部分结账流程。
- 季节性原因（如假期、学校休息等）。
- 结束促销活动，导致更多人放弃购物车。
- 特定产品导致。

(4) 移动应用问题的假设：激活率下降的原因。

- 产品中的某些内容已发生变化，导致整体激活率下降。
- 一群新的（和不同的）用户开始尝试该产品。

在分析数据之前，假设问题的多个可能原因非常重要。有时在对数据进行探索时我们还可能会产生一个新的假设，之后需要测试该假设。一个好的假设需要满足以下几点：

- 它涉及一个自变量和一个因变量；
- 它是可测试的；
- 它是可证伪的。

自变量是原因（可以改变或控制），因变量是效果（可测试结果）。可证伪意味着假设可被证明是错误的。确保有一个可证伪的假设的有效方法是在这个问题中放弃变量：如果自变量 / 原因发生，因变量 / 效果是真还是假？无论是正式的还是非正式的，我们的假设将在下一步使用数据得到证实或反驳。分析将从最有可能的假设开始，然后在找到原因之前继续进行。

1.4.4 数据怎么说

通常人们通过假设并将观察与假设进行比较来发现真相。前面的假设已经探索了有几个可能的原因，现在要做的就是看看数据是怎么说的。这是一个简单的数据分析过程，通过这一过程可以测试我们的假设。

1. 确定并分割相关数据

根据之前的假设，判断需要查看哪些数据，哪些指标可以帮助证明或推翻可能的原因。通过隔离可能与问题相关或导致问题的不同数据，能够更轻松地发现趋势或异常。例如，可以按国家、地区、渠道

或网络会话持续时间来细分注册用户，据此测试假设（解决前面提到的营销问题）。

2. 探索数据

有经验的数据分析人员肯定知道什么情况下指标是“正常”的；基于这些知识并运用常识，会注意到什么；数据的任何方面是否出现异常。例如，针对安卓（Android）手机用户的注册量下降了20%这一问题，如果尚未为“正常”建立基线，就需要使用历史数据作为起点，可以将本月的注册与去年同月的注册进行比较，或者看看过去12个月的注册趋势。

3. 评估异常或趋势的影响

这是一个完整性检查，以查看发现的趋势、异常是否足以解释潜在问题。数据分析人员最常听到的就是这样的问题：“结果是否具有统计意义？”不过解决日常业务问题意味着我们正在寻找数据中的异常或趋势，这些异常或趋势不仅具有统计意义，而且具有实际意义。换句话说，我们需要弄清楚是什么将对我们的用户注册、问题响应时间、购物车弃购率和激活率产生实际影响。

1.4.5 数据分析中应该避免的典型问题

数据并不总是有价值的，在数据分析中需要牢记数据有时会对我们有用！下面将讨论一些常见的数据谬论，以帮助大家在数据分析过程中避免它们的发生。最常见的谬论是假设数据集是值得信赖的，直到后来的分析才发现它不是。在数据收集中经常会遇到如下陷阱。

1. 单方论证

诺贝尔经济学奖获得者罗纳德·科斯说：“如果你对数据进行足够长时间的折磨，它就会承认任何事情。”数据分析中，我们有时会陷入单方论证的陷阱（即Cherry Picking [\(1\)](#)），仅选择支持自己观点的数据，同时丢弃不支持自己观点的部分。例如，我们可能会注意到产品某个新功能相关的支持问题响应时间增加了。如果只是着眼于此，可能会得出结论，响应时间增加是新产品功能导致。但如果查看过去两个月的所有客户支持问题，可能会看到整体响应时间增加，是因为问题数量增加了。我们需要保持中立，并且不要爱上某个假设！

2. 错误因果关系

我们经常会因为两个事件同时发生，就认为二者相关，这可能会导致错误因果关系（False Causality）。有时似乎相关的模式可能与第三个独立因子相关，而不是彼此相关。然而更好的方法是，收集更多数据并查看可能的第三方原因。例如，我们发现放弃在线购物车的潜在客户往往具有较低的总购物价值。然而当我们深入挖掘时，可能会发现实际是由于运费的原因导致购物车弃购率上升，因为免费送货仅适用于超过特定金额的订单。

3. 幸存者偏差

幸存者偏差（Survivorship Bias）是一种常见的逻辑谬误（而不是“偏差”），指的是只能看到经过某种筛选而产生的结果，而没有意识到筛选的过程，因此忽略了被筛选掉的关键信息。即从不完整的数据集中得出结论，因为这些数据也仅仅是碰巧符合了一些选择标准。分析数据时，一个很重要的步骤是问一下自己有什么缺失的数据。有时可能没办法掌握数据的整体情况就是因为它们只反映了一部分。例如，在第二次世界大战中，英美空军为了加强战斗机的保护措施，对参战飞机中弹区域进行了详细统计，结果显示机翼部位中弹最

密集，而机舱部位中弹最少。于是军方决定对飞机机翼进行加固，但一名统计学家站出来反对。他表示真正需要加固的是机舱，因为机舱中弹的飞机大概率无法返航，才导致了这样的统计结果。最终军方采纳了他的建议，战斗机坠毁率果然降低。这就是所谓的幸存者偏差，也称为“死人不会说话”效应，幸存者的经验往往误导了我们的判断。

4. 采样偏差

由于我们并不总是能获得全部数据，那么数据能代表总体样本就变得至关重要。我们经常会发现在进行A/B测试时，某种产品改进确实提高了转化率，但是当产品上线后转化率反而下降了。通常这种问题的发生都与试验设计时样本的采样偏差（Sampling Bias）有关。

[\[1\]](https://en.wikipedia.org/wiki/Cherry_picking) https://en.wikipedia.org/wiki/Cherry_picking

第2章 搭建数据科学开发环境

工欲善其事，必先利其器。

——《论语》

在数据分析过程中需要考虑如何获取数据、整合数据源；对数据进行清洗、整理；之后可能还需要进行统计分析、数据可视化；有时还需要进行数据挖掘，根据模型做出预测；有的情况下还需要据此开发出一个数据应用。因此，在选择对应的数据科学开发环境时就需要考虑软件能否提供强大的数据爬取、多数据源整合、统计分析、数据可视化、机器学习等功能。本章将介绍：

- 为什么选择基于Python的数据科学开发环境；
- Python数据科学开发栈；
- 基于Anaconda的Python环境安装；
- Conda环境基本使用；
- 利用Jupyter Notebook进行可重复的数据分析。

2.1 为什么选择Python

2.1.1 人生苦短，我用Python

Python开发领域流传着一句话：“人生苦短，我用Python”，很多人还把它印在了衣服上，如图2.1所示。某种程度上，这也表明了Python的功能强大与简洁。



图2.1 人生苦短，我用Python

这可不是一句戏言，先来看一组统计数据，在著名的程序员问答社区Stack Overflow 2019年的开发者年度调查报告中，Python成为增长最快的开发语言，仅次于Rust语言，并成为最受喜爱的语言之一，如图2.2所示。

不仅程序员对Python钟爱有加，Python也是数据科学家的标配语言。在著名的数据科学竞赛网站Kaggle (www.kaggle.com) 2018年对数据科学家使用何种语言的调查中，Python也是拔得头筹，如图2.3所示。

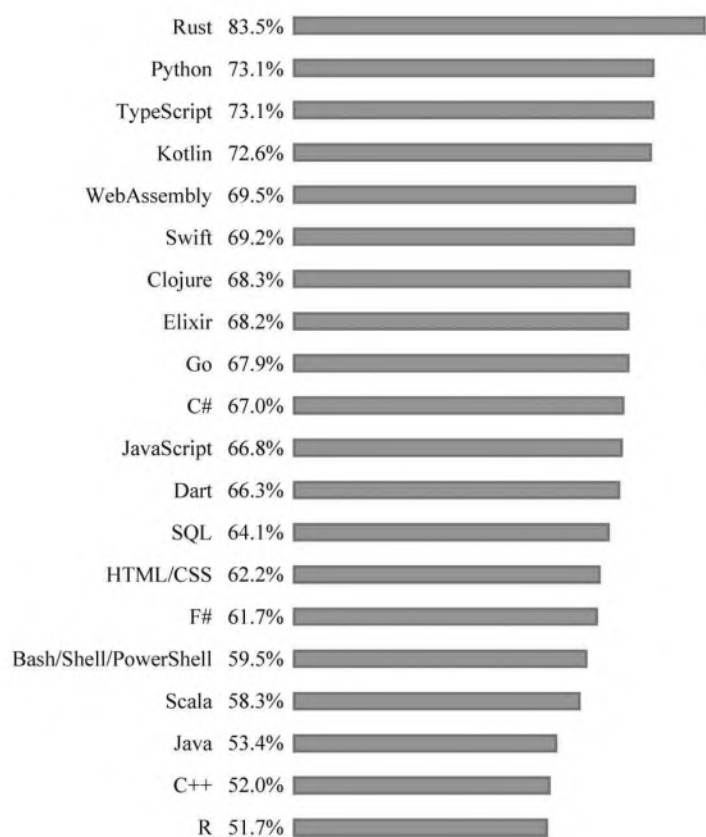


图2.2 开发语言统计

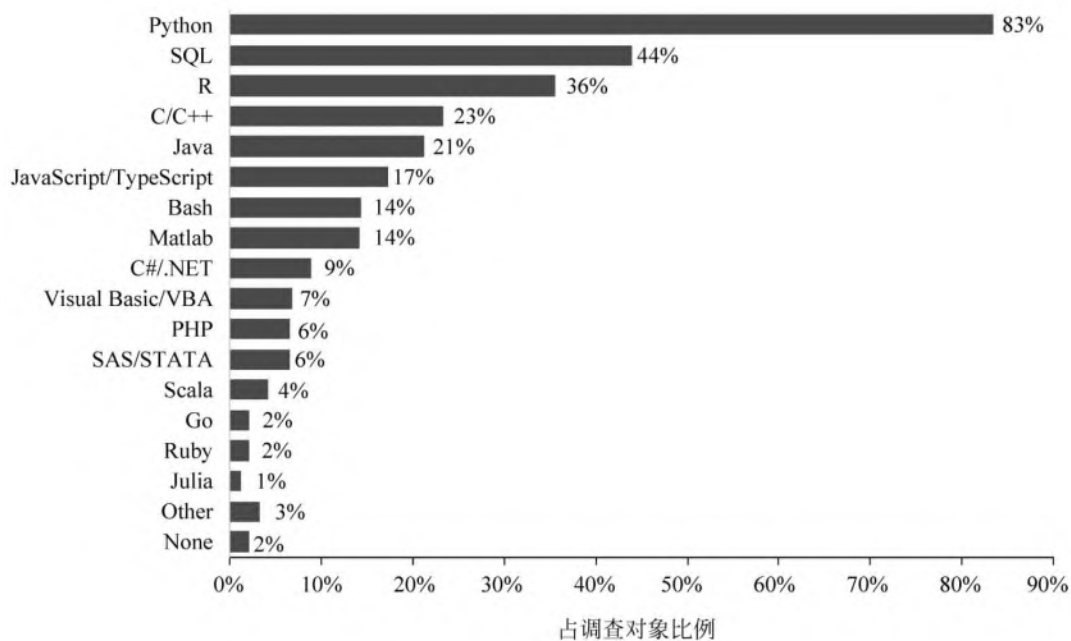


图2.3 数据工作中使用语言的统计

毋庸置疑，Python的流行也有其原因。

(1) 快速开发。除了Python语言自身的设计哲学以外，用户还有丰富的Python三方库可以使用，如在数据科学中常用的用于数值处理和科学计算的NumPy、SciPy；用于数据处理的Pandas；用于数据可视化的Matplotlib；用于机器学习的Scikit-learn等。这些库提供了大量的基础实现，数据分析人员在编码的过程中，可以方便地使用这些库，从而避免了大量代码编写过程。

(2) 代码简洁，容易理解。以机器学习为例，利用传统的Java和Python完成同一个算法实现时，Python的实现代码量明显少于Java。程序代码量的下降意味着开发周期的缩短，这在一定程度上减轻了程序员的开发负担。程序员可以利用节省的时间做更多有意义的事情，如算法设计。

(3) 语言生态健全。Python语言目前在Web开发、大数据开发、人工智能开发、后端服务开发和嵌入式开发等领域都有广泛的应用，成熟案例非常多，所以采用Python完成代码实现的时候往往具有较小的风险。

伴随着大数据和人工智能的发展，Python语言的上升趋势非常明显，相信未来Python语言在人工智能、物联网会有更加广泛的应用。从这个角度来看，利用Python进行数据分析也是一个绝佳选择。

2.1.2 为何Python是数据科学家的最佳选择

首先，Python不像R语言，它不是一门统计计算语言。然而正因为如此，Python成为了一门用于统计计算的最佳语言。从事数据分析工作的人一定都对图2.4并不陌生。



图2.4 数据科学

首先，数据科学工作需要用到数学和统计科学的知识，因此选择数据科学语言时要考虑其对数值处理、统计分析、矩阵运算等的良好支持。Python中提供了第三方包NumPy和SciPy，它们很好地提供了这些功能。其次，从事数据科学工作还需要整合各种数据源、开发数据应用。这些需要语言能支持各种数据库的连接、不同数据格式文件读取、与企业当前的应用整合以及外部数据爬取等功能。Python作为一门高级面向对象编程语言可以完美完成上述工作。最后，数据科学家还需要结合领域知识完成数据分析、机器学习，最终形成报告。而利用基于Python的Jupyter Notebook，数据科学家可以进行可重复的数据分析，快速与团队分享，编写分析报告，同时基于Python第三方软件包Scikit-learn、Tensorflow、Pytorch和Keras，既可以完成传统的机器学习应用开发，又可以完成最前沿的深度学习应用开发。

2.2 Python数据科学开发栈

武侠作家古龙曾经写过一本书叫《七种武器》，书中描述了长生剑、孔雀翎、碧玉刀等武器，而基于Python的数据武器远不止7种，它是以一套数据科学开发栈的方式来呈现给大家的，如图2.5所示。

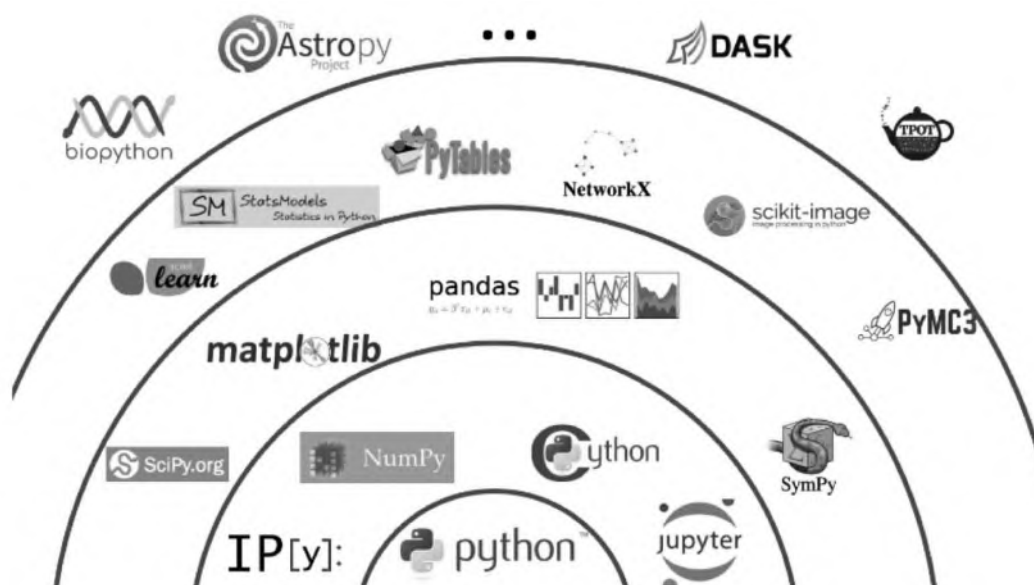


图2.5 Python数据科学开发栈

图2.5中最核心的部分当然还是Python；第二层的Cython、NumPy、IPython、Jupyter Notebook为基于Python的程序交互运行、数值计算等提供了支持；第三层的SciPy、Matplotlib、Pandas、SymPy提供了基础的数据处理、科学计算、数据可视化、符号运算的支持；基于前面3层的支持，StatsModels、Scikit-learn、NetworkX、PyMC3等模块更进一步提供了统计分析、机器学习、网络分析、概率编程等的支持；最后第五层提供了各种不同应用领域的软件包，如用于生物分析领域、天文领域的biopython、Astropy，大数据方面的DASK，自动机器学习的TPOT，此外还有深度学习领域的Keras和Pytorch等。下面将选取部分软件包进行简单介绍。

2.2.1 Cython

Cython是用C语言实现Python，是目前应用最广泛的解释器。Python最新的语言特性都是在这个上面先实现，Linux、OS X等自带的也是这个版本，包括Anaconda里面用的也是Cython。Cython是官方版

本加上对于C/Python API的全面支持，基本包含了所有第三方库支持，如NumPy、SciPy等。

2.2.2 NumPy

NumPy是一个运行速度非常快的数学库，主要用于数组计算，包含：

- 一个强大的 N 维数组对象；
- 广播功能函数；
- 整合C/C++/FORTRAN代码的工具；
- 线性代数、傅里叶变换、随机数生成等功能。

同时，NumPy内部解除了Python的PIL（全局解释器锁），运算效率极高，是大量机器学习框架的基础库。关于NumPy的更详细介绍请参考网站<https://www.numpy.org/>。

2.2.3 IPython

IPython是Interactive Python的简称，即交互式Python，作为一个增强的Python解释器，由Fernando Perez于2001年启动，并由此发展为一个项目。用Perez的原话来说，该项目致力于提供“科学计算的全生命周期开发工具”。如果将Python看作数据科学任务的引擎，那么IPython就是一个交互式控制面板。它比默认的python shell好用得多，支持变量自动补全、自动缩进、bash shell命令，同时IPython还内置了许多很有用的功能和函数，包括：

- 强大的交互式shell；
- 供Jupyter Notebooks使用的Jupyter内核；
- 交互式的数据可视化工具；
- 灵活、可嵌入的解释器；

- 易于使用，高性能的并行计算工具。

关于IPython的更详细介绍请参考网站<https://ipython.org/>。

2.2.4 Jupyter

在2011年，IPython首次发布10年后，IPython Notebook被引入。这个基于Web的IPython接口把代码、文本、数学表达式、内联图、交互式图形、小部件、图形界面以及其他丰富的媒体集成到一个独立的可共享的Web文档中。该平台为交互式科学计算和数据分析提供了理想的门户。IPython已经成为研究人员、工程师、数据科学家、教师和学生不可或缺的工具。

几年后，IPython在科学界和工程界获得了不可思议的盛誉。Notebook开始支持越来越多的Python以外的编程语言。2014年，IPython开发者宣布了Jupyter项目，该项目旨在改进Notebook的实现并通过设计使其与语言无关。目前，Notebook支持3种主要科学计算语言：Julia、Python和R。至此，Jupyter成为一个完整生态系统，同时提供几种可供选择的Notebook界面（JupyterLab、Hydrogen等）、交互式可视化库、与Notebook兼容的创作工具。它的优点也显而易见，具体如下。

- 所有内容聚合在一个地方。Jupyter Notebook是一个基于Web的交互式环境，它将代码、富文本、图像、视频、动画、数学公式、图表、地图、交互式图形、小部件以及图形用户界面组合成一个文档。
- 易于共享。Notebook保存为结构化文本文件（JSON格式），这使得它们可以轻松共享。
- 易于转换。Jupyter附带了一个特殊的工具nbconvert，可将Notebook转换为其他格式，如HTML和PDF。另一个在线工具

nbviewer 允许用户直接在浏览器中渲染一个公共可用的 Notebook。

- 独立于语言。Jupyter的架构与语言无关。客户端和内核之间的解耦使用任何语言编写内核成为可能。
- 易于创建内核包装器。Jupyter为可以用Python包装的内核语言提供了一个轻量级接口。包装内核可以实现可选的方法，特别是自动代码补齐和代码检查。
- 易于定制。Jupyter界面可用于在Jupyter Notebook（或其他客户端应用程序，如控制台）中创建完全定制的体验。
- 自定义魔术命令的扩展。使用自定义魔术命令创建IPython扩展，使交互式计算变得更加简单。许多第三方扩展和魔术命令都存在，例如，允许在Notebook中直接编写Cython代码的`%%cython`指令。
- 轻松可重复实验。Jupyter Notebook可以帮助用户轻松进行高效且可重复的交互式计算实验。它可以让用户保存详细的工作记录。此外，Jupyter Notebook的易用性意味着用户不必担心可重复性，只需要在Notebook上做所有的互动工作，将它们置于版本控制之下，并定期提交。
- 有效的教学和学习工具。Jupyter Notebook不仅是科学研究和数据分析的工具，而且是教学的好工具。
- 交互式代码和数据探索。ipywidgets包提供了许多用于交互式浏览代码和数据的通用用户界面控件。

关于Jupyter的更详细内容请参考网站<https://jupyter.org/>，本书后续章节也将对Jupyter Notebook的使用进行更详细的介绍。

2.2.5 SciPy

SciPy是基于Python的NumPy构建的一个集成了多种数学算法和函数的Python模块。它为用户提供了一些高层的命令和类，SciPy在Python交互式会话中，大大增加了操作和可视化数据的能力。通过SciPy，Python的交互式会话变成了一个数据处理和系统原型设计环境，足以与Matlab、IDL、Octave、R-Lab和SciLab等系统相媲美。

更重要的是，在Python中使用SciPy，还可以同时用一门强大的语言——Python来开发复杂和专业的程序。用SciPy写科学应用，还能获得世界各地的开发者开发的模块的帮助。从并程序到网络应用，再到数据库子例程和各种类，Python都已经有可用的模块。SciPy最强大的功能还是在于它的数学库。SciPy中提供的具体功能如表2.1所示。

表2.1 SciPy中的核心包

功 能	描 述	功 能	描 述
SciPy. cluster	向量计算/K-means	SciPy. odr	正交距离回归
SciPy. constants	物理和数学常量	SciPy. optimize	优化
SciPy. fftpack	傅里叶变换	SciPy. signal	信号处理
SciPy. integrate	积分程序	SciPy. sparse	稀疏矩阵
SciPy. interpolate	插值	SciPy. spatial	空间数据结构和算法
SciPy. io	数据输入和输出	SciPy. special	一些特殊数学函数
SciPy. linalg	线性代数程序	SciPy. stats	统计
SciPy. ndimage	<i>n</i> 维图像包		

关于SciPy的更详细介绍，请参考网站<https://www.scipy.org/>。

2.2.6 Matplotlib

Matplotlib是利用Python实现的绘图套件，其中包含两个最重要的模块——pylab和pyplot。pylab已经几乎实现了在学术界最常用的软件Matlab所支持的绘图功能，或者说pylab其实就是Matlab的Python版本；而pyplot是将pylab再加上Python中有名的数学计算软件——NumPy，让使用者在使用pyplot时，可以直接调用NumPy的函数做计算后再以图形的方式呈现。Matplotlib功能十分强大，可以说是基

于Python的图形可视化标准，利用Matplotlib，用户几乎可以完成任何图形的绘制，从传统的线图、条形图（如图2.6所示）、统计图（如图2.7所示）到图像、热图、等高图（如图2.8所示）等，同时用户还可以利用Matplotlib的子图功能实现多图绘制、定制图形（如图2.9所示）等。

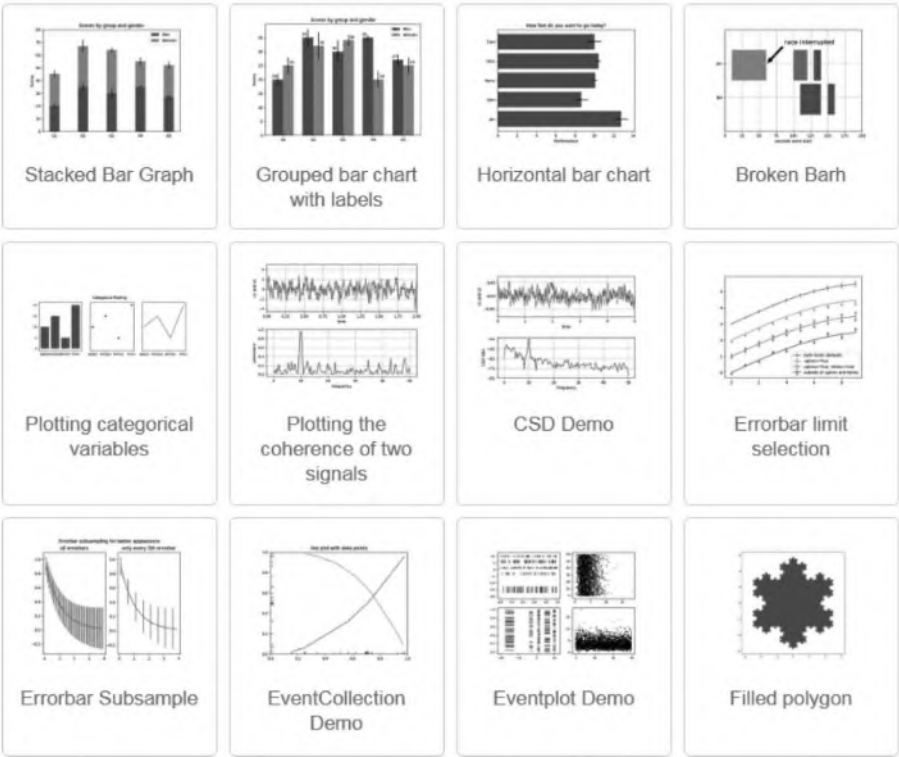


图2.6 线图与条形图

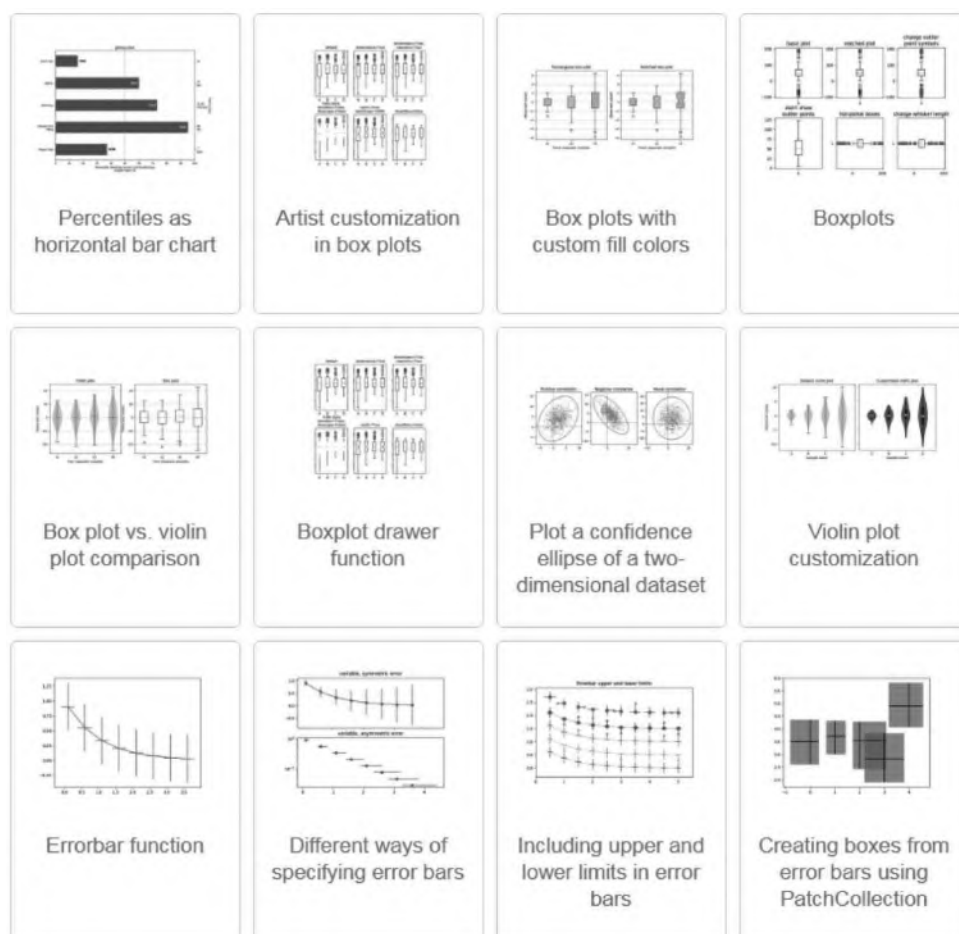


图2.7 统计图

当然，Python中的图形可视化工具不只有Matplotlib，还有专门进行统计图形可视化的Seaborn，交互图形可视化的Plotly、Bokeh等。本书后续章节将对Matplotlib和Seaborn进行更详细介绍。

上述包的更详细介绍可以参考下列网站：

- Matplotlib: <https://matplotlib.org/>;
- Seaborn: <https://seaborn.pydata.org/>;
- Plotly: <https://plot.ly/>;
- Bokeh: <https://bokeh.pydata.org>。

2.2.7 Pandas

Pandas为Python提供了高效的数据处理、数据清洗与整理的工具。关于Pandas的更详细介绍，请参考网站<https://pandas.pydata.org/>。本书的后续章节将对Pandas的功能进行深入介绍。

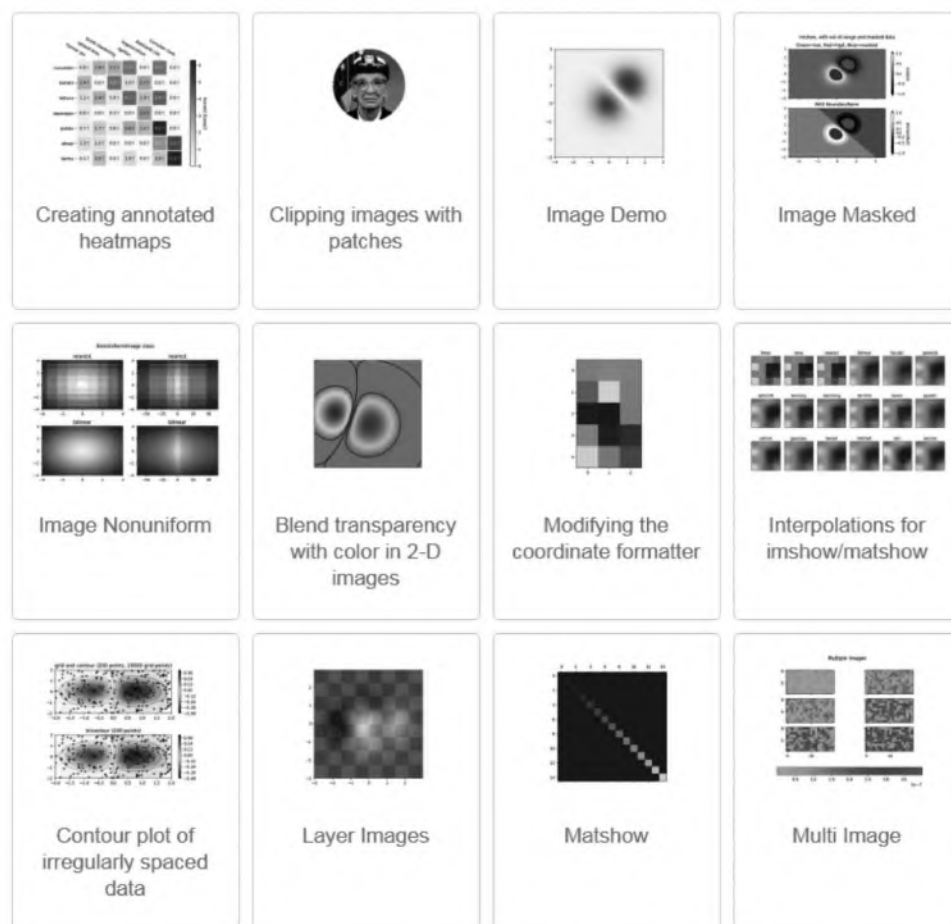


图2.8 图像、热图、等高图

2.2.8 Scikit-learn

Scikit-learn 的官方解释很简单：Machine Learning in Python，即用Python进行机器学习。Scikit-learn的优点也非常明显，具体如下。

- 构建于现有的NumPy、SciPy、Matplotlib、IPython、Pandas之上，做了易用性的封装。
- 简单且高效的数据挖掘、数据分析工具。
- 对所有人开放，且在很多场景易于复用。
- 基于BSD（Berkly Software Distribution）授权的开源实现。

Scikit-learn是一个非常强大的机器学习库，它包含了从数据预处理到训练模型的各个方面。Scikit-learn不仅提供了各种分类、聚类、回归、降维等经典算法的实现（如图2.10所示），同时还提供了机器学习中必需的数据预处理功能。使用Scikit-learn可以极大地节省开发人员编写代码的时间并减少程序代码量，使用户有更多的精力去分析数据分布、调整模型和优化超参数。

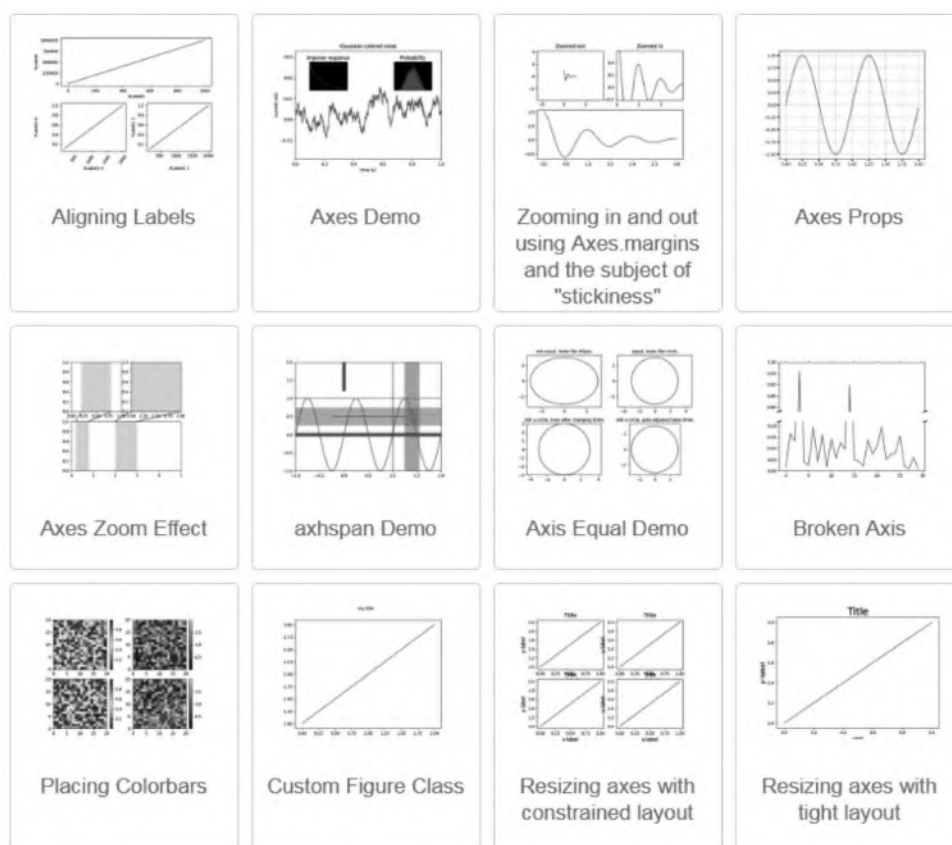


图2.9 子图功能

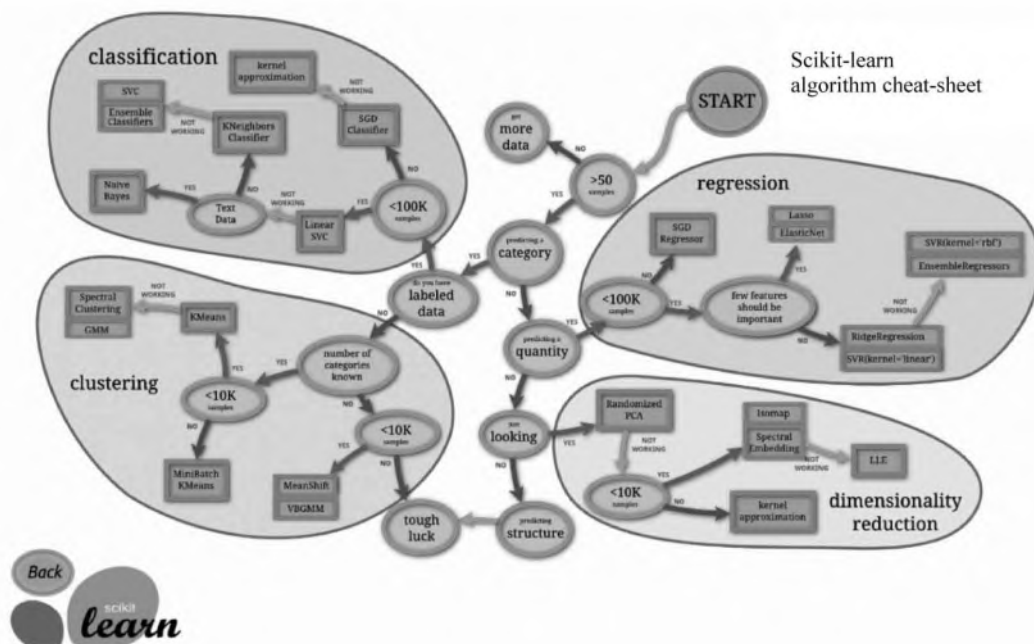


图2.10 Scikit-learn中的算法

关于Scikit-learn的更详细介绍请参考网站<https://scikit-learn.org/>。

2.2.9 NetworkX

网络的研究是运筹学的一个经典而重要的分支，是很多领域研究的基础工具，在交通运输、经济管理和工业工程等都有广泛的应用。从广义上讲，我们都生活在一个由网络组成的世界里，各种网络组成了现下复杂的生活环境。故而要分析身边各个领域的规律，就需要对其组成基础——网络的特性进行分析。NetworkX作为Python的一个开源包，便于用户对复杂网络进行创建、操作和学习。利用NetworkX可以以标准化和非标准化的数据格式存储网络、生成多种随机网络和经典网络、分析网络结构、建立网络模型、设计新的网络算法、进行网络绘制等。

关于 NetworkX 的更详细介绍请参考网站 <https://networkx.github.io/>。

2.2.10 PyMC3

历史上，帮助在不确定性下做出决策的方法之一就是使用概率推理系统。而计算机运算能力的提高，相应的编程工具的出现，使得概率编程逐渐成为可能。概率编程允许在用户自定义的概率模型上进行自动贝叶斯推断。新的MCMC（Markov Chain Monte Carlo）采样方法允许在复杂模型上进行推断。这类MCMC采样方法被称为HMC（Hamiltonian Monte Carlo），但是其推断需要的梯度信息有时候是无法获得的。而PyMC3是一个用Python编写的开源的概率编程框架，使用Theano通过变分推理进行梯度计算，并使用了C实现加速运算。不同于其他概率编程语言，PyMC3允许使用Python代码来定义模型。这种没有作用域限制的语言极大地方便了模型定义和直接交互。

关于PyMC3的更详细介绍请参考网站<https://docs.pymc.io/>。

2.2.11 数据科学领域中最新的一些Python包

近年随着数据科学、人工智能的快速发展，不断有新的基于Python的数据科学应用软件包出现，这里选取几个典型软件包进行简单介绍。

1. DASK

DASK是一个用于分析计算的灵活的并行计算库。因为DASK中的DataFrame与Pandas库中的相同，它的Array对象的工作方式类似于

NumPy，能够并行化以纯Python编写。因此，通过只更改几行代码，用户可以快速对现有代码进行并行处理。

2. TPOT

自动化机器学习（Automatic Machine Learning, AML）是一种流水线（也称管道），它能够让用户自动执行机器学习（Machine Learning, ML）问题中的重复步骤，从而节省时间，让数据科学家专注于使其专业知识发挥更高价值的工作。实际上，AML是在Scikit-learn中应用的网格搜索的扩展，而不是迭代这些值预先定义的集合和其组合，它通过搜索方法、特征、变换和参数值来获得最佳解决方案。因此，AML网格搜索不需要在可能的配置空间上进行详尽的搜索。TPOT就是提供了自动化机器学习功能的包，它提供了像遗传算法这样的应用，可用来在某个配置中混合各个参数并达到最佳设置。数据科学家甚至还可以把DASK与TPOT结合起来，进行分布式的自动化机器学习。

3. Keras

Keras 是一个高层神经网络应用程序接口（Application Programming Interface, API），由纯Python编写而成，用户可以选择Tensorflow、Theano以及CNTK后端。Keras开发的宗旨是为支持快速实验而生，能够把开发人员的想法迅速转换为结果。如果数据科学家需要完成简易和快速的原型设计，同时支持卷积神经网络、循环神经网络或二者的结合，以及CPU和GPU无缝切换，那么Keras将是最佳选择。

上述包的更详细介绍可以参考下列网站：

- DASK: <https://dask.org/>;

- TPOT: <https://epistasislab.github.io/tpot/>;
- Keras: <https://keras.io/>。

2.3 Anaconda的安装与使用

Python中除了前面已经介绍过的包以外，还有各种第三方包。这是一个优势，因为用户总可以找到某个特定功能的包，但这也给Python的用户带来了许多困扰。实际开发中经常会遇到这样的问题，开发某一应用使用的分别是A包的1.0.5版本，B包的2.0.6版本，C包的0.8版本，Python使用的是2.7版本；而另一应用则需要使用Python 3.6版本，A包的1.1.2版本，B包的2.2.0版本，C包的0.8版本，如何管理这些不同的环境是开发人员的一大难题。对数据分析人员而言，通常需要同时使用SciPy、NumPy、Matplotlib、Pandas、DASK、Scikit-learn等包，而不同包之间还存在兼容问题，如何快速完成这些包的安装和升级也是一个问题。

开源Anaconda正是为解决这些问题应运而生，它是在Linux、Windows和Mac OS X上搭建基于Python语言和R语言数据科学、机器学习环境的最简单方法。Anaconda在全球有超过1100万用户，利用它可以实现以下功能。

- 快速下载1500多个Python/R数据科学包。
- 使用Conda管理库、依赖项和环境。
- 使用Scikit-learn、TensorFlow和Theano开发和训练机器学习和深度学习模型。
- 使用DASK、NumPy、Pandas和Numba分析具有可伸缩性的数据。
- 使用Matplotlib、Bokeh、Datashader和Holoviews可视化结果。

2.3.1 安装Anaconda

首先访问网站<https://www.anaconda.com/distribution/>，根据对应的操作系统和期望的Python版本，单击Download按钮完成Anaconda应用的下载，如图2.11所示。

本书以Windows操作系统，Python 3.7版本的Anaconda为例进行讲解，针对其他系统的安装，读者可参考网站<https://docs.anaconda.com/anaconda/install/>。

(1) 在Windows系统中完成Anaconda程序的下载后，只需要直接运行Anaconda3-2019.03-Windows-x86_64.exe，单击Next按钮，如图2.12所示。

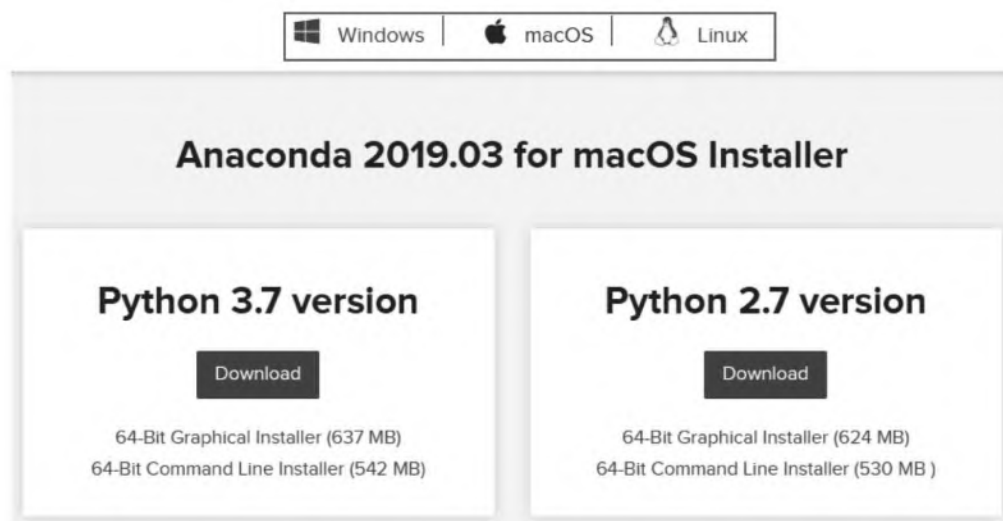


图2.11 Anaconda下载

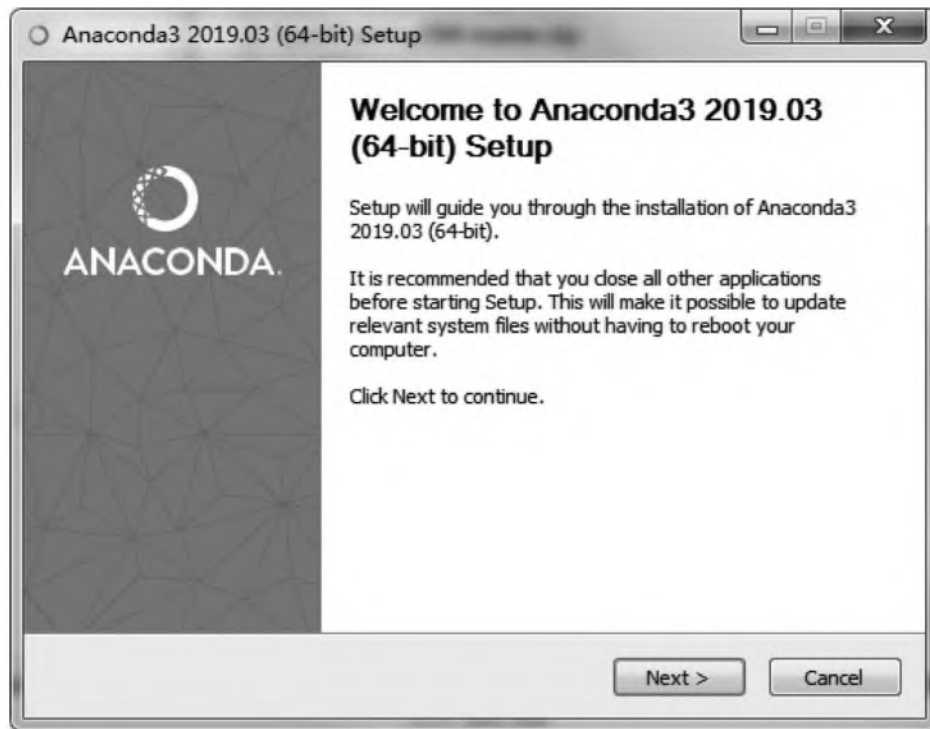


图2.12 安装Anaconda

(2) 选择 “I Agree” 接受用户许可协议，进入如图2.13所示的界面，选择安装类型是当前用户使用还是系统全部用户都可以使用。第二种情况需要具有Windows系统管理员权限。

(3) 选择程序的安装目录，如图2.14所示。

(4) 选择是否把Anaconda注册为系统默认的Python环境，除非想在系统里面安装多个不同版本的Anaconda或Python，那么通常建议使用系统默认推荐，如图2.15所示。

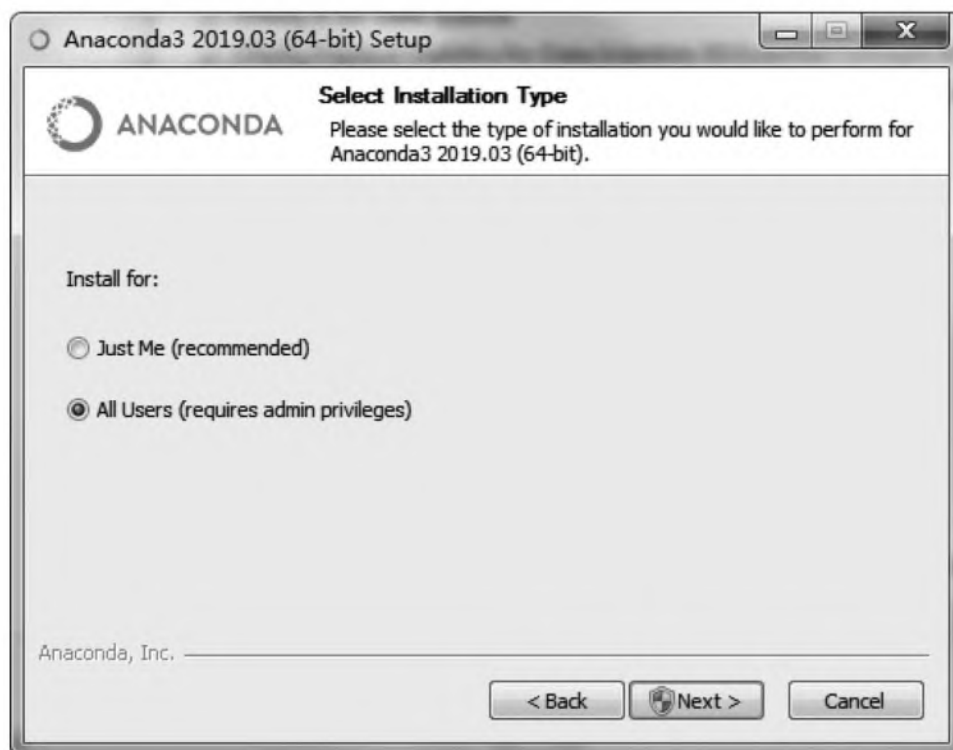


图2.13 选择安装类型

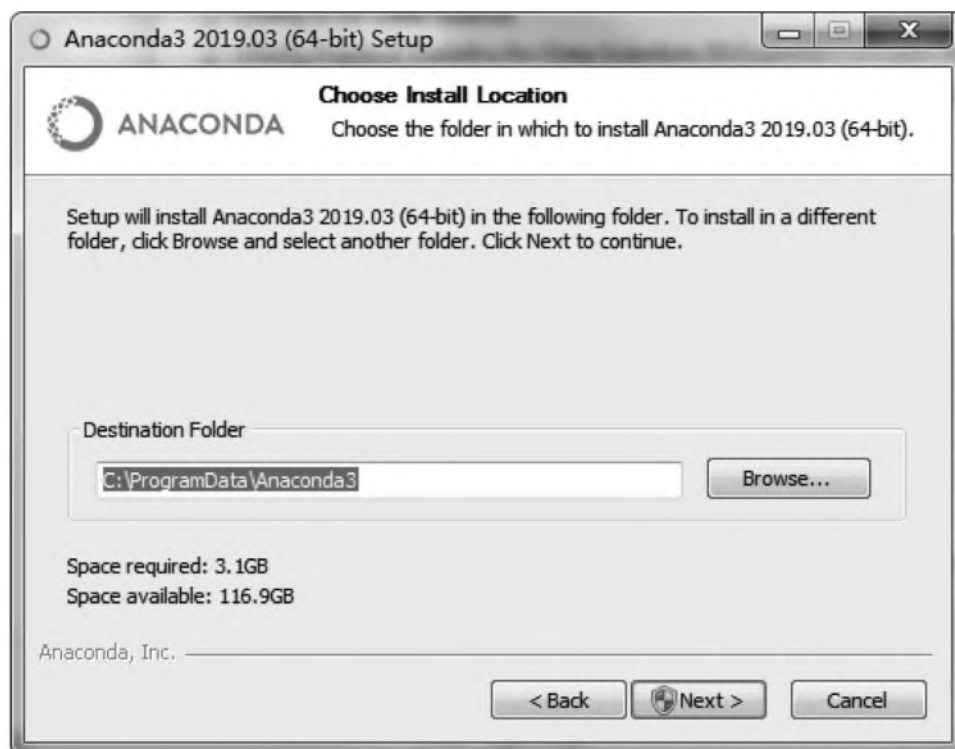


图2.14 选择安装目录

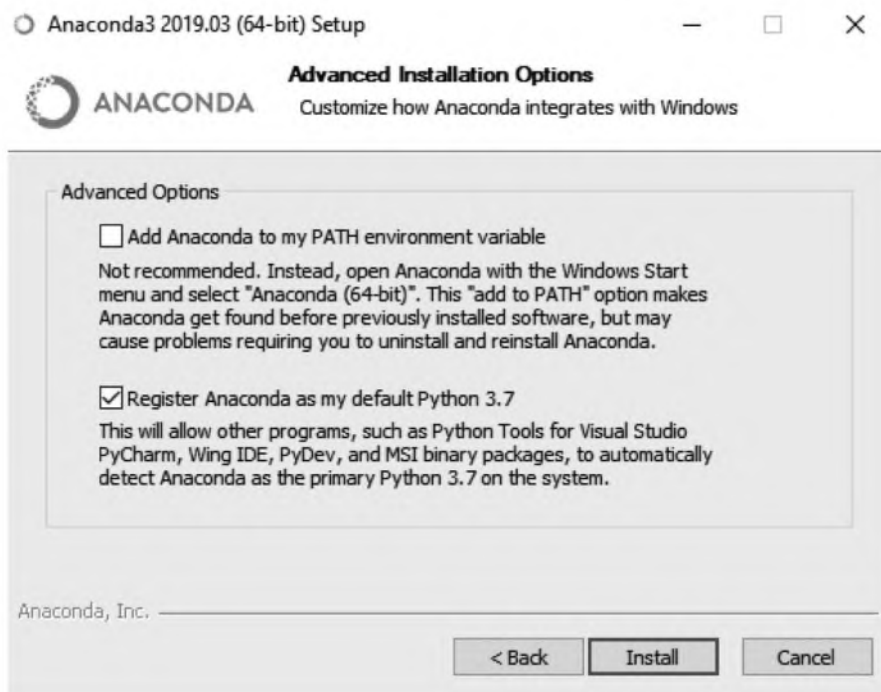


图2.15 注册Anaconda

（5）单击Install按钮后，Anaconda将开始安装，当完成安装后，程序将提示是否安装PyCharm，如图2.16所示，这里直接单击Next按钮，进入下一步。



图2.16 PyCharm安装

(6) 直接单击Finish按钮完成Anaconda安装，如图2.17所示。



图2.17 完成Anaconda安装

成功安装Anaconda后，可以在Windows的开始菜单中找到Anaconda的启动菜单，如图2.18所示。



图2.18 Anaconda启动菜单

2.3.2 利用Conda管理Python环境

Conda是Anaconda提供的包及其依赖项和环境的管理工具，利用它可以：

- 快速安装、运行和升级包及其依赖项；
- 在计算机中便捷地创建、保存、加载和切换环境。

有了Anaconda就可以利用它提供的Conda命令来管理环境，从而解决前面提到的Python版本问题和各种数据分析中用到的包管理问题。要使用Conda，首先需要在系统菜单中单击“Anaconda Prompt”进入

命令行模式，如图2.19所示（对Linux/Mac OS，Conda命令可以直接在命令行运行）。需要注意的是，这里是以管理员身份运行的“Anaconda Prompt”。

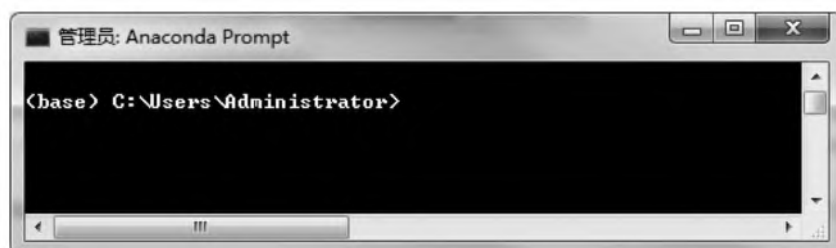


图2.19 Anaconda Prompt

进入Anaconda命令行模式后的第一件事就是更新Anaconda，在命令行输入：

```
conda update conda
```

并在提示是否更新的时候输入y（Yes）让更新继续。

1. 创建环境

完成了Conda更新，接下来就可以创建一个新的环境了，默认打开Anaconda命令行的时候进入的是系统的base环境（图2.19），通常开发中会创建不同的环境以满足不同应用对各种Python版本以及第三方包的要求。要创建一个新的环境，可以在命令行输入：

```
conda create-n env_name package_names
```

上面的命令中，env_name是设置环境的名称（-n指该命令后面的env_name是要创建环境的名称），package_names是要安装在所创建环境中的包名称，同时还可以指定版本。例如，要创建名为py37的环境，并在其中安装最新的Python版本，可以在终端中输入：

```
conda create-n py37 python=3
```


此时Conda将创建一个全新的环境，环境名为py37，使用的Python版本为3，由于没有指定小版本号，此时系统将选择安装最新的Python 3版本，本书写作时最新版本是3.7.3，如图2.20所示。

同样地，如果想创建一个环境，使用Python 3.6，同时安装Pandas 0.24.0版本，以及NumPy包，可以通过如下命令完成。

```
conda create -n py36 python=3.6 numpy pandas=0.24.0
```

如果想在创建环境同时安装其他的包，只需要在创建环境时添加对应的包就可以了。

```
管理员: Anaconda Prompt - conda create -n py37 python=3
(base) C:\Users\Administrator>conda create -n py37 python=3
Collecting package metadata: done
Solving environment: done

## Package Plan ##

environment location: C:\ProgramData\Anaconda3\envs\py37

added / updated specs:
- python=3

The following packages will be downloaded:

package | build | size
-----|-----|-----
pip-19.1 | py37_0 | 1.8 MB
python-3.7.3 | h8c8aaf0_1 | 17.8 MB
setuptools-41.0.1 | py37_0 | 680 KB
sqlite-3.28.0 | he774522_0 | 945 KB
vs2015_runtime-14.15.26706 | h3a45250_4 | 2.4 MB
Total: 23.5 MB

The following NEW packages will be INSTALLED:

ca-certificates | pkgs/main/win-64::ca-certificates-2019.1.23-0
certifi | pkgs/main/win-64::certifi-2019.3.9-py37_0
openssl | pkgs/main/win-64::openssl-1.1.1b-he774522_1
pip | pkgs/main/win-64::pip-19.1-py37_0
python | pkgs/main/win-64::python-3.7.3-h8c8aaf0_1
setuptools | pkgs/main/win-64::setuptools-41.0.1-py37_0
sqlite | pkgs/main/win-64::sqlite-3.28.0-he774522_0
vc | pkgs/main/win-64::vc-14.1-h0510ff6_4
vs2015_runtime | pkgs/main/win-64::vs2015_runtime-14.15.26706-h3a45250_4
wheel | pkgs/main/win-64::wheel-0.33.1-py37_0
wincertstore | pkgs/main/win-64::wincertstore-0.2-py37_0

Proceed [Y]/N)?
```

图2.20 创建py37环境

2. 切换环境

在Windows上，可以使用`activate my_env`进入对应的环境；而在OS X/Linux上则使用`source activate my_env`进入环境。进入环境后，用户会在终端提示符中看到环境名称。图2.21所示为进入前面创建的py37环境。

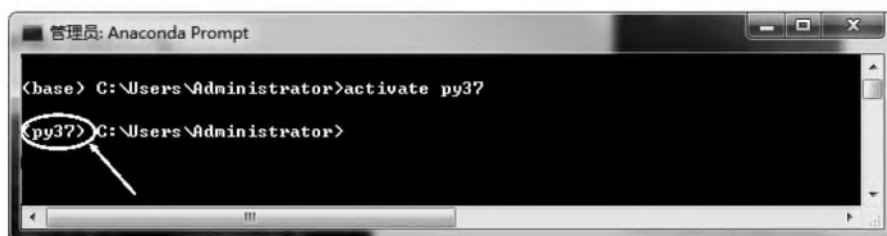


图2.21 进入py37环境

进入环境后可以通过命令`conda list`来列出当前环境已经安装的包，以及对应版本，如图2.22所示。

可以用下面的命令来检查当前环境的Python版本。

```
python -version
```

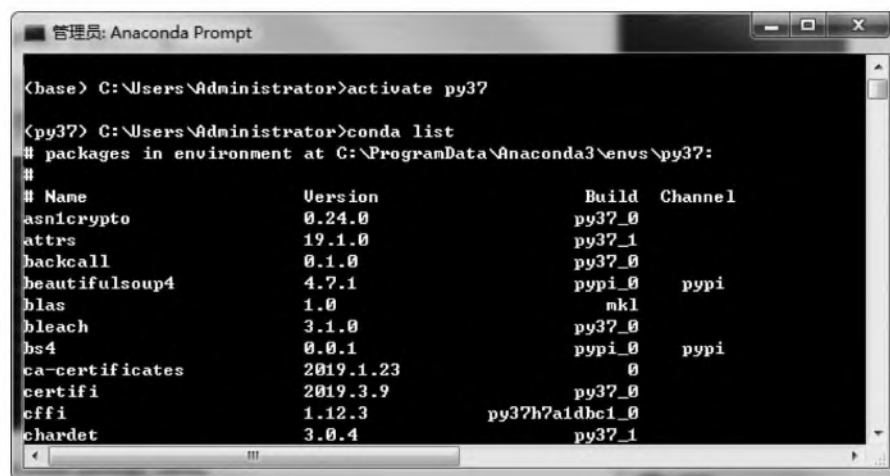


图2.22 列出当前环境中的包

这里要特别强调的是不同的环境类似于在系统里面建立了隔离沙箱，相互之间不影响，这样开发人员就可以通过切换不同环境满足不

同的开发需求。

3. 离开环境

要离开当前环境，在命令行输入：

```
conda deactivate
```

4. 环境共享

环境共享是非常实用的功能，它能保证让整个项目的协作人员都使用相同的软件包，并确保这些包的版本正确。例如，当前数据分析师李四正在进行网络促销数据分析，他需要提交应用给另一部门的张三来部署项目，但是张三并不知道数据分析时使用的是哪个Python版本，以及使用了哪些包和包的版本。这时应该怎么办呢？李四就可以在当前的环境终端输入：

```
conda env export > environment.yaml
```

将当前的环境的配置（包括Python版本和所有包的名称）保存到一个YAML文件中。命令的第一部分`conda env export`用于输出环境中所有包的名称（包括Python版本），第二部分是对应的文件名。那么张三拿到了导出的环境文件，在其他电脑环境中如何使用呢？首先在Conda中进入当前的环境，如：

```
activate py37
```

然后再使用以下命令更新环境。

```
conda env update -f=/path/to/environment.yml
```

其中，`-f`表示要使用的环境文件在本地的路径，读者将`/path/to/environment.yml`替换成本地的实际路径即可。

5. 列出环境

有时用户可能会忘记自己创建的环境名称，这时就可以用`conda env list`命令来列出本地创建的所有环境。用户会看到本地所有环境的列表，而当前所在环境的旁边会有一个星号，Anaconda命令行默认的环境（即还没有选定环境时使用的环境）名为`base`，如图2.23所示。

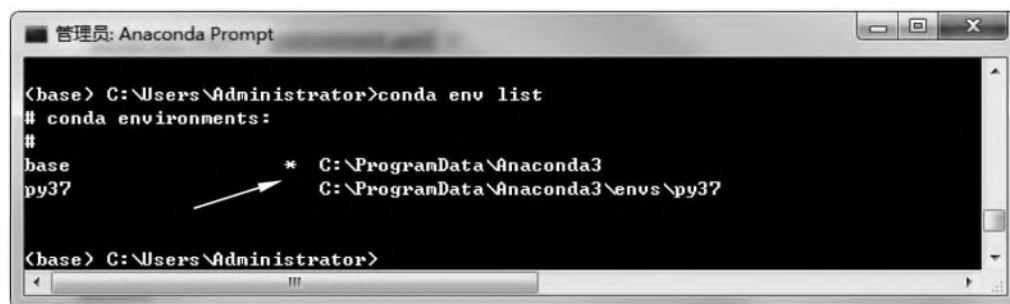


图2.23 列出环境

6. 删除环境

如果不再使用某个环境，可以使用以下命令删除指定的环境，这里的`env_name`为想删除的本地环境名。

```
conda env remove -n env_name
```

7. 查看环境信息

如果想了解当前环境，可以使用以下命令。

```
conda info
```

此命令会列出该环境的所有信息，如图2.24所示。

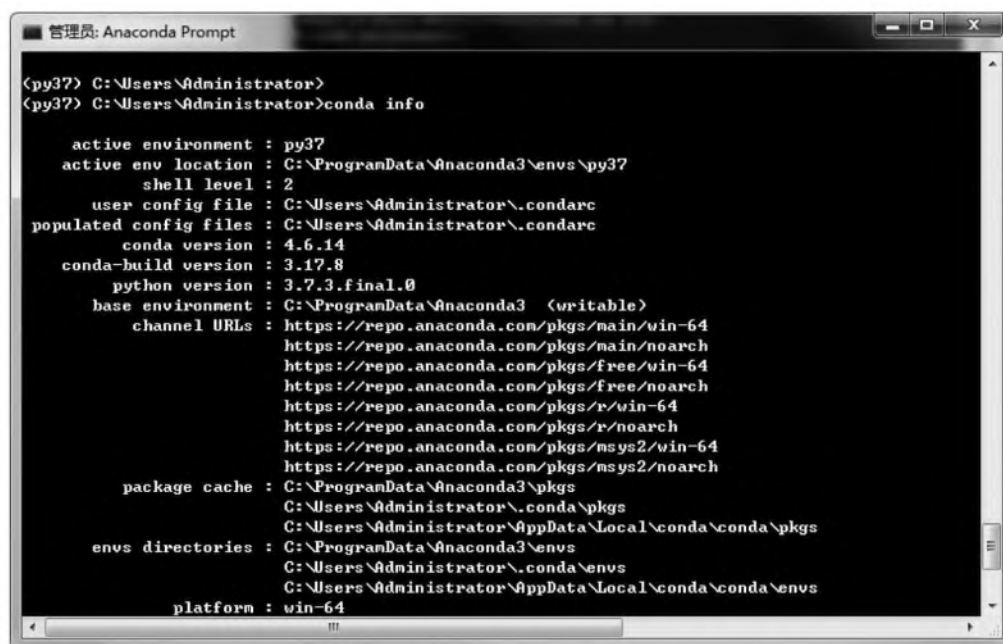


图2.24 查看环境信息

2.3.3 利用Conda管理Python包

创建好对应的工作环境后，通常还需要对环境中的包进行管理。利用Conda命令可以实现各种包的安装、升级以及卸载。

1. 安装包

例如，想安装requests这个包，进入创建好的py37环境后，在命令行模式下输入：

```
conda search requests
```

将在Anaconda提供的库（repository）里面搜索这个包是否存在，以及有哪些版本。之后可以利用如下命令来完成该包的安装。

```
conda install requests
```

在安装时用户也可以指定对应的requests版本。此外，如果在Anaconda提供的库里面找不到想安装的包，或者想安装更新的版本，那么也可以通过社区维护的conda-forge来安装。例如，如果想使用conda-forge来安装Pandas，可以使用如下命令。

```
conda install -c conda-forge pandas
```

如果在上面的库都无法找到想安装的包，也可以用标准的Python包管理命令pip来完成在当前环境中第三方包的安装。例如，用来获取国内财经以及股票数据的tushare包，就可以通过在当前环境下使用如下命令来完成安装。

```
pip install tushare
```

2. 包的卸载与升级

包的卸载可以通过如下命令完成。

```
conda uninstall packages_name
```

这里的packages_name就是当前环境中要删除的包。如果想在当前环境中删除另一个环境中的包，可以通过如下命令完成。

```
conda uninstall my_env packages_name
```

这里的my_env就是对应的环境名。同时，对于任何Conda命令都可以通过获取帮助的方式来详细了解命令的使用方法，如：

```
conda uninstall -help  
conda install -help
```

如果想对已经安装的包进行升级，则可以输入：

```
conda update my_env packages_name
```

如果是对当前环境中包升级，则可以略去环境名。如果想对所有包升级，则可以输入：

```
conda update-all
```

3. 为Anaconda添加新的库

有时在国内使用Anaconda提供的库来安装包会比较缓慢，这时可以考虑添加国内镜像来解决这个问题。例如，通过如下命令可以使用中国科学技术大学的镜像。

```
conda config--add channels
https://mirrors.ustc.edu.cn/anaconda/pkgsg/free/
conda config --set show_channel_urlsyes
```

上面的第二个命令是显示当前有哪些镜像地址。可以通过如下命令来将刚才添加的镜像地址移除，用conda config-show来确认该地址已经移除。

```
conda config --remove channels
'https://mirrors.ustc.edu.cn/anaconda/pkgsg/free/'
conda config-show
```

2.3.4 安装本书所需的包

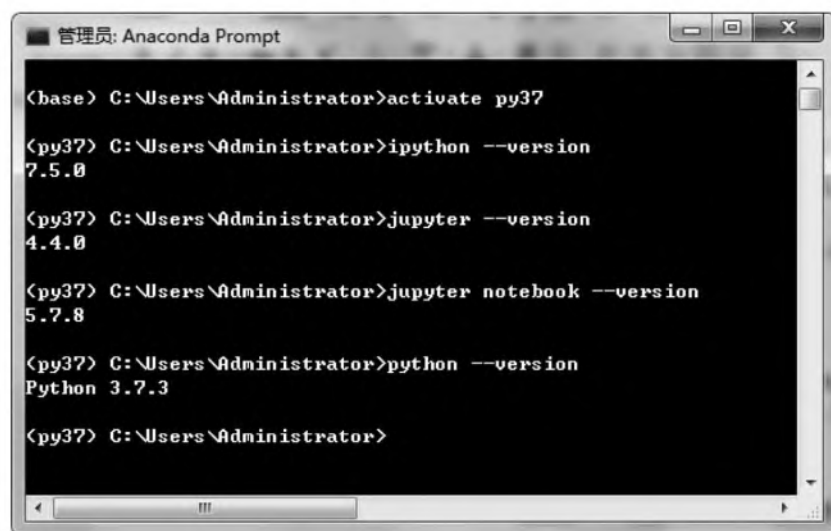
前面已经创建好了一个使用Python 3.7的环境py37，在Anaconda命令行模式使用activate py37进入该环境，接下来需要在该环境中安装本书所需的软件包。在命令行模式输入：

```
conda install scipynumpy statsmodels pandas scikit-learn
matplotlib seaborn ipython jupyter
```

即可完成所有本书中用到的包的安装。本书中用到的包的版本可以通过如下方式获得。

```
python --version
ipython--version
jupyter--version
jupyter notebook --version
```

分别输入上述命令，可以获得环境中用到的包的版本信息，如图2.25所示。除此之外，也可以利用conda list方式获得版本信息，如conda list ipython可以查询IPython包的信息。

A screenshot of an Anaconda Prompt window titled "管理员: Anaconda Prompt". The window shows a series of commands and their outputs. The first command is "activate py37", which changes the prompt from "(base)" to "(py37)". Subsequent commands are "ipython --version" (output: 7.5.0), "jupyter --version" (output: 4.4.0), and "jupyter notebook --version" (output: 5.7.8). The final command is "python --version" (output: Python 3.7.3). The prompt then returns to "(py37)".

```
(base) C:\Users\Administrator>activate py37

(py37) C:\Users\Administrator>ipython --version
7.5.0

(py37) C:\Users\Administrator>jupyter --version
4.4.0

(py37) C:\Users\Administrator>jupyter notebook --version
5.7.8

(py37) C:\Users\Administrator>python --version
Python 3.7.3

(py37) C:\Users\Administrator>
```

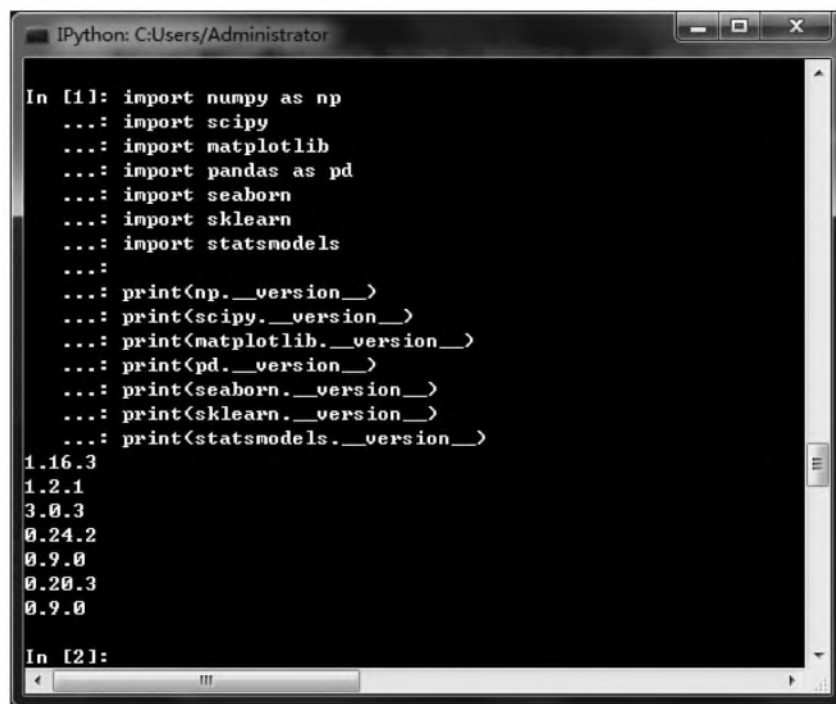
图2.25 检查软件包版本

对于Pandas、SciPy等包的版本信息，可以首先在命令行输入ipython，进入IPython的交互模式后查询，具体代码如下。


```
>>> import numpy as np
>>> import scipy
>>> import matplotlib
>>> import pandas as pd
>>> import seaborn
>>> import sklearn
>>> import statsmodels

>>> print(np.__version__)
>>> print(scipy.__version__)
>>> print(matplotlib.__version__)
>>> print(pd.__version__)
>>> print(seaborn.__version__)
>>> print(sklearn.__version__)
>>> print(statsmodels.__version__)
```

输入上述代码，将看到如图2.26所示的输出。



```
IPython: C:\Users\Administrator

In [1]: import numpy as np
...: import scipy
...: import matplotlib
...: import pandas as pd
...: import seaborn
...: import sklearn
...: import statsmodels
...:
...: print(np.__version__)
...: print(scipy.__version__)
...: print(matplotlib.__version__)
...: print(pd.__version__)
...: print(seaborn.__version__)
...: print(sklearn.__version__)
...: print(statsmodels.__version__)
1.16.3
1.2.1
3.0.3
0.24.2
0.9.0
0.20.3
0.9.0

In [2]:
```

图2.26 Pandas等的版本信息

2.4 使用Jupyter Notebook进行可重复数据分析

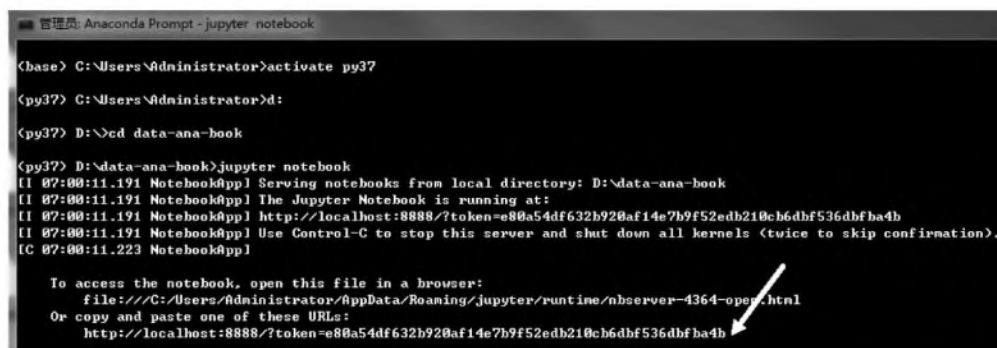
说到每个数据科学家都应该使用或必须了解的工具，那非Jupyter Notebook莫属了（之前也被称为IPython笔记本）。Jupyter Notebook功能强大，数据科学家可以在其中进行数据可视化，也可以创建和共享他们的文档，无论代码还是整个报告都可以利用它完成。Jupyter Notebook能帮助数据科学家简化工作流程，实现更高的生产力和更便捷的协作，因此是数据科学家最常用的工具之一。本书在开始正式的数据分析学习之前将先简单介绍Jupyter Notebook的使用。需要说明的是，本书的代码全部在Jupyter Notebook中完成，同时也将以Notebook的形式提供给读者下载。

2.4.1 Jupyter Notebook的配置

在py37环境中已经安装了Jupyter Notebook，要运行它，只需在进入Anaconda命令行模式后，输入：

```
activate py37
jupyter notebook
```

我们将看到如图2.27所示的提示，此时只需要按照提示把箭头所指的那一行复制到浏览器，就可以看到Jupyter Notebook的启动页面。



```
管理员: Anaconda Prompt - jupyter notebook

(base) C:\Users\Administrator>activate py37

(py37) C:\Users\Administrator>d:

(py37) D:\>cd data-ana-book

(py37) D:\data-ana-book>jupyter notebook
[I 07:00:11.191 NotebookApp] Serving notebooks from local directory: D:\data-ana-book
[I 07:00:11.191 NotebookApp] The Jupyter Notebook is running at:
[I 07:00:11.191 NotebookApp] http://localhost:8888/?token=e80a54df632b920af14e7b9f52edb210cb6dbf536dbfba4b
[I 07:00:11.191 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 07:00:11.223 NotebookApp]

To access the notebook, open this file in a browser:
    file:///C:/Users/Administrator/AppData/Roaming/jupyter/runtime/nbserver-4364-open.html
Or copy and paste one of these URLs:
    http://localhost:8888/?token=e80a54df632b920af14e7b9f52edb210cb6dbf536dbfba4b
```

图2.27 启动Jupyter Notebook

每次都复制一遍会比较麻烦，所以我们通常会修改配置，采用密码方式启动Jupyter Notebook，同时也会修改Jupyter Notebook从工作目录启动，具体方法如下。

1. 生成一个Notebook配置文件

默认情况下，配置文件 `jupyter_notebook_config.py` 是不存在的，使用如下命令生成配置文件。

```
jupyter notebook --generate-config
```

该命令将在当前计算机中生成一个新的配置文件，对于Windows用户，通常它位于 `C:\Users\Administrator\.jupyter`（这里用的是管理员身份，所以是Administrator用户，如果是其他用户，Administrator通常会替换为对应的用户名）；对于Linux用户，通常它位于 `~/.jupyter/jupyter_notebook_config.py`（如果是以Root用户运行命令，需要使用 `jupyter notebook--generate-config--allow-root`）。

2. 生成密码

从Jupyter Notebook 5.0版本开始，Jupyter Notebook提供了一个命令来设置密码，在Anaconda命令行模式输入：

```
jupyter notebook password
```

根据提示输入你的密码，假设这里输入的密码为“datatbook”，之后生成的密码存储在文件 `jupyter_notebook_config.json` 中，通常该文件和前面生成的配置文件位于同一目录。打开该文件，可以发现文件中 `password` 后面有一段密码，例如作者的 `jupyter_notebook_config.json` 文件内容为：

```
{
  "NotebookApp": {
    "password": "sha1:0223a0449c12:796970c4a4f4d6b5c2de90b077e7372ed59a5188"
  }
}
```

3. 修改配置文件

在 `jupyter_notebook_config.py` 中找到 `c.NotebookApp.password` 所在行，取消注释，并将前面提到的密码复制到后面，代码如下。

```
c.NotebookApp.password =
'sha1:0223a0449c12:796970c4a4f4d6b5c2de90b077e7372ed59a5188'
```

此外还可通过修改 `c.NotebookApp.port` 来自行指定 Notebook 运行的端口。

4. 修改 Notebook 启动时的工作目录

在 `jupyter_notebook_config.py` 中找到下面的行：

```
## The directory to use for notebooks and kernels.
#c.NotebookApp.notebook_dir= ''
```

取消 `c.NotebookApp.notebook_dir` 行的注释，并将目录修改为想要的目录，例如本书中使用的目录为 `D:\data-ana-book\notebook`，那么需要将该行修改为：

```
c.NotebookApp.notebook_dir = 'D:\\data-ana-
book\\notebook'
```

需要说明的是，在 Windows 系统下，由于转义符的原因需要使用“\\”，对于 Linux 是不需要这样做的。现在我们已经为 Jupyter Notebook 设置了密码，并配置了默认的启动目录，再次在 Anaconda 命令行模式输入：

```
jupyter notebook
```

将看到浏览器启动，并提示输入密码，此时输入前面设置的密码“databook”，将进入Jupyter Notebook界面，如图2.28所示。

由于当前工作目录D:\data-ana-book\notebook下面没有任何文件，所以没有任何Notebook显示出来。此时可以通过单击右侧“New”菜单，选择“Python 3”来创建新的Notebook。细心的读者可能会发现，前面不是创建了一个py37的环境吗？为什么这里没有提供用该环境来创建Notebook的选项？要使Jupyter Notebook支持虚拟环境，还需要安装一个新的包：nb_conda。首先回到命令行窗口，连续按两次Ctrl+C组合键来退出Jupyter Notebook，然后输入：

```
conda install nb_conda
```



图2.28 Jupyter Notebook启动界面

完成nb_conda包的安装后，再次启动Jupyter Notebook，在右侧菜单选择“New”，就可以看到如图2.29所示的界面。

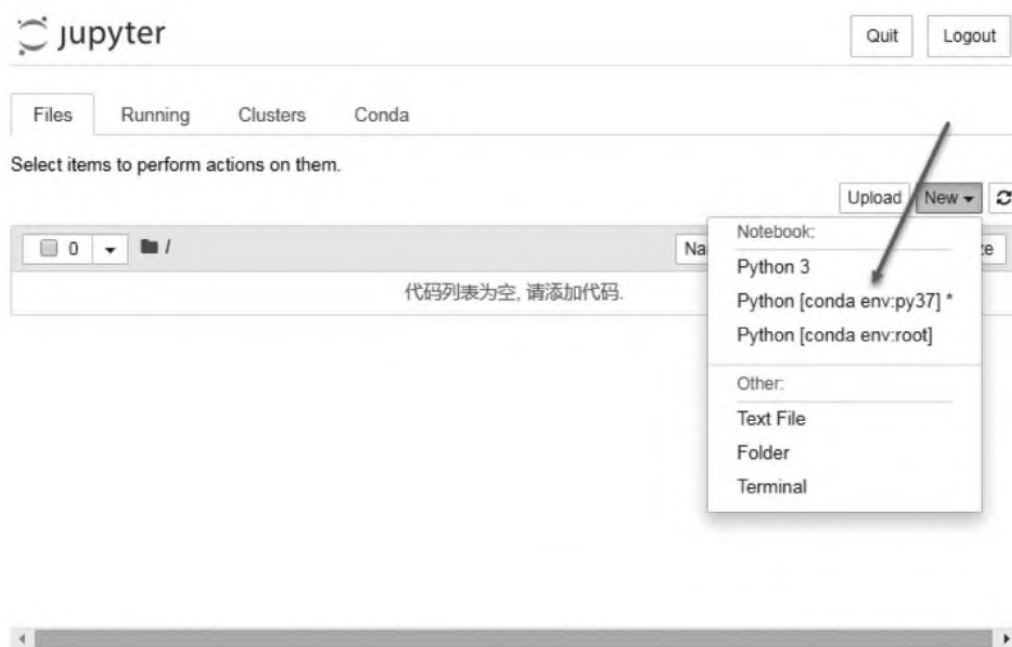


图2.29 支持虚拟环境的Jupyter

下拉菜单中出现了之前创建的py37环境选项以及Anaconda安装时默认创建的环境选项。切换窗口回到Anaconda命令提示窗口，可以看到Jupyter启动时也确实使用的是前面设置的目录D:\data-ana-book\notebook，如图2.30所示。

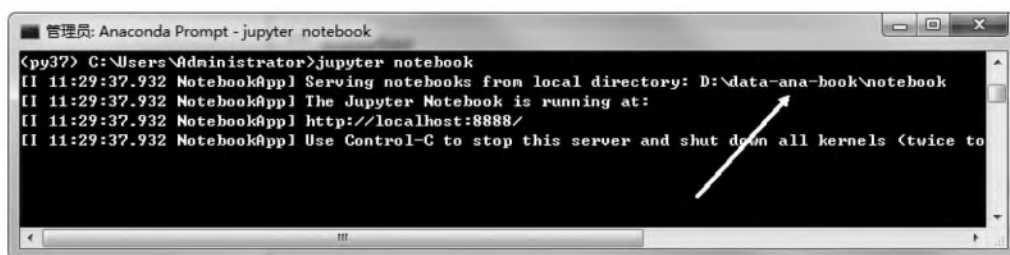


图2.30 Jupyter默认启动目录

2.4.2 Jupyter Notebook中的单元格

首先进入py37环境运行命令 `jupyter notebook`，单击右侧菜单“New”，选择使用py37环境创建新的Notebook，此时浏览器将在新的标签页中创建一个Notebook。该Notebook的文件名为Untitled，可以单击Notebook中的“Untitled”对文件名进行修改，如图2.31所示。

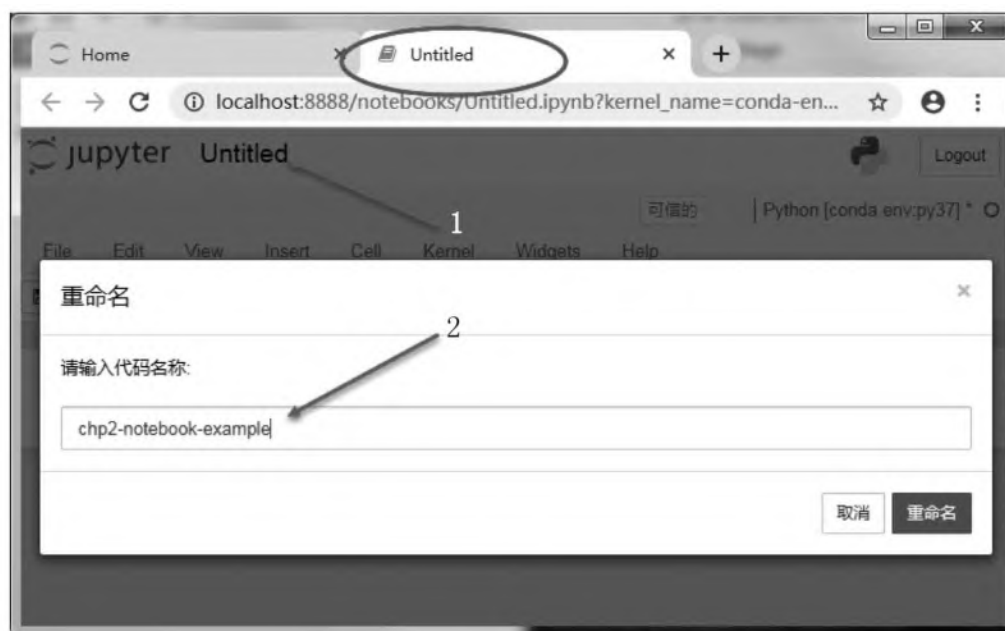


图2.31 修改Notebook名称

单击“重命名”按钮确认修改，就可以开始编辑这个名为chp2-notebook-example的文件了，如图2.32所示。

可以看到最顶部是对应的Notebook的名称，接下来是菜单栏 / 工具栏，读者可以通过菜单栏 / 工具栏执行选择不同操作。接下来就是Notebook中最重要的部分——单元格（Cell），单元格可分为3种不同类型：Code、Markdown和Raw NBConvert，这里需要重点掌握前面两种。如果一个单元格为Code类型，那么这意味着在这个单元格中可以输入一行或多行的Python代码，同时可以运行该代码。如图2.33所示，我们在Code单元格中输入了一行代码，然后单击工具栏中的“运行”按钮，Notebook中将打印输出：My first line of code in Jupyter！

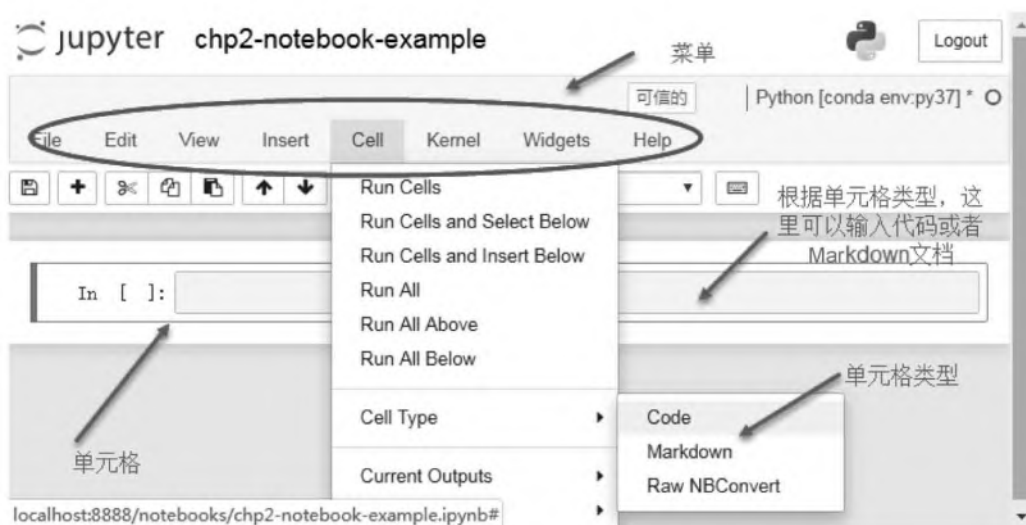


图2.32 Notebook界面介绍

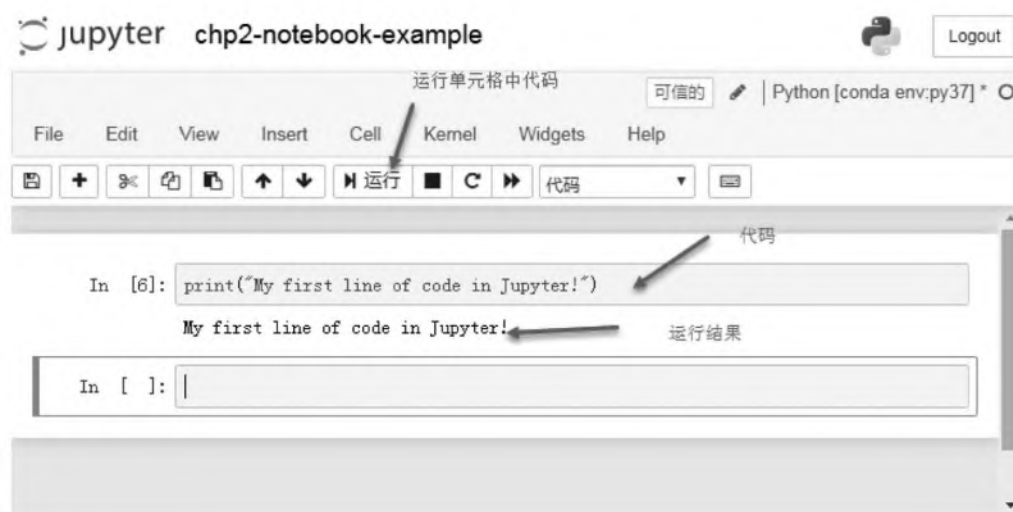


图2.33 运行单元格中代码

而如果单元格类型为Markdown，那么输入的就是Markdown格式的文档，Notebook将按照Markdown文档的规范显示该段文档。例如，在新的单元格中输入如下内容。

```
# 搭建数据科学开发环境
## 为什么选择Python?
### 人生苦短，我用Python
Python开发领域流传着一句话：“人生苦短，我用Python”，很多人还把它印在了衣服上，如图2.1所
```


示. 某种程度这也表明了Python的功能强大与简洁
![人生苦短](1.jpg)

选择运行单元格，将看到如图2.34所示的内容。



图2.34 Markdown单元格

这里的单元格使用了Markdown语法，其中：

- #代表一级标题；
- ##代表二级标题，以此类推；
- ![]（图片地址）是一个图片链接。

关于Markdown语法更详细的说明可以参考网站<https://markdown-zh.readthedocs.io/en/latest/>。

2.4.3 Jupyter Notebook中的命令模式与编辑模式键

Jupyter Notebook有两种模式：编辑模式与命令模式。编辑模式下可以输入代码或文档，而命令模式下可以执行Jupyter Notebook命令。在编辑和命令模式之间切换，分别使用Esc键和Enter键。无论当前为何种模式，按下Esc键就可以进入命令模式，此时：

- 按Up和Down键可以向上和向下移动单元格；
- 按A键在活动单元格上方插入一个新单元格；
- 按B键在活动单元格下方插入一个新单元格；
- 按M键将活动单元格转换为Markdown单元格；
- 按Y键将激活的单元格设置为一个代码单元格；
- D+D（按两次D键）将删除活动单元格；
- 按Z键将撤销单元格删除；
- 按住Shift键，同时按Up或Down键，一次选择多个单元格；
- 按下Ctrl+Shift+-组合键，在编辑模式下，将在光标处拆分活动单元格；
- 选中单元格，按下Enter键将进入编辑模式。

更多的快捷键命令可以通过在命令模式下按H键来获得帮助，即首先按Esc键，然后按H键，此时将看到Jupyter帮助页面，如图2.35所示。



图2.35 Jupyter帮助页面

2.4.4 使用Jupyter Notebook进行数据分析

在 py37 环境中启动 Jupyter Notebook，并创建一个新的 Notebook，文件名为 chp2-adult-data-analysis。按 Esc 键进入命令模式，在命令模式按 M 键将单元格设置为 Markdown 类型。在该单元格中按下 Enter 键进入编辑模式，输入：

```
# 业务理解
"adult dataset", 也称为"Census Income"数据集来自1994年人口普查数据库.人口普查是一项广泛的活动...包括但不限于人口统计状况（使用最终，年龄，性别，原籍国），社会和经济状况（教育和就业）相关数据)等.
# 数据统计
```

按下Shift+Enter组合键运行该单元格，并自动在当前单元格下方创建一个新代码单元格（如果按Ctrl+Enter组合键将只运行单元格，不创建新单元格）。在新单元格中输入：

```
# Loading required modules
>>> import numpy as np
>>> import pandas as pd
>>> import seaborn as sns
>>> import matplotlib.pyplot as plt
>>> %matplotlib inline

# loading the data
>>> file_name = "../data/adult.data"

>>> columns = ["age", "work-class", "fnlwt", "education", \
               "education-num", "marital-status", "occupation", "relationship", \
               "race", "sex", "capital-gain", "capital-loss", "hours-per-week", \
               "native-country", "income"]
>>> data = pd.read_csv(file_name, names=columns, sep=',', \
                       na_values='?', skipinitialspace=True)
```

再次按下Shift+Enter组合键，在新单元格中输入：

```
>>> data.head()
```

再次按下Shift+Enter组合键，此时运行结果如图2.36所示。可以看出，Notebook既是文档，又是代码，显著提高了数据分析与交流的效率。

接下来按Esc键进入命令模式，再次按B键在当前单元格输入：

```
# Data Visualization
```

按Esc键，再按M键将单元格变成Markdown类型单元格。按Shift+Enter组合键后，在新单元格输入：

```
# Plotting histogram for numerical values
>>> numerical_attributes = data.select_dtypes(include = ['int64'])
>>> numerical_attributes.hist(figsize = (12,12))
```



图2.36 Notebook运行结果

按Shift+Enter组合键运行该单元格，将看到Notebook中出现如图2.37所示的内容。

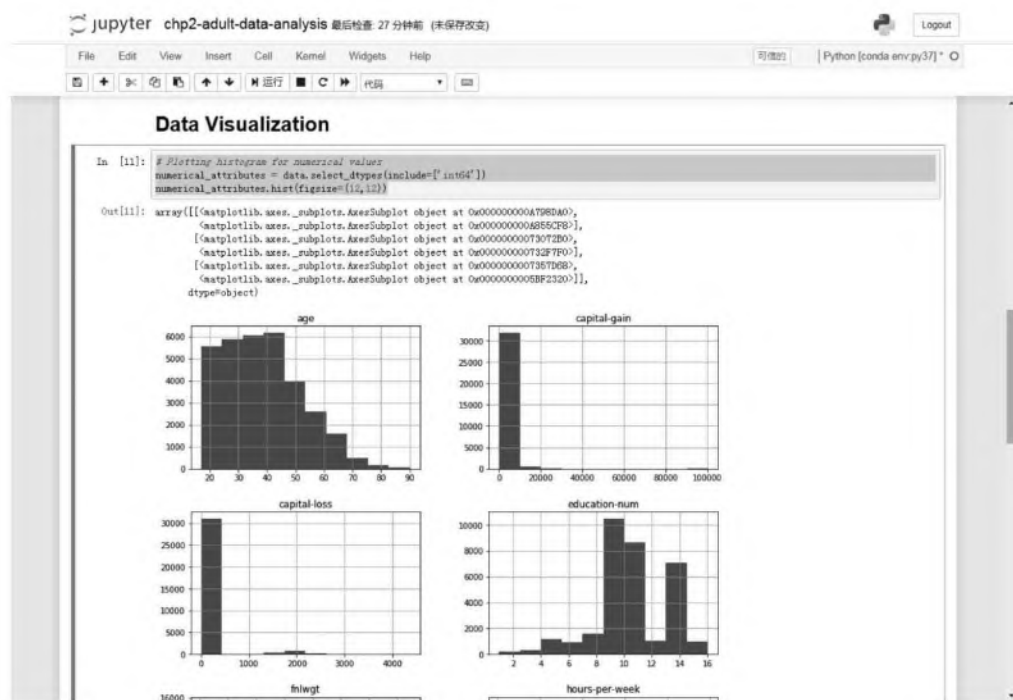
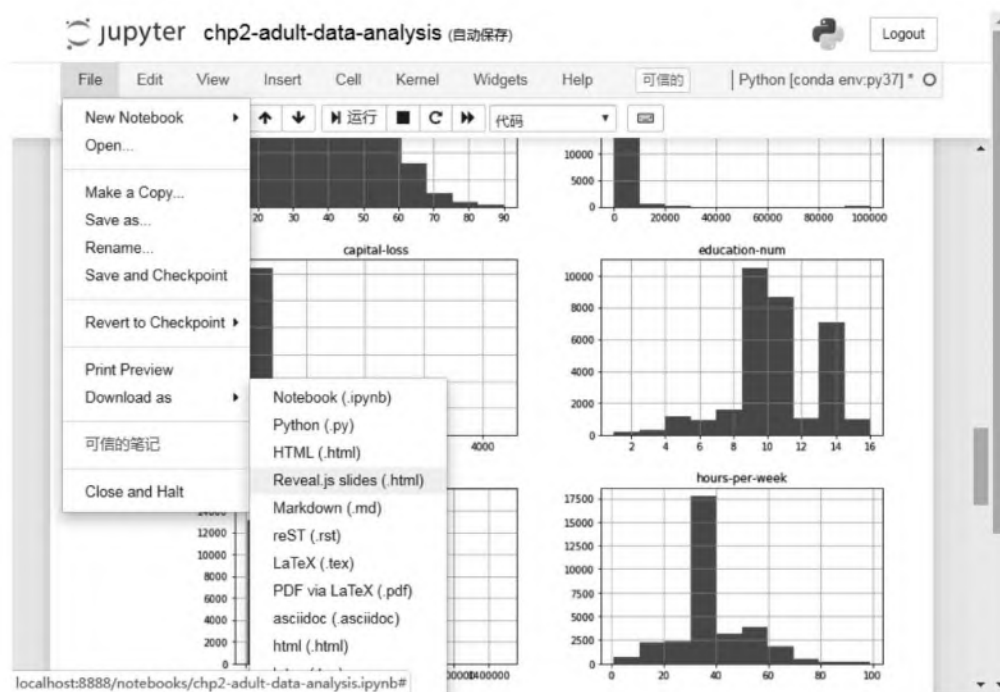


图2.37 Notebook中绘图

可见，在Notebook中不仅可以运行代码，还可以很好地完成数据可视化。最后，当完成了Notebook代码后，我们既可以将结果以Notebook格式（即保存为ipynb格式文件）分享给其他人，也可以输出为HTML、PDF等格式，这样即使管理人员也可以很方便地阅读，如图2.38所示。



通过这样一个简单示例，说明Notebook可以完美地把数据分析文档、分析代码、分析结果可视化整合到一个文件中，它是数据分析师的重要工具之一。

第3章 Pandas基础

九层之台，始于垒土。

——《老子》

使用Python进行数据挖掘和数据分析时，一个必不可少的利器就是Pandas库。利用Pandas库可以快速地完成数据读写、数据分片 / 分组统计、数据整理等操作。Pandas的所有功能都是构建在两个最基础的数据结构之上：Series与DataFrame。本章将围绕这两个基础的数据结构，详细讨论以下内容：

- 如何理解DataFrame，以及它的基本要素；
- DataFrame中的数据类型；
- 如何选择DataFrame中的某一行；
- 什么是Series，以及针对Series的链式方法；
- 如何对DataFrame的列名、行名、索引进行修改；
- 如何选择DataFrame中的多列；
- 面向DataFrame的操作符。

3.1 什么是DataFrame

Pandas中有两种基础数据类型：Series与DataFrame。Series是一种类似于一维数组的对象，由一组数据（各种NumPy数据类型）以及一组与之相关的数据标签（即索引）组成。而DataFrame则是一个表格型的数据结构，包含一组有序的列，每列可以是不同的值类型（数值、字符串、布尔型等），DataFrame既有行索引，也有列索引，可以看作是由Series组成的字典。在程序中要使用Pandas，第一步需要导入该

库。为了方便，通常在导入Pandas时会给它一个别名“pd”，后面如果需要使用该库，只需要输入pd就可以了。此外，在Jupyter Notebook中，我们还可以根据自己的需要设置数据显示多少行、多少列，超出设置的部分就会用省略号代替。下面的代码完成了Pandas库的导入，同时设置Jupyter Notebook中显示数据时，最多显示7行6列，超出部分用省略号代替。

```
>>> import pandas as pd
# set max display row, columns
>>> pd.set_option('display.max_rows', 7)
>>> pd.set_option('display.max_columns', 6)
```

3.1.1 DataFrame的基本要素

DataFrame有3个基本要素，它们分别是索引、列、数据。首先运行如下代码。

```
>>> retail_data = pd.read_csv("../data/Online_Retail_Fake.csv")
>>> retail_data.head()
```

如果是在Jupyter Notebook中运行上述代码，那么会得到如图3.1所示的输出。



图3.1 DataFrame三要素

上面的代码中，第一行首先使用了`read_csv()`函数来将本地目录下的数据文件`Online_Retail_Fake.csv`读入到一个`DataFrame`中。关于读取数据部分，我们在第8章会有更深入的介绍。读者现在只需要知道`read_csv()`函数可以将CSV格式的文件的內容读入到一个`DataFrame`中就可以了。现在数据已经读入到`retail_data`这个`DataFrame`中，我们就可以利用`DataFrame`对象中的方法（函数）`head()`来查看数据内容，在Jupyter中`DataFrame`的列名以及索引均以黑体显示，如图3.1所示。一个`DataFrame`包含了3个基本组件：索引、列和数据。通过指定对应的索引标签（Index Label）和列可以快速访问一个`DataFrame`的指定行或列的数据。在多个Series或`DataFrame`合并时，索引会用来做数据对齐。实际上`DataFrame`把索引和列都当作轴（axis），一个`DataFrame`有一个竖轴（Index）和一个横轴（Columns）。Pandas借用了NumPy的命名，分别用0/1来表示竖/横轴。图3.1中最左边一列的0, 1, 2, 3, 4就是`DataFrame`的索引，我们也把索引（Index）称为`DataFrame`的0轴，后面经常会遇到在函数中指定参数`axis=0`的情况，就是指函数是作用于行。而具体的每一个索引标签（Index Label）对应于一行数据，如图3.1中的3就对应了数据的第4行（Pandas中的索引也采用了从0开始计数的方式）。图的最顶端一行则代表了该数据的列（Columns），也可以用`axis=1`表示。例如，`StockCode`就是其中一列的名称，正如图中显示的一样，每一列的数据都是同一类型，要么是数字，要么是字符串。虽然`DataFrame`中每一列数据要求是相同类型，但是不同列却可以是不同数据类型。对于数据缺失的情况，Pandas中采用了NaN进行表示。实际数据分析中的数据经常是几千甚至几万行，所以为了方便分析，Jupyter Notebook提供了设置最多显示多少行、多少列的功能。对于超出设置的部分，Jupyter Notebook会用省略号进行表示，如图3.1所示。Jupyter Notebook还具有代码补齐功能，按下Tab键可以帮助补齐后面的代码。

如图3.2所示，按下Tab键后，Notebook中会给出对应`retail_data`对象的方法，可以从列表中选择对应方法来补齐代码。

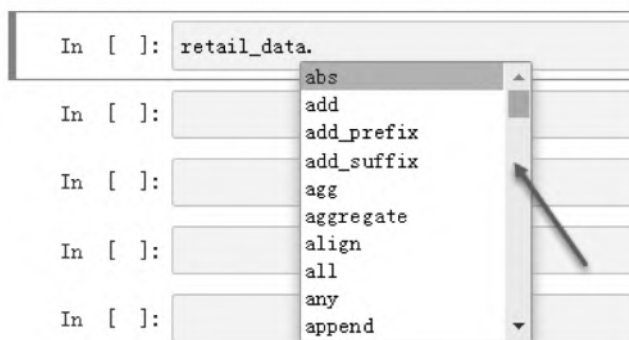


图3.2 代码补齐

如果对某一个函数的使用存在任何疑问，也可以在函数名称后面输入“?”后按下Shift+Enter组合键运行该行代码来查看函数帮助，如图3.3所示。

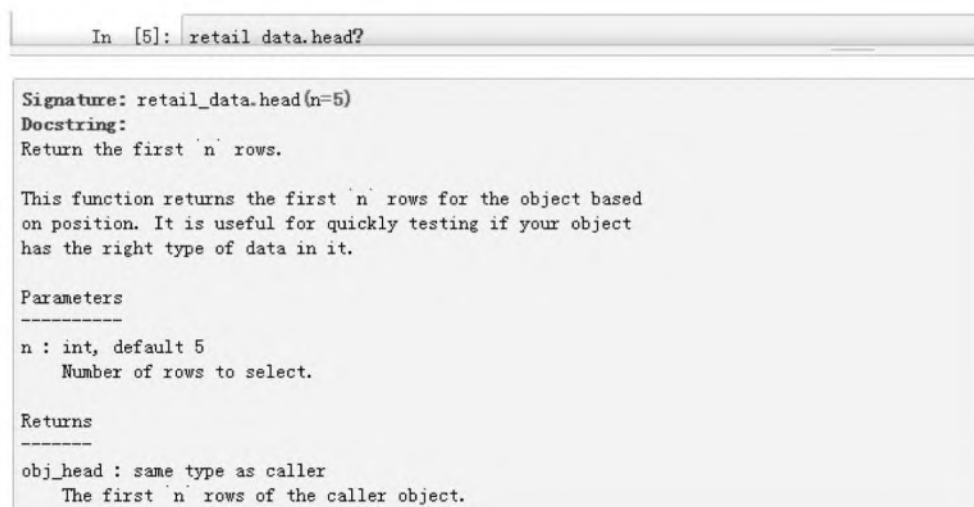


图3.3 查看函数帮助

由于`index`、`columns`、`values`都是`DataFrame`对象中的属性，所以我们可以直接访问这些属性。例如，我们可以用如下代码来获取上述属性的值。

```
>>> index = retail_data.index
>>> columns = retail_data.columns
>>> data = retail_data.values
```

上面3行代码直接获取了retail_data对象的index、columns、values属性，并赋值给了变量index、columns、data。接下来，运行如下代码依次查看这3个变量的值。

```
>>> index
RangeIndex(start = 0, stop = 541909, step = 1)
>>> columns
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity',
      'InvoiceDate', 'UnitPrice', 'CustomerID', 'Country'],
      dtype = 'object')
>>> data
array([[ '536365', '85123A', 'WHITE HANGING HEART T-LIGHT HOLDER', ..., 2.55, 17850.0, 'United
Kingdom'], [ '536365', '71053', 'WHITE METAL LANTERN', ..., nan, 17850.0, 'United Kingdom'],
       [ '536365', '84406B', nan, ..., 2.75, 17850.0,
'United Kingdom'], ..., [ '581587', '23254',
'CHILDRENS CUTLERY DOLLY GIRL ', ..., 4.15, 12680.0, 'France'],
       [ '581587', '23255', 'CHILDRENS CUTLERY CIRCUS PARADE', ...,
4.15, 12680.0, 'France'], [ '581587', '22138',
'BAKING SET 9 PIECE RETROSPOT ', ..., 4.95,
12680.0, 'France']], dtype = object)
```

从输出内容可知index变量为RangeIndex类型，columns变量为Index类型，而data变量为ndarray类型。通过issubclass（）方法可以发现RangeIndex本质上就是Index的子类，代码如下。

```
>>> type(index)
pandas.core.indexes.range.RangeIndex
>>> type(columns)
pandas.core.indexes.base.Index
>>> type(data)
numpy.ndarray
>>> issubclass(pd.RangeIndex, pd.Index)
True
```

index的子类除了RangeIndex外，还包括CategoricalIndex、MultiIndex、IntervalIndex、Int64Index、UInt64Index、

Float64Index、RangeIndex、TimedeltaIndex、DatetimeIndex、PeriodIndex等，在后面的学习中读者也会逐渐接触到它们。我们知道Pandas基于NumPy而构建，因此它大量地使用了ndarray，例如，index、columns、data底层实际都是ndarray，下面的代码就是很好的说明。

```
>>> index.values
array([ 0, 1, 2, ..., 541906, 541907, 541908], dtype = int64)
>>> columns.values
array(['InvoiceNo', 'StockCode', 'Description', 'Quantity',
      'InvoiceDate', 'UnitPrice', 'CustomerID', 'Country'], dtype = object)
```

3.1.2 数据类型

数据分析中通常将数据简单地分为连续变量和离散变量，而Pandas对数据的类型有更详细的分类。表3.1给出了Pandas中的主要数据类型。

表3.1 Pandas中的数据类型

Pandas Type	Python Type	NumPy Type	使用场景
object	str	str	文本
int64	int	int,int8,int16,int32,int64,uint8,uint16,uint32,uint64	整数
float64	float	float	浮点数
bool	bool	bool	布尔类型
datetime64	NA	NA	日期
timedelta[ns]	NA	NA	日期间隔
category	NA	NA	分类变量

通过3.1.1节，我们已经知道DataFrame的每一列只能是一种数据类型，而不同列则可以是有不同数据类型。如果要具体了解一个DataFrame对象中列的数据类型，可以通过dtypes属性来查看，代码如下。

```
>>> retail_data.dtypes
InvoiceNo      object
StockCode      object
Description    object
Quantity       int64
InvoiceDate    object
UnitPrice      float64
CustomerID     float64
Country        object
dtype: object
```

这里`retail_data`的`UnitPrice`列中每个数据都是64位浮点数，而`Quantity`列则都是64位整数。对应`object`类型，它可以是任何的Python对象，不过通常在数据分析中遇到的`object`类型都是字符串。要方便地查看`DataFrame`中每种数据类型各有几列，可以通过`get_dtype_counts()`方法，代码如下。

```
>>> retail_data.get_dtype_counts()
float64      2
int64        1
object       5
dtype: int64
```

3.1.3 了解Series

`Series`就是构成`DataFrame`的列，每一列就是一个`Series`。一个`Series`是一个一维的数据类型，其中每一个元素都有一个标签，类似于NumPy中元素带标签的数组。其中，标签可以是数字或字符串。如果访问`DataFrame`中的某一列，可以采用如下代码。

```
>>> retail_data['Country']
0          United Kingdom
1          United Kingdom
...
541907          France
541908          France
Name: Country, Length: 541909, dtype: object
>>> type(retail_data.Country)
pandas.core.series.Series
```

这一段代码访问的是Country列，正如我们所料，返回的是一个Series。它包含Index和数据，同时该Series的Name（即列名）为Country，数据类型为object，数据长度为541909。除了用“[]”操作符来访问某一列，DataFrame还提供了另一种用“.”操作符来访问列的方法，代码如下。

```
>>> retail_data.Country
0          United Kingdom
1          United Kingdom
...
541907          France
541908          France
Name: Country, Length: 541909, dtype: object
```

通常情况下，我们不建议采用这种方法，原因有二：其一，如果对应的列名包含特殊字符，这种方法可能无效，如果列名为Country Name，那么无法通过代码retail_data.Country Name来访问对应列；其二，如果列名与DataFrame的方法重名，那么也会无效，如果数据中某列名为mean，此时也是无法采用该方法访问列的。获取了某一列数据后，Python中支持的大部分运算操作符都可以应用于Series，代码如下。

```

>>> retail_data['UnitPrice']
0          2.55
1          NaN
...
541907     4.15
541908     4.95
Name: UnitPrice, Length: 541909, dtype: float64
>>> retail_data['UnitPrice'] + 1
0          3.55
1          NaN
...
541907     5.15
541908     5.95
Name: UnitPrice, Length: 541909, dtype: float64
>>> retail_data['UnitPrice'] * 2
0          5.1
1          NaN
...
541907     8.3
541908     9.9
Name: UnitPrice, Length: 541909, dtype: float64
>>> retail_data['UnitPrice'] > 2
0          True
1          False
...
541907     True
541908     True
Name: UnitPrice, Length: 541909, dtype: bool

```

上述代码分别将UnitPrice列的每个数据都加1，乘以2，将UnitPrice列中的每个数据与2比较，返回取值为True/False的新Series。

除了数值操作，还可以将一个Series与字符串进行比较，例如：

```

>>> country = retail_data['Country']
>>> country == 'France'
0          False
1          False
...
541907     True
541908     True
Name: Country, Length: 541909, dtype: bool

```


这里将Country这个Series中的每个数据与字符串France进行逻辑比较，返回一个新的取值为True/False的Series。数据分析中会大量用到类似的布尔判断来对数据进行过滤，后续章节会对此进行更详细的介绍。

3.1.4 链式方法

在Python这一面向对象的高级语言中，一切皆对象，只要是对象，就有着自己的属性和方法，而对这些属性和方法的调用可能会返回新的对象和方法。这种通过“.”操作来顺序调用对象方法的方式称为链式方法。在Pandas库中，DataFrame和Series的很多操作都将返回一个新的DataFrame或Series，此时我们又可以对新的DataFrame和Series进行方法调用，下面来看一段关于链式方法的代码。

```
>>> customers = retail_data['CustomerID']
>>> customers.value_counts().head()
17841.0    7983
14911.0    5903
14096.0    5128
12748.0    4642
14606.0    2782
Name: CustomerID, dtype: int64
>>> retail_data['Country'].value_counts().head()
United Kingdom    495477
Germany           9495
France            8557
EIRE              8196
Spain             2533
Name: Country, dtype: int64
```

上述第一段代码先获取了CustomerID列，并将其赋值给了一个新变量customers。之后用value_counts（）方法统计了customers这个Series中每个顾客的购物记录总数，最后用head（）方法显示前5行。而第二段代码直接对Country列中来自不同国家的购物记录进行汇总，

之后再显示前5行。数据分析中经常会用到链式方法来对DataFrame或Series中的缺失值进行统计，例如：

```
>>> retail_data['UnitPrice'].isnull()
0      False
1       True
...
541907  False
541908  False
Name: UnitPrice, Length: 541909, dtype: bool
>>> retail_data['UnitPrice'].isnull().sum()
3
```

第一段代码使用isnull（）方法来判断UnitPrice列中的每一行数据是否缺失，如果是，则返回True。最终将得到一个新的Series，它的长度与UnitPrice列一样。通过链式方法可以继续调用sum（）函数，该函数将对Series求和，这样就可以得出UnitPrice列共有3行数据缺失。当然，除了采用求和的方式统计总的缺失数据个数，也可以用求平均值函数mean（）来获取缺失数据所占比例，代码如下。

```
>>> retail_data['UnitPrice'].isnull().mean()
5.535984824020269e-06
```

此外，也可以利用fillna（）函数对缺失数据进行填充，之后再利用mean（）函数查看缺失数据，此时可以发现UnitPrice中已经没有缺失数据了，代码如下。

```
>>> retail_data['UnitPrice'].fillna(0).isnull().mean()
0.0
```

fillna（）函数是处理缺失数据时常用到的一个方法，在后续章节还将更详细地介绍该方法。同时，对于任何函数，在Jupyter中都可以通过“？”操作符来获取详细的使用说明，读者可以通过帮助详细了解该函数的使用。

另一个Pandas新用户经常会遇到的问题就是DataFrame和Series中大部分对原有数据的修改操作都是产生一个全新的Series或DataFrame，而不是对原数据的修改。例如，前面的代码`retail_data['UnitPrice'].fillna(0)`产生了一个新的Series，那么对它修改不会对原来的DataFrame（即`retail_data`）产生任何改变。通过运行如下代码可以得到验证。

```
>>> retail_data['UnitPrice'].fillna(0).head()
0    2.55
1    0.00
2    2.75
3    3.39
4    3.39
Name: UnitPrice, dtype: float64
>>> retail_data.head()
0    2.55
1    NaN
2    2.75
3    3.39
4    3.39
Name: UnitPrice, dtype: float64
```

如果想直接在原DataFrame上进行修改，通常只需要在函数中加上参数`inplace=True`就可以了。

3.2 索引与列

3.2.1 修改索引与列

在数据分析过程中经常会遇到需要修改DataFrame中的索引与列名的问题，这里以gapminder数据集为例进行讲解，首先读入该数据，代码如下。

```
>>> gapminder = pd.read_csv("../data/gapminder.csv")
>>> gapminder.head()
```

显示结果如图3.4所示。

	Unnamed: 0	1800	1801	...	2015	2016	Life expectancy
0	0	NaN	NaN	...	NaN	NaN	Abkhazia
1	1	28.21	28.20	...	NaN	NaN	Afghanistan
2	2	NaN	NaN	...	NaN	NaN	Akrotiri and Dhekelia
3	3	35.40	35.40	...	NaN	NaN	Albania
4	4	28.82	28.82	...	NaN	NaN	Algeria

5 rows x 219 columns

图3.4 gapminder原始显示结果

从数据的显示看出gapminder采用了默认的索引值，取值为0，1，…然而分析中想要的是用Life expectancy列的数据作为gapminder的索引，因此可以用如下代码完成修改。

```
>>> gapminder = pd.read_csv("../data/gapminder.csv",  
                             index_col = 'Life expectancy')  
>>> gapminder.head()
```

修改结果如图3.5所示。

	Unnamed: 0	1800	1801	...	2014	2015	2016
Life expectancy							
Abkhazia	0	NaN	NaN	...	NaN	NaN	NaN
Afghanistan	1	28.21	28.20	...	NaN	NaN	NaN
Akrotiri and Dhekelia	2	NaN	NaN	...	NaN	NaN	NaN
Albania	3	35.40	35.40	...	NaN	NaN	NaN
Algeria	4	28.82	28.82	...	NaN	NaN	NaN

5 rows x 218 columns

图3.5 修改结果（1）

通过index_col='Life expectancy'参数可以让read_csv（）函数在读取数据时就将某列设置为索引。当然，Pandas也支持在读入数据后再进行修改，代码如下。

```
>>> gapminder = pd.read_csv("../data/gapminder.csv")
>>> gapminder.set_index('Life expectancy')
>>> gapminder.head()
```

修改结果如图3.6所示。set_index() 函数可以将指定的列设置为对应DataFrame对象的Index，但是从代码输出来看似乎设置没有成功，我们用gapminder.head() 看到的还是原来的Index。细心的读者应该会记得前面已经指出Pandas中大部分函数执行后返回的都是一个新的对象。因此，如果想在原来的对象上修改，有两种方法：一种就是使用inplace=True参数；另一种就是将返回的对象赋值给原来的变量，代码如下。

	Unnamed: 0	1800	1801	...	2015	2016	Life expectancy
0	0	NaN	NaN	...	NaN	NaN	Abkhazia
1	1	28.21	28.20	...	NaN	NaN	Afghanistan
2	2	NaN	NaN	...	NaN	NaN	Akrotiri and Dhekelia
3	3	35.40	35.40	...	NaN	NaN	Albania
4	4	28.82	28.82	...	NaN	NaN	Algeria

5 rows x 219 columns

图3.6 修改结果（2）

```
>>> gapminder = pd.read_csv("../data/gapminder.csv")
>>> gapminder = gapminder.set_index('Life expectancy')
>>> gapminder.head()
```

既然可以设置索引，那么必然有将其恢复到列的方法，如下代码就可以将索引重新恢复到列中，结果如图3.7所示。

```
>>> gapminder.reset_index()
```

	Life expectancy	Unnamed: 0	1800	...	2014	2015	2016
0	Abkhazia	0	NaN	...	NaN	NaN	NaN
1	Afghanistan	1	28.21	...	NaN	NaN	NaN
...	...	...	...	...	...	...	...
778	Åland	258	NaN	...	NaN	NaN	NaN
779	South Sudan	259	NaN	...	56.1	56.1	56.1

780 rows x 219 columns

图3.7 修改结果（3）

修改后返回一个新的DataFrame，如果想在原来的DataFrame上修改，请使用inplace=True参数。掌握了索引的修改，接下来看看如何同时修改索引与列名，示例代码如下。

```
>>> gapminder = pd.read_csv("../data/gapminder.csv",
    index_col = 'Life expectancy')
>>> idx_rename = {'Abkhazia': 'Abk', 'Afghanistan': 'Afg'}
>>> col_rename = {'Unnamed: 0': 'Wrong Column', '1801': '1801Y'}
## 再次强调返回的是一个新的 DataFrame 而不是对原有 DataFrame 的修改
>>> gapminder_new = gapminder.rename(index = idx_rename,
    columns = col_rename)
>>> gapminder_new.head()
```

修改结果如图3.8所示。上述代码通过rename（）函数完成了对gapminder的索引和列名的修改。其中，idx_rename和col_rename均为字典对象，字典的key对应了需要修改的内容，value则是对应要修改成的那个值。同时，index和columns也都可以转换为列表对象，可以通过直接将一个列表赋值给index和columns的方法来完成修改，具体代码如下。

	Wrong Column	1800	1801Y	...	2014	2015	2016
Life expectancy							
Abk	0	NaN	NaN	...	NaN	NaN	NaN
Afg	1	28.21	28.20	...	NaN	NaN	NaN
Akrotiri and Dhekelia	2	NaN	NaN	...	NaN	NaN	NaN
Albania	3	35.40	35.40	...	NaN	NaN	NaN
Algeria	4	28.82	28.82	...	NaN	NaN	NaN

5 rows x 218 columns

图3.8 修改结果（4）

```
>>> index = gapminder.index
>>> columns = gapminder.columns
>>> index_list = index.tolist()
>>> column_list = columns.tolist()
>>> print(index_list[0])
Abkhazia
>>> index_list[0] = 'Abk'
>>> index_list[1] = 'Afg'
>>> column_list[0] = 'Wrong Column'
>>> gapminder.index = index_list
>>> gapminder.columns = column_list
>>> gapminder.head()
```

修改结果如图3.9所示。

	Wrong Column	1800	1801	...	2014	2015	2016
Abk	0	NaN	NaN	...	NaN	NaN	NaN
Afg	1	28.21	28.20	...	NaN	NaN	NaN
Akrotiri and Dhekelia	2	NaN	NaN	...	NaN	NaN	NaN
Albania	3	35.40	35.40	...	NaN	NaN	NaN
Algeria	4	28.82	28.82	...	NaN	NaN	NaN

5 rows x 218 columns

图3.9 修改结果（5）

正如我们所期望的，gapminder的列名和前两行的索引标签都已经修改为想要的内容。

3.2.2 添加、修改或删除列

在数据分析过程中经常需要添加新的列，或对当前列的数据进行修改，有时还需要删除某些列。本节将一一讨论如何实现它们。首先使用如下代码读入新的数据，并查看前5行。

```
>>> retail_data = pd.read_csv("../data/Online_Retail_Fake.csv")
>>> retail_data.head()
```

如图3.10所示，`retail_data`数据包含了单价`UnitPrice`列和数量`Quantity`列，但是没有总价信息，分析中如果想添加新的一列`TotalPrice`到`retail_data`中，那么可以采取如下方法，结果如图3.11所示。

	InvoiceNo	StockCode	Description	...	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	...	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	...	NaN	17850.0	United Kingdom
2	536365	84406B		NaN	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	...	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	...	3.39	17850.0	United Kingdom

5 rows x 8 columns

图3.10 读入数据

```
>>> retail_data['Total_price'] = retail_data['UnitPrice'] * \
    retail_data['Quantity']
>>> retail_data[['Total_price', 'UnitPrice', 'Quantity']]
```

	Total_price	UnitPrice	Quantity
0	15.30	2.55	6
1	NaN	NaN	6
2	22.00	2.75	8
...	...	...	...
541906	16.60	4.15	4
541907	16.60	4.15	4
541908	14.85	4.95	3

541909 rows x 3 columns

图3.11 添加新列

从上述代码可以看出，添加新列是非常简单的，直接给该列赋值就可以完成。上面的第二行代码表示只取这3列数据，在第4章数据筛选中还会更详细地介绍，现在只需要理解`['Total_price', 'UnitPrice', 'Quantity']`代表只取这个列表中的列就可以了。默认情况下采用上述代码增加的新列都会作为最后一

列，如果想在DataFrame的中间增加列，此时就需要采用insert（）方法。例如，如果要在CustomerID列后添加刚才的数据，代码如下。

```
>>> totalPrice_index = retail_data.columns.get_loc(
('CustomerID') + 1
>>> totalPrice_index
7
>>> retail_data.insert(loc = totalPrice_index,
                        column = 'New_totalPrice',
                        value = retail_data['Total_price'])
>>> retail_data.head()
```

运行结果图3.12所示。

	InvoiceNo	StockCode	Description	Quantity	...	CustomerID	New_totalPrice	Country	Total_price
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	...	17850.0	15.300000	United Kingdom	15.300000
1	536365	71053	WHITE METAL LANTERN	6	...	17850.0	27.666699	United Kingdom	27.666699
2	536365	84406B	NaN	8	...	17850.0	22.000000	United Kingdom	22.000000
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	...	17850.0	20.340000	United Kingdom	20.340000
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	...	17850.0	20.340000	United Kingdom	20.340000

5 rows x 10 columns

图3.12 insert（）插入列

从代码输出可以看出新的New_totalPrice列位于CustomerID后。了解了如何在DataFrame中添加新列后，下面来看一下如何删除列。删除列需要使用drop（）方法，如果想删除New_totalPrice列，则可以运行如下代码。

```
>>> retail_data.drop('New_totalPrice',axis = 'columns')
```

运行结果如图3.13所示。

	InvoiceNo	StockCode	Description	Quantity	...	UnitPrice	CustomerID	Country	Total_price
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	...	2.550000	17850.0	United Kingdom	15.300000
1	536365	71053	WHITE METAL LANTERN	6	...	4.611117	17850.0	United Kingdom	27.666699
...	...	...	...	...	...	...	...	...	...
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	...	4.950000	12680.0	France	14.850000
541909	581587	22138	Wrong booking	3	...	4.950000	12680.0	France	14.850000

541910 rows x 9 columns

图3.13 删除列

需要注意的是，`drop()`方法返回的是一个新的DataFrame，而不是在原来的DataFrame进行列的删除。因此，如果需要保留这个新的DataFrame，可以使用参数`inplace=True`来指明修改在原来的DataFrame上进行。或者也可以将返回的DataFrame赋值给一个新的变量。除了采用`drop()`方法以外，还可以用如下代码直接在原来的DataFrame中删除列。

```
>>> del retail_data['New_totalPrice']
```

除了常规的数学运算，还可以对DataFrame的列进行逻辑运算，之后将其赋值给新的列，例如：

```
>>> retail_data['key_customer'] = retail_data['Total_price'] > 1000
>>> retail_data['key_customer'].sum()
388
```

通过上述代码将购物总金额超过1000的定义为关键客户，首先将Total_price列的数据与1000比较，此时返回的是一个取值为True/False的Series。然后将该Series添加到retail_data中，命名该列为key_customer，对该列求和就可以得出关键客户总数为388。这种将DataFrame的列与某个值进行比较后得到一个取值为布尔值Series的方法在后面章节还会多次使用，希望读者能熟练掌握。

3.3 选择多列

前面已经学习了如何选择DataFrame中的某一列，那么如果想选择多列，应该怎么操作呢？例如，选择retail_data中的3列，依次为Total_price、UnitPrice、Quantity，那么可以用如下代码完成，结果如图3.14所示。

```
>>> retail_data[['Total_price', 'UnitPrice', 'Quantity']]
```

	Total_price	UnitPrice	Quantity
0	15.300000	2.550000	6
1	27.666696	4.611116	6
...	...	...	...
541907	16.600000	4.150000	4
541908	14.850000	4.950000	3

541909 rows x 3 columns

图3.14 选择结果（1）

上述代码中用了一个列表（即 ['Total_price', 'UnitPrice', 'Quantity']）作为变量在 retail_data 中指出想要选择的列，此时将返回一个新的 DataFrame，即使该列表中只有一个元素，返回的也是一个 DataFrame。下面的代码可以帮助我们对此有更好的理解。

```
>>> type(retail_data[['Total_price']])
pandas.core.frame.DataFrame
>>> type(retail_data['Total_price'])
pandas.core.series.Series
```

利用 type（）函数进行检查，可以确认输入为列表时返回的是 DataFrame，而直接输入 Total_price 返回的则是 Series。除了通过明确指定列名的方式来选择列以外，还可以根据列的数据类型或列的名称字符串匹配来选择列，代码如下。

```
>>> retail_data.get_dtype_counts()
float64    4
int64      1
object     5
dtype: int64
>>> retail_data.select_dtypes(include = ['float64']).head()
```

运行结果如图3.15所示。

	UnitPrice	CustomerID	New_totalPrice	Total_price
0	2.550000	17850.0	15.300000	15.300000
1	4.611116	17850.0	27.666696	27.666696
2	2.750000	17850.0	22.000000	22.000000
3	3.390000	17850.0	20.340000	20.340000
4	3.390000	17850.0	20.340000	20.340000

图3.15 选择结果（2）

第一段代码统计了retail_data中各种数据类型的数量，其中float64类型有4列。第二段代码则通过select_dtypes（）函数指定选择float64数据类型的列。当然，Pandas也支持一次选择多种数据类型，例如：

```
>>> retail_data.select_dtypes(include = ['float64', 'int']).head()
```

这段代码同时选择了float64和int类型的列。除了根据数据类型选择列，Pandas中还可以根据列名称来进行选择，代码如下。

```
# 字符串部分匹配
>>> retail_data.filter(like = 'Price')

# 正则表达式
>>> retail_data.filter(regex = '_').head()
```

运行结果如图3.16和图3.17所示。

	UnitPrice	New_totalPrice
0	2.550000	15.300000
1	4.611116	27.666696
2	2.750000	22.000000
...	...	...
541906	4.150000	16.600000
541907	4.150000	16.600000
541908	4.950000	14.850000

541909 rows x 2 columns

图3.16 选择结果（3）

	New_totalPrice	Total_price
0	15.300000	15.300000
1	27.666696	27.666696
2	22.000000	22.000000
3	20.340000	20.340000
4	20.340000	20.340000

图3.17 选择结果（4）

`filter()` 方法有3个互斥的参数，分别为`items`、`regex`和`like`。通过前面的代码已经知道了`regex`和`like`参数的用法，接下来看看`items`参数，该参数实际对应的就是列名。例如，如下代码将选择`Total_price`和`UnitPrice`列，结果如图3.18所示。

```
>>> retail_data.filter(items = ['Total_price', 'UnitPrice']).head()
```

使用`items`参数的一个好处是，如果`items`中的列名不存在，此时不会返回`KeyError`错误，而只是返回包含正确列的`DataFrame`，代码如下。

```
>>> retail_data.filter(items = ['Total_price', 'UnitPrice',
                                'wrong']).head()
```

	Total_price	UnitPrice
0	15.300000	2.550000
1	27.666696	4.611116
2	22.000000	2.750000
3	20.340000	3.390000
4	20.340000	3.390000

图3.18 选择结果（5）

前面学习了如何选择特定列，利用这一方法还可以对数据中的列进行重排。例如，根据列数据类型（连续变量或离散变量）组织`DataFrame`中的列，或者将关联的列顺序排列。如果从数据分析角度看，`retail_data`中`UnitPrice`、`Quantity`、`Total_price`列是一组关联

数据，InvoiceNo、InvoiceDate、StockCode、Description列为另一组数据，CustomerID和Country为一组，那么可以采用如下方式完成列的重排。

```
>>> price_info = ['UnitPrice', 'Quantity', 'Total_price']
>>> invoice_info = ['InvoiceNo', 'InvoiceDate', 'StockCode',
                    'Description']
>>> customer_info = ['CustomerID', 'Country']
>>> columns_new = price_info + invoice_info + customer_info
>>> set(columns_new) == set(retail_data.columns)
True
```

上述代码首先定义了3个列表，分别包含了想要的3组列的名称。然后将3个列表汇总到columns_new这个新的列表，它里面的列名称实际上和原来的retail_data一样，只是列的排序不一样。通过集合比较的输出也可以确认二者是一致的。直接将columns_new这个列表传入到retail_data的“[]”操作符就完成了列的选择，新的DataFrame中列的顺序即是所期望的顺序了，代码如下。

```
>>> retail_data_new = retail_data[columns_new]
>>> retail_data_new.head()
```

运行结果如图3.19所示。

	UnitPrice	Quantity	Total_price	InvoiceNo	InvoiceDate	StockCode	Description	CustomerID	Country
0	2.550000	6	15.300000	536365	2010/12/1 8:26	85123A	WHITE HANGING HEART T-LIGHT HOLDER	17850.0	United Kingdom
1	4.611117	6	27.666699	536365	2010/12/1 8:26	71053	WHITE METAL LANTERN	17850.0	United Kingdom
2	2.750000	8	22.000000	536365	2010/12/1 8:26	84406B	NaN	17850.0	United Kingdom
3	3.390000	6	20.340000	536365	2010/12/1 8:26	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	17850.0	United Kingdom
4	3.390000	6	20.340000	536365	2010/12/1 8:26	84029E	RED WOOLLY HOTTIE WHITE HEART.	17850.0	United Kingdom

图3.19 列的重排

第4章 数据筛选

众里寻他千百度。

蓦然回首，那人却在，灯火阑珊处。

——辛弃疾《青玉案·元夕》

如果用一句话来概括数据分析，可以认为数据分析的过程就是找到所关心的问题的答案。而要回答这一问题，我们需要像炼金术士一样对数据不断提炼。在这一过程中需要频繁选取特定数据放到放大镜下进行观察。在Pandas中，选择特定行、列有两种方式：一种是操作符“[]”访问方式，“[]”称为Indexing Operator（索引操作符）；另一种是通过.loc或.iloc的方式，即Indexer（索引器）来选择特定行、列。通过第3章的学习，读者对使用操作符方式访问DataFrame的行和列已经有所了解，本章将重点介绍通过索引器选择行、列的方法。

4.1 使用.loc和.iloc筛选行与列数据

Series和DataFrame具有极大的灵活性，数据分析人员既可以像访问Python中的列表那样利用整数索引的方式对Series和DataFrame中的数据进行访问，也可以像Python中的字典一样通过标签（Label）来访问。其中.loc支持通过索引标签（Index Label）的方式访问数据，而.iloc支持通过整数索引的方式访问数据。在开始本节的讲解之前，首先读入要使用的数据，同时完成一些Jupyter Notebook中的基本设置，代码如下。


```

>>> import pandas as pd
# set max display row, columns
>>> pd.set_option('display.max_rows', 10)
>>> pd.set_option('display.max_columns', 10)
>>> retail_data = pd.read_csv("../data/Online_Retail_Fake.csv")
>>> retail_data['UnitPrice'] = retail_data['UnitPrice']. \
    fillna(retail_data['UnitPrice'].mean())
>>> retail_data['Quantity'] = retail_data['Quantity']. \
    fillna(retail_data['Quantity'].mode())
>>> retail_data['Total_price'] = retail_data['UnitPrice'] * \
    retail_data['Quantity']

```

上述代码完成了Pandas模块的导入，同时设置Jupyter Notebook中行、列显示限制为10。然后程序读入数据到retail_data中，并使用fillna（）函数将UnitPrice列缺失数据用均值来填充，Quantity列缺失数据用众数填充，最后在retail_data中添加了新列Total_price。

4.1.1 选择Series和DataFrame中的行

.loc和.iloc在Series和DataFrame的工作方式基本一样，首先来看一下Series中使用整数索引方式来选择数据的方法——.iloc。

```

>>> country = retail_data['Country']
>>> country.head()
0    United Kingdom
1    United Kingdom
2    United Kingdom
3    United Kingdom
4    United Kingdom
Name: Country, dtype: object

```

上述代码通过“[]”操作访问retail_data的Country列，并将其赋值给了变量country，通过前面章节的学习，读者应该知道该变量为Series类型。此时如果想选择country的某一特定行，应该怎么办呢？通过.iloc可以实现，代码如下。


```
>>> country.iloc[0],country.iloc[7279],country.iloc[541909]
('United Kingdom', 'Belgium', 'France')
```

这段代码分别选择了country的第0行、第7279行和第541909行数据。如果想访问多行数据，可以直接输入一个列表来完成，例如：

```
>>> country.iloc[[0,7279,541909]]
0      United Kingdom
7279      Belgium
541909      France
Name: Country, dtype: object
```

这一行代码就可以同时完成对多行数据的选择。除了支持采用列表方式选择离散的多行数据以外，.iloc还支持更多样的选择行的方式，即切片（Slicing），例如：

```
>>> country.iloc[10:50:10]
10      United Kingdom
20      United Kingdom
30      France
40      France
Name: Country, dtype: object
```

上述代码中的10:50:10，代表从整数索引10开始到50，每隔10个选择一个，其中50不包括在里面。所以上述代码得到的是对应country这个Series中排在第10，20，30，40的数据。需要强调的是，这里所指的索引是指数据在Series中的排列位置，而不是Series的索引（Index）。而当前变量country的索引只是碰巧也是数字而已。下面的代码很好地说明了其中的区别。

```

>>> s1 = pd.Series([1, 2, 3, 4], index = ['A', 'B', 'C', 'D'])
>>> s2 = pd.Series([1, 2, 3, 4], index = [4, 1, 3, 2])
>>> s1
A    1
B    2
C    3
D    4
dtype: int64
>>> s2
4    1
1    2
3    3
2    4
dtype: int64
>>> s1.iloc[[1,3]]
B    2
D    4
dtype: int64
>>> s2.iloc[[1,3]]
1    2
2    4
dtype: int64

```

上述代码中s1变量的索引为['A', 'B', 'C', 'D']，而s2的索引为[4, 1, 3, 2]。如果利用.iloc选择Series中的第2行和第4行（对应的是下标1和3，因为计数从0开始），那么对应s2返回的值是2和4，而不是对应于索引值1和3的数据。希望读者能理解索引和值在Series中的顺序的区别。有时索引和顺序值一样，但是不一定总是如此。例如，对如下的数据college，两者就不一样，如图4.1所示。

```

>>> college = pd.read_csv("../data/College.csv", index_col = 0)
>>> college.head()

```

	Private	Apps	Accept	Enroll	Top10perc	...	Terminal	S.F.Ratio	perc.alumni	Expend	Grad.Rate
Abilene Christian University	Yes	1660	1232	721	23	...	78	18.1	12	7041	60
Adelphi University	Yes	2186	1924	512	16	...	30	12.2	16	10527	56
Adrian College	Yes	1428	1097	336	22	...	66	12.9	30	8735	54
Agnes Scott College	Yes	417	349	137	60	...	97	7.7	37	19016	59
Alaska Pacific University	Yes	193	146	55	16	...	72	11.9	2	10922	15

5 rows × 18 columns

图4.1 college数据

college中的索引是各学校的名称，而不是像前面的retail_data中的索引刚好为0, 1, ..., 和顺序值一样。对于这种类型的数据，通常会考虑使用.loc来选择对应的行，例如：

```
>>> college.loc['Yale University']
Private      Yes
Apps        10705
Accept       2453
Enroll       1317
Top10perc    95
...
Terminal     96
S.F.Ratio    5.8
perc.alumni  49
Expend       40386
Grad.Rate    99
Name: Yale University, Length: 18, dtype: object
```

这一行代码就是选择对应索引标签为Yale University的行。如果要一次访问多行，也可以采用如下代码。

```
>>> college.loc['Abilene Christian University':\
'Alaska Pacific University']
```

与.iloc类似，.loc也支持数据分片，因此也可以用如下代码来选择数据。

```
>>> college.loc['Abilene Christian University':'Yale University':50]
```

选择结果如图4.2所示。

	Private	Apps	Accept	Enroll	Top10perc	...	Terminal	S.F.Ratio	perc.alumni	Expend	Grad.Rate
Abilene Christian University	Yes	1660	1232	721	23	...	78	18.1	12	7041	60
Bethany College	Yes	878	816	200	16	...	68	11.6	29	7718	48
...	...	...	...	...	...	...	...	...	...	...	...
University of Wisconsin at Madison	No	14901	10932	4631	36	...	96	11.5	20	11006	72
Westminster College of Salt Lake City	Yes	917	720	213	21	...	83	10.5	34	7170	50

16 rows x 18 columns

图4.2 数据选择结果（1）

上述代码，在索引标签Abilene Christian University与Yale University之间，每隔50个选择一行数据。

4.1.2 同时选择行与列

掌握了选择行的方式后，可以将类似方法应用到列中。使用类似下面代码的方式，可以同时选择列和行。

```
df.iloc[rows, columns]
df.loc[rows, columns]
```

其中，rows代表了行的选择，它既可以是列表，也可以是数据切片方式的输入，columns代表了列的选择。例如，如下代码选择了索引标签值从Westminster College of Salt Lake City开始到最后一行，以Accept列为结束的前面所有列。这里需要注意，与.iloc方式不同的是，Accept列是包含在其中的，因为切片时如果采用的是数字，那么后面的值是不包含的，而在不是数字时则是包含的，结果如图4.3所示。

```
>>> college.loc['Westminster College of Salt Lake City':,
                : 'Accept'].head(3)
```

另外需要说明的是，如果“:”后没有内容，那么代表直到结束；如果是前面没有内容，那么代表直到开始。如果说只是“:”，那么代表所有行或列，如下代码就选择了所有行。

```
>>> college.loc[:, ['Apps', 'Accept']].head(3)
```

	Private	Apps	Accept
Westminster College of Salt Lake City	Yes	917	720
Westmont College	No	950	713
Wheaton College IL	Yes	1432	920

图4.3 数据选择结果（2）

选择结果如图4.4所示。

	Apps	Accept
Abilene Christian University	1660	1232
Adelphi University	2186	1924
Adrian College	1428	1097

图4.4 数据选择结果（3）

Pandas中的数据切片除了上面提到的几种用法，还有更多用法，例如：

```
>>> college.iloc[-5:, :]
```

选择结果如图4.5所示。

	Private	Apps	Accept	Enroll	Top10perc	...	Terminal	S.F.Ratio	perc.alumni	Expend	Grad.Rate
Worcester State College	No	2197	1515	543	4	...	60	21.0	14	4469	40
Xavier University	Yes	1959	1805	695	24	...	75	13.3	31	9189	83
Xavier University of Louisiana	Yes	2097	1915	695	34	...	75	14.4	20	8323	49
Yale University	Yes	10705	2453	1317	95	...	96	5.8	49	40386	99
York College of Pennsylvania	Yes	2989	1855	691	28	...	75	18.1	28	4509	99

5 rows x 18 columns

图4.5 数据选择结果（4）

这段代码选择了最后5行和所有列，其中的-5代表了倒数第5行。而如下代码则选择最后5行中的第1列和第2列（分别对应的是Apps和Accept列，因为它们在列中的顺序是1，2），结果如图4.6所示。

```
>>> college.iloc[-5:,[1,2]]
```

	Apps	Accept
Worcester State College	2197	1515
Xavier University	1959	1805
Xavier University of Louisiana	2097	1915
Yale University	10705	2453
York College of Pennsylvania	2989	1855

图4.6 数据选择结果（5）

在不输入列的情况下，默认选择所有列，代码如下。

```
>>> college.iloc[:5]
```

结果如图4.7所示。这段代码的效果和如下代码一样。

```
>>> college.iloc[:5,:]
```

	Private	Apps	Accept	Enroll	Top10perc	...	Terminal	S.F.Ratio	perc.alumni	Expend	Grad.Rate
Abilene Christian University	Yes	1660	1232	721	23	...	78	18.1	12	7041	60
Adelphi University	Yes	2186	1924	512	16	...	30	12.2	16	10527	56
Adrian College	Yes	1428	1097	336	22	...	66	12.9	30	8735	54
Agnes Scott College	Yes	417	349	137	60	...	97	7.7	37	19016	59
Alaska Pacific University	Yes	193	146	55	16	...	72	11.9	2	10922	15

5 rows x 18 columns

图4.7 数据选择结果（6）

有时列的内容太多，在不知道该列的数字顺序时，可以通过 `get_loc()` 函数来获得该列位于第几列，代码如下。

```
>>> college.columns.get_loc('Expend')
16
>>> start = college.columns.get_loc('Apps')
>>> end = college.columns.get_loc('Top10perc')
>>> college.iloc[10:100:10,start:end]
```

	Apps	Accept	Enroll
Alfred University	1732	1425	472
Antioch University	713	661	252
Augustana College	761	725	306
Beaver College	1163	850	348
Bethany College	878	816	200
Bowdoin College	3356	1019	418
Brown University	12586	3239	1462
Calvin College	1784	1512	913
Carthage College	1616	1427	434

图4.8 `get_loc()` 函数获得顺序值

结果如图4.8所示。这段代码首先采用`get_loc()`函数获得了Apps和Top10perc列的顺序值，之后使用该值进行列的选择。此外，有的读者可能在某些书中看到使用`.ix`方式也可以实现对行和列的选择，它还可以支持顺序值和索引混用。这里要说明的是该方式在新的版本中已经不再推荐，请读者谨慎使用。

4.2 布尔选择

4.1节学习了如何对指定的行列进行筛选，然而在数据分析中应用最广泛的还是利用布尔选择的方式来筛选数据。该方法的基本思路是通过对Pandas中的Series和DataFrame进行逻辑运算后得到一个新的Series或DataFrame，其中的数据是布尔值（True/False）。利用这些布尔值数据，分析人员就可以完成数据的筛选。本节将以CWUR数据为例来对布尔选择进行说明。CWUR指的是世界大学排名中心（Center for World University Rankings），该中心以“对学校教学质量的评估尤为关注”而闻名世界。CWUR世界大学排名是除了QS世界大学排名、THE世界大学排名、USNews全球大学排名和上海软科世界大学学术排名外，另一大备受关注的世界大学排名，可以通过网站<https://cwur.org/>进行访问。该排名通过以下几个指标完成：

- 教育质量 (Quality of Education) , 25%;
- 校友就业 (Alumni Employment) , 25%;
- 师资力量 (Quality of Faculty) , 10%;
- 研究成果 (Research Output) , 10%;
- 高质量出版物 (High Quality Publications) , 10%;
- 影响力 (Influence) , 10%;
- 引用 (Citations) , 10%。

首先读入数据，如图4.9所示。

```
>>> cwur = pd.read_csv('../data/cwurData.csv')
>>> cwur.head()
```

	world_rank	institution	country	national_rank	quality_of_education	...	citations	broad_impact	patents	score	year
0	1	Harvard University	USA	1	7	...	1	NaN	5	100.00	2012
1	2	Massachusetts Institute of Technology	USA	2	9	...	4	NaN	1	91.67	2012
2	3	Stanford University	USA	3	17	...	2	NaN	15	89.50	2012
3	4	University of Cambridge	United Kingdom	1	10	...	11	NaN	50	86.17	2012
4	5	California Institute of Technology	USA	4	2	...	22	NaN	18	85.21	2012

5 rows x 14 columns

图4.9 读入CWUR数据

4.2.1 计算布尔值

现在已经有了大学排名数据，如果想选择那些score > 85的大学，应该如何操作呢？首先需要构建一个布尔值构成的Series，方法如下。

```
>>> cwur_85 = cwur['score'] > 85
>>> cwur_85.head(5)
0    True
1    True
2    True
3    True
4    True
Name: score, dtype: bool
```


这里得到的 `cwur_85` 是一个新的 Series，它的长度和原来的 `cwur['score']` 列一样，而取值则是列中的每一个值和85比较后返回的布尔值。对该值求和可以得出共有多少所学校得分大于85，代码如下。

```
>>> cwur_85.sum()
35
```

通过下面的代码还可以计算得分大于85的学校比例是多少。

```
>>> cwur_85.mean()
0.015909090909090907
```

因为 `mean()` 函数的计算是求和后求平均，而 `True=1`，`False=0`，因此求平均就得到了比例。除了直接用 “`>`” 操作符的方式进行比较，还可以利用函数进行比较。例如，下面的一行代码也可以完成对得分大于85的学校比例的统计。

```
>>> cwur['score'].gt(85).mean()
```

类似的函数还有 `eq()`、`ne()`、`le()`、`lt()`、`ge()`，读者可以通过查看帮助来了解它们的用法。除了与某一个值进行比较可以构造用于布尔选择的 Series，还可以用在 DataFrame 中的列之间进行比较的方式进行数据选择，例如：

```
>>> edu_faculty = cwur['quality_of_education'] > \
    cwur['alumni_employment']
```

这一段代码就是比较学校的教育质量得分与职员得分列。现在分别构造学校得分大于85分的一个布尔条件选择，以及学校教育质量得分大于职员得分的另一个布尔条件选择，之后可以将两者进行逻辑运算。

```
>>>(cwur_85&edu_faculty).head()
0    False
1    False
2     True
3    False
4    False
dtype: bool
```

需要强调的是，在进行这样的布尔运算时一定要使用括号。例如，运行如下代码就会遇到错误。

```
>>> cwur['score']> 85 &\
      cwur['quality_of_education']> cwur['alumni_employment']
```

在这种情况下，正确的代码应该是：

```
>>>(cwur['score']> 85) & \
      cwur['quality_of_education']> cwur['alumni_employment'])
0    False
1    False
...
2198  False
2199  False
Length: 2200, dtype: bool
```

4.2.2 多条件筛选数据

掌握了构造数据的选择条件后，后面的数据选择就是很简单的事情了。例如，如下代码就筛选了中国历年在全球排名前100的大学。

```
>>> crit_1 = cwur['world_rank']< 100
>>> crit_2 = cwur['country'] == 'China'
>>> criterial = crit_1&crit_2
>>> cwur[criterial]
```

筛选结果如图4.10所示。条件1是排名前100，条件2是中国，将两者求“与”，就得到同时满足这两个条件的新变量criterial，将该条件应用到cwur数据中就可以筛选出满足条件的数据了。类似地，还可

以构造各种更复杂的条件，如全球排名前100，专利大于100，影响力大于200的大学可以通过如下方式筛选。

	world_rank	institution	country	national_rank	quality_of_education	...	citations	broad_impact	patents	score	year
254	55	Peking University	China	1	355	...	250	155.0	7	55.30	2014
286	87	Tsinghua University	China	2	294	...	134	162.0	16	52.60	2014
1255	56	Peking University	China	1	182	...	182	125.0	20	54.26	2015
1277	78	Tsinghua University	China	2	309	...	65	156.0	30	52.21	2015

4 rows x 14 columns

图4.10 筛选结果（1）

```
>>> crit_3 = cwur['world_rank'] < 100
>>> crit_4 = cwur['patents'] > 100
>>> crit_5 = cwur['broad_impact'] > 200
>>> criteria2 = crit_3 & (crit_4 | crit_5)
>>> cwur[criteria2].head()
```

筛选结果如图4.11所示。

	world_rank	institution	country	national_rank	quality_of_education	...	citations	broad_impact	patents	score	year
5	6	Princeton University	USA	5	8	...	26	NaN	101	82.50	2012
10	11	University of Chicago	USA	9	15	...	28	NaN	101	73.82	2012
20	21	Rockefeller University	USA	15	1	...	96	NaN	101	61.74	2012
28	29	University of Texas Southwestern Medical Center	USA	21	19	...	84	NaN	101	56.43	2012
34	35	University of Toronto	Canada	1	101	...	18	NaN	101	53.43	2012

5 rows x 14 columns

图4.11 筛选结果（2）

对于复杂的条件，通常需要采用构造新的条件选择变量的方式来完成，对于一些逻辑比较清楚的选择则可以直接进行数据的筛选。例如，如下代码直接筛选了CWUR数据集中所有中国的大学。

```
>>> cwur[cwur['country'] == 'China'].head()
```

筛选结果如图4.12所示。

	world_rank	institution	country	national_rank	quality_of_education	...	citations	broad_impact	patents	score	year
254	55	Peking University	China	1	355	...	250	155.0	7	55.30	2014
286	87	Tsinghua University	China	2	294	...	134	162.0	16	52.60	2014
388	189	Fudan University	China	3	355	...	310	230.0	100	48.14	2014
394	195	Shanghai Jiao Tong University	China	4	325	...	250	234.0	138	48.02	2014
405	206	Zhejiang University	China	5	355	...	493	290.0	94	47.76	2014

5 rows x 14 columns

图4.12 筛选结果（3）

最后用一个实际的例子来完成本章的内容，假如从事股票投资时想知道上证指数在过去一年里出现极端情况（涨跌大于或等于2.5%）的比例有多少，我们应该怎么办呢？首先获取上证指数的数据，代码如下。

```
# import tushare as ts
# sh = ts.get_h_data('000001', start = '2009-01-01', \
    end = '2019-09-01', index = True)
>>> sh = pd.read_csv('../data/sh.csv', index_col = 'date')
>>> sh.head()
```

	open	high	close	low	volume	amount
date						
2019-08-30	2907.383	2914.577	2886.237	2874.103	19395995100	224751169931
2019-08-29	2895.999	2898.605	2890.919	2878.588	17861308200	196332770521
2019-08-28	2901.627	2905.435	2893.756	2887.012	18309790300	201805050637
2019-08-27	2879.515	2919.644	2902.193	2879.406	20814179400	230999692857
2019-08-26	2851.016	2870.494	2863.567	2849.238	16989536300	191036667851

图4.13 获取上证指数数据

结果如图4.13所示。这里作者安装了tushare库来获取上证指数过去一年的数据，读者既可以使用作者提供的数据来完成后面的练习，也可以自己使用tushare下载实时数据。由于数据是倒序排列，所以需要按照索引重新排序，代码如下。

```
>>> sh.sort_index(inplace = True)
>>> sh.head()
```

重新排序结果如图4.14所示。

	open	high	close	low	volume	amount
date						
2009-01-05	1849.020	1880.716	1880.716	1844.094	6713671200	46100959232
2009-01-06	1878.827	1938.690	1937.145	1871.971	9906675200	69012570112
2009-01-07	1938.974	1948.233	1924.012	1920.515	9236008800	63931166720
2009-01-08	1890.242	1894.171	1878.181	1862.263	8037400000	55076814848
2009-01-09	1875.164	1909.349	1904.861	1875.164	7122477600	50131263488

图4.14 重新排序

由于这里得到的是每日收盘价，为了转换成每天的涨跌幅，需要利用pct_change（）函数来实现该功能，代码如下。

```
>>> percent = sh['close'].pct_change()
>>> sh['perc'] = percent
>>> sh.head()
```

转换结果如图4.15所示。得到涨跌幅后，下一步就是构造条件，并计算极端值的比例。这里使用了shape[0]来得到极端值出现的次数（shape是DataFrame对象的属性，它返回的是数据的总行数、总列数，更详细的介绍请读者参考本书第5章），代码如下。

	open	high	close	low	volume	amount	perc
date							
2009-01-05	1849.020	1880.716	1880.716	1844.094	6713671200	46100959232	NaN
2009-01-06	1878.827	1938.690	1937.145	1871.971	9906675200	69012570112	0.030004
2009-01-07	1938.974	1948.233	1924.012	1920.515	9236008800	63931166720	-0.006780
2009-01-08	1890.242	1894.171	1878.181	1862.263	8037400000	55076814848	-0.023821
2009-01-09	1875.164	1909.349	1904.861	1875.164	7122477600	50131263488	0.014205

图4.15 换算涨跌幅

```
>>> extreme = sh[(sh['perc'] >= 0.025) | (sh['perc'] <= -0.025)].shape[0]
>>> normal = sh.shape[0]
>>> extreme/normal
0.07797805642633229
```

从计算结果可以看出上证指数出现极端值的比例很低，不到8%。

第5章 开始利用Pandas进行数据分析

纸上得来终觉浅，绝知此事要躬行。

——陆游《冬夜读书示子聿》

在开始正式的数据分析之前，数据分析人员通常都会进行探索性数据分析。这一工作的目标是对要分析的数据有一个初步了解：各项数据的定义如何、数据有无缺失、是否有异常数据、数据的分布如何。在探索性数据分析过程中，数据分析人员还经常会对数据进行描述性统计、简单的数据可视化等操作以发现数据中的模式，帮助他们提出假设。

5.1 了解元数据

元数据（Metadata），又称中介数据、中继数据，它是描述数据的数据（Dataabout Data），主要是描述数据属性（Property）的信息。Pandas数据分析中所指的元数据主要包括了数据维度、数据类型、大小以及数据字典等。在了解元数据之前，首先完成相关库导入、基本设置以及数据读入等工作，代码如下。

```

>>> import numpy as np
>>> import pandas as pd
# set max display row, columns
>>> pd.set_option('display.max_rows', 10)
>>> pd.set_option('display.max_columns', 10)
>>> retail_data = pd.read_csv("../data/Online_Retail_Fake.csv")
>>> retail_data['UnitPrice'] = retail_data['UnitPrice'].\
    fillna(retail_data['UnitPrice'].mean())
>>> retail_data['Quantity'] = retail_data['Quantity'].\
    fillna(retail_data['Quantity'].mode())
>>> retail_data['Total_price'] = retail_data['UnitPrice'] * \
    retail_data['Quantity']

```

上述代码读入了一个修改过的在线网站销售数据，其中的缺失值分别采用了均值（mean）和众数（mode）来填充。接下来可以利用 head（）函数查看数据的前5行，这是数据分析人员进行任何数据分析前的例行工作，如图5.1所示。通过查看数据的前几行或最后几行（利用tail（）函数），数据分析人员可以对要分析的数据有一个初步印象，对后续分析也大有帮助。

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Total_price
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010/12/1 8:26	2.550000	17850.0	United Kingdom	15.300000
1	536365	71053	WHITE METAL LANTERN	6	2010/12/1 8:26	4.611117	17850.0	United Kingdom	27.666699
2	536365	84406B	NaN	8	2010/12/1 8:26	2.750000	17850.0	United Kingdom	22.000000
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010/12/1 8:26	3.390000	17850.0	United Kingdom	20.340000
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010/12/1 8:26	3.390000	17850.0	United Kingdom	20.340000

图5.1 head（）函数查看数据

```

>>> retail_data.head()

```

大致了解了数据后，接下来就需要查看数据的大小，利用 DataFrame提供的shape属性可以得到该信息。输入如下代码。

```

>>> retail_data.shape
(541909, 9)

```


从代码输出可以看出数据有541909行，9列。而数据总数则可以通过size属性获得，代码如下。

```
>>> retail_data.size
4877181
```

此外，还可以用ndim属性查看数据的维度，当然要分析的数据通常都是二维的，所以在实际分析中使用不多。

```
>>> retail_data.ndim
2
```

在对数据有了第一印象后，接下来要了解的元数据信息是数据的类型、占用内存大小信息。这可以通过info（）函数得到，代码如下。

```
>>> retail_data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 9 columns):
InvoiceNo      541909 non-null object
StockCode      541908 non-null object
Description    540453 non-null object
Quantity       541909 non-null int64
InvoiceDate    541909 non-null object
UnitPrice      541909 non-null float64
CustomerID     406827 non-null float64
Country        541908 non-null object
Total_price    541909 non-null float64
dtypes: float64(3), int64(1), object(5)
memory usage: 37.2 + MB
```

从info（）函数的输出可以看出retail_data中有3列float64类型的数据，1列int64类型的数据，5列object类型的数据。此外，从上面代码的输出还可以发现CustomerID和Description列均存在缺失数据。除了以上的元数据信息，有的数据文件还会提供一个数据字典，该数

据字典通常会提供各数据列的名称、说明、数据类型、取值范围等信息。下面就是一个简单的数据字典信息，如图5.2所示。

```
>>> lc = pd.read_excel('../data/LCDataDictionary.xlsx',
    sheet_name = 'LoanStats')
>>> lc.head()
```

	LoanStatNew	Description
0	addr_state	The state provided by the borrower in the loan...
1	annual_inc	The self-reported annual income provided by th...
2	annual_inc_joint	The combined self-reported annual income provi...
3	application_type	Indicates whether the loan is an individual ap...
4	collection_recovery_fee	post charge off collection fee

图5.2 一个简单的数据字典信息

5.2 数据类型转换

数据分析人员通常会把数据分为连续变量和离散变量。然而在Python数据科学库中却有着更详细的数据类型，图5.3所示是SciPy中的详细数据类型。

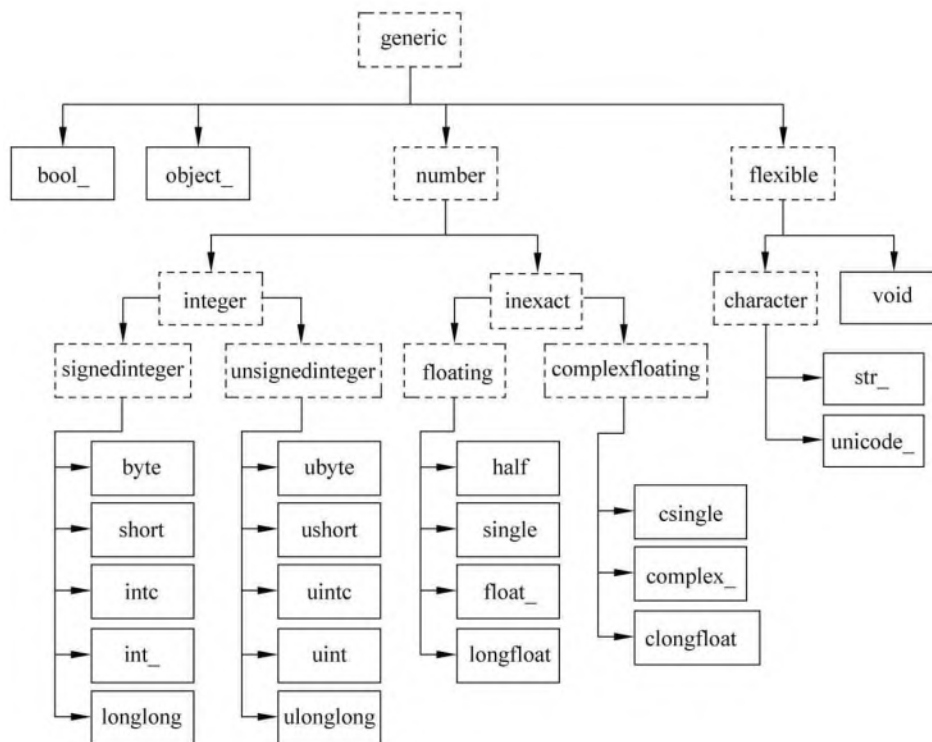


图5.3 SciPy中的数据类型

对一些只有几万行或十几万行的小型数据集而言，数据所占用的内存可能不是特别重要，但是在分析上百万行的数据时，由于Pandas是将数据先读取到内存中，此时正确的数据类型就变得无比重要了。因此，数据分析人员需要详细地了解每一列的数据类型，要完成这一工作，可以采用如下代码。

```

>>> retail_data.dtypes
InvoiceNo      object
StockCode      object
Description    object
Quantity       int64
InvoiceDate    object
UnitPrice      float64
CustomerID     float64
Country        object
Total_price    float64
dtype: object

```

通常在数据分析时还会对数据所占内存进行精确统计，例如：

```
>>> mem = retail_data.memory_usage(deep = True)
>>> mem
Index          80
InvoiceNo      34149621
StockCode      33648312
Description    45252183
Quantity       4335280
InvoiceDate    39006854
UnitPrice      4335280
CustomerID     4335280
Country        38137522
Total_price    4335280
dtype: int64
```

上述代码统计了retail_data中每一列数据所占的存储空间（字节数），其中的deep=True参数代表是否计算引用的对象的内存使用情况。此外，也可以将其转换为以兆字节方式显示，代码如下。

```
>>> round(mem.sum()/(1024 * 1024))
198.0
```

在对各列数据类型、所占内存大小有了了解后，就需要根据具体需求来调整数据的类型。例如，Country列的可能取值并不多，完全可以转换为Pandas中的category变量以减少存储空间，那么可以采用如下代码。

```

>>> print('Unique country: ', retail_data['Country'].nunique())
>>> retail_data['Country'] = retail_data['Country'].astype('category')
>>> new_mem = retail_data.memory_usage(deep = False)
>>> print(new_mem)
Unique country: 38
Index          80
InvoiceNo      4335272
StockCode      4335272
Description    4335272
Quantity       4335272
InvoiceDate    4335272
UnitPrice      4335272
CustomerID     4335272
Country        543493
Total_price    4335272
dtype: int64

```

可以看出Country的取值一共只有38个国家，经过转换后存储空间明显减少，只有原来存储空间的1.4%。

```

>>> round(new_mem['Country']/mem['Country'],3)
0.014

```

类似地，还可以对CustomerID列进行数据类型转换，目前CustomerID是浮点数类型，而实际上该数据应该是整数。不过如果直接运行如下代码：

```

>>> retail_data['CustomerID'] = retail_data['CustomerID'].astype('int')

```

会得到错误提示“ValueError: Cannot convert non-finite values (NA or inf) to integer”，这是由于CustomerID列中有缺失值。如果要进行数据类型转换，需要先使用如下代码将缺失值补齐。

```

>>> retail_data['CustomerID'] = retail_data['CustomerID'].fillna(0)

```

关于如何处理缺失值，本书会在5.3节进行更详细的介绍。最后需要处理的是InvoiceDate列，该列应该是时间类型，to_datetime（）

函数可以完成时间类型的转换。因此需要用如下代码进行转换。

```
>>> retail_data['InvoiceDate'] = pd.to_datetime(\
    retail_data['InvoiceDate'])
>>> retail_data[['InvoiceDate']].dtypes
InvoiceDate    datetime64[ns]
dtype: object
```

5.3 缺失数据与异常数据处理

理想总是很丰满，现实总是很骨感。数据也是一样，真实数据分析中的数据经常会有各种问题，要么有缺失，要么有重复，有的情况下数据还有错误，有时数据需要进行变换才能用于分析。本节将讨论如何处理这些问题。

5.3.1 缺失值与重复值

前面已经介绍过先用head（）和tail（）函数查看一下数据的前5行和后5行，对数据有一个基本了解。例如，从retail_data的前5行可以发现UnitPrice中存在取值NaN（即数据缺失），Description列也存在缺失值，如图5.4所示。

```
>>> retail_data_na = pd.read_csv("../data/Online_Retail_Fake.csv")
>>> retail_data_na.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010/12/1 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010/12/1 8:26	NaN	17850.0	United Kingdom
2	536365	84406B	NaN	8	2010/12/1 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010/12/1 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010/12/1 8:26	3.39	17850.0	United Kingdom

图5.4 存在缺失值

既然已经知道数据中有缺失值，下一步就需要统计有多少缺失数据，代码如下。

```
>>> retail_data_na.isnull().sum()
InvoiceNo      0
StockCode      1
Description    1456
Quantity       0
InvoiceDate    0
UnitPrice      3
CustomerID    135082
Country        1
dtype: int64
>>> retail_data_na.isnull().sum().sum()
136543
```

`isnull()` 函数会对 `retail_data` 中的所有数据是否存在缺失值 (NaN) 进行判断，它返回的是一个与原数据维度大小相同的 `DataFrame`，而取值 (True/False) 由该处是否有缺失值决定。有缺失值的情况取值为 True，反之则为 False。由于 `isnull()` 函数运行后得到的是一个新的 `DataFrame`，所以需要先通过 `sum()` 函数对每列求和得到该列的缺失值总数，之后再次利用 `sum()` 函数求和来计算总的缺失值个数。此外，Pandas 中还存在一个与 `isnull()` 函数对应的 `notnull()` 函数，取值刚好相反，分析中通常用它来过滤数据。然而现实中缺失值并不总是以 NaN 的形式存在，一旦开始数据分析，就会发现各种五花八门的代表缺失值的方式，如 0 代表缺失、-999 代表缺失等。那么应该如何处理呢？Pandas 中的数据读入函数已经充分考虑到了各种可能性，所以可以在读入数据时就完成这样的映射，例如：

```
>>> retail_data_na = pd.read_csv("../data/Online_Retail_Fake.csv",
    na_values = [0])
>>> retail_data_na.isnull().sum()
InvoiceNo          0
StockCode          1
Description       1456
Quantity          0
InvoiceDate        0
UnitPrice         2518
CustomerID       135082
Country           1
dtype: int64
```

na_values=[0]代表如果数据中有取值为0的，那么该值代表了缺失值，加了这个参数后可以发现Country、StockCode、UnitPrice列存在更多的缺失值。na_values参数的取值也可以是一个列表，例如：

```
>>> retail_data_na = pd.read_csv("../data/Online_Retail_Fake.csv",
    na_values = [0, 'Wrong booking'])
>>> retail_data_na.isnull().sum()
```

对缺失数据了解后，下一步通常需要查看数据是否存在重复，duplicated（）函数刚好提供了这一功能，代码如下。

```
>>> retail_data_new.duplicated().sum()
5269
```

数据中存在这么多的重复值还是很少见的情况，因此，需要进一步分析到底有哪些重复值，如图5.5所示。

```
>>> retail_data_new[retail_data_new.duplicated()].head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Total_price
517	536409	21866	UNION JACK FLAG LUGGAGE TAG	1	2010/12/1 11:45	1.25	17908.0	United Kingdom	1.25
527	536409	22866	HAND WARMER SCOTTY DOG DESIGN	1	2010/12/1 11:45	2.10	17908.0	United Kingdom	2.10
537	536409	22900	SET 2 TEA TOWELS I LOVE LONDON	1	2010/12/1 11:45	2.95	17908.0	United Kingdom	2.95
539	536409	22111	SCOTTIE DOG HOT WATER BOTTLE	1	2010/12/1 11:45	4.95	17908.0	United Kingdom	4.95
555	536412	22327	ROUND SNACK BOXES SET OF 4 SKULLS	1	2010/12/1 11:49	2.95	17920.0	United Kingdom	2.95

图5.5 分析重复值

从前5行数据的输出没有看出什么问题，那么需要具体筛选某一行数据来看看有什么异常。利用已经掌握的数据筛选功能可以很轻松地构建如下的条件选择来查看InvoiceNo为536409的数据行，如图5.6所示。

```
>>> retail_data_new[(retail_data_new['InvoiceNo'] == '536409')\
    &(retail_data_new['StockCode'] == '21866')]
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Total_price
494	536409	21866	UNION JACK FLAG LUGGAGE TAG	1	2010/12/1 11:45	1.25	17908.0	United Kingdom	1.25
517	536409	21866	UNION JACK FLAG LUGGAGE TAG	1	2010/12/1 11:45	1.25	17908.0	United Kingdom	1.25

图5.6 查看InvoiceNo为536409的数据行

从上面输出的两行数据看，很可能是用户购买了两件相同物品，但是本章开始就提到了待分析数据来源于某在线商店，因此很容易提出疑问：为什么不直接把数量记为2呢？这样的问题我们目前无法回答，但是如果是真实的数据分析，就需要去寻找答案了。再来看看duplicated（）函数，该函数还可以指定subset和keep参数，subset可以指定只针对特定列数据来判断重复。例如，如果CustomerID是唯一的，那么检查的时候可以指定subset='CustomerID'。而keep参数取值可以为'first'，'last'，False，取值为'first'代表了第一个取值不标记为重复，'last'则相反，False则代表只要相同就是重复。

5.3.2 处理缺失数据

首先利用pd.DataFrame（）函数来构造一个包含缺失数据的DataFrame，如图5.7所示，该函数的参数依次是数据，数据的索引，数据列名，代码如下。

```
>>> df = pd.DataFrame(np.random.randint(0,100,15).reshape(5, 3),
                        index = ['a', 'b', 'c', 'd', 'e'],
                        columns = ['c1', 'c2', 'c3'])
>>> df['c4'] = np.nan
>>> df.loc['f'] = np.arange(10,14)
>>> df.loc['g'] = np.nan
>>> df['c5'] = np.nan
>>> df['c4']['a'] = 18
>>> df
```

利用isnull（）和notnull（）函数判断是否存在缺失值，如图5.8和图5.9所示。

```
>>> df.isnull()
>>> df.notnull()
```

	c1	c2	c3	c4	c5
a	96.0	96.0	40.0	18.0	NaN
b	25.0	5.0	48.0	NaN	NaN
c	76.0	99.0	28.0	NaN	NaN
d	52.0	67.0	16.0	NaN	NaN
e	75.0	76.0	70.0	NaN	NaN
f	10.0	11.0	12.0	13.0	NaN
g	NaN	NaN	NaN	NaN	NaN

图5.7 包含缺失数据的DataFrame

	c1	c2	c3	c4	c5
a	False	False	False	False	True
b	False	False	False	True	True
c	False	False	False	True	True
d	False	False	False	True	True
e	False	False	False	True	True
f	False	False	False	False	True
g	True	True	True	True	True

图5.8 isnull（）判断结果

如果想丢弃c4列的重复值，可以使用如下代码。

```
>>> df['c4'].dropna() # inplace = False
a    18.0
f    13.0
Name: c4, dtype: float64
```

如果只要某行有缺失就丢弃，可以使用如下代码。

```
>>> df.dropna()
```

由于df中每行都有缺失数据，因此这样处理后得到的数据就是空的。通常这种情况会给dropna（）函数加上参数，如how='all'代表了要整行都是缺失数据才丢弃，代码如下。

```
df.dropna(how='all')
```

结果如图5.10所示。

	c1	c2	c3	c4	c5
a	True	True	True	True	False
b	True	True	True	False	False
c	True	True	True	False	False
d	True	True	True	False	False
e	True	True	True	False	False
f	True	True	True	True	False
g	False	False	False	False	False

图5.9 notnull（）判断结果

	c1	c2	c3	c4	c5
a	38.0	19.0	86.0	18.0	NaN
b	24.0	91.0	84.0	NaN	NaN
c	20.0	23.0	77.0	NaN	NaN
d	62.0	41.0	87.0	NaN	NaN
e	59.0	65.0	24.0	NaN	NaN
f	10.0	11.0	12.0	13.0	NaN

图5.10 整行数据缺失

而axis=1代表了要整列都是缺失数据才丢弃该列，代码如下。

```
>>> df.dropna(how = 'all', axis = 1)
```

结果如图5.11所示。更灵活方式是指定一个阈值thresh，要求至少有几列数据不是缺失值，代码如下。

```
>>> df.dropna(thresh = 4)
```

结果如图5.12所示。

	c1	c2	c3	c4
a	96.0	96.0	40.0	18.0
b	25.0	5.0	48.0	NaN
c	76.0	99.0	28.0	NaN
d	52.0	67.0	16.0	NaN
e	75.0	76.0	70.0	NaN
f	10.0	11.0	12.0	13.0
g	NaN	NaN	NaN	NaN

图5.11 整列数据缺失

	c1	c2	c3	c4	c5
a	96.0	96.0	40.0	18.0	NaN
f	10.0	11.0	12.0	13.0	NaN

图5.12 指定阈值

5.3.3 NumPy与Pandas对缺失数据的不同处理方式

虽然Pandas构建在NumPy之上，但是它们对缺失数据却有着不同的处理方式，下面的代码可以很好地说明二者的不同。

```
>>> a = np.array([np.nan, 1, 2, np.nan, 3])
>>> a
array([nan,  1.,  2., nan,  3.])
>>> s = pd.Series(a)
>>> s
0    NaN
1    1.0
2    2.0
3    NaN
4    3.0
dtype: float64
```

上述代码中a是一个NumPy中的数组（np.array），而s是Pandas中的一个Series，两者取值一样，但是如果对它们求平均值，会得到不同结果。

```
>>> a.mean(), s.mean()
(nan, 2.0)
```

二者输出不同的原因在于NumPy中只要有数据缺失就会返回NaN，而在Pandas中则会跳过该值对剩余的数值进行相应计算，请读者一定要理解二者不同的处理方式。接下来再来构造一个新的DataFrame，如图5.13所示，以更进一步说明Pandas中进行运算时对缺失值的处理。

```

>>> df2 = df.copy()
>>> df2.loc['g'].c1 = 0
>>> df2.loc['g'].c3 = 0
>>> df2
>>> df2['c4'] + 1
a    19.0
b     NaN
c     NaN
d     NaN
e     NaN
f    14.0
g     NaN
Name: c4, dtype: float64
>>> df2['c4'].cumsum()
a    18.0
b     NaN
c     NaN
d     NaN
e     NaN
f    31.0
g     NaN
Name: c4, dtype: float64

```

	c1	c2	c3	c4	c5
a	96.0	96.0	40.0	18.0	NaN
b	25.0	5.0	48.0	NaN	NaN
c	76.0	99.0	28.0	NaN	NaN
d	52.0	67.0	16.0	NaN	NaN
e	75.0	76.0	70.0	NaN	NaN
f	10.0	11.0	12.0	13.0	NaN
g	0.0	NaN	0.0	NaN	NaN

图5.13 新构造DataFrame

从上述代码看出任何数值与NaN进行运算都返回NaN，在进行累加求和时，中间的缺失值会忽略，只对非缺失值进行累加。

5.3.4 填充缺失值

在缺失数据较少又不影响数据分析时，数据分析人员第一选择可能会考虑将数据丢弃。有时则会考虑采用不同方式来尝试填补这个缺失值，利用fillna（）函数则可以完成这个功能，如图5.14所示。

```
>>> fill_0 = df.fillna(0)
>>> fill_0
```

例如，上述代码中fillna（0）代表用指定值0进行填充。或者在填充时指定一个limit，如每列中最多填充3个，之后就不再填充，代码如下，填充结果如图5.15所示。

```
>>> df.fillna(0,limit=3) # 每列中的 limit
```

	c1	c2	c3	c4	c5
a	38.0	19.0	86.0	18.0	0.0
b	24.0	91.0	84.0	0.0	0.0
c	20.0	23.0	77.0	0.0	0.0
d	62.0	41.0	87.0	0.0	0.0
e	59.0	65.0	24.0	0.0	0.0
f	10.0	11.0	12.0	13.0	0.0
g	0.0	0.0	0.0	0.0	0.0

图5.14 填充结果（1）

当然，数据分析中更常用的方式是用均值进行填充，代码如下，填充结果如图5.16所示。

```
>>> df.fillna(df.mean())
```

对于有一定时序的数据，此时可以考虑用它前面或后面的值来填充。例如，下面的代码分别用缺失数据前一行和后一行数据来填充。

```
>>> df['c4'].fillna(method = 'ffill')
a    18.0
b    18.0
c    18.0
d    18.0
e    18.0
f    13.0
g    13.0
Name: c4, dtype: float64
>>> df['c4'].fillna(method = 'bfill')
a    18.0
b    13.0
c    13.0
d    13.0
e    13.0
f    13.0
g     NaN
Name: c4, dtype: float64
```

	c1	c2	c3	c4	c5
a	96.0	96.0	40.0	18.0	0.0
b	25.0	5.0	48.0	0.0	0.0
c	76.0	99.0	28.0	0.0	0.0
d	52.0	67.0	16.0	0.0	NaN
e	75.0	76.0	70.0	NaN	NaN
f	10.0	11.0	12.0	13.0	NaN
g	0.0	0.0	0.0	NaN	NaN

图5.15 填充结果（2）

	c1	c2	c3	c4	c5
a	96.000000	96.0	40.000000	18.0	NaN
b	25.000000	5.0	48.000000	15.5	NaN
c	76.000000	99.0	28.000000	15.5	NaN
d	52.000000	67.0	16.000000	15.5	NaN
e	75.000000	76.0	70.000000	15.5	NaN
f	10.000000	11.0	12.000000	13.0	NaN
g	55.666667	59.0	35.666667	15.5	NaN

图5.16 填充结果（3）

某些特殊情况下，还可以手工补齐某几个指定的缺失值，下面的代码完成了指定索引标签处的值用指定值来代替的功能。

```
>>> fill_values = pd.Series([1,2], index = ['b', 'c'])
>>> df['c4'].fillna(fill_values)
a    18.0
b     1.0
c     2.0
d     NaN
e     NaN
f    13.0
g     NaN
Name: c4, dtype: float64
```

除了以上几种简单的填充缺失值的方式，还可以利用 `interpolate()` 插值函数来完成缺失值的补齐，例如：

```
>>> s = pd.Series([1,2,np.nan,5,np.nan,9])
>>> s
0    1.0
1    2.0
2    NaN
3    5.0
4    NaN
5    9.0
dtype: float64
>>> s.interpolate()
0    1.0
1    2.0
2    3.5
3    5.0
4    7.0
5    9.0
dtype: float64
```

对于索引不是时间的插值，计算的时候只是根据数据的值来进行。例如，在2和5之间插入3.5就可以了。然而，如果索引为时间，插值时就需要思考是否需要处理时间的间隔，例如：

```

>>> import datetime
>>> ts = pd.Series([1, np.nan, 2],
                    index=[datetime.datetime(2016, 1, 1),
                           datetime.datetime(2016, 2, 1),
                           datetime.datetime(2016, 4, 1)])

>>> ts
2016-01-01    1.0
2016-02-01    NaN
2016-04-01    2.0
dtype: float64
>>> ts.interpolate()
2016-01-01    1.0
2016-02-01    1.5
2016-04-01    2.0
dtype: float64
>>> ts.interpolate(method='time')
2016-01-01    1.000000
2016-02-01    1.340659
2016-04-01    2.000000
dtype: float64

```

不考虑时间间隔，上述代码的插值结果就应该是1.5，而如果考虑了时间，那么中间还有2016-03-01，因此插值的时候2016-02-01得到的就是1.340659了。对非时间索引也可以采用类似方法指定参数method='values'。下面代码中第一种插值方式就是不考虑索引的插值，第二种插值方式是考虑了索引的插值。

```

>>> s = pd.Series([0, np.nan, 20], index=[0, 1, 10])
>>> s
0      0.0
1      NaN
10     20.0
dtype: float64
>>> s.interpolate()
0      0.0
1     10.0
10     20.0
dtype: float64
>>> s.interpolate(method='values')
0      0.0
1      2.0
10     20.0
dtype: float64

```

5.4 处理重复数据

假设有如下数据：

```
>>> data = pd.DataFrame({'a': ['x'] * 3 + ['y'] * 4,  
                          'b': [1, 1, 2, 3, 3, 4, 4]})  
  
>>> Data  
a  b  
0  x  1  
1  x  1  
2  x  2  
3  y  3  
4  y  3  
5  y  4  
6  y  4
```

通过`data.duplicated()` 函数，可以输出哪些行有重复数据。

```
>>> data.duplicated()  
0    False  
1     True  
2    False  
3    False  
4     True  
5    False  
6     True  
dtype: bool
```

之后利用`data.drop_duplicates()` 函数可以将其丢弃。

```
>>> data.drop_duplicates()  
a  b  
0  x  1  
2  x  2  
3  y  3  
5  y  4
```

与`data.duplicated()` 函数类似，也可以选择是保留前面的数据还是后面的重复数据。

```
>>> data.drop_duplicates(keep = 'last')
a  b
1  x  1
2  x  2
4  y  3
6  y  4
```

还有的情况下，数据集中只要某几列数据相同，那么就是重复数据，此时可以在`drop_duplicates()`函数中输入参数来指定检查的列来完成该功能，代码如下。

```
>>> data['c'] = np.arange(7)
>>> data
   a  b  c
0  x  1  0
1  x  1  1
2  x  2  2
3  y  3  3
4  y  3  4
5  y  4  5
6  y  4  6
>>> data.drop_duplicates(['a', 'b'])
   a  b  c
0  x  1  0
2  x  2  2
3  y  3  3
5  y  4  5
```

上述代码首先在原来的数据中添加了新的一列c，这样data中就没有重复数据了，参数['a', 'b']代表检查的是这两列，那么只要这两列中有重复数据就会丢弃。

5.5 异常值

数据分析过程中，除了重复值和缺失值，有时还会遇到异常值。异常值的判断一方面来自常识（如天气温度不可能是100℃，人的寿命不会是200岁等），另一方面来自数据字典给出的取值范围。此外，还

可以通过数据的标准差来判断，例如认为两个标准差以外的数据需要特别进行观察，代码如下。

```
>>> df = pd.DataFrame({'Data':np.random.normal(size = 200)})
>>> df.head()
>>> df[np.abs(df['Data'] - df['Data'].mean()) <= (2 * df['Data'].std())]
   Data
0  0.746054
1  0.656542
...  ...
198  0.866292
199  -0.481225
190 rows × 1 columns
```

上述代码利用随机数构造了一个DataFrame，如果某个数据在两个标准差外的话，该数据就是异常，那么可以构造一个条件选择如下。

```
>>> mask = np.abs(df.Data - df.Data.mean()) >= (2 * df.Data.std())
>>> df[mask]
   Data
21  2.619704
41  2.008573
...  ...
141  -2.139558
150  -2.192969
10 rows × 1 columns
```

对符合该条件的数据用均值替代。

```
df[mask] = df.Data.mean()
df[mask]
   Data
21  2.619704
41  2.008573
...  ...
141  -2.139558
150  -2.192969
10 rows × 1 columns
```

5.6 描述性统计

假设所有的缺失数据、重复数据、异常值都已经处理完毕，那么数据分析的下一步工作就是对数据进行描述性统计了，通过描述性统计可以对数据分布有更深入的了解。下面的代码获得了retail_data中所有类型为数值列的描述性统计，结果如图5.17所示。

```
>>> retail_data.describe(include = [np.number]).T
```

	count	mean	std	min	25%	50%	75%	max
Quantity	541910.0	9.552237	218.080957	-80995.00	1.00	3.00	10.00	80995.0
UnitPrice	541910.0	4.611117	96.759764	-11062.06	1.25	2.08	4.13	38970.0
CustomerID	406828.0	15287.695176	1713.600085	12346.00	13953.00	15152.00	16791.00	18287.0
Total_price	541910.0	17.987805	378.810474	-168469.60	3.40	9.75	17.40	168469.6

图5.17 描述性统计

describe（）函数默认会给出数据计数、均值、标准差、最大值、最小值和第1，2，3分位数。上面最后的.T代表求转置，即行列变换。此外，describe（）函数还可以指定具体的分位数，如下代码分别指定了函数输出要显示的分位数，结果如图5.18所示。

```
>>> retail_data.describe(include = [np.number],
                           percentiles = [.05, .10, .25, .5, .75, .9, .95]).T
```

	count	mean	std	min	5% ...	50%	75%	90%	95%	max
Quantity	541910.0	9.552237	218.080957	-80995.00	1.00 ...	3.00	10.00	24.00	29.00	80995.0
UnitPrice	541910.0	4.611117	96.759764	-11062.06	0.42 ...	2.08	4.13	7.95	9.95	38970.0
CustomerID	406828.0	15287.695176	1713.600085	12346.00	12626.00 ...	15152.00	16791.00	17719.00	17905.00	18287.0
Total_price	541910.0	17.987805	378.810474	-168469.60	0.83 ...	9.75	17.40	31.80	59.40	168469.6

4 rows x 12 columns

图5.18 指定分位数

对数值类型，describe（）函数输出均值等信息，而对其他类型，该函数的输出则是计数、不同值个数、出现频率等信息，代码如下。

```
>>> retail_data.describe(include = [np. object]).T
```

结果如图5. 19所示。

	count	unique	top	freq
InvoiceNo	541910	25900	573585	1114
StockCode	541909	4070	85123A	2313
Description	540454	4224	WHITE HANGING HEART T-LIGHT HOLDER	2369
InvoiceDate	541910	23260	2011/10/31 14:41	1114
Country	541909	38	United Kingdom	495477

图5.19 describe（）函数对其他类型的输出

此外，针对不同值个数的统计，还可以用nunique（）函数来获得，代码如下。

```
>>> retail_data.nunique()
InvoiceNo      25900
StockCode      4070
...
Country        38
Total_price    6206
Length: 9, dtype: int64
```

对基本统计信息有了一定了解后，通常数据分析人员还会查看一定数量的最大、最小值。例如，下面的代码分别查看了Total_price最大的50个和最小的10个数据，结果如图5. 20和图5. 21所示。

```
>>> retail_data.nlargest(50, 'Total_price').head()
>>> retail_data.nsmallest(10, 'Total_price').head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Total_price
540421	581483	23843	PAPER CRAFT , LITTLE BIRDIE	80995	2011/12/9 9:15	2.08	16446.0	United Kingdom	168469.60
61619	541431	23166	MEDIUM CERAMIC TOP STORAGE JAR	74215	2011/1/18 10:01	1.04	12346.0	United Kingdom	77183.60
222680	556444	22502	PICNIC BASKET WICKER 60 PIECES	60	2011/6/10 15:28	649.50	15098.0	United Kingdom	38970.00
15017	537632	AMAZONFEE	AMAZON FEE	1	2010/12/7 15:08	13541.33	NaN	United Kingdom	13541.33
299982	A563185	B	Adjust bad debt	1	2011/8/12 14:50	11062.06	NaN	United Kingdom	11062.06

图5. 20 查看最大值

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Total_price
540422	C581484	23843	PAPER CRAFT , LITTLE BIRDIE	-80995	2011/12/9 9:27	2.08	16446.0	United Kingdom	-168469.60
61624	C541433	23166	MEDIUM CERAMIC TOP STORAGE JAR	-74215	2011/1/18 10:17	1.04	12346.0	United Kingdom	-77183.60
222681	C556445	M	Manual	-1	2011/6/10 15:31	38970.00	15098.0	United Kingdom	-38970.00
524602	C580605	AMAZONFEE	AMAZON FEE	-1	2011/12/5 11:36	17836.46	NaN	United Kingdom	-17836.46
43702	C540117	AMAZONFEE	AMAZON FEE	-1	2011/1/5 9:55	16888.02	NaN	United Kingdom	-16888.02

图5.21 查看最小值

数据也可以通过sort_values（）排序的方式得到，代码如下。

```
>>> retail_data.sort_values('Total_price',ascending = False).head()
```

结果如图5.22所示。

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Total_price
540421	581483	23843	PAPER CRAFT , LITTLE BIRDIE	80995	2011/12/9 9:15	2.08	16446.0	United Kingdom	168469.60
61619	541431	23166	MEDIUM CERAMIC TOP STORAGE JAR	74215	2011/1/18 10:01	1.04	12346.0	United Kingdom	77183.60
222680	556444	22502	PICNIC BASKET WICKER 60 PIECES	60	2011/6/10 15:28	649.50	15098.0	United Kingdom	38970.00
15017	537632	AMAZONFEE	AMAZON FEE	1	2010/12/7 15:08	13541.33	NaN	United Kingdom	13541.33
299982	A563185	B	Adjust bad debt	1	2011/8/12 14:50	11062.06	NaN	United Kingdom	11062.06

图5.22 排序显示

ascending=False 参数表明是降序排列，升序则为True。sort_values（）函数还可以同时指定多列排序，例如首先对Total_price列排序，如果有重复的，则根据Country列决定先后顺序，运行如下代码，结果如图5.23所示。

```
>>> retail_data.sort_values(['Total_price','Country'],ascending = True).head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Total_price
540422	C581484	23843	PAPER CRAFT , LITTLE BIRDIE	-80995	2011/12/9 9:27	2.08	16446.0	United Kingdom	-168469.60
61624	C541433	23166	MEDIUM CERAMIC TOP STORAGE JAR	-74215	2011/1/18 10:17	1.04	12346.0	United Kingdom	-77183.60
222681	C556445	M	Manual	-1	2011/6/10 15:31	38970.00	15098.0	United Kingdom	-38970.00
524602	C580605	AMAZONFEE	AMAZON FEE	-1	2011/12/5 11:36	17836.46	NaN	United Kingdom	-17836.46
43702	C540117	AMAZONFEE	AMAZON FEE	-1	2011/1/5 9:55	16888.02	NaN	United Kingdom	-16888.02

图5.23 多列排序

第6章 数据整理

Structuring datasets to facilitate analysis.

——*Hadley Wickham*

有统计表明数据科学家80%的时间都是用于数据清洗和准备的过程。数据清洗的第一步目标可以总结为“完全合一”。

- 完整性：单条数据是否存在空值，统计的字段是否完善。
- 全面性：观察某一列的全部数值，可以看到该列的平均值、最大值、最小值。我们可以通过常识来判断该列是否有问题，如数据定义、单位标识、数值本身等方面。
- 合法性：数据的类型、内容、大小的合法性，如销售额为负、年龄超过了200岁等。
- 唯一性：数据是否存在重复记录，因为数据通常来自不同渠道的汇总，重复的情况是常见的。行数据、列数据都需要是唯一的，例如一个人不能重复记录多次，而相同数据不能多列重复。

关于数据空值、异常值、重复值处理，前面已经有详细讨论。本章的目标是重点讨论数据清洗的第二步目标：数据整理（Tidy Data）。Tidy Data得名于著名数据科学家Hadley Wickham于2014年在 *Journal of Statistical Software* 发表的同名论文。在该论文中，Hadley给出了Tidy Data的定义，同时对5种典型的混乱数据格式进行了讨论。基于该理念，Hadley还开发了基于R语言的软件包：reshape、reshape2、plyr和dplyr，本章将完全按照Hadley的论文用Pandas软件包的功能来完成他提出的5种混乱数据的整理。

6.1 什么是数据整理

6.1.1 数据的语义

如Hadley的论文中所指出，现实中大多数统计数据是矩形的表格，由行和列构成。各列几乎总带有标签，而各行有时也带有标签。表6.1展示了关于一个假想实验的数据，其格式很常见。表格有2列3行，行列都有标签。

表6.1 假想实验数据

观察对象 \ 治疗方案	治疗方案 a	治疗方案 b
John Smith	—	2
Jane Doe	16	11
Mary Johnson	3	11

这是其中一种数据的展现方式，同样，数据也可以用表6.2的方式展示。表6.2展示的数据和表6.1相同，但是行列被转置。

表6.2 转置表示

治疗方案 \ 观察对象	John Smith	Jane Doe	Mary Johnson
治疗方案 a	—	16	3
治疗方案 b	2	11	1

数据相同，但是展示的布局不同。显然，除了数据的外观，需要一种统一方式来描述表格所展示数值的语义或含义。因此，需要对数据进行下列定义。

- 一个数据集是一组“值”的集合，通常不是数字（定量的）就是字符串（定性的）。
- 每个“值”属于一个变量和一个观察对象。
- 一个变量包含测量各个观察单元同一内在属性的所有值（如高度、温度、时长）。

- 一个观察对象包含测量该对象各不同属性的所有值（如一个人、某一天或一场比赛）。

基于此，表6.3将前面的数据重新排列，使各个值、变量和观察值更加清晰。

表6.3 数据重新排列

观察对象	治疗方案	结果
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

这个数据集包含3个变量、6个观察对象的18个变量，具体包括：

- 人：有3个可能的值（John Smith、Jane Doe 和 Mary Johnson）；
- 治疗方案：有两个可能的值（a和b）；
- 结果：有5个或6个可能的值（—，16，3，2，11，1），当然这取决于如何看待缺失值。

6.1.2 整齐的数据

整齐的数据是指数据含义和其结构的标准化匹配方式。一个数据集是混乱还是整齐，取决于行、列、表格与观察对象、变量和类型如何匹配。在整齐的数据中：

- 每个变量组成一列；
- 每个观察对象的所有属性组成一行；
- 每个观察单元的类型组成一个表格。

显然，表6.3是表6.1和表6.2的整齐版本。每行代表一个观察，即某一治疗方案对某个人的效果，每列是一个变量。整齐的数据能让分析师或计算机更容易地提取所需变量，因为它提供了一个构成数据集的标准方式。不幸的是，真实的数据集将会而且经常会以各种可能的方式违反整齐数据的3个原则。典型的混乱数据通常会具有以下5个常见的问题。

- 列标题是值，而非变量名。
- 多个变量存储在一列中。
- 变量既在列中存储，又在行中存储。
- 多个观测单元存储在表中。
- 一个观测单元存储在多个表中。

6.2 数据整理实战

本节将针对上面列出的5个常见混乱数据的问题，提出解决方案。

6.2.1 列标题是值，而非变量名

第一种常见的混乱数据集是为展示而设计的表格数据，变量既构成列，又构成行，列标题是值，而非变量名。图6.1展示了这类典型数据集的一个子集。这个数据集记录的是在美国部分居民收入和宗教信仰的关系。数据来源于皮尤研究中心（Pew Research Center）所做的一个报告。

	religion	<\$10k	\$10-20k	\$20-30k	\$30-40k	\$40-50k	\$50-75k
0	Agnostic	27	34	60	81	76	137
1	Atheist	12	27	37	52	35	70
2	Buddhist	27	21	30	34	33	58
3	Catholic	418	617	732	670	638	1116
4	Dont know/refused	15	14	15	11	10	35
5	Evangelical Prot	575	869	1064	982	881	1486
6	Hindu	1	9	7	9	11	34
7	Historically Black Prot	228	244	236	238	197	223
8	Jehovahs Witness	20	27	24	24	21	30
9	Jewish	19	19	25	25	30	95

图6.1 宗教信仰与收入关系

这个数据表将“< \$10k”“\$10-20k”这些收入变量的值作为了列名，而表中的数值应该是对应另一变量——人数。因此，该数据集本来应该有3个变量：宗教、收入和人数，对其进行整理，需要把它融合（又称融化）或堆叠（有的数据分析师也把这一过程称为将宽的数据集变长或变高，本书后面统一用融合来代表这一含义）。所谓融合是指将已经是变量的各列的列表（简称为id_vars）进行参数化，其他的列转换为两个变量：一个被称作列（var_name）的新变量，它包含重复的列标题；另一个被称作值（value_name）的新变量，它包含从之前分离的列中提供联系的数据值。图6.2通过一个简单的数据集对这一过程进行了说明，这里以row变量作为id_vars，将原来的列名融合后成为新列column中的值（对应var_name），原来列中的值则融合到了另一列（对应value_name）。

row	a	b	c
A	1	4	7
B	2	5	8
C	3	6	9

(a) 原始数据

row	column	value
A	a	1
B	a	2
C	a	3
A	b	4
B	b	5
C	b	6
A	c	7
B	c	8
C	c	9

(b) “熔化” 的数据

图6.2 数据的融合

融合的结果是一个“熔化”的数据集，下面用如下数据来做一个示例，如图6.3所示。

```
>>> df = pd.read_csv("../data/pew - raw.csv")
>>> df.head(5)
```

	religion	<\$10k	\$10-20k	\$20-30k	\$30-40k	\$40-50k	\$50-75k
0	Agnostic	27	34	60	81	76	137
1	Atheist	12	27	37	52	35	70
2	Buddhist	27	21	30	34	33	58
3	Catholic	418	617	732	670	638	1116
4	Dont know/refused	15	14	15	11	10	35

图6.3 数据示例

上面的数据中，列标题实际应该是变量income的取值，因此需要将religion作为id_vars，原来的列名将变成新变量income，由参数var_name指定，对其进行融合操作，代码如下。

```
>>> formatted_df = pd.melt(df, id_vars = ["religion"],
    var_name = "income", value_name = "freq")
>>> formatted_df = formatted_df
    .sort_values(by = ["religion"])
>>> formatted_df.head(7)
```

	religion	income	freq
0	Agnostic	<\$10k	27
30	Agnostic	\$30-40k	81
40	Agnostic	\$40-50k	76
50	Agnostic	\$50-75k	137
10	Agnostic	\$10-20k	34
20	Agnostic	\$20-30k	60
41	Atheist	\$40-50k	35

图6.4 melt（）函数操作结果

结果如图6.4所示。这里采用了melt（）函数来完成这一功能，该函数有5个参数，各参数含义如下。

- frame：需要处理的数据框。
- id_vars：保持原样的数据列。
- value_vars：需要被转换成变量值的数据列。
- var_name：转换后变量的列名。
- value_name：数值变量的列名。

在本例中通过融合这一操作，指定religion列不变，所有列转换为对应income（参数var_name指定）变量的变量值，原表中的数值变量列名为freq（参数value_name指定）。除了采用融合操作，也可以利用堆叠（stack）功能来完成数据变换，代码如下。

```
>>> formatted_df = df.set_index('religion')
>>> formatted_df = formatted_df.stack()
>>> formatted_df.index = formatted_df.index.rename('income', level=1)
>>> formatted_df.name = 'freq'
>>> formatted_df = formatted_df.reset_index()
>>> formatted_df.head(7)
```

结果如图6.5所示。上述代码中，首先利用set_index（）将religion列设置为了行索引，之后利用stack（）函数将所有的列堆叠，即将其列作为二级索引，得到的数据如图6.6所示。

	religion	income	freq
0	Agnostic	<\$10k	27
30	Agnostic	\$30-40k	81
40	Agnostic	\$40-50k	76
50	Agnostic	\$50-75k	137
10	Agnostic	\$10-20k	34
20	Agnostic	\$20-30k	60
41	Atheist	\$40-50k	35

图6.5 stack（）函数操作结果

religion		
Agnostic	<\$10k	27
	\$10-20k	34
	\$20-30k	60
	\$30-40k	81
	\$40-50k	76
	\$50-75k	137
Atheist	<\$10k	12
	\$10-20k	27
	\$20-30k	37
	\$30-40k	52
	\$40-50k	35
	\$50-75k	70
Buddhist	<\$10k	27
	\$10-20k	21
	\$20-30k	30
	\$30-40k	34
	\$40-50k	33
	\$50-75k	58

图6.6 堆叠后的数据

之后利用 . index.rename（）函数，将第二级索引名修改为 income，这里的第一级索引是前面提到的 religion，堆叠操作将原来的所有列名作为了第二层索引的值。接下来将 formatted_df.name 修改为 freq，最后再将所有的索引还原到列，就得到了和融合操作一样的数据，显然采用 melt（）函数实现起来更简单。下面再来看一个更复杂一点的例子来加深对融合操作的理解，这里采用的是一个75周的歌曲排行榜数据集，数据内容如图6.7所示。


```
>>> df = pd.read_csv("../data/billboard.csv", encoding = "mac_latin2")
>>> df.head(5)
```

与前一数据表类似，查看前5行数据，很容易发现所有这些类似x72nd. week的列名实际应该对应变量week的取值，而原来这些列中对应的值其实是对应变量rank的取值。因此，需要做如下变换，结果如图6.8所示。

	year	artist.inverted	track	time	genre	...	x72nd.week	x73rd.week	x74th.week	x75th.week	x76th.week
0	2000	Destiny's Child	Independent Women Part I	3:38	Rock	...	NaN	NaN	NaN	NaN	NaN
1	2000	Santana	Maria, Maria	4:18	Rock	...	NaN	NaN	NaN	NaN	NaN
2	2000	Savage Garden	I Knew I Loved You	4:07	Rock	...	NaN	NaN	NaN	NaN	NaN
3	2000	Madonna	Music	3:45	Rock	...	NaN	NaN	NaN	NaN	NaN
4	2000	Aguilera, Christina	Come On Over Baby (All I Want Is You)	3:38	Rock	...	NaN	NaN	NaN	NaN	NaN

5 rows x 83 columns

图6.7 歌曲排行榜数据集

```
>>> id_vars = ["year", "artist.inverted", "track",
               "time", "genre", "date.entered", "date.peakd"]
>>> df = pd.melt(frame = df, id_vars = id_vars,
                 var_name = "week", value_name = "rank")
>>> df.head()
```

	year	artist.inverted	track	time	genre	date.entered	date.peakd	week	rank
0	2000	Destiny's Child	Independent Women Part I	3:38	Rock	2000-09-23	2000-11-18	x1st.week	78.0
1	2000	Santana	Maria, Maria	4:18	Rock	2000-02-12	2000-04-08	x1st.week	15.0
2	2000	Savage Garden	I Knew I Loved You	4:07	Rock	1999-10-23	2000-01-29	x1st.week	71.0
3	2000	Madonna	Music	3:45	Rock	2000-08-12	2000-09-16	x1st.week	41.0
4	2000	Aguilera, Christina	Come On Over Baby (All I Want Is You)	3:38	Rock	2000-08-05	2000-10-14	x1st.week	57.0

图6.8 变换结果

虽然从数据整理的角度已经完成了列标题是值而非变量名的问题处理，但是从数据分析角度来看，数据整理的工作远远未完成。week变量值应该对应数值1, 2, 3, ...而不是字符串，需要将其转化为数值，代码如下。

```
>>> df["week"] = df['week'].str.extract('(\d+)',
                                         expand = False).astype(int)
```

结果如图6.9所示。上述代码通过字符串函数extract（），采用'（\d+）'正则表达式提取了所有的数值，之后转换为整数类型，至此得到的数据中week已经转换为了数值。

	year	artist.inverted	track	time	genre	date.entered	date.peaked	week	rank
0	2000	Destiny's Child	Independent Women Part I	3:38	Rock	2000-09-23	2000-11-18	1	78.0
1	2000	Santana	Maria, Maria	4:18	Rock	2000-02-12	2000-04-08	1	15.0
2	2000	Savage Garden	I Knew I Loved You	4:07	Rock	1999-10-23	2000-01-29	1	71.0
3	2000	Madonna	Music	3:45	Rock	2000-08-12	2000-09-16	1	41.0
4	2000	Aguilera, Christina	Come On Over Baby (All I Want Is You)	3:38	Rock	2000-08-05	2000-10-14	1	57.0

图6.9 将week对应变量转化为数值

6.2.2 多个变量存储在一列中

如图6.10所示，下列数据集来源于国际卫生组织，该数据按照国家、年和人口统计学分组记录了确诊的肺结核（Tuberculosis）病例数。人口统计学分组根据性别（m，f）、年龄（0-14，15-25，25-34，35-44，45-54，55-64，unknown）划分。具体数据内容如下。

	country	year	m014	m1524	m2534	...	m4554	m5564	m65	mu	f014
0	AD	2000	0.0	0.0	1.0	...	0	0	0.0	NaN	NaN
1	AE	2000	2.0	4.0	4.0	...	5	12	10.0	NaN	3.0
2	AF	2000	52.0	228.0	183.0	...	129	94	80.0	NaN	93.0
3	AG	2000	0.0	0.0	0.0	...	0	0	1.0	NaN	1.0
4	AL	2000	2.0	19.0	21.0	...	24	19	16.0	NaN	3.0

5 rows × 11 columns

图6.10 肺结核病例数据集

```
>>> df = pd.read_csv("../data/tb-raw.csv")
>>> df.head()
```

	country	year	cases	sex	age
0	AD	2000	0.0	m	0-14
10	AD	2000	0.0	m	15-24
20	AD	2000	1.0	m	25-34
30	AD	2000	0.0	m	35-44
40	AD	2000	0.0	m	45-54

图6.11 性别、年龄提取结果

此数据将性别和年龄这两个变量都放入了列名中，那么数据整理工作的第一步就是利用melt（）将列名转换为变量，之后利用正则表达式通过函数extract（“（\D）（\d+）（\d{2}）”，expand=False）分别提取性别和年龄下限和上限信息。有的情况下，可能需要保留原来的年龄区间记录方式，那么还需要将提前出来的数据再还原为之前的记录方式，代码如下。

```
>>> df = pd.melt(df, id_vars = ["country", "year"],
                 value_name = "cases", var_name = "sex_and_age")
>>> # 正则表达式提取性别, 年龄下限, 上限值
>>> tmp_df = df["sex_and_age"].str.extract("(\D)(\d+)(\d{2})", expand = False)
>>> tmp_df.columns = ["sex", "age_lower", "age_upper"]
>>> tmp_df["age"] = tmp_df["age_lower"] + "-" + tmp_df["age_upper"]
>>> # Merge
>>> df = pd.concat([df, tmp_df], axis = 1)
>>> 清理不需要的列和行
>>> df = df.drop(['sex_and_age', 'age_lower', 'age_upper'], axis = 1)
>>> df = df.dropna()
>>> df = df.sort_values(ascending = True,
                       by = ["country", "year", "sex", "age"])
>>> df.head(5)
```

提取结果如图6.11所示。

6.2.3 变量既在列中存储，又在行中存储

当变量既在列中存储，又在行中存储时，就会出现最复杂形式的混乱数据。读入要分析的数据，如图6.12所示。

```
>>> df = pd.read_csv("../data/weather - raw.csv")
>>> df.head()
```

	id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
0	MX17004	2010	1	tmax	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	MX17004	2010	1	tmin	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	MX17004	2010	2	tmax	NaN	27.3	24.1	NaN	NaN	NaN	NaN	NaN
3	MX17004	2010	2	tmin	NaN	14.4	14.4	NaN	NaN	NaN	NaN	NaN
4	MX17004	2010	3	tmax	NaN	NaN	NaN	NaN	32.1	NaN	NaN	NaN

图6.12 天气数据集

上面的数据来源于全球历史气象网，提供了墨西哥MX17004气象站2010年5个月的每日天气数据。其中的日期被作为了列名（d1～d8），该问题可以用melt（）函数处理，而tmin和tmax代表每日最低和最高温度，显然应该是两个变量，在本例中它们代表了观测对象“天”的属性。因此，需要用pivot_table（）函数将其拆分为两列。利用融合操作处理列名实际是变量的值的问题，代码如下。

```
>>> df = pd.melt(df, id_vars=["id", "year", "month", "element"],
                 var_name = "day_raw")
>>> df.head(10)
```

融合结果如图6.13所示。经过melt（）处理，日期数据已经转换为了列day_raw的取值，进一步需要处理day_raw以取出具体的第几天。此外，日期信息需要合并year、month和day_raw 3列才能得到，因此需要用如下代码整理数据，结果如图6.14所示。

```

>>> df["day"] = df["day_raw"].str.extract("d(\\d+)",
      expand = False)
>>> df["id"] = "MX17004"
>>> df[["year", "month", "day"]] = df[["year", "month", "day"]].apply(
      lambda x: pd.to_numeric(x, errors = 'ignore'))
>>> def create_date_from_year_month_day(row):
      return datetime.datetime(year = row["year"],
      month = int(row["month"]), day = row["day"])
>>> df["date"] = df.apply(lambda row:
      create_date_from_year_month_day(row), axis = 1)
>>> df = df.drop(['year', "month", "day", "day_raw"], axis = 1)
>>> df.head(10)

```

	id	year	month	element	day_raw	value
0	MX17004	2010	1	tmax	d1	NaN
1	MX17004	2010	1	tmin	d1	NaN
2	MX17004	2010	2	tmax	d1	NaN
...	...	...	...	...	...	...
7	MX17004	2010	4	tmin	d1	NaN
8	MX17004	2010	5	tmax	d1	NaN
9	MX17004	2010	5	tmin	d1	NaN

10 rows × 6 columns

图6.13 天气数据融合结果

	id	element	value	date
0	MX17004	tmax	NaN	2010-01-01
1	MX17004	tmin	NaN	2010-01-01
2	MX17004	tmax	NaN	2010-02-01
...	...	...	...	...
7	MX17004	tmin	NaN	2010-04-01
8	MX17004	tmax	NaN	2010-05-01
9	MX17004	tmin	NaN	2010-05-01

10 rows × 4 columns

图6.14 合并日期信息

上述代码首先使用extract（）函数提取了原始日期中的具体第几天信息，之后利用apply（）函数将year、month、day列转换为了数字（转换函数为lambda函数），最后再次通过apply（）函数将create_date_from_year_month_day（）函数作用于df来建立新的date

列。现在已经解决了数据的第一个问题，下一步要做的就是拆分 tmin、tmax到列，这里使用pivot_table（）函数，在进行这一操作前，因为value列有大量缺失数据，所以需要使用dropna（）函数将缺失数据丢弃，代码如下。

```
>>> df = df.dropna()
>>> df = df.pivot_table(index=["id","date"],
                        columns="element", values="value")
>>> df
```

结果如图6.15所示。上述代码中，pivot_table（）函数将原来的id、date列变换成为行的索引，因此还需要利用reset_index（）函数将多级索引id、date还原到列，即可完成全部的数据处理，结果如图6.16所示。

```
>>> df.reset_index(drop=False, inplace=True)
>>> df
```

		element	tmax	tmin
id	date			
	MX17004	2010-02-02	27.3	14.4
		2010-02-03	24.1	14.4
		2010-03-05	32.1	14.2

图6.15 丢弃缺失数据

	element	id	date	tmax	tmin
0	MX17004	2010-02-02	27.3	14.4	
1	MX17004	2010-02-03	24.1	14.4	
2	MX17004	2010-03-05	32.1	14.2	

图6.16 将多级索引还原到列

6.2.4 多个观测单元存储在同一表中

再次回到6.2.1节中处理过的歌曲排行榜数据，仔细观察一下，该数据实际上包含了两个观测对象，一个是排行榜，一个是歌曲信息。排行榜提供了每年每首歌在每周的榜单中的名次，而歌曲信息提供了每首歌的风格、长度等信息。显然，这一数据集是把多个观测单元存储在了同一表中。回到数据分析的本源，数据分析人员需要关注的是使用此数据的目的是什么。显然目的是想知道某一天某一首歌在榜单上的排名，而这样保存数据不利于数据分析人员快速做出回答。要解决此问题，需要将数据拆分到两张表中，首先用如下代码来获取榜单信息，如图6.17所示。

```
>>> df = df.dropna()
>>> df['date'] = pd.to_datetime(df['date.entered']) + \
    pd.to_timedelta(df['week'], unit='w') - \
    pd.DateOffset(weeks=1)
>>> df = df[["year", "artist.inverted", "track",
    "time", "genre", "week", "rank", "date"]]
>>> df = df.sort_values(ascending=True,
    by=["year", "artist.inverted", "track", "week", "rank"])
>>> billboard = df
>>> df.head()
```

	year	artist.inverted	track	time	genre	week	rank	date
246	2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	1	87.0	2000-02-26
563	2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	2	82.0	2000-03-04
880	2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	3	72.0	2000-03-11
1197	2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	4	77.0	2000-03-18
1514	2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	5	87.0	2000-03-25

图6.17 获取榜单信息

上述代码利用上榜日期date.entered与week变量，就可以获得新的当前日期date，基于此可以构建整理后的数据集，该表提供与每周的歌曲排行榜信息。接下来利用如下代码再建立一个关于歌曲的数据表，如图6.18所示。

```
>>> songs_cols = ["year", "artist.inverted", "track", "time", "genre"]
>>> songs = billboard[songs_cols].drop_duplicates()
>>> songs = songs.reset_index(drop = True)
>>> songs["song_id"] = songs.index
>>> songs.head()
```

	year	artist.inverted	track	time	genre	song_id
0	2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	0
1	2000	2Ge+her	The Hardest Part Of Breaking Up (Is Getting Ba...	3:15	R&B	1
2	2000	3 Doors Down	Kryptonite	3:53	Rock	2
3	2000	3 Doors Down	Loser	4:24	Rock	3
4	2000	504 Boyz	Wobble Wobble	3:35	Rap	4

图6.18 歌曲数据表

最后将歌曲数据表和原来的排行榜数据表进行合并，就可以很快找出某首歌在某个时间的排名了，代码如下。

```
>>> ranks = pd.merge(billboard, songs,
                      on=["year", "artist.inverted", "track", "time", "genre"])
>>> ranks = ranks[["song_id", "date", "track", "rank", ]]
>>> ranks.head(10)
```

合并结果如图6.19所示。

	song_id	date	track	rank
0	0	2000-02-26	Baby Don't Cry (Keep Ya Head Up II)	87.0
1	0	2000-03-04	Baby Don't Cry (Keep Ya Head Up II)	82.0
2	0	2000-03-11	Baby Don't Cry (Keep Ya Head Up II)	72.0
...	...	...	...	...
7	1	2000-09-02	The Hardest Part Of Breaking Up (Is Getting Ba...	91.0
8	1	2000-09-09	The Hardest Part Of Breaking Up (Is Getting Ba...	87.0
9	1	2000-09-16	The Hardest Part Of Breaking Up (Is Getting Ba...	92.0

10 rows x 4 columns

图6.19 合并结果

6.2.5 一个观测单元存储在多个表中

实际的数据分析中经常会遇到一个观测单元的数据被存放在不同的表里面的情况。典型的拆分方式有两种，一种是按照某个变量拆分，如按年拆分为2016年、2017年，或者按地理位置、渠道等进行拆分；另一种则是按度量的不同属性拆分，例如前面的天气数据就可能用多个表保存，一张表是来源于温度传感器的数据，另一张是来源于湿度传感器的数据。对于第二种情况，只要各数据集的记录中有一个唯一标识，如日期、身份证号等，那么后续分析中可以通过pd.merge将各个数据集连接起来，这部分内容将在第8章数据整合中详细介绍。本节重点介绍第一种情况。对这类问题的处理也很简单，可以先编写一个读取数据的函数，遍历目录中的文件，并将文件名作为单独的列加入到DataFrame，最后使用pd.concat 进行合并。例如，现有如图6.20所示的数据，该数据是对美国伊利诺伊州每年的小孩取名的统计，每年的数据作为一个数据集，实际上所有这些数据集应该合并为一个。

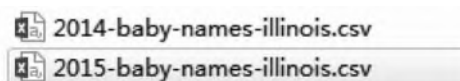


图6.20 姓名数据集

现在的数据整理问题就变成了如何将不同年份的数据读取并合并，首先来解决读取问题，代码如下。

```
>>> def extract_year(string):
    match = re.match(". + (\d{4})", string)
    if match != None: return match.group(1)

>>> path = '../data'
>>> allFiles = glob.glob(path + "/201* - baby - names - illinois.csv")
>>> df_list = []
>>> for file_ in allFiles:
    df = pd.read_csv(file_, index_col = None, header = 0)
    df.columns = map(str.lower, df.columns)
    df["year"] = extract_year(file_)
    df_list.append(df)
```

上述代码利用了Python中的标准库glob。glob提供了文件名模式匹配，不用遍历整个目录判断每个文件是不是符合通配符。利用201*可以完成对2014和2015的匹配。之后利用extract_year（）函数可以将年的信息从文件名中提取出来，将其作为year列的信息。完成上述工作后，每年的数据保存到了列表df_list中，此时只需要用pd.concat将列表中的数据合并就可以了，代码如下。

```
>>> df = pd.concat(df_list)
>>> df.head(5)
```

合并结果如图6.21所示。

6.2.6 思考

如图6.22所示，现在有如下数据，该数据记录了埃博拉病毒在2014到2015年期间每月各国家的病例以及死亡人数，请问此数据有什么问题，需要如何处理？

	rank	name	frequency	sex	year
0	1	Noah	837	Male	2014
1	2	Alexander	747	Male	2014
2	3	William	687	Male	2014
3	4	Michael	680	Male	2014
4	5	Liam	670	Male	2014

图6.21 合并不同年份的数据

```
>>> ebola = pd.read_csv('../data/country_timeseries.csv')
>>> ebola.head()
```

	Date	Day	Cases_Guinea	Cases_Liberia	Cases_SierraLeone	...	Deaths_Nigeria	Deaths_Senegal	Deaths_UnitedStates	Deaths_Spain	Deaths_Mali
0	1/5/2015	289	2776.0	NaN	10030.0	...	NaN	NaN	NaN	NaN	NaN
1	1/4/2015	288	2775.0	NaN	9780.0	...	NaN	NaN	NaN	NaN	NaN
2	1/3/2015	287	2769.0	8166.0	9722.0	...	NaN	NaN	NaN	NaN	NaN
3	1/2/2015	286	NaN	8157.0	NaN	...	NaN	NaN	NaN	NaN	NaN
4	12/31/2014	284	2730.0	8115.0	9633.0	...	NaN	NaN	NaN	NaN	NaN

5 rows × 18 columns

图6.22 埃博拉病毒病例数据集

第7章 分组统计

横看成岭侧成峰，
远近高低各不同。

——苏轼《题西林壁》

话说天下大势，分久必合，合久必分。数据分析也是如此，我们经常要对数据进行分组与聚合，以对不同组的数据进行深入解读。本章将介绍如何利用Pandas中的GroupBy操作函数来完成数据的分组、聚合以及统计。

7.1 分组、应用和聚合

“分而治之”（Divide and Conquer）方法（又称为“分治术”），是有效算法设计中普遍采用的一种技术。所谓“分而治之”，就是把一个复杂的算法问题按一定的“分解”方法分为等价的规模较小的若干部分，然后逐个解决，分别找出各部分的解，把各部分的解组成整个问题的解。这种朴素的思想来源于人们生活与工作的经验，也完全适用于技术领域。以海量数据处理为例，由于数据量太大，导致无法在较短时间内迅速解决，或无法一次性装入内存。那么如何解决该问题呢？无非只有一个办法——大而化小。规模太大，就把规模大的化为规模小的，各个击破。例如，从海量日志数据中提取出某日访问次数最多的那个IP，把整个大文件映射为1000个小文件，再找出每个小文件中出现频率最高的IP及相应的频率，然后从这1000个最大的IP中，找出那个频率最高的IP，即为所求。这也是大数据编程模型MapReduce的基本思想。

Pandas中同样存在着“分而治之”的思想，即Pandas的GroupBy，从英文的字面意义上理解就是“根据（By）一定的规则进行分组（Group）”。它的作用就是通过一定的规则将一个数据集划分成若干小的区域，然后针对若干个小区域进行数据处理。简单地说，GroupBy就是Split-Apply-Combine，如图7.1所示。首先将数据按照不同的key进行分割（Split），然后将求和函数sum（）应用（Apply）于各组，最后再将数据合并（Combine）到一起，得到最终结果。

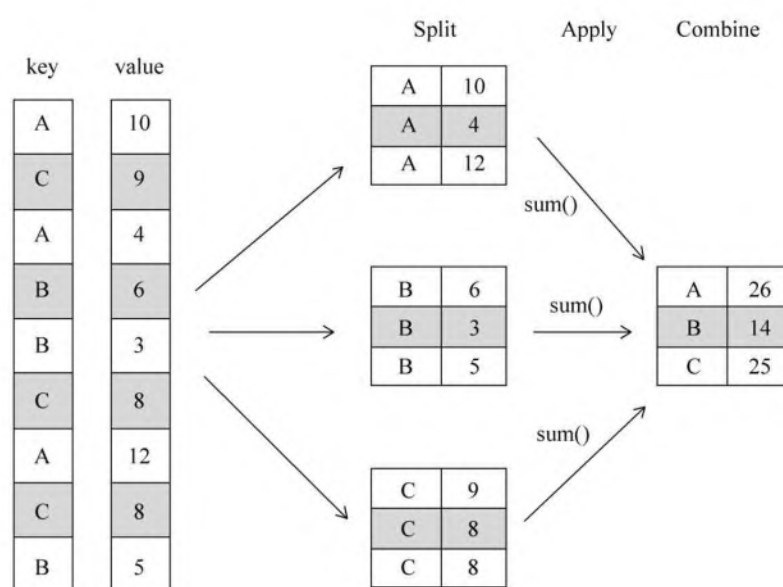


图7.1 Split-Apply-Combine

7.2 Pandas中的GroupBy操作

本节主要以Seaborn中自带的tips数据集为例对GroupBy进行讲解。数据前5行内容如下。

```
>>> df_tips = sns.load_dataset('tips')
>>> df_tips.head()
   total_bill  tip  sex  smoker  day  time  size
0    16.99    1.01 Female    No   Sun  Dinner    2
1    10.34    1.66  Male    No   Sun  Dinner    3
2    21.01    3.50  Male    No   Sun  Dinner    3
3    23.68    3.31  Male    No   Sun  Dinner    2
4    24.59    3.61 Female    No   Sun  Dinner    4
```

7.2.1 单列数据分组统计

以tips数据集为例，如果想按照不同性别来对数据进行统计，应该怎么办呢？首先我们需要创建一个DataFrameGroupBy对象，代码如下。

```
>>> df_tips.groupby(by = 'sex')
< pandas.core.groupby.generic.DataFrameGroupBy object at
0x000000000BA8DA20 >
```

此时我们得到的只是一个DataFrameGroupBy对象，也就是只完成了图7.1中的Split工作，接下来要做的是Apply和Combine。例如，我们想知道tips分组里面男性（Male）和女性（Female）各有多少，代码如下。

```
>>> df_tips.groupby(by = 'sex').size()
sex
Male      157
Female     87
dtype: int64
```

size（）即是DataFrameGroupBy对象提供的一个分组聚合函数，该函数将自动统计Male组和Female组中的数据大小，之后将其汇总到一个新的Series中，可以通过如下代码进行验证。

```
>>> type(df_tips.groupby(by = 'sex').size())
pandas.core.series.Series
>>> groups = df_tips.groupby(by = 'sex')
>>> for group in groups:
    print(group)
('Male', total_bill  tip    sex    smoker  day    time    size
1      10.34      1.66  Male    No     Sun    Dinner  3
2      21.01      3.50  Male    No     Sun    Dinner  3
...      ...      ...    ...    ...    ...    ...    ...
241     22.67      2.00  Male    Yes    Sat    Dinner  2
242     17.82      1.75  Male    No     Sat    Dinner  2

[157 rows x 7 columns])
('Female', total_bill  tip    sex    smoker  day    time    size
0      16.99      1.01  Female    No     Sun    Dinner  2
4      24.59      3.61  Female    No     Sun    Dinner  4
...      ...      ...    ...    ...    ...    ...    ...
240     27.18      2.00  Female    Yes    Sat    Dinner  2
243     18.78      3.00  Female    No     Thur   Dinner  2

[87 rows x 7 columns])
```

上面的第二段代码对分组对象中的组依次进行了遍历。除了对组进行遍历，我们还可以通过`get_group()`函数来获取指定组，例如：

```
>>> groups.get_group('Female').head(3)
   total_bill  tip  sex  smoker  day  time  size
0    16.99     1.01 Female    No   Sun  Dinner  2
4    24.59     3.61 Female    No   Sun  Dinner  4
11   35.26     5.00 Female    No   Sun  Dinner  4
```

在完成分组后，我们就可以针对各组进行聚合运算。例如，我们想看tips数据集中男性、女性买单时总账单、小费以及用餐人数的均值，那么可以采用如下代码。

```
>>> df_tips.groupby(by = 'sex').mean()
   total_bill  tip      size
sex
Male    20.744076  3.089618  2.630573
Female  18.056897  2.833448  2.459770
```

上述代码对分组中每列都进行聚合运算，有的时候我们只需要对某一列进行聚合运算。例如，我们只想统计男性组与女性组的总账单均值，可以采用如下代码。

```
>>> df_tips.groupby(by='sex')['total_bill'].mean()
sex
Male      20.744076
Female    18.056897
Name: total_bill, dtype: float64
```

DataFrameGroupBy对象除了提供了前面已经用过的聚合函数外，还提供了如下的聚合函数。

- sum（）：求和
- mean（）：求平均值
- count（）：统计所有非空值
- size（）：统计所有值
- max（）：求最大值
- min（）：求最小值
- std（）：计算标准差

这里重点讲一下size（）和count（）的区别。有如下数据：

```
>>> data = {'person': ['Dan', 'Dan', 'Jamie', 'Jamie'],
            'ride_duration_minutes': [4, np.NaN, 8, 10]}
>>> df_rides = pd.DataFrame(data)
>>> df_rides
   person  ride_duration_minutes
0   Dan      4.0
1   Dan      NaN
2  Jamie      8.0
3  Jamie     10.0
```

如果分别使用size（）和count（）这两个聚合函数，得到的结果将不同。


```
>>> df_rides.groupby(by = 'person')['ride_duration_minutes'].size()
person
Dan      2
Jamie    2
Name: ride_duration_minutes, dtype: int64
>>> df_rides.groupby(by = 'person')['ride_duration_minutes'].count()
person
Dan      1
Jamie    2
Name: ride_duration_minutes, dtype: int64
```

得到不同结果的原因是由于count（）函数不会统计空值，而size（）函数只是统计组的大小，不管取值是否为空。除了直接对分组对象使用聚合函数来完成分组统计，我们还可以使用agg（）或aggregate（）函数来进行分组统计，例如下面的代码与使用mean（）函数效果完全一样。

```
>>> df_tips.groupby('sex')['tip'].agg('mean')
sex
Male      3.089618
Female    2.833448
Name: tip, dtype: float64
>>> df_tips.groupby('sex')['tip'].aggregate('mean')
sex
Male      3.089618
Female    2.833448
Name: tip, dtype: float64
```

既然两者效果一样，为什么Pandas中要提供agg（）函数呢？这是因为agg（）函数提供了更好的灵活性，我们如果想同时统计各分组的小费均值、最小值、最大值，只需要执行一次agg（）函数就可以完成，代码如下。

```
>>> df_tips.groupby('sex')['tip'].agg(['mean', 'min', 'max'])
      mean  min  max
sex
Male  3.089618  1.0  10.0
Female 2.833448  1.0   6.5
```

其中，agg（）函数中的参数['mean'，'min'，'max']即是聚合函数列表。此外，我们还可以对聚合后的列进行重命名，例如：

```
>>> df_tips.groupby('sex')['tip'].agg([('tip_mean', 'mean'),\
    ('tip_min', 'min'), ('tip_max', 'max')])
      tip_mean  tip_min  tip_max
sex
Male    3.089618    1.0    10.0
Female  2.833448    1.0     6.5
```

与前一段代码不同的是，这里以元组的方式来指定聚合函数。例如，（'tip_mean'，'mean'）代表了我们要执行的聚合函数为mean，聚合运算后得到的列名为tip_mean。如果完成聚合后，想将Index去掉，那么可以直接使用reset_index（）函数，代码如下。

```
>>> grouped = df_tips.groupby('sex')['tip'].agg(\
    [('tip_mean', 'mean'), ('tip_min', 'min'), ('tip_max', 'max')])
>>> grouped.reset_index()
      sex  tip_mean  tip_min  tip_max
0  Male    3.089618    1.0    10.0
1  Female  2.833448    1.0     6.5
```

7.2.2 多列数据分组统计

上一小节是将sex列作为分组基准，如果想同时基于sex列和day列进行分组统计男女每天的消费，可采用如下代码。

```
>>> df_tips.groupby(by = ['sex', 'day']).size()
sex    day
Male   Thur    30
        Fri    10
        Sat    59
        Sun    58
Female Thur    32
        Fri     9
        Sat    28
        Sun    18
dtype: int64
>>> df_tips.groupby(by = ['sex', 'day'])['total_bill'].sum()
sex    day
Male   Thur    561.44
        Fri    198.57
        Sat   1227.35
        Sun   1269.46
Female Thur    534.89
        Fri    127.31
        Sat    551.05
        Sun    357.70
Name: total_bill, dtype: float64
```

上述两段代码分别统计了tips数据集中男性与女性每天总就餐次数以及账单总额。与7.2.1节类似，我们也可以利用如下代码对聚合后的列进行重命名，如图7.2所示。

```
>>> df_tips.groupby(by = ['sex', 'day'])['tip'].agg(\
    [('tip_mean', 'mean'), ('tip_min', 'min'), ('tip_max', 'max')])
```

Pandas的分组统计还提供了更加灵活的方式，对于分组后的对象，我们还可以针对不同的列进行不同聚合运算。例如针对tip列和total_bill列，我们想统计不同的内容，那么可以采用如下代码。

```
>>> df_tips.groupby('sex').agg({'tip': [('avg_tip', 'mean'), \
    ('max_tip', 'max')], 'total_bill': [('avg_bill', 'mean')]})
```

统计结果如图7.3所示。

		tip_mean	tip_min	tip_max
sex	day			
Male	Thur	2.980333	1.44	6.70
	Fri	2.693000	1.50	4.73
	Sat	3.083898	1.00	10.00
	Sun	3.220345	1.32	6.50
Female	Thur	2.575625	1.25	5.17
	Fri	2.781111	1.00	4.30
	Sat	2.801786	1.00	6.50
	Sun	3.367222	1.01	5.20

图7.2 重命名结果

sex	tip		total_bill
	avg_tip	max_tip	avg_bill
Male	3.089618	10.0	20.744076
Female	2.833448	6.5	18.056897

图7.3 统计结果

输出数据出现了多级Index，可以用如下代码验证。

```
>>> grouped = df_tips.groupby('sex').agg(\
    {'tip': [('avg_tip', 'mean'), ('max_tip', 'max')], \
     'total_bill': [('avg_bill', 'mean')]})
>>> grouped.columns
MultiIndex(levels = [['tip', 'total_bill'],
 ['avg_bill', 'avg_tip', 'max_tip']], codes = [[0, 0, 1], [1, 2, 0]])
```

其中，第一级Index为tip和total_bill，第二级则是avg_tip、max_tip、avg_bill。如果我们想对其进行修改，可以直接利用修改列名的方式来完成，代码如下。

```
>>> grouped.columns = ['avg_tip', 'max_tip', 'avg_bill']
>>> grouped
      avg_tip  max_tip  avg_bill
sex
Male    3.089618    10.0    20.744076
Female  2.833448     6.5    18.056897
```

7.2.3 使用自定义函数进行分组统计

如果Pandas中提供的聚合函数不能满足我们的要求，我们还可以自己编写自定义函数来完成聚合功能。例如，我们想统计男性组与女性组中账单最大值和最小值的差异，可以利用如下代码完成。

```
>>> df_tips.groupby(by = 'sex').agg({\
    'total_bill': lambda bill: bill.max() - bill.min()})
    total_bill
sex
Male      43.56
Female    41.23
```

上述代码定义了一个lambda函数来完成各组中账单最大值与最小值差的计算。除了对某列进行聚合运算，还可以对不同列定义不同的自定义函数，示例如下。

```
>>> df_tips.groupby(by = 'sex').agg({\
    'total_bill': lambda bill: bill.max() - bill.min(),\
    'tip': lambda tip: tip.max()})
    total_bill  tip
sex
Male      43.56   10.0
Female    41.23    6.5
```

lambda函数通常用于相对简单的函数定义，如果是复杂一点的，我们可以自己定义新函数后使用。如下代码定义了一个名为max_deviation()的函数。

```
>>> def max_deviation(s):
    std_score = (s - s.mean()) / s.std()
    return std_score.abs().max()
>>> df_tips.groupby('sex')['tip'].agg(max_deviation)
```

上述代码中`max_deviation()`函数的参数`s`实际对应于分组对象的`tip`列，因此`s.mean()`是对该列求平均。在有的情况下，自定义函数还可以带参数，如果我们想知道男性和女性组总账单中金额为30~60的比例，可以采用如下代码。

```
>>> def bill_between(s, low, high):
        return s.between(low, high).mean()
>>> df_tips.groupby('sex')['total_bill'].agg(bill_between, 30, 60)
sex
Male      0.159236
Female    0.080460
Name: total_bill, dtype: float64
```

上述代码中`bill_between()`函数中的参数，直接通过`agg(bill_between, 30, 60)`函数传入。

7.2.4 数据过滤与变换

有的时候我们对数据进行分组不是为了分组统计，而是为了对数据进行过滤或变换，此时可以使用`filter()`和`transform()`函数来完成。例如，我们想知道`tips`数据集中每天消费总额大于20的账单，代码如下。

```
>>> df_tips.groupby('day').filter( \
        lambda x : x['total_bill'].mean() > 20 )
```

数据过滤结果如图7.4所示。

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
...	...	...	...	...	...	...	...
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2

163 rows x 7 columns

图7.4 数据过滤

上述代码首先对数据按day进行分组，`x['total_bill'].mean()`
 > 20将过滤消费总额大于20的数据。如果我们需要对分组数据进行变换，则使用`transform()`函数。例如，如下代码对按day分组的数据求均值后，将其作为新列添加回原来的`df_tips`中，结果如图7.5所示。

```
>>> df_tips['day_average'] = df_tips.groupby('day')['total_bill'].
    transform( lambda x : x.mean())
>>> df_tips.head()
```

	total_bill	tip	sex	smoker	day	time	size	day_average
0	16.99	1.01	Female	No	Sun	Dinner	2	21.41
1	10.34	1.66	Male	No	Sun	Dinner	3	21.41
2	21.01	3.50	Male	No	Sun	Dinner	3	21.41
3	23.68	3.31	Male	No	Sun	Dinner	2	21.41
4	24.59	3.61	Female	No	Sun	Dinner	4	21.41

图7.5 数据变换

除了`filter()`和`transform()`操作，我们也可以对组对象执行`apply`操作。例如，我们可以按性别分组后计算小费占总账单的比例，代码如下。

```
>>> df_tips.groupby('sex').apply(lambda x :x['tip']/x['total_bill'])
sex
Male    1      0.160542
        2      0.166587
        3      0.139780
        5      0.186240
        6      0.228050
        ...
Female  226     0.198216
        229     0.130199
        238     0.130338
        240     0.073584
        243     0.159744
Length: 244, dtype: float64
```


第8章 数据整合

泰山不让土壤，故能成其大。

——李斯《上书秦始皇》

数据无处不在，有的以CSV格式存在，有的是文本文件，还有的存在于数据库、网络中。因此，在数据分析过程中经常需要考虑如何将不同数据读入、整合。本章将详细讲解如何读入不同格式的数据，以及如何合并数据。

8.1 数据读入

8.1.1 基本数据读入方法

为了更好地整合不同格式的数据，Pandas提供了丰富的数据读写函数，常用的数据读取函数有：

- `read_csv()`
- `read_excel()`
- `read_hdf()`
- `read_sql()`
- `read_json()`
- `read_html()`
- `read_stata()`
- `read_sas()`
- `read_clipboard()`
- `read_pickle()`

与读数据对应，Pandas也提供了对应格式的写文件函数：

- `to_csv()`
- `to_excel()`
- `to_hdf()`
- `to_sql()`
- `to_json()`
- `to_html()`
- `to_stata()`
- `to_clipboard()`
- `to_pickle()`

除了对上述格式数据的支持，Pandas还支持读取数据库、网页、网络文件。本节将从`read_csv()`函数开始讲解，掌握了这种格式后，其他格式的文件处理基本都是一致的。经过了前面章节的学习，相信读者对下面的代码已经很熟悉了，输出结果如图8.1所示。

```
>>> import numpy as np
>>> import pandas as pd
# set max display row, columns
>>> pd.set_option('display.max_rows', 5)
>>> pd.set_option('display.max_columns', 10)
>>> eu12 = pd.read_csv('../data/Eueo2012.csv')
>>> eu12.head()
```

	Team	Goals	Shots on target	Shots off target	Shooting Accuracy	...	Yellow Cards	Red Cards	Subs on	Subs off	Players Used
0	Croatia	4	13	12	51.9%	...	9	0	9	9	16
1	Czech Republic	4	13	18	41.9%	...	7	0	11	11	19
2	Denmark	4	10	10	50.0%	...	4	0	7	7	15
3	England	5	11	18	50.0%	...	5	0	11	11	16
4	France	3	22	24	37.9%	...	6	0	11	11	19

5 rows x 35 columns

图8.1 读入Eueo2012.csv

这段代码将Eueo2012.csv中的数据读入到eu12这个DataFrame中，读者也可以在读入的时候指定用Team来作为index的列，代码如下。

```
>>> eu12 = pd.read_csv('../data/Eueo2012.csv', index_col = 'Team')
>>> eu12.head()
```

运行结果如图8.2所示。

	Goals	Shots on target	Shots off target	Shooting Accuracy	% Goals-to-shots	...	Yellow Cards	Red Cards	Subs on	Subs off	Players Used
Team											
Croatia	4	13	12	51.9%	16.0%	...	9	0	9	9	16
Czech Republic	4	13	18	41.9%	12.9%	...	7	0	11	11	19
Denmark	4	10	10	50.0%	20.0%	...	4	0	7	7	15
England	5	11	18	50.0%	17.2%	...	5	0	11	11	16
France	3	22	24	37.9%	6.5%	...	6	0	11	11	19

5 rows x 34 columns

图8.2 用Team作为index的列

有时待读入的数据不包含列名，如果读的时候不进行处理就会出现下面的错误，如图8.3所示。

```
>>> eu12_header = pd.read_csv('../data/Eueo2012_no_header.csv')
>>> eu12_header.head()
```

此时本应该是第一行的数据，却变成了每列的标题。对于这种情况，需要在读入的时候指明header=None参数，如图8.4所示。

	Croatia	4	13	12	51.90%	16.00%	32	0	0.1	0.2	...	13.1	81.30%	41	62	2.1	9	0.4	9.1	9.2	16
0	Czech Republic	4	13	18	41.90%	12.90%	39	0	0	0	...	9	60.10%	53	73	8	7	0	11	11	19
1	Denmark	4	10	10	50.00%	20.00%	27	1	0	0	...	10	66.70%	25	38	8	4	0	7	7	15
2	England	5	11	18	50.00%	17.20%	40	0	0	0	...	22	88.10%	43	45	6	5	0	11	11	16
3	France	3	22	24	37.90%	6.50%	65	1	0	0	...	6	54.60%	36	51	5	6	0	11	11	19
4	Germany	10	32	32	47.80%	15.60%	80	2	1	0	...	10	62.60%	63	49	12	4	0	15	15	17

5 rows x 35 columns

图8.3 待读入数据不包含列名

```
>>> eu12_header = pd.read_csv('../data/Eueo2012_no_header.csv',
                                header = None)
>>> eu12_header.head()
```

		0	1	2	3	4	5	6	7	8	9	...	25	26	27	28	29	30	31	32	33	34
0	Croatia	4	13	12	51.90%	16.00%	32	0	0	0	...	13	81.30%	41	62	2	9	0	9	9	16	
1	Czech Republic	4	13	18	41.90%	12.90%	39	0	0	0	...	9	60.10%	53	73	8	7	0	11	11	19	
2	Denmark	4	10	10	50.00%	20.00%	27	1	0	0	...	10	66.70%	25	38	8	4	0	7	7	15	
3	England	5	11	18	50.00%	17.20%	40	0	0	0	...	22	88.10%	43	45	6	5	0	11	11	16	
4	France	3	22	24	37.90%	6.50%	65	1	0	0	...	6	54.60%	36	51	5	6	0	11	11	19	

5 rows x 35 columns

图8.4 指明header=None参数

如果已经提前知道列名，那么还可以在读入的时候指定列名，代码如下。

```
>>> columns = eu12.columns
>>> eu12_header = pd.read_csv('../data/Eueo2012_no_header.csv',
names = columns)
>>> eu12_header.head()
```

运行结果如图8.5所示。

	Goals	Shots on target	Shots off target	Shooting Accuracy	% Goals-to-shots	...	Yellow Cards	Red Cards	Subs on	Subs off	Players Used
Team											
Croatia	4	13	12	51.9%	16.0%	...	9	0	9	9	16
Czech Republic	4	13	18	41.9%	12.9%	...	7	0	11	11	19
Denmark	4	10	10	50.0%	20.0%	...	4	0	7	7	15
England	5	11	18	50.0%	17.2%	...	5	0	11	11	16
France	3	22	24	37.9%	6.5%	...	6	0	11	11	19

5 rows x 34 columns

图8.5 指定列名

此外，read_csv（）函数还可以支持预先指定列的数据类型。例如，下面的代码将Goals列指定为浮点数。

```
>>> eu12_header = pd.read_csv('../data/Eueo2012_no_header.csv',
    names = columns, dtype = {'Goals':np.float64})
>>> eu12_header.dtypes
Goals                float64
Shots on target      int64
...
Subs off             int64
Players Used         int64
Length: 34, dtype: object
```

某些情况下只需要读取部分列的数据，此时可以通过usecols参数来进行设置，代码如下。

```
>>> eu12 = pd.read_csv('../data/Eueo2012.csv',
    index_col = 0, usecols = ['Team', 'Goals', 'Shots on target'])
>>> eu12.head()
```

结果如图8.6所示。上述代码选择了只读取Team、Goals和Shots on target列的数据，同时对index_col参数采用数字进行设置，即将第0列作为index。既然有读入，那么必然有对应的写文件函数to_csv（），该函数的参数用法基本上和read_csv（）一致，读者可以自行查看帮助了解更多内容，如下代码即是将eu12中的数据写入到Eueo2012_save.csv文件中。

```
>>> eu12.to_csv('../data/Eueo2012_save.csv')
```

	Goals	Shots on target
Team		
Croatia	4	13
Czech Republic	4	13
Denmark	4	10
England	5	11
France	3	22

图8.6 读取部分列

8.1.2 文件读取进阶

默认的情况下，`read_csv()` 函数认为文件中数据间用“，”分隔，遇到不是这种格式的文件，就需要指定分隔符。例如，如下代码对采用“?”分隔的数据进行了读取。

```
>>> eu12 = pd.read_csv('../data/Eueo2012_del.txt', sep = '?', index_col = 0)
>>> eu12.head()
```

还有的情况是，数据分析人员想在读取的时候跳过数据文件的前几行，那么可以采用`skiprows`参数进行设置，代码如下。

```
>>> eu12 = pd.read_csv('../data/Eueo2012_skip.txt',
                        sep = '?', skiprows = [0,1], index_col = 0)
>>> eu12
```

结果如图8.7所示。某些情况下，数据文件可能很大，进行分析时经常会先读入其中一部分数据，在对数据有一定了解后再读入全部数据，此时就可以通过指定`nrows`参数来设置，代码如下。

```
>>>> eu12 = pd.read_csv('../data/Eueo2012_del.txt',
                        header = 0, skiprows = 5, nrows = 5,
                        sep = '?', index_col = 0, names = ['Goals', 'Shots on target'])
>>> eu12
```

运行结果如图8.8所示。

	Goals	Shots on target
Team		
Croatia	4.0	13.0
Czech Republic	4.0	13.0
...	...	...
Ukraine	2.0	7.0
This is the end.	NaN	NaN

17 rows × 2 columns

图8.7 设置skiprows参数

	Goals	Shote on target
Germany	10	32
Greece	5	8
Italy	6	34
Netherlands	2	12
Poland	2	15

图8.8 设置nrows参数

8.1.3 读取其他格式文件

在数据分析过程中，数据文件除了常用的CSV格式，还会使用到EXCEL、JSON或数据库格式文件，本节将介绍如何处理这些格式的文件。下面的代码完成了EXCEL格式文件的读入，结果如图8.9所示。

```
>>> eu12 = pd.read_excel('../data/Eueo2012_excel.xlsx',
    sheet_name = 'Eueo2011')
>>> eu12.head()
```

	Team	Goals	Shots on target	Shots off target	Shooting Accuracy	...	Yellow Cards	Red Cards	Subs on	Subs off	Players Used
0	Croatia	8	13	12	0.519	...	9	0	9	9	16
1	Czech Republic	4	13	18	0.419	...	7	0	11	11	19
2	Denmark	4	10	10	0.500	...	4	0	7	7	15
3	England	5	11	18	0.500	...	5	0	11	11	16
4	France	3	22	24	0.379	...	6	0	11	11	19

5 rows x 35 columns

图8.9 EXCEL格式文件的读入

上述代码中read_excel（）函数支持对EXCEL格式文件的读入，同时通过参数sheet_name可以指定读入哪一个表。如果要将数据写入到EXCEL文件，可以使用to_excel（）函数，代码如下。

```
# conda install openpyxl
>>> eu12.to_excel('../data/test.xlsx')
>>> eu12.to_excel('../data/test1.xlsx', sheet_name = 'eu2012')
```

下面的代码则可以将两个不同的DataFrame写入到同一个EXCEL文件的不同表中，这里需要使用xlwt库，如果读者没有安装该库，则需要使用conda install或pip install命令进行安装。

```
>>> from pandas import ExcelWriter
# conda install xlwt
>>> with ExcelWriter("../data/test3.xls") as writer:
    eu12.to_excel(writer, sheet_name = 'eu12')
    eu12.to_excel(writer, sheet_name = 'eu11')
```

除了EXCEL格式的数据，另一种常见的文件格式就是JSON，对于这种格式，可以利用read_json（）函数来读取，代码如下。

```
>>> eu12_json = pd.read_json("../data/eueo2012.json")
>>> eu12_json.head(5)
```

需要注意的是，JSON是无顺序格式的文件，因此数据读入后可能不是想要的顺序，此时可以通过前面章节介绍的列选择、行排序等方法进行处理，这里不再赘述。接下来再来看看如何处理数据库和HDF5格式的数据，对于数据库，为了简单起见，本书以sqlite3数据库为例，其他数据库的使用也与之类似。

```
>>> import sqlite3
>>> connection = sqlite3.connect("../data/eueo2012.sqlite")
>>> eu12.to_sql("EU12_DATA", connection, if_exists = "replace")
# commit the SQL and close the connection
>>> connection.commit()
>>> connection.close()
```

上述代码首先导入sqlite3库，接下来利用sqlite3.connect（）函数构造了一个面向sqlite数据库的连接，之后将eu12这个DataFrame通过to_sql（）函数写入到数据库中的表EU12_DATA中。其中，

if_exists="replace"代表如果数据表已经存在就替换，当然也可以使用添加模式，即if_exists="append"。最后要使写入生效，需要执行commit（）函数，并关闭数据库连接。如果想将数据读出来，可以使用read_sql（）函数，代码如下。

```
>>> connection = sqlite3.connect("../data/eueo2012.sqlite")
>>> eu_sql = pd.read_sql('select * from EU12_DATA',connection)
>>> eu_sql.head()
```

结果如图8.10所示。read_sql（）函数支持各种SQL语句，理论上也可以用它来完成所有的数据选择工作。而对于HDF5格式文件的处理，它与前面的CSV、EXCEL格式稍有不同，先通过HDFStore（）函数指定要存储HDF5文件到哪里，再以字典形式将DataFrame赋值，代码如下。

	index	Team	Goals	Shots on target	Shots off target	...	Yellow Cards	Red Cards	Subs on	Subs off	Players Used
0	0	Croatia	8	13	12	...	9	0	9	9	16
1	1	Czech Republic	4	13	18	...	7	0	11	11	19
2	2	Denmark	4	10	10	...	4	0	7	7	15
3	3	England	5	11	18	...	5	0	11	11	16
4	4	France	3	22	24	...	6	0	11	11	19

5 rows x 36 columns

图8.10 数据库文件的读入

```
>>> df = pd.DataFrame(np.random.randn(10, 3),
                        index = pd.date_range('11/11/2016', periods = 10),
                        columns = ['A', 'B', 'C'])
# conda install pytables
>>> hdf = pd.HDFStore('../data/hdf.h5')
>>> hdf['df'] = df
>>> hdf
<class 'pandas.io.pytables.HDFStore'>
File path: ../data/hdf.h5
```

而对数据进行修改并写回到HDF5文件中也需要进行类似操作，代码如下。

```
>>> read_hdf = pd.HDFStore("../data/hdf.h5")
>>> df = read_hdf['df']
>>> df.iloc[0].A = 1
>>> hdf['df'] = df
>>> pd.HDFStore("../data/hdf.h5")['df'].head(2)
```

	A	B	C
2016-11-11	1.000000	-1.014995	0.920577
2016-11-12	-0.532871	0.120090	0.044471

最后来看看读取网页，Pandas支持对网页中的表格数据的直接读取，下面的代码完成了对ESPN网站上的数据读取，结果如图8.11所示。

```
>>> table = pd.read_html(f'http://www.espn.com/nfl/qbr/_/\
    type/alltime-season/page/{1}/order/true/qualified/true')
>>> type(table), len(table)
>>> table[0].head()
```

	0	1	2	3	4	5	6	7	8	9	10
0	RK	PLAYER	YEAR	PTS ADDED	PASS	RUN	PENALTY	TOTAL EPA	QB PLAYS	RAW QBR	TOTAL QBR
1	1	Lamar Jackson, BAL	2019	9.7	5.1	-0.0	0.2	5.3	27	99.4	99.4
2	2	Dak Prescott, DAL	2019	9.5	10.9	0.7	0.3	11.9	37	97.4	96.9
3	3	Derek Carr, OAK	2019	5.9	8.9	-0.3	-0.1	8.4	30	94.1	93.6
4	4	Carson Wentz, PHI	2019	8.1	9.9	0.7	0.2	10.8	44	93.2	92.6

图8.11 网页数据的读取

8.2 数据合并

8.2.1 认识merge操作

在一个数据分析项目中，数据来源于不同格式的文件是很常见的事情，此时数据分析前首先要解决的问题就变成了如何将不同的数据合并。Pandas提供了merge()函数来完成此功能。首先构造两个不同的DataFrame，其中customers数据集包含了客户ID、姓名和地址，而orders数据集包含了订单客户的ID和订单日期。具体代码如下。

```

>>> customers = {'CustomerID': [10, 11],
                  'Name': ['MikeWang', 'JackMa'],
                  'Address': ['Address for MikeWang', 'Address for JackMa']}
>>> customers = pd.DataFrame(customers)
>>> customers
  CustomerID  Name      Address
0      10    MikeWang  Address for MikeWang
1      11    JackMa    Address for JackMa
>>> import datetime
>>> orders = {'CustomerID': [10, 11, 10],
              'OrderDate': [datetime.date(2016, 12, 1),
                           datetime.date(2016, 12, 1),
                           datetime.date(2016, 12, 2)]}
>>> orders = pd.DataFrame(orders)
>>> orders
  CustomerID  OrderDate
0      10    2016-12-01
1      11    2016-12-01
2      10    2016-12-02

```

现在想要构建一个包含了以上全部信息的数据集，需要将两个数据集进行合并，由于CustomerID在两个数据集中都存在，因此merge（）函数会自动以该列为key来进行合并，结果如下。

```

>>> customers.merge(orders)
  CustomerID  Name      Address      OrderDate
0      10    MikeWang  Address for MikeWang  2016-12-01
1      10    MikeWang  Address for MikeWang  2016-12-02
2      11    JackMa    Address for JackMa    2016-12-01

```

上述情况是最简单的合并数据方式，下面来看一个更复杂的情形，假设有如下两个数据集：

```

>>> left_data = {'key1': ['a', 'b', 'c'],
                  'key2': ['x', 'y', 'z'],
                  'lval1': [0, 1, 2]}
>>> right_data = {'key1': ['a', 'b', 'c'],
                   'key2': ['x', 'a', 'z'],
                   'rval1': [6, 7, 8]}
>>> left = pd.DataFrame(left_data, index=[0, 1, 2])
>>> right = pd.DataFrame(right_data, index=[1, 2, 3])
>>> left
   key1  key2  lval1
0    a    x      0
1    b    y      1
2    c    z      2
>>> right
   key1  key2  rval1
1    a    x      6
2    b    a      7
3    c    z      8

```

运行merge（）函数，将得到如下结果。

```

>>> left.merge(right)
   key1  key2  lval1  rval1
0    a    x      0      6
1    c    z      2      8

```

这是由于left和right中都有key1和key2列，那么合并的时候将以这两列为key来进行合并，同时默认情况下merge（）函数合并时的参数how='inner'，因此合并的时候left、right中key1、key2列取值都一样的数据才会进行合并，最后得到的新数据就只有两行，列变成了4列。如果合并时指定key，那么merge（）函数就会以key来进行合并。例如，下面的代码就是以key1列进行合并。

```

>>> left.merge(right,on='key1')
   key1  key2_x  lval1  key2_y  rval1
0    a      x      0      x      6
1    b      y      1      a      7
2    c      z      2      z      8

```

这种情况下，因为left和right中都有key2列，因此merge（）函数会重命名这两列为key2_x和key2_y。除了按照key进行合并，merge（）函数还支持按照索引进行合并，同样是刚才的数据，将得到如下结果。

```
>>> pd.merge(left, right, left_index=True, right_index=True)
   key1_x  key2_x  lval1  key1_y  key2_y  rval1
1      b      y      1      a      x      6
2      c      z      2      b      a      7
```

两个数据集中都有1和2这两个索引标签（Index Label），因此合并的时候会以此为依据，对于名字相同的列会各自加上后缀_x和_y。

8.2.2 merge进阶

除了上一节的用法，merge（）函数还支持通过how参数来指定不同的合并方法。how的参数取值可以有以下4种：

- inner：两个DataFrame的key交集，也称内连接；
- outer：两个DataFrame的key并集，也称外连接；
- left：只使用左边的DataFrame的key，也称左连接；
- right：只使用右边的DataFrame的key，也称右连接。

使用参数outer，将得到如下结果。

```
>>> left.merge(right, how='outer')
   key1  key2  lval1  rval1
0     a     x    0.0    6.0
1     b     y    1.0    NaN
2     c     z    2.0    8.0
3     b     a    NaN    7.0
```

由于是取并集，那么合并的时候会将key1和key2在两个数据集中的组合方式合并起来，因此合并后得到的数据有4行，key1和key2对应

的取值为b和y的情况仅在left数据中存在，那么对应的rval1列就会填充NaN。接下来再看使用left中的key的情况，代码如下。

```
left.merge(right,how = 'left')
   key1 key2 lval1 rval1
0    a    x     0   6.0
1    b    y     1   NaN
2    c    z     2   8.0
```

此时显然保留left，用right中的数据来进行补充。反之，也可以选择使用右连接，代码如下。

```
>>> left.merge(right,how = 'right')
   key1 key2 lval1 rval1
0    a    x   0.0     6
1    c    z   2.0     8
2    b    a   NaN     7
```

8.2.3 join与concat

除了使用merge（）函数，Pandas还提供了join（）函数来合并DataFrame的列，例如，下面的代码采用左连接方式基于索引进行合并，左边数据集的列会加上后缀_left，右边的加上后缀_right，结果如图8.12所示。

```
>>> left.join(right, lsuffix = '_left', rsuffix = '_right')
```

	key1_left	key2_left	lval1	key1_right	key2_right	rval1
0	a	x	0	NaN	NaN	NaN
1	b	y	1	a	x	6.0
2	c	z	2	b	a	7.0

图8.12 左连接

下面的代码分别实现了外连接和内连接，结果如图8.13和图8.14所示。

```
>>> left.join(right,how='outer',lsuffix='_left',rsuffix='_right')
>>> left.join(right,lsuffix='_left',rsuffix='_right',how='inner')
```

	key1_left	key2_left	lval1	key1_right	key2_right	rval1
0	a	x	0.0	NaN	NaN	NaN
1	b	y	1.0	a	x	6.0
2	c	z	2.0	b	a	7.0
3	NaN	NaN	NaN	c	z	8.0

图8.13 外连接

	key1_left	key2_left	lval1	key1_right	key2_right	rval1
1	b	y	1	a	x	6
2	c	z	2	b	a	7

图8.14 内连接

最后来看看concat（）操作，假设有如下两个DataFrame：

```
>>> df1 = pd.DataFrame(np.arange(9).reshape(3, 3),
                        columns = ['a', 'b', 'c'])
>>> df2 = pd.DataFrame(np.arange(9, 18).reshape(3, 3),
                        columns = ['a', 'b', 'c'])
>>> df1
   a  b  c
0  0  1  2
1  3  4  5
2  6  7  8
>>> df2
   a  b  c
0  9 10 11
1 12 13 14
2 15 16 17
```

如果运行concat（）函数，那么默认的情况下会按照行进行合并，得到如下结果。

```
>>> pd.concat([df1, df2])
```

	a	b	c
0	0	1	2
1	3	4	5
2	6	7	8
0	9	10	11
1	12	13	14
2	15	16	17

由于df1和df2的列名都是一样的，所以concat（）函数会直接将df2的行添加到df1后面，就完成了合并。但是如果存在两边列不相同的情况，合并将自动补充NaN，示例如下。

```
>>> df1 = pd.DataFrame(np.arange(9).reshape(3, 3),
                        columns = ['a', 'b', 'c'])
>>> df2 = pd.DataFrame(np.arange(9, 18).reshape(3, 3),
                        columns = ['a', 'c', 'd'])
>>> pd.concat([df1, df2], sort = False)
```

	a	b	c	d
0	0	1.0	2	NaN
1	3	4.0	5	NaN
2	6	7.0	8	NaN
0	9	NaN	10	11.0
1	12	NaN	13	14.0
2	15	NaN	16	17.0

上述代码中使用sort参数是由于Pandas的新版本中默认不排序，默认采用接受该方式。对于合并后的数据，如果想知道它来自哪一个DataFrame，那么可以在合并的时候指定key参数，此时key将作为level0的索引存在，代码如下。

```
>>> c = pd.concat([df1, df2], keys = ['df1', 'df2'], sort = False)
>>> c
```


		a	b	c	d
df1	0	0	1.0	2	NaN
	1	3	4.0	5	NaN
	2	6	7.0	8	NaN
df2	0	9	NaN	10	11.0
	1	12	NaN	13	14.0
	2	15	NaN	16	17.0

图8.15 指定key作为level0的索引

运行结果如图8.15所示。

在这种情况下，也可以通过level 0的索引来访问原来的df1中的数据，代码如下。

```
>>> c.loc['df1']
   a  b   c  d
0  0  1.0  2 NaN
1  3  4.0  5 NaN
2  6  7.0  8 NaN
```

当然，concat（）函数其实也支持按照列进行连接，只需要指定axis=1就可以了，下面的代码就可以实现将df1和df2按索引进行列连接。

```
>>> pd.concat([df1, df2], axis=1)
   a  b  c  a   c  d
0  0  1  2  9  10 11
1  3  4  5 12 13 14
2  6  7  8 15 16 17
```

第9章 数据可视化

图片的最大价值就是，让我们看到我们从未看到过的东西。

——*John Tukey*

可视化作为利用数据、信息来达到目的、解决问题的最后一环，在很多时候其重要性往往高于前面的所有工作。在经过了定义问题、收集数据、数据清洗、数据建模、评价的环节以后，终于到了要将数据分析的结果展示出来的时候，它直接决定了分析的结果是否可以落地，是否可以实现价值。本章将重点介绍Python中最典型的两种可视化工具：Matplotlib和Seaborn。

9.1 Matplotlib

Matplotlib是Python的一个绘图库，它包含了大量的工具，也提供了极大的灵活性，理论上使用者可以利用它绘制任何图形。正是由于这一特性，使得Matplotlib学习曲线相对陡峭，本节将以初学者的视角带领读者由浅入深一步步地掌握Matplotlib的各项功能。

9.1.1 绘制第一个散点图

Jupyter Notebook中几乎所有利用Matplotlib绘图的代码都会包含下面两句。

```
>>> import matplotlib.pyplot as plt
>>> %matplotlib inline
```

上述代码首先将matplotlib.pyplot以别名plt导入，它是整个Matplotlib库的核心，包含了所有绘图相关的函数，在Notebook中输入`dir (plt)`可以查看所有函数。而`%matplotlib inline`则是Jupyter Notebook中的特殊命令，有了它，所有绘图将在Notebook中自动显示，如果没有这一行代码，就需要调用`plt.show ()`函数让Notebook显示图形。导入了Matplotlib库后，就可以用`plt.plot ()`函数来绘制第一个图形，运行如下代码将得到图9.1所示图形。

```
>>> plt.plot([1,2,3,4,8])
```

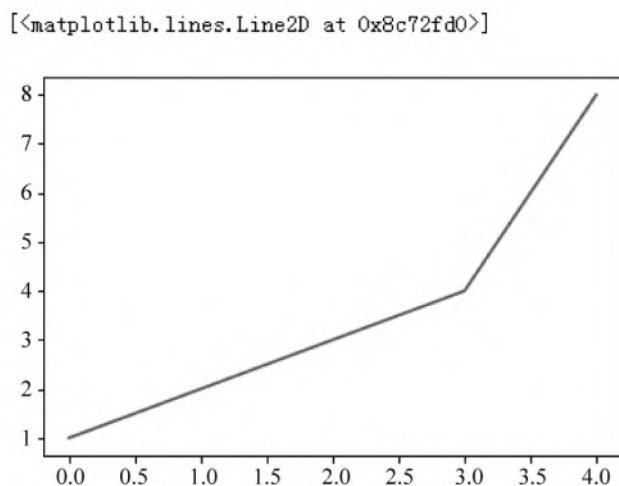


图9.1 折线图

上述代码将一个列表的数据传给了`plt.plot ()`函数，该函数自动绘制了一幅折线图，其中的Y轴取值来自输入的数据列表，而X轴取值则对应输入列表的索引，因此，最后得到如图9.1所示的图形。细心的读者还会注意到图形输出上方有一行显示[<matplotlib.lines.Line2Dat0x8c72fd0>]，其实这是`plt.plot ()`函数返回的图形对象，如果不想有这样的显示，在代码后面加上“;”就可以不输出该内容，当然通过在代码下一行执行`plt.show ()`函数也可以达到同样效果。不过前面不是说要画散点图吗？现在`plt.plot ()`怎么默认画的是折线图呢？要理解原因，就需要了解

`plt.plot()` 的参数使用，这个函数有3个基本输入项（`x`，`y`，`format`）。其中，`format`包含以下3种参数的组合：`{color}{marker}{line}`。例如，“`go-`”依次代表的是绿色（对应`g`）[\(1\)](#)，对应（`x`，`y`）处画点（对应`o`），各个点间用线连接（对应`-`）。如果给出的`format`组合是“`go`”，此时就将得到绿色散点图。为帮助大家更深入理解这种绘图格式，下面再来看几个例子。

- `r*-`：带红色星型点的虚线，`r`代表红色，`*`代表星型，`-`代表虚线。
- `ks.`：带黑色方块点的实线，`k`代表黑色，`s`代表方块，`.`代表点。
- `bd-.`：带蓝色钻石点的点画线，`b`代表蓝色，`D`代表钻石点，`-.`代表点画线。

关于 `{color}{marker}{line}` 的更详细内容，读者可以通过 `help(plt.plot)` 获取更深入的说明。接下来再来看一个画红色散点图的例子，运行下面代码将得到如图9.2所示输出。由于代码后面加了“`;`”，所以不再看到如图9.1那样的对象地址输出。

```
>>> plt.plot([1,2,3,4,5], [1,2,3,6,10], 'ro');
```

前面画的是一组散点图，如果要画两组数据的散点图呢？很简单，只需要画两次就可以得到如图9.3所示的散点图，代码如下。

```
>>> plt.plot([1,2,3,4,5], [1,2,3,6,10], 'g* ')
>>> plt.plot([2,5,6,8,9], [5,3,7,9,11], 'bD')
>>> plt.show()
```

一幅完整的图形不只是图形部分，还包含标题、坐标轴等，接下来为上面的图加上标题、坐标轴、图例等，运行如下代码，将得到如图9.4所示的图形。

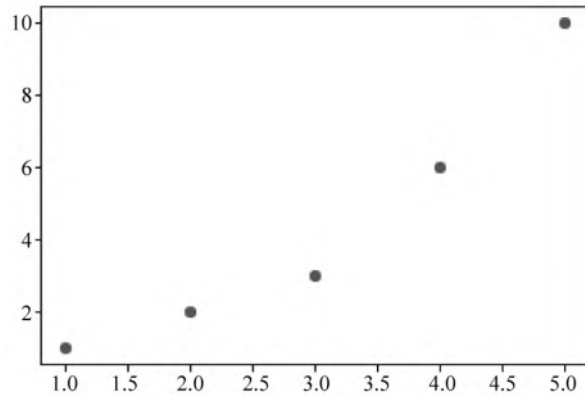


图9.2 散点图

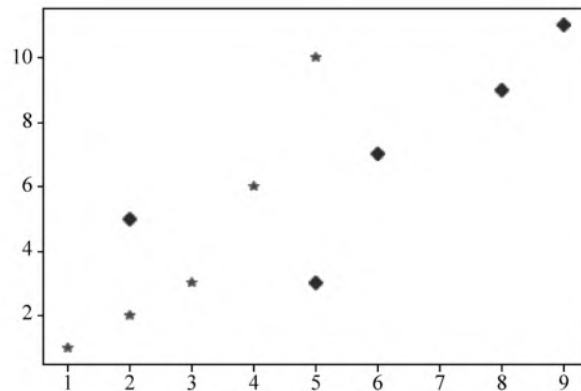


图9.3 两组数据散点图

```
>>> plt.plot([1,2,3,4,5], [1,2,3,6,10], 'g*',label="Green Star")
>>> plt.plot([2,5,6,8,9], [5,3,7,9,11], 'bD',label="Blue Diamond")
>>> plt.title('Scatterplot With Legend')
>>> plt.xlabel('X')
>>> plt.ylabel('Y')
>>> plt.legend(loc='best')
>>> plt.show()
```

上述代码中，`plt.plot()` 函数通过`label`参数给出了图例信息，之后的`plt.legend(loc='best')`给出图例的位置，`best`代表自适应选择最佳位置，更多的`loc`参数取值读者可以通过`help(plt.legend)`获得。`plt.xlabel('X')`和`plt.ylabel('Y')`分别给出了两个坐标轴的名称。`plt.title('Scatterplot With Legend')`则给出了整个

图形的标题。如果图形中有中文字符，为了使中文能正确显示，需要指定字体，运行如下代码将得到能显示中文字符的图形，如图9.5所示。

```
>>> plt.rcParams['font.sans-serif'] = ['SimHei'] # 显示中文标签
# 坐标轴有负号的时候可以显示负号
>>> plt.rcParams['axes.unicode_minus'] = False
>>> plt.plot([1,2,3,4,5], [1,2,3,6,10], 'g*', label="绿星")
>>> plt.plot([2,5,6,8,9], [5,3,7,9,11], 'bD', label="蓝钻")
>>> plt.title('带有图例的散点图')
>>> plt.xlabel('X')
>>> plt.ylabel('Y')
>>> plt.legend(loc='best')
>>> plt.show()
```

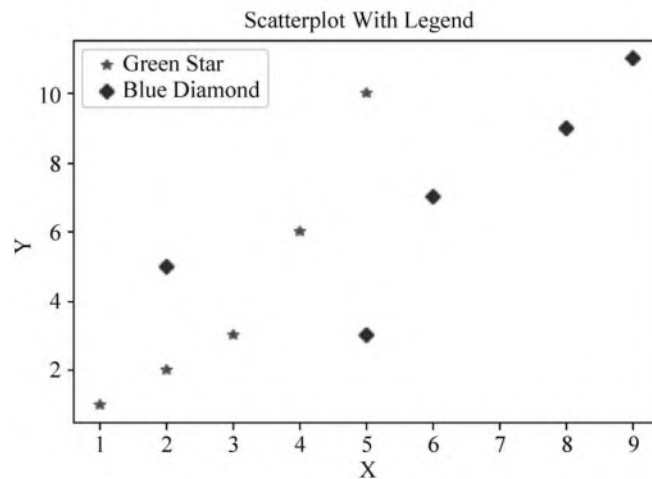


图9.4 带图例和标题的散点图

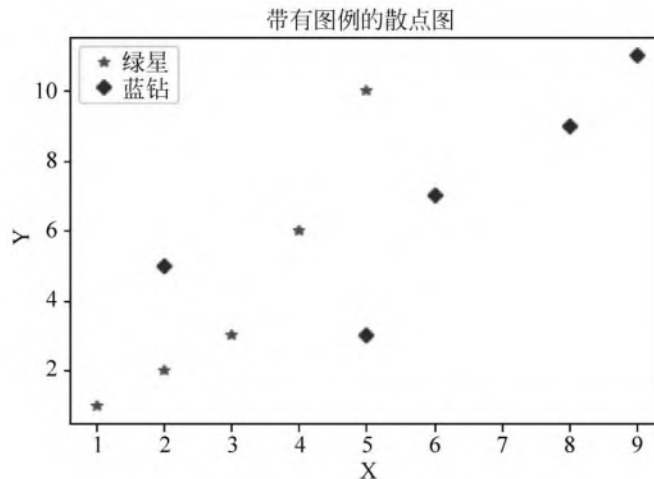


图9.5 中文标题散点图

上述代码中，第一行代码给出了字体，这里使用了SimHei字体，当然如果读者的计算机中有其他中文字体，也可以指定使用该字体。对于Linux系统，通常可以通过命令`fc-list:lang=zh`查看计算机中的中文字体。如果使用的是Windows系统，通常可以通过“控制面板”中的“字体”进行查看。第二行代码则是解决当坐标轴有负号的时候可以显示负号。现在图形有了标题、图例，但是整个图看起来似乎有点小，应该怎么办呢？同时，如果要修改X轴和Y轴显示的范围，应该如何操作呢？下面的代码给出了方法。

```
>>> plt.figure(figsize=(12,9))
>>> plt.xlim(0,12)
>>> plt.ylim(0,12)
>>> plt.rcParams['font.sans-serif'] = ['SimHei'] # 显示中文标签
# 坐标轴有负号的时候可以显示负号
>>> plt.rcParams['axes.unicode_minus'] = False
>>> plt.plot([1,2,3,4,5], [1,2,3,6,10], 'g*',label="绿星")
>>> plt.plot([2,5,6,8,9], [5,3,7,9,11], 'bD',label="蓝钻")
>>> plt.title('带有图例的散点图')
>>> plt.xlabel('X')
>>> plt.ylabel('Y')
>>> plt.legend(loc='best')
>>> plt.show()
```

其中，`plt.xlim(0,12)`和`plt.ylim(0,12)`给出了绘图时显示的X轴和Y轴范围。`plt.figure(figsize=(12,9))`则给出了图形的长与宽。

9.1.2 理解figure与axes

现在读者对Matplotlib中的绘图已经有了基本的了解，是时候来深入了解其绘图机制了。在Matplotlib绘图语言中有两个基本术语：figure与axes，读者可以把figure想象成绘画的画布，在Matplotlib中每幅图都是在figure上面绘制的，这个对象包含了所有的子图以及图形元素。图9.6详细说明了一张Matplotlib图形中存在的图形元素，实际上绘图就是添加、调整这些元素。

理解了figure，那么axes是做什么的呢？要回答这个问题，需要了解Matplotlib中的另一种绘图方式。假设现在想画一幅图，这幅图包含两个散点图水平排列，请问应该如何绘制呢？此时就需要通过`plt.subplots(1,2)`使用子图（也称为axes）来完成这一操作。利用该函数创建两个对象：figure对象和在figure对象中的axes对象。通过如下代码，就可以得到如图9.7所示的图形。

```
>>> fig, (ax1, ax2) = plt.subplots(1,2,
    figsize=(6,4),
    sharey=True,dpi=120)
>>> plt.rcParams['font.sans-serif'] = ['SimHei']
>>> plt.rcParams['axes.unicode_minus'] = False
>>> ax1.plot([1,2,3,4,5], [1,2,3,6,10], 'g* ')
>>> ax2.plot([2,5,6,8,9], [5,3,7,9,11], 'bD')
>>> ax1.set_title('绿星散点图');ax2.set_title('蓝钻散点图');
>>> ax1.set_xlabel('X');ax2.set_xlabel('X')
>>> ax1.set_ylabel('Y');ax2.set_ylabel('Y')
>>> ax1.set_xlim(0, 12);ax2.set_xlim(0, 12)
>>> ax1.set_ylim(0, 12);ax2.set_ylim(0, 12)
>>> plt.tight_layout()
>>> plt.show()
```

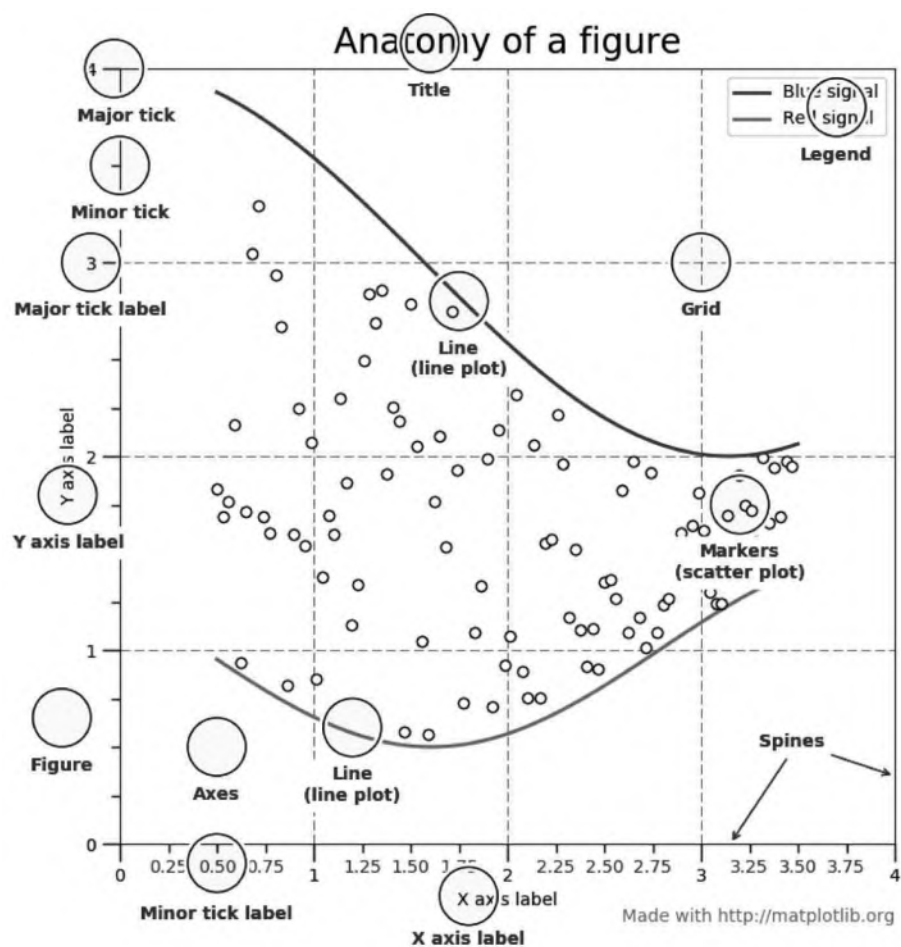



图9.6 Matplotlib中的图形元素

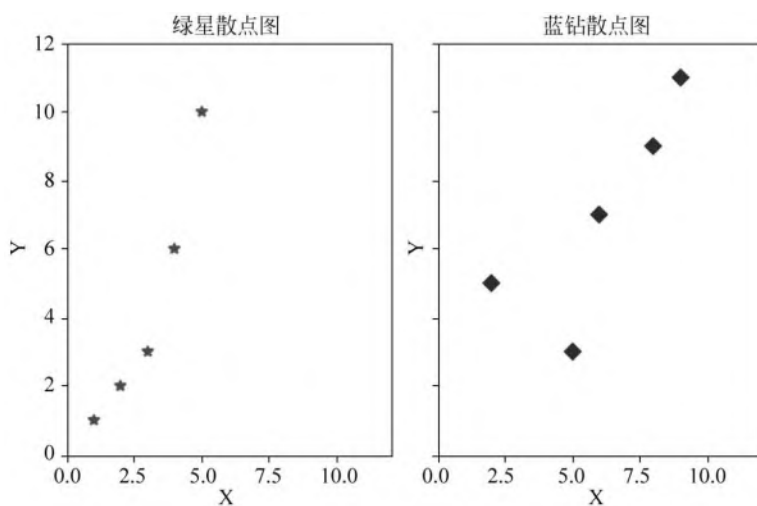


图9.7 并列子图

其中，`plt.subplots (1, 2)` 可以创建一个有两个子图的figure对象，返回值包括了figure对象以及axes对象列表。`sharey=True`参数表示两个子图共用Y轴，`dpi`参数表示分辨率，指每英寸长度上的点数。子图对象 `ax1`、`ax2` 与 `plt` 一样，提供了 `set_title ( )`、`set_xlabel ( )`、`set_ylabel ( )` 等函数。实际上`plt.title ( )` 底层调用的就是当前axes对象的 `set_title ( )` 函数。类似地，`plt.xlabel ( )` 对应 `ax.set_xlabel ( )`，`plt.ylabel ( )` 对应 `ax.set_ylabel ( )`，`plt.xlim ( )` 对应 `ax.set_xlim ( )`，`plt.ylim ( )` 对应 `ax.set_ylim ( )`。为了简化代码，也可以通过axes对象的`set ( )` 函数一次完成这些设置，代码如下。

```
>>> ax1.set(title='绿星', xlabel='X',
            ylabel='Y', xlim=(0,12), ylim=(0,12))
>>> ax2.set(title='蓝钻', xlabel='X',
            ylabel='Y', xlim=(0,12), ylim=(0,12))
```

9.1.3 Matplotlib中面向对象与类Matlab语法的区别

随着对Matplotlib的学习的深入，读者会很自然地发现，似乎Matplotlib中有两种不同的绘图语法。一种是类似Matlab的语法，另一种就是面向对象语法。也正是由于这一原因，很多人都觉得Matplotlib学习曲线陡峭，对新手一点都不友好。由于Matplotlib最初的目的是在Python中实现Matlab的绘图功能，所以它采用了类似的语法，即无状态语法（stateful）。所谓无状态是指`plt`会记录当前的axes是哪个，当使用`plt. {绘图函数}`绘图时，它将在当前的那个子图上绘制。简单来说，面向对象和类Matlab语法的区别就是面向对象是在指定axes对象上面绘图，而类Matlab语法中所有的绘图则是使用的`plt ( )` 在当前那个axes对象上绘制。那么问题来了，如果想如前一节那样绘制两幅并列的图，采用类Matlab的语法该怎么实现呢？其实很

简单，只要每次先建立一幅子图（使用`plt.subplot（）`或`plt.add_subplot（）`函数），然后使用`plt.plot（）`或`plt.{其他绘图属性函数}`来修改对应的子图（`axes`）就可以了。这里需要牢记，不管调用的是`plt`的哪个方法，绘图都是在当前的`axes`（子图）上进行。下面来看一下代码。

```
>>> plt.figure(figsize=(6,4), dpi=120)
>>> plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签
# 左边子图
>>> plt.subplot(1,2,1) # (行,列,第几个子图)
>>> plt.plot([1,2,3,4,5], [1,2,3,6,10], 'g*')
>>> plt.title('绿星散点图')
>>> plt.xlabel('X'); plt.ylabel('Y')
>>> plt.xlim(0, 12); plt.ylim(0, 12)
# 右边子图
>>> plt.subplot(1,2,2)
>>> plt.plot([2,5,6,8,9], [5,3,7,9,11], 'bD')
>>> plt.title('蓝钻散点图')
>>> plt.xlabel('X'); plt.ylabel('Y')
>>> plt.xlim(0, 12); plt.ylim(0, 12)
>>> plt.show()
```

这段代码产生的输出与图9.7相同，与前一段代码的区别在于这段代码使用的是类Matlab语法绘图，没有明确指明`axes`对象。其中，代码`plt.subplot（1，2，1）`中前两个参数（1，2）分别代表了新建的子图的行数与列数，这里是1行2列，第三个参数“1”指定当前子图为第1个子图，后续对应的`plt`函数将在该子图进行图形绘制。为了方便获取或设定`axes`与`figure`，Matplotlib中提供了`plt.gca（）`和`plt.gcf（）`函数来获取当前`axes`和`figure`。而`plt.cla（）`和`plt.clf（）`函数则可以清除当前`axes`和`figure`。看起来面向对象的绘图方式与类Matlab方式相比似乎没有太大优势，不过如果要绘制如图9.8所示的图形呢？

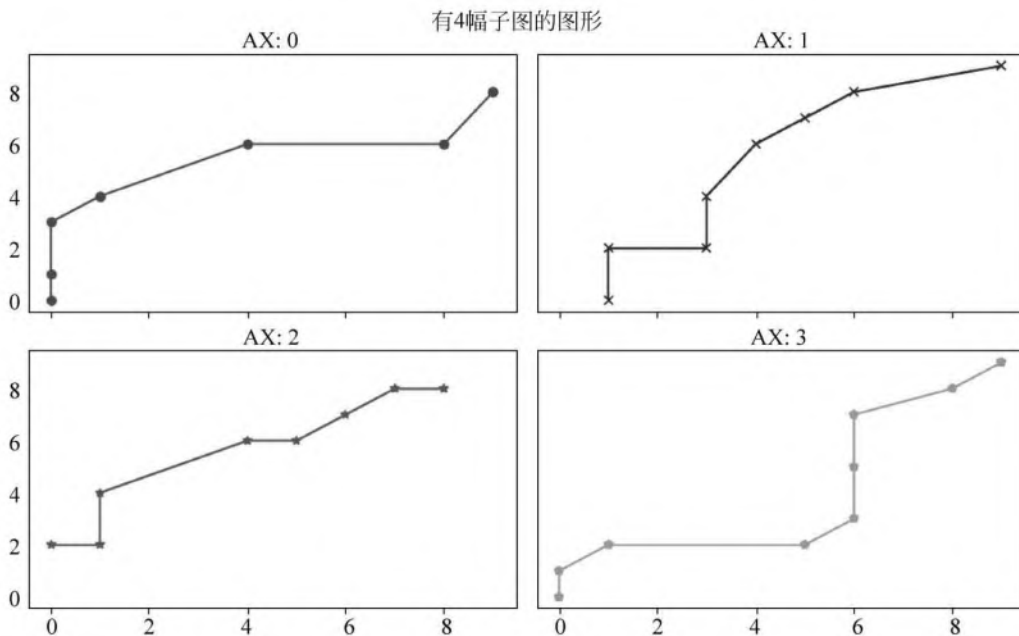


图9.8 有4幅子图的图形

此时如果采用无状态方式绘图，代码将大量重复，而采用面向对象方式则相对简单，示例如下。

```
>>> import numpy as np
>>> from numpy.random import seed, randint
>>> seed(10)
# 创建 figure, subplots
>>> fig, axes = plt.subplots(2, 2, figsize = (10, 6),
                             sharex = True, sharey = True, dpi = 120)
>>> plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签
# 定义要使用的颜色与标记
>>> colors = {0:'g', 1:'b', 2:'r', 3:'y'}
>>> markers = {0:'o', 1:'x', 2:'*', 3:'p'}
# 绘制各子图
>>> for i, ax in enumerate(axes.ravel()):
    ax.plot(sorted(randint(0, 10, 10)), sorted(randint(0, 10, 10)),
            marker = markers[i], color = colors[i])
    ax.set_title('Ax: ' + str(i))
    ax.yaxis.set_ticks_position('none')
>>> plt.suptitle('有 4 幅子图的图形', verticalalignment = 'bottom',
                fontsize = 16)
>>> plt.tight_layout()
>>> plt.show()
```

这段代码通过循环遍历axes对象的方式，将重复性的工作放到了一起，绘图代码得到了极大简化。此外，还可以使用plt.suptitle()函数为整个绘图添加一个标题（plt.title()是给当前的子图添加标题），verticalalignment='bottom'参数给出了对齐方式是底部。最后，这段代码使用了ax.yaxis.set_ticks_position('none')使得Y轴不绘制刻度标记。这也是使用面向对象的好处，理论上用户可以获得axes对象的所有子对象，之后调用该子对象的方法或属性对图形进行修改，下面将进一步讨论这种设置图形属性的方式。

9.1.4 修改坐标轴属性

读者已经知道了一幅Matplotlib图形中有多个属性，图形的每个属性都可以进行调整。这一方面使得Matplotlib功能极其强大，理论上可以绘制出任何图形，当然另一方面也使得它学习起来有些困难。本节将一步步带领大家掌握Matplotlib中如何对各图形属性进行调节。首先来看看经常用到的关于坐标轴以及标签的修改方法。

- 修改坐标轴的刻度位置以及标签（plt.xticks()/ax.setxticks()/ax.setxticklabels()）。
- 控制坐标轴的刻度显示位置（上 / 下 / 左 / 右），使用plt.tick_params()。
- 格式刻度标签。

假设要绘制如图9.9所示的图形，应该怎么办呢？

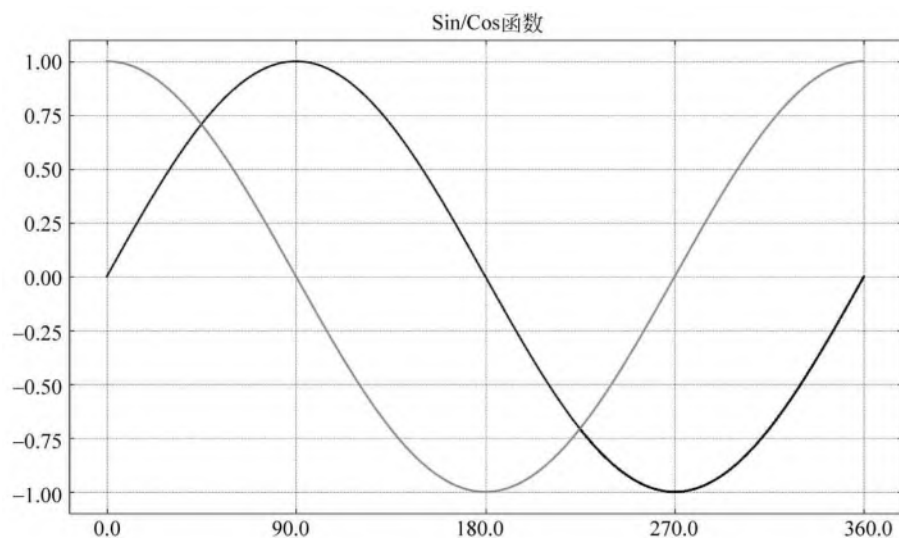


图9.9 正弦余弦曲线

要设置X轴的刻度和标签，需要用到 `xticks()` 函数。`plt.xticks()` 一次完成了X轴的刻度、标签字体、方向等的设置，实际上底层也是通过调用 `ax.set_xticks()` 和 `ax.set_xticklabels()` 来实现。`plt.tick_params()` 则对每个坐标轴的刻度进行了设置，这里设置了每个轴（上 / 下 / 左 / 右）都显示主刻度标记，同时标记显示在内部（`direction='in'`），网格采用蓝色虚线。最后因为要在X轴显示度数，所以需要对X轴的刻度标签进行从弧度到角度的转换，首先利用 `FuncFormatter` 定义了一个 `formatter`，之后使用 `plt.gca().xaxis.set_major_formatter(formatter)` 将该 `formatter` 应用到X轴。具体代码如下。

```

>>> from matplotlib.ticker import FuncFormatter
# 当坐标轴有负号的时候可以显示负号
>>> plt.rcParams['axes.unicode_minus'] = False
>>> def rad_to_degrees(x, pos):
    '角度弧度转换'
    # 两个参数分别是值与 tick 位置
    return round(x * 57.2985, 2)
>>> plt.figure(figsize=(12,7), dpi=100)
# 0-2pi 间均匀生成 1000 个点
>>> X = np.linspace(0, 2 * np.pi, 1000)
>>> plt.plot(X, np.sin(X))
>>> plt.plot(X, np.cos(X))
# 1 弧度 = 57.2985 度
>>> plt.xticks(ticks = np.arange(0, 440/57.2985, 90/57.2985),
    fontsize=12, rotation=30, ha='center', va='top')
# tick 参数, 每个轴的都画刻度, 方向是朝内, 蓝色网格
>>> plt.tick_params(axis='both', bottom=True, top=True,
    left=True, right=True, direction='in', which='major',
    grid_color='blue')
# x 轴的弧度转度
>>> formatter = FuncFormatter(rad_to_degrees)
>>> plt.gca().xaxis.set_major_formatter(formatter)
>>> plt.grid(linestyle='--', linewidth=0.5, alpha=0.5)
>>> plt.title('Sin/Cos 函数', fontsize=14)
>>> plt.show()

```

9.1.5 修改图形属性

Matplotlib 中各图形组件的外观都可以通过其属性进行修改，对应到代码就是通过 `rcParams` 进行全局设置。要了解详细的图形属性，可以通过 `mpl.rcParams()` 查看，代码如下。

```
>>> import matplotlib as mpl
>>> mpl.rc_params()
RcParams({'_internal.classic_mode': False,
          'agg.path.chunksize': 0,
          'animation.avconv_args': [],
          'animation.avconv_path': 'avconv',
          'animation.bitrate': -1,
          'animation.codec': 'h264',
          'animation.convert_args': [],
          'animation.convert_path': 'convert',
          ...})
```

上述代码中只是列出了一部分属性，这里以字体属性为例，来说明如何进行修改，其他属性修改基本类似。例如，STIX字体是一种常用字体，读者可以输入以下代码来一次完成设置。

```
>>> mpl.rcParams.update({'font.size': 18,
                        'font.family': 'STIXGeneral', 'mathtext.fontset': 'stix'})
```

如果想恢复原来的设置，则可以通过下面的代码实现。

```
>>> mpl.rcParams.update(mpl.rcParamsDefault)
```

单个修改各种属性的确很灵活，但是人工设置各图形属性，一是比较麻烦，另外就是可能使整个图形的样式存在这样或那样的不一致情况。因此，Matplotlib提供了各种设置好的样式供用户选择，通过如下代码，可以查看当前系统中已安装的样式。

```
>>> plt.style.available
['bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 ...
 'tableau-colorblind10',
 '_classic_test']
```


要选择使用哪种样式绘图也非常简单，只需要在绘图时进行指定就可以了。例如，如下代码就使用了不同样式进行绘图。

```
>>> import numpy as np
>>> import matplotlib as mpl
>>> mpl.rcParams.update({'font.size': 18,
    'font.family': 'STIXGeneral', 'mathtext.fontset': 'stix'})
>>> def plot_sine_cosine_wave(style = 'ggplot'):
    plt.style.use(style)
    plt.figure(figsize = (7,4), dpi = 80)
    X = np.linspace(0, 2 * np.pi, 1000)
    plt.plot(X, np.sin(X)); plt.plot(X, np.cos(X))
    plt.xticks(ticks = np.arange(0, 440/57.2985, 90/57.2985),
        labels = [r'$0$', r'$\frac{\pi}{2}$', r'$\pi$',
            r'$\frac{3\pi}{2}$', r'$2\pi$'])
    plt.gca().set(ylim = (-1.25, 1.25), xlim = (-.5, 7))
    plt.title(style, fontsize = 18)
    plt.show()
>>> plot_sine_cosine_wave('seaborn-notebook')
>>> plot_sine_cosine_wave('ggplot')
>>> plot_sine_cosine_wave('fivethirtyeight')
```

上述代码首先定义了一个绘图函数`plot_sine_cosine_wave()`，该函数通过参数`style`来决定选择的绘图样式，然后用`plt.style.use(style)`来完成设置。最后得到的图形如图9.10～图9.12所示，可以看出不同样式对应了不同字体、大小、颜色等属性。此外，上述代码中对于X轴的刻度标签的显示特意采用了另一种方式，即使用了参数`labels=[r'$0$', r'$\frac{\pi}{2}$', r'$\pi$', r'$\frac{3\pi}{2}$', r'$2\pi$']`来完成，这里是利用了Latex的语法，如果读者经常需要写科技论文，那么应该对此非常熟悉。如果有不清楚的地方可以去参考Latex的相关文档。

Matplotlib中提供了3组不同的颜色，分别是CSS4风格、XKCD风格和系统的基础色，使用者可以通过如下代码查看具体颜色。

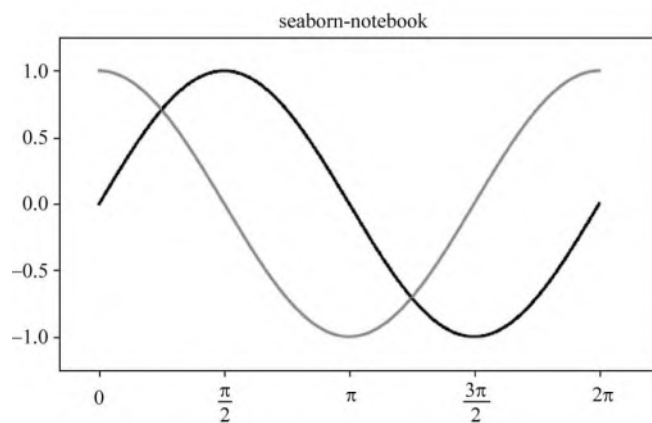


图9.10 seaborn-notebook

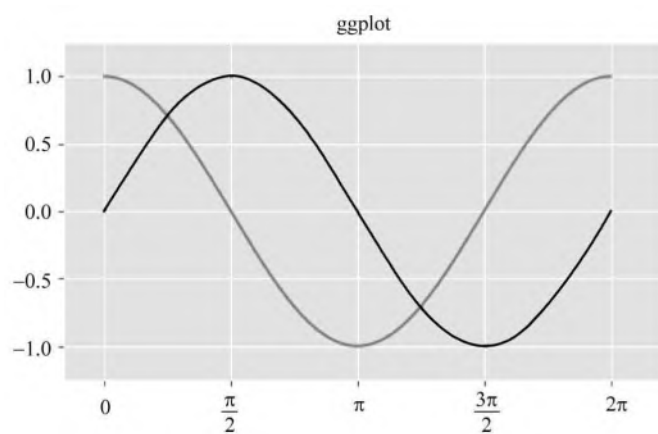


图9.11 ggplot

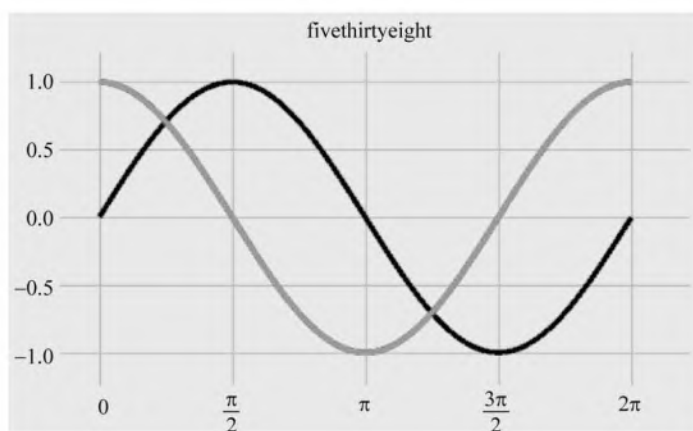


图9.12 fivethirtyeight

```
>>> mpl.colors.CSS4_COLORS # 148 colors
>>> mpl.colors.XKCD_COLORS # 949 colors
>>> mpl.colors.BASE_COLORS # 8 colors
```

除此之外，为了方便用户更好地使用颜色对比，Matplotlib中还提供了调色板，通过如下代码可以进行查看。

```
>>> dir(plt.cm)[:10]
['Accent',
 'Accent_r',
 'Blues',
 'Blues_r',
 'BrBG',
 'BrBG_r',
 'BuGn',
 'BuGn_r',
 'BuPu',
 'BuPu_r']
```

9.1.6 定制图例，添加标注

绘图过程中经常会遇到图例说明、在特定位置添加标注以及文本说明的情况，本节将介绍如何在Matplotlib中完成这一功能。首先来看图例的定制，仍然是绘制正弦余弦曲线，如果想得到如图9.13所示图形，应该如何完成呢？

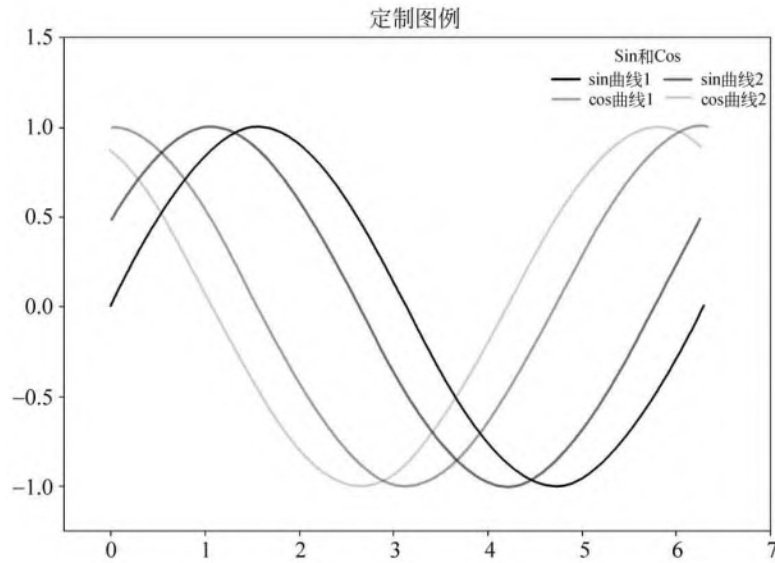


图9.13 定制图例

图例的设置主要通过`plt.legend()`函数来完成，可以使用如下代码。

```
>>> mpl.rcParams.update(mpl.rcParamsDefault)
>>> plt.rcParams['font.sans-serif'] = ['SimHei']
>>> plt.rcParams['axes.unicode_minus'] = False
>>> plt.style.use('seaborn-notebook')
>>> plt.figure(figsize=(10,7), dpi=80)
>>> X = np.linspace(0, 2 * np.pi, 1000)
>>> sine_1 = plt.plot(X, np.sin(X))
>>> cosine_1 = plt.plot(X, np.cos(X))
>>> sine_2 = plt.plot(X, np.sin(X + .5))
>>> cosine_2 = plt.plot(X, np.cos(X + .5))
>>> plt.gca().set(ylim=(-1.25, 1.5), xlim=(-.5, 7))
>>> plt.title('定制图例', fontsize=18)
# 修改图例
>>> plt.legend([sine_1[0], cosine_1[0], sine_2[0], cosine_2[0]],
               ['sin 曲线 1', 'cos 曲线 1', 'sin 曲线 2', 'cos 曲线 2'],
               frameon=False,           # legend border
               framealpha=1,           # transparency of border
               ncol=2,                 # num columns
               shadow=True,            # shadow on
               borderpad=1,            # thickness of border
               title='Sin 和 Cos',
               loc='best')
>>> plt.show()
```

其中，`frameon`指明是否画边框，`framealpha`指明了边框透明度，`ncol`指明分成两列，`shadow`指明是否有阴影，`boderpad`指明边框厚度，`title`指明图例标题。通过`plt.text()`和`plt.annotate()`函数还可以给图形添加标注以及文本说明，例如如下代码可以完成图形标注功能，绘制出如图9.14所示的图形。

```
>>> plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签
>>> plt.figure(figsize=(14,7), dpi=120)
>>> X = np.linspace(0, 8 * np.pi, 1000)
>>> sine = plt.plot(X, np.sin(X), color='tab:blue');
# 使用 ArrowProps 和 bbox 设置标注
>>> plt.annotate('峰', xy=(90/57.2985, 1.0), xytext=(90/57.2985, 1.5),
                 bbox=dict(boxstyle='square', fc='green', linewidth=0.1),
                 arrowprops=dict(facecolor='green', shrink=0.01, width=0.1),
                 fontsize=12, color='white', horizontalalignment='center')
# 文本
>>> for angle in [440, 810, 1170]:
    plt.text(angle/57.2985, 1.05, str(angle) + "\n度",
             horizontalalignment='center', color='green')
>>> for angle in [270, 630, 990, 1350]:
    plt.text(angle/57.2985, -1.3, str(angle) + "\n度",
             horizontalalignment='center',
             color='red')
>>> plt.gca().set(ylim=(-2.0, 2.0), xlim=(-.5, 26))
>> plt.title('标注与文本示例', fontsize=18)
>>> plt.show()
```

其中，`plt.annotate()`函数通过`xy`参数决定标注箭头位置，`xytext`决定了文本位置，`bbox`指定了文本框的样式，`arrowprops`指定了箭头的样式，而`plt.text()`函数则指定了文本位置。

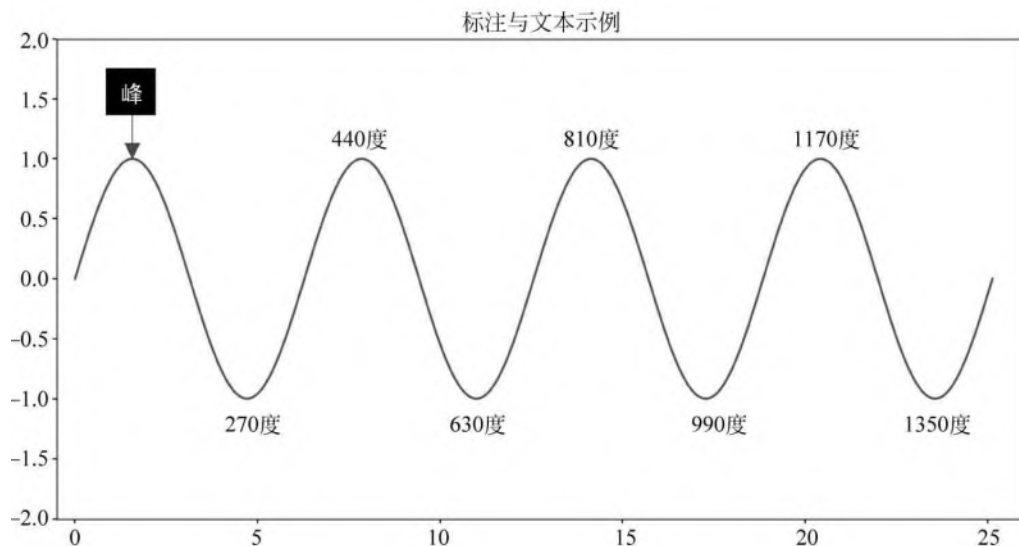


图9.14 标注与文本示例

此外，在`plt.text()`函数中还有一个参数`transform`，它可以用于窗口坐标与子图坐标之间的变换，读者可以通过文档了解 (https://matplotlib.org/3.1.1/tutorials/advanced/transforms_tutorial.html)，这里用一个简单例子来说明。

```
>>> plt.figure(figsize=(14,7), dpi=80)
>>> X = np.linspace(0, 8 * np.pi, 1000)
>>> plt.text(0.50, 0.02, "中心位置相对于 data: (0.50, 0.02)",
            transform=plt.gca().transData,
            fontsize=14,
            ha='center',
            color='blue')
>>> plt.text(0.50, 0.02, "中心位置相对于 axes: (0.50, 0.02)",
            transform=plt.gca().transAxes,
            fontsize=14,
            ha='center',
            color='blue')
>>> plt.text(0.50, 0.02, "中心位置相对于 axes: (0.50, 0.02)",
            transform=plt.gcf().transFigure,
            fontsize=14,
            ha='center',
            color='blue')
>>> plt.gca().set(ylim=(-2.0, 1), xlim=(0, 1))
>>> plt.title('文本位置相对 data, axes 和 figure 坐标', fontsize=18)
>>> plt.show()
```


上述代码中第二行`plt.subplot2grid()`函数的第一个参数给出了整个图形是 3×3 的网格，现在首先绘制 $(0, 0)$ 位置子图，`colspan=2`代表该子图占列方向两个网格，`rowspan=2`代表占行方向两个网格。接下来的`plt.subplot2grid()`函数就只能从 $(0, 2)$ 位置开始了，因为网格的行方向0和1已经被第一个子图占用，`rowspan=3`则代表了将行方向3个网格占满。依此类推就得到了图9.16的布局，利用此方法，用户可以构造任何图形布局。同样的功能，利用`plt.GridSpec()`函数也可以完成，例如如下代码将得到如图9.17所示的输出。

```
>>> import matplotlib.gridspec as gridspec
>>> fig = plt.figure()
>>> grid = plt.GridSpec(2, 3)      # 2行3列
>>> plt.subplot(grid[0, :2])      # 占据第1行,前2列
>>> plt.subplot(grid[0, 2])       # 占据第1行,第3列
>>> plt.subplot(grid[1, :1])
>>> plt.subplot(grid[1, 1:])
>>> fig.tight_layout()
```

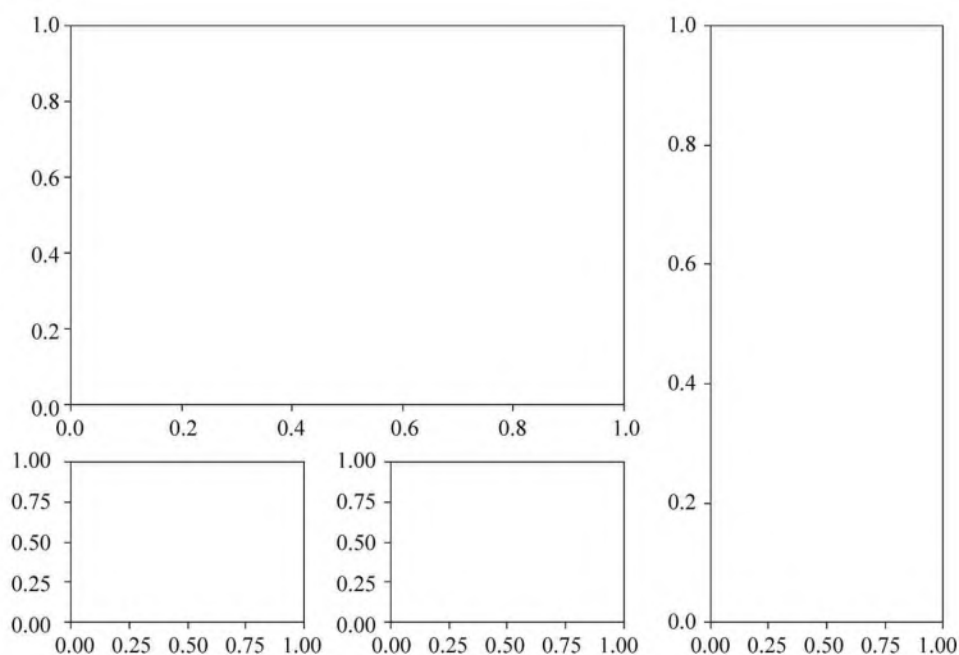


图9.16 subplot2grid实现子图

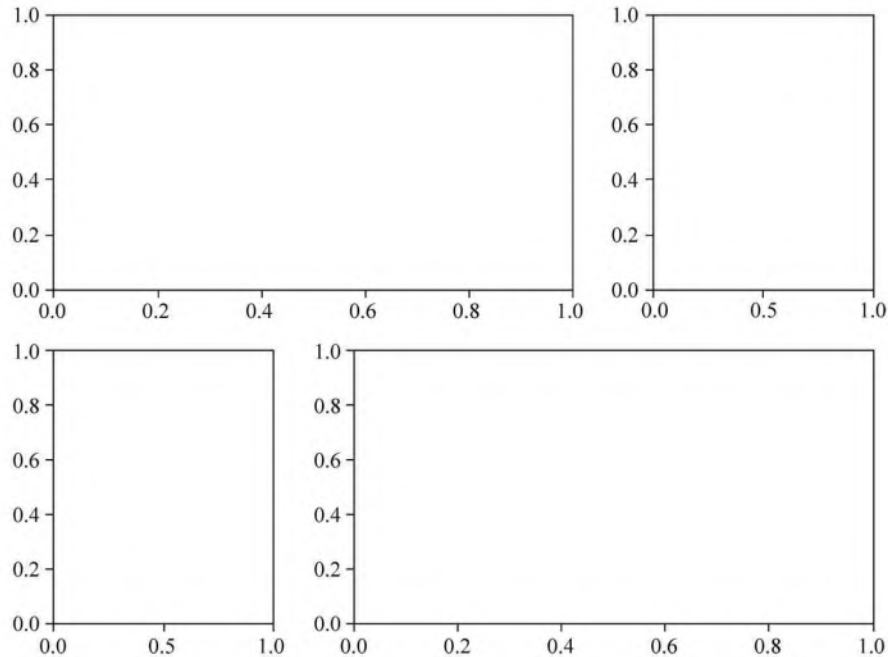


图9.17 GridSpec实现子图

除了子图，数据分析中有时还会绘制图中图，即在原图形区域中创建一个新的图形区域来放大显示某部分图形。例如，运行如下代码将得到如图9.18所示的图中图。

```
>>> plt.style.use('seaborn-whitegrid')
>>> plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签
>>> plt.rcParams['axes.unicode_minus'] = False
>>> fig, ax = plt.subplots(figsize=(10,6))
>>> x = np.linspace(-0.50, 1., 1000)
# 外部图形
>>> ax.plot(x, x**2)
>>> ax.plot(x, np.sin(x))
>>> ax.set(xlim=(-0.5, 1.0), ylim=(-0.5, 1.2))
>>> fig.tight_layout()
# 内部绘图
>>> inner_ax = fig.add_axes([0.2, 0.55, 0.35, 0.35])
>>> inner_ax.plot(x, x**2)
>>> inner_ax.plot(x, np.sin(x))
>>> inner_ax.set(title='放大', xlim=(-.2, .2), ylim=(-.01, .02),
>>> yticks=[-0.01, 0, 0.01, 0.02], xticks=[-0.1, 0, .1])
>>> ax.set_title("图形内绘图", fontsize=20)
>>> plt.show()
>>> mpl.rcParams.update(mpl.rcParamsDefault)
```

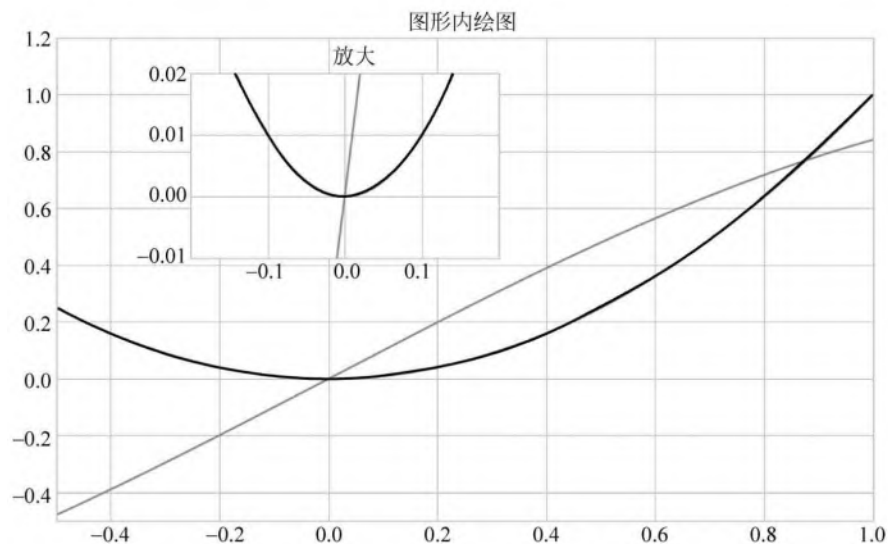


图9.18 图中图

上述代码中最关键部分就是`inner_ax=fig.add_axes ([0.2, 0.55, 0.35, 0.35])`，其中给出的参数依次代表`left`，`bottom`，`width`，`height`，但它并不是坐标位置，而是`figure`的百分比，即左边从`figure`的20%位置开始，下边从55%位置开始，宽高是`figure`的35%。

9.1.8 利用Matplotlib绘制各种图形

经过前面章节的学习，读者可以说已经完成了Matplotlib中的内功修炼，只需要再学习些不同招式就可以绘制各种图形了。先来看一个气泡图绘制的例子。

```

>>> import pandas as pd
>>> midwest = pd.read_csv("../data/midwest_filter.csv")
>>> plt.rcParams['font.sans-serif'] = ['SimHei']
>>> fig = plt.figure(figsize=(14, 7), dpi=80,
    facecolor='w', edgecolor='k')
>>> plt.scatter('area', 'poptotal', data=midwest,
    s='dot_size', c='popdensity',
    cmap='Reds', edgecolors='black', linewidths=.5)
>>> plt.title("气泡图 PopTotal vs Area\n(颜色: 'popdensity' & \
    大小: 'dot_size')", fontsize=16)
>>> plt.xlabel('面积', fontsize=18)
>>> plt.ylabel('人口总数', fontsize=18)
>>> plt.colorbar()
>>> plt.show()

```

上面的大部分代码，读者朋友应该已经很熟悉了。需要说明的有3个地方，`plt.figure()` 函数中的两个新参数 `facecolor` 和 `edgecolor`，分别代表了背景色为白色，边框颜色为黑色。`plt.scatter()` 函数代表画散点图，同时它提供了更多参数。‘area’和‘poptotal’分别代表了X和Y的取值，而data则指定了对应数据集，所以前面的‘area’和‘poptotal’实际指的是midwest中的对应列。s代表散点大小由‘dot_size’列决定，c=‘popdensity’指定了散点颜色，cmap指定了调色板。最后，`plt.colorbar()` 函数会在图形右侧绘制色条。`plt.scatter()` 和 `plt.colorbar()` 函数的更多用法，读者可以查看它的帮助，这里不再赘述。通过上述代码将得到如图9.19所示的输出。

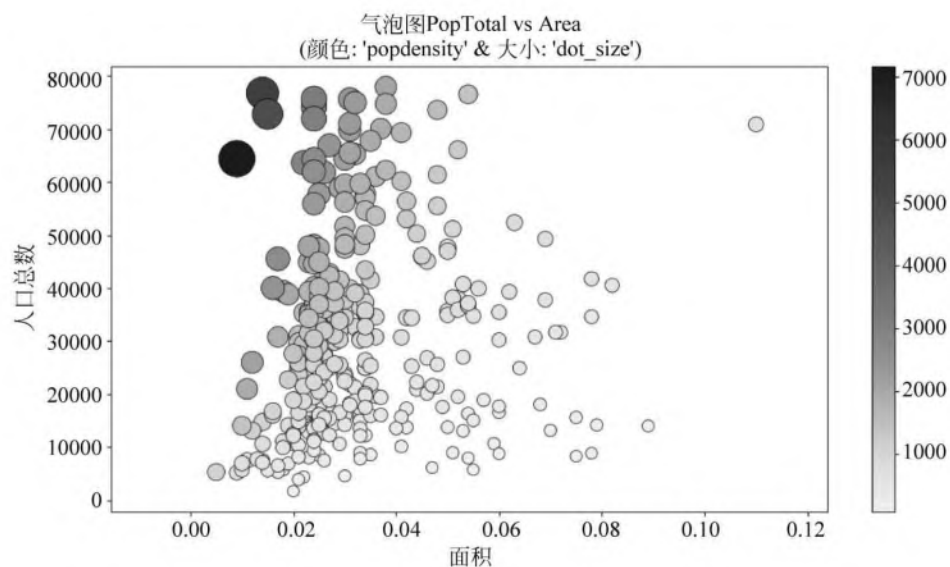


图9.19 气泡图

接下来再来看一个更复杂图形，图9.20所示的图形将各种图形的绘制整合到了一个图中。

图9.20将散点图、步进图、箱线图、直方图等整合在了一个图里面。下面一段一段地讲解代码。

```
>>> import pandas as pd
>>> fig = plt.figure(figsize=(10, 5))
>>> ax1 = plt.subplot2grid((2,4), (0,0))
>>> ax2 = plt.subplot2grid((2,4), (0,1))
>>> ax3 = plt.subplot2grid((2,4), (0,2))
>>> ax4 = plt.subplot2grid((2,4), (0,3))
>>> ax5 = plt.subplot2grid((2,4), (1,0), colspan=2)
>>> ax6 = plt.subplot2grid((2,4), (1,2))
>>> ax7 = plt.subplot2grid((2,4), (1,3))
# 绘图用的数据
n = np.array([0,1,2,3,4,5])
x = np.linspace(0,5,10)
xx = np.linspace(-0.75, 1., 100)
```

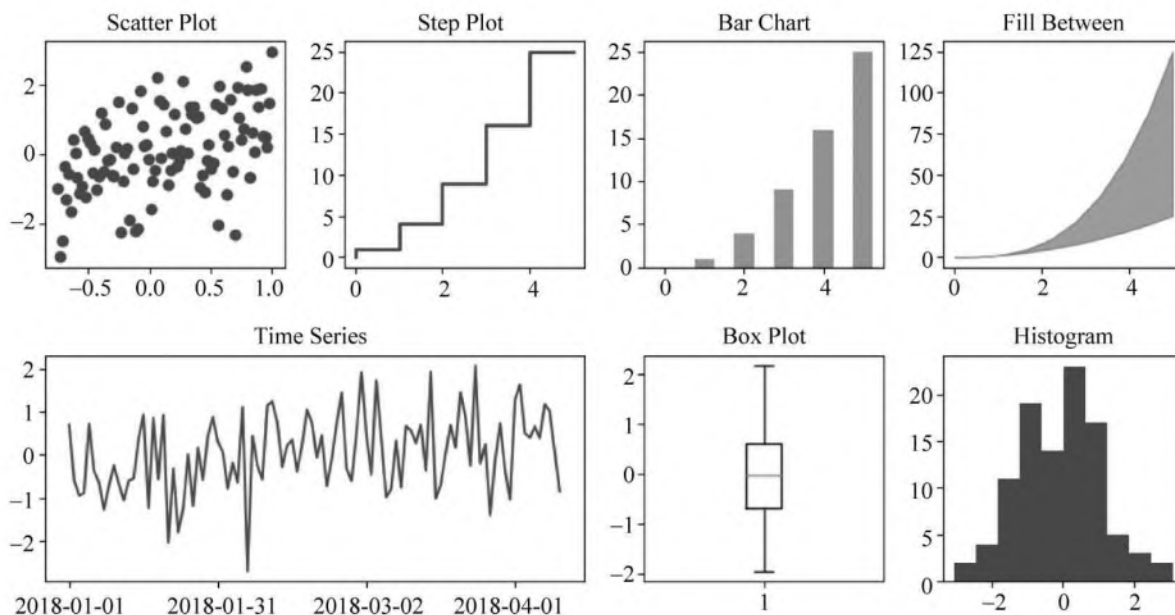


图9.20 各种Matplotlib图形

上述代码定义了图形的布局为2行4列，其中第5个图占据第2行的前2列。接下来的代码完成各图形绘制。

```
# 散点图
>>> ax1.scatter(xx, xx + np.random.randn(len(xx)))
>>> ax1.set_title("Scatter Plot")
# 步进图
ax2.step(n, n**2, lw=2)
ax2.set_title("Step Plot")
# 条形图
ax3.bar(n, n**2, align="center", width=0.5, alpha=0.5)
ax3.set_title("Bar Chart")
# 区域填充图
ax4.fill_between(x, x**2, x**3, color="steelblue", alpha=0.5);
ax4.set_title("Fill Between");
# 时间序列
dates = pd.date_range('2018-01-01', periods = len(xx))
ax5.plot(dates, xx + np.random.randn(len(xx)))
ax5.set_xticks(dates[::30])
ax5.set_xticklabels(dates.strftime('%Y-%m-%d')[::30])
ax5.set_title("Time Series")
# 箱线图
ax6.boxplot(np.random.randn(len(xx)))
ax6.set_title("Box Plot")
# 直方图
ax7.hist(xx + np.random.randn(len(xx)))
ax7.set_title("Histogram")
fig.tight_layout()
```

其中，`scatter()` 函数完成散点图绘制，`step()` 函数完成步进图绘制，`bar()` 函数完成条形图绘制，`fill_between()` 函数完成区域填充图绘制，`date_range()` 函数生成一个时间序列，更多关于时间序列的内容将在第20章进行更深入的讨论。`boxplot()` 函数完成箱线图绘制，`hist()` 函数完成直方图绘制。以上图形的定制都可以参考前面几小节内容完成，同时读者还可以查看帮助了解更多内容，这里就不再详细讨论。

最后用图9.21这样一个具有双Y轴的图形来结束Matplotlib绘图这一节，代码如下。

```
# Import Data
>>> plt.rcParams['font.sans-serif'] = ['SimHei']
>>> df = pd.read_csv('../data/economics.csv')
>>> x = df['date']; y1 = df['psavert']; y2 = df['unemploy']
>>> # 左边 Y 轴对应曲线
>>> fig, ax1 = plt.subplots(1,1,figsize=(16,7), dpi=80)
>>> ax1.plot(x, y1, color='tab:red')
>>> # 右边 Y 轴对应曲线
>>> ax2 = ax1.twinx() # 新建 axes, 共享同一 X 轴
>>> ax2.plot(x, y2, color='tab:blue')
>>> ax1.set_xlabel('日期', fontsize=16)
>>> ax1.set_ylabel('储蓄率', color='tab:red', fontsize=16)
>>> ax1.tick_params(axis='y', rotation=0, labelcolor='tab:red')
>>> # ax2 (right Y axis)
>>> ax2.set_ylabel("失业人数(1000)", color='tab:blue', fontsize=16)
>>> ax2.tick_params(axis='y', labelcolor='tab:blue')
>>> ax2.set_title("储蓄率 vs 失业率", fontsize=16)
>>> ax2.set_xticks(np.arange(0, len(x), 60))
>>> ax2.set_xticklabels(x[::60], rotation=90,
>>>                      fontdict={'fontsize':10})
>>> plt.show()
```



图9.21 双Y轴示例

上述代码中最关键的一行代码就是`ax2=ax1.twinx()`，这段代码基于`ax1`新建了一个`axes`对象`ax2`，该对象与`ax1`共享X轴，因此就可以得到图9.21所示的双Y轴图形。

9.2 Pandas绘图

由于大量的数据分析都是基于Pandas完成的，而在分析过程中数据分析人员经常会快速对数据进行可视化，此时的目标不是图形美观，而是迅速发现数据中的模式和规律。因此，为了实现这一目标，Pandas中为`DataFrame`集成了`plot()`函数，该函数对`plt.plot()`函数进行了封装以完成快速绘图。本节将学习如何直接使用`DataFrame`中的`plot()`函数，以及如何整合Matplotlib和Pandas中提供的绘图功能。

9.2.1 Pandas基础绘图

`DataFrame`中的`plot()`函数是为了快速探索数据而产生，因此绘图方法中只提供了除默认线图之外的少数绘图方法。这些方法可以作

为plot（）函数的kind关键字参数提供。具体包括：

- bar或barh：条形图或水平条形图；
- hist：直方图；
- boxplot：箱线图，也称为盒子图、盒型图；
- area：区域填充图，也称为面积图；
- scatter：散点图。

下面向读者展示如何绘制这些图形。首先，如果要绘制如图9.22所示的条形图，那么需要如下代码。

```
>>> df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d'])
>>> df.plot(kind='bar')
```

其中，第一行代码新建了一个10行4列的DataFrame，kind='bar'代表绘制条形图。通过指定参数方式和使用df.plot.bar（）函数的效果是完全等价的，读者可以自行尝试一下使用df.plot.bar（）进行绘图。如果想绘制的是如图9.23所示的堆叠条形图，只需要修改一个参数stacked=True就可以了。

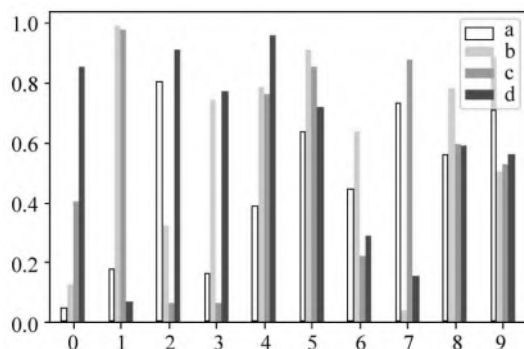


图9.22 条形图

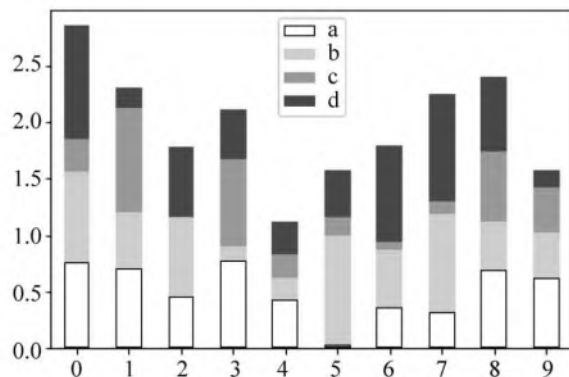


图9.23 堆叠条形图

如果要绘制水平条形图，那么使用如下代码可以输出图9.24所示的图形。

```
>>> df.plot(kind = 'barh', stacked = True)
```

但是如果数据有多列时，使用如下代码绘制直方图，将得到图9.25所示图形，而不是期望的每列一个直方图。

```
>>> df = pd.DataFrame({'a':np.random.randn(1000) + 1,
                        'b':np.random.randn(1000)},columns = ['a', 'b'])
>>> df.plot(kind = 'hist') # df.plot.hist(bins = 20)
```

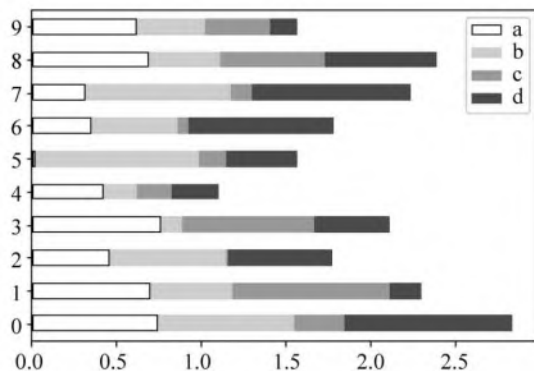


图9.24 水平条形图

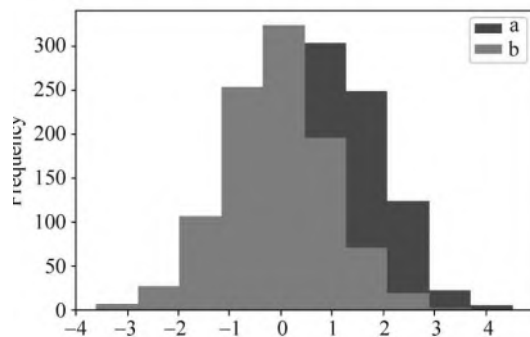


图9.25 重叠的直方图

此时应该改为使用`df.hist()`函数，才能针对每列数据绘制单独的直方图，如图9.26所示。

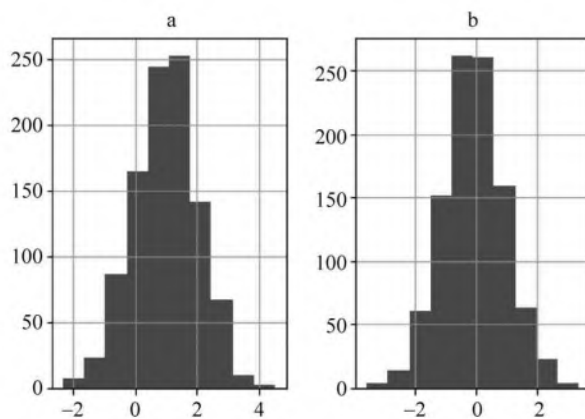


图9.26 针对每列绘制直方图

接下来绘制如图9.27所示的箱线图，代码如下。

```
>>> df = pd.DataFrame(np.random.rand(10, 5),
                        columns = ['A', 'B', 'C', 'D', 'E'])
>>> df.plot(kind = 'box') # df.plot.box()
```

绘制如图9.28所示的面积填充图，则可以使用如下代码。

```
>>> df = pd.DataFrame(np.random.rand(10, 4),
                        columns = ['a', 'b', 'c', 'd'])
>>> df.plot(kind = 'area') # df.plot.area()
```

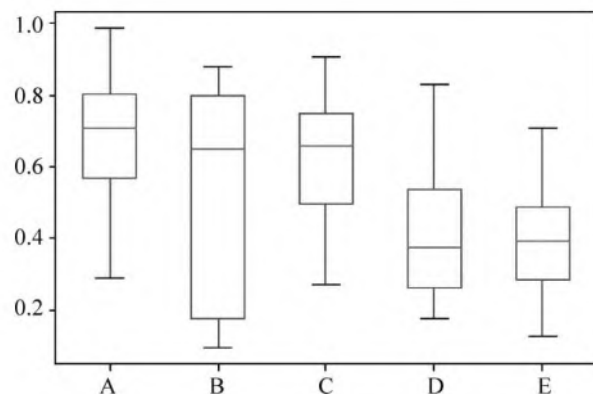


图9.27 箱线图

最后再来看一下如图9.29所示的饼图，不过一般情况数据可视化中饼图是被错误使用最多的图形。读者在进行可视化时，如果要绘制饼图，最好思考一下是否可以用条形图代替。

```
>>> df = pd.DataFrame(100 * np.random.rand(5),
                        index=['a', 'b', 'c', 'd', 'e'], columns=['sales'])
>>> df.plot(kind='pie', subplots=True) # df.plot.pie(subplots=True)
```

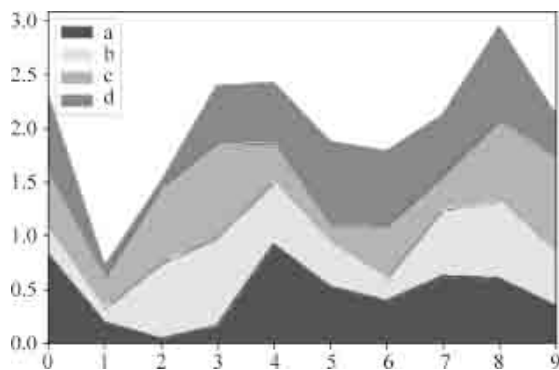


图9.28 面积填充图

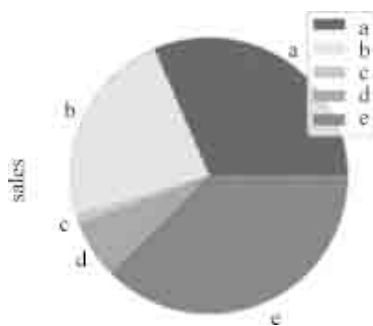


图9.29 饼图

9.2.2 整合Pandas绘图与Matplotlib绘图

将Pandas的快速绘图与Matplotlib的精细化图形设置结合起来，可以快速完成更加吸引人的图形。例如，现在有如下数据需要分析，如图9.30所示。

```
>>> df = pd.read_excel('../data/sample-salesv3.xlsx')
>>> df.head()
```

首先需要对数据进行分组统计，按每客户统计总购买额和购买数量，取前十大客户，代码如下。

```
>>> top_10 = (df.groupby('name')['ext price', 'quantity'].\
               agg({'ext price': 'sum', 'quantity': 'count'})\
               .sort_values(by = 'ext price', ascending = False)[:10].\
               reset_index())
>>> top_10.rename(columns = {'name': 'Name',
                             'ext price': 'Sales', 'quantity': 'Purchases'}, inplace = True)
```

	account number	name	sku	quantity	unit price	ext price	date
0	740150	Barton LLC	B1-20000	39	86.69	3380.91	2014-01-01 07:21:51
1	714466	Trantow-Barrows	S2-77896	-1	63.16	-63.16	2014-01-01 10:00:47
2	218895	Kulas Inc	B1-69924	23	90.70	2086.10	2014-01-01 13:24:58
3	307599	Kassulke, Ondricka and Metz	S1-65481	41	21.05	863.05	2014-01-01 15:05:22
4	412290	Jerde-Hilpert	S2-34077	6	83.21	499.26	2014-01-01 23:26:55

图9.30 需要分析的数据

有了数据，下一步要考虑的就是选择什么图形来可视化，这里推荐读者阅读网页<https://www.perceptualedge.com/blog/?p=2080>。该网站给出了进行数据可视化时的图形选择方法，如图9.31所示。图形选择的第一步明确可视化的目的是要展示分布还是对比，又或者是相互关系和组合，之后再根据不同目标来选择对应图形。

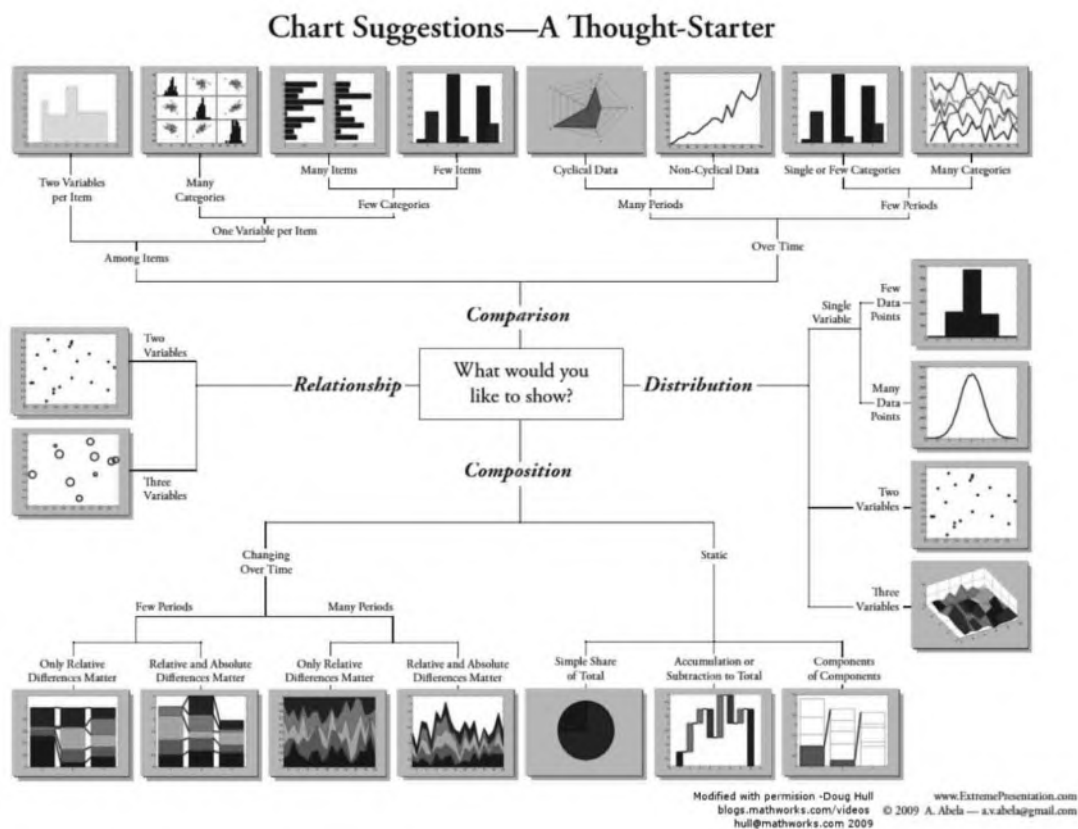


图9.31 图形选择

此时本例的目标是展示前十大客户的销售额，这是多个不同类型数值的对比，因此采用水平条形图，可视化如图9.32所示，代码如下。

```
>>> plt.style.use('fivethirtyeight')
>>> plt.rcParams['font.sans-serif'] = ['SimHei']
>>> fig, ax = plt.subplots(figsize=(5, 6))
>>> top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
>>> ax.set_xlim([-10000, 140000])
>>> ax.set(title='2014 年销售额', xlabel='销售额', ylabel='客户')
>>> ax.legend().set_visible(False)
>>> plt.show()
```

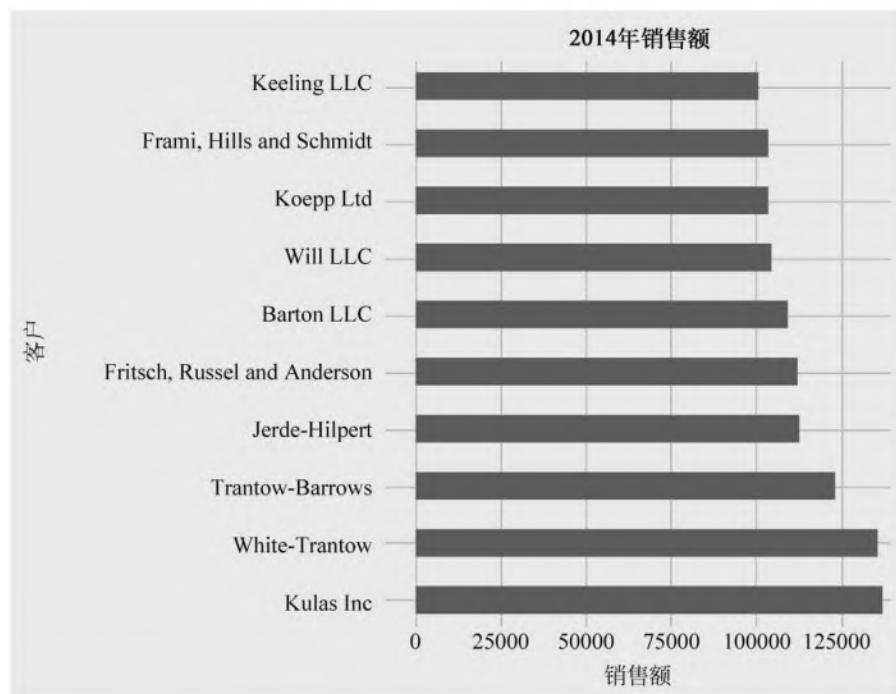


图9.32 十大客户

上述代码首先进行绘图样式选择，这里采用了作者个人比较喜欢的fivethirtyeight样式，因为中文显示问题，需要单独设置字体。可以说上面的图形已经基本达到了可视化的目标，但是它还不够完美，如X轴以美元显示，可以进一步简化为常用的以千美元 / 百万美元来计数的方式，这时需要用到前面学习过的FuncFormatter。此外，对于前十中的某些客户，有些是新客户需要在图形中标记出来，同时能标记前十客户的平均销售额，那么可以采用如下代码。

```

>>> def currency(x, pos):
    if x >= 1000000:
        return '$ {:.1f}M'.format(x * 1e-6)
    return '$ {:.1f}K'.format(x * 1e-3)
>>> fig, ax = plt.subplots()
>>> top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
>>> avg = top_10['Sales'].mean()
>>> ax.set_xlim([-10000, 140000])
>>> ax.set(title='2014 年销售额', xlabel='销售额', ylabel='客户')
>>> ax.axvline(x=avg, color='b', label='Average',
    linestyle='--', linewidth=1)
>>> for cust in [3, 5, 8]:
    ax.text(115000, cust, "新发展客户")
>>> formatter = FuncFormatter(currency)
>>> ax.xaxis.set_major_formatter(formatter)
>>> ax.legend().set_visible(False)

```

上述代码中currency（）函数定义了X轴的变换方式，100万美元以下以K记，大于100万用M来表示，之后将FuncFormatter（currency）定义的formatter应用到X轴就完成了转换。而ax.axvline（）完成竖直的虚线绘制，ax.text（）在指定位置添加标注，最终得到如图9.33所示的图形。



图9.33 前十客户销售额

在探索性分析的时候，数据可视化的目标是高效发现数据模式和规律，而数据分析最终的可视化是为了向读者展示结果，此时的图形

需要不断重构、美化，从客户的角度来考虑，希望读者能细心体会这一区别。

9.3 Seaborn

Matplotlib是Python中最常见的绘图包，强大之处不言而喻。然而由于Matplotlib比较底层，想要绘制漂亮的图非常麻烦，需要写大量的代码。为了解决这一问题，Michael Waskom开发了Seaborn，如果Matplotlib是力图让简单的事情简单，困难的事情成为可能，那么Seaborn就是让一系列定义好的困难的事情也变得简单。Seaborn是在Matplotlib基础上进行了高级API封装，图表装饰更加容易，使用者可以用更少的代码作出更美观的图。与Matplotlib相比，Seaborn更好地提供了：

- 计算多变量间关系的面向数据集接口；
- 可视化类别变量的观测与统计；
- 可视化单变量或多变量分布并与其子数据集比较；
- 控制线性回归的不同因变量并进行参数估计与作图；
- 对复杂数据进行易行的整体结构可视化；
- 对多表统计图的制作高度抽象并简化可视化过程；
- 提供多个内建主题渲染Matplotlib的图像样式；
- 提供调色板工具生动再现数据；
- 同时支持矩阵数据和DataFrame。

而以上这些正是探索性数据分析所需要的。

9.3.1 Seaborn中的样式

Seaborn装载了一些默认主题风格，主题风格的选择通过`sns.set()`方法实现。`sns.set()`可以设置5种风格的图表背景：

darkgrid、whitegrid、dark、white、ticks，通过参数style设置，默认情况下为darkgrid风格。下面以图9.34所示数据为例进行具体讲解。

```
>>> import pandas as pd
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset('tips')
>>> tips.head()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

图9.34 示例数据

假如要可视化total_bill与tip的关系，同时考虑性别的影响，那么可以用如下代码。

```
>>> sns.set() # 默认为 darkgrid 风格
>>> sns.relplot(x='total_bill', y='tip', col='sex',
                hue='smoker', size='size', data=tips)
```

上述代码将得到图9.35所示的输出，从图形可以看出Seaborn默认对整个图形样式（字体、大小、颜色、坐标轴等）进行了设置，这里使用的是darkgrid风格。relplot（）函数画的是散点图，col参数表明需要按不同性别进行可视化，hue参数代表根据是否吸烟来着色，size参数根据吃饭人数来决定散点大小。

同样的绘图代码，如果需要使用另一样式，通过代码sns.set（style='white'）将得到如图9.36所示的可视化图。Seaborn在后台已经默默帮我们完成了底层的一切调整。

同时还可以修改调色板，如`sns.set (palette='Paired')`改为对比色，将得到如图9.37所示的图形。

感兴趣的读者还可以试试其他的样式，这里不再一一举例。

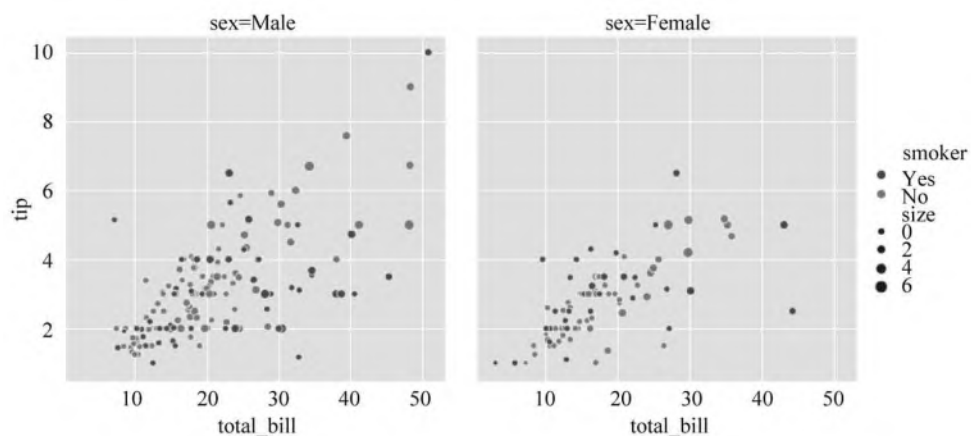


图9.35 账单与消费关系 (darkgrid)

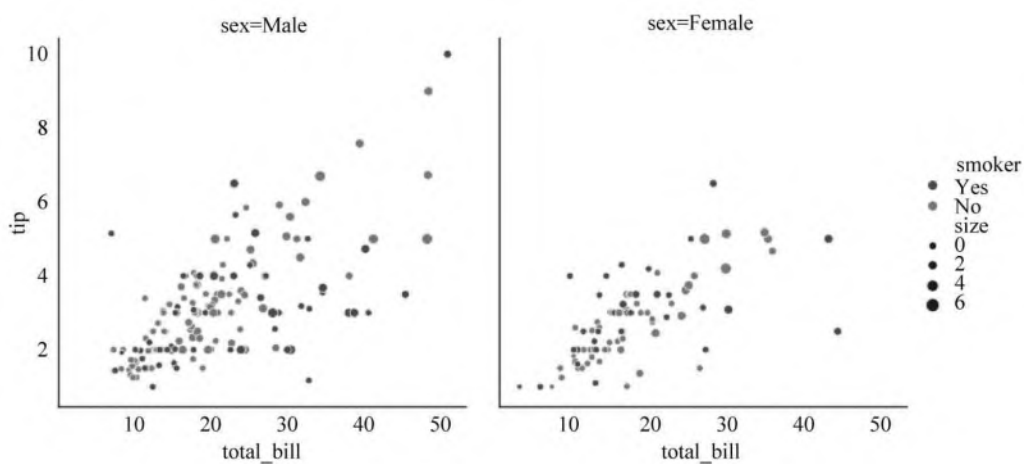


图9.36 账单与消费关系 (white)

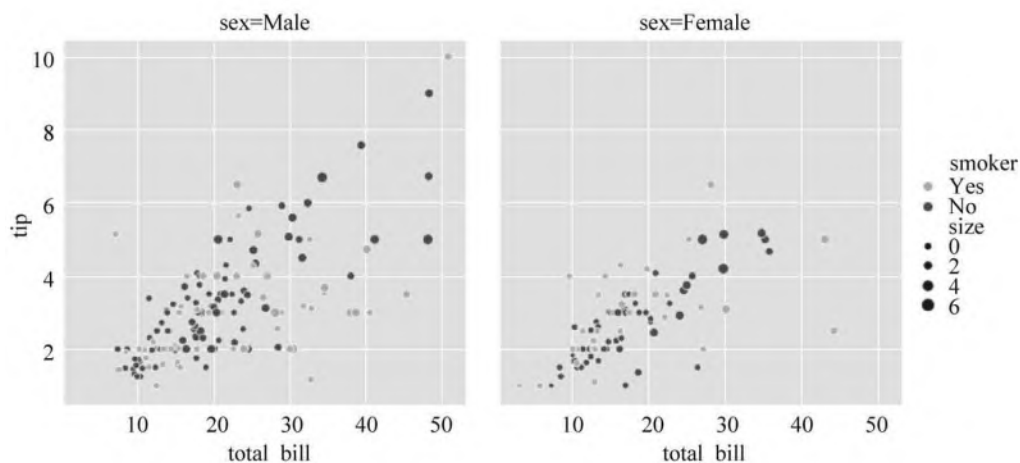


图9.37 账单与消费关系（对比色）

9.3.2 Seaborn绘制统计图形

Seaborn另一强大的功能就是统计图形的绘制，下面以任天堂的精灵宝可梦游戏数据为例进行说明，如图9.38所示。

```
>>> df = pd.read_csv('../data/Pokemon.csv', index_col = 0)
>>> df.head()
```

#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False

图9.38 游戏数据示例

上面的数据给出了各代宝可梦精灵的属性、攻击、防守能力等。首先来查看一下各种精灵的某个属性分布情况，此时可以通过分布图，使用如下代码来完成。

```
>>> sns.distplot(df['Attack'])
```

上述代码将输出如图9.39所示的叠加了概率密度曲线的直方图。

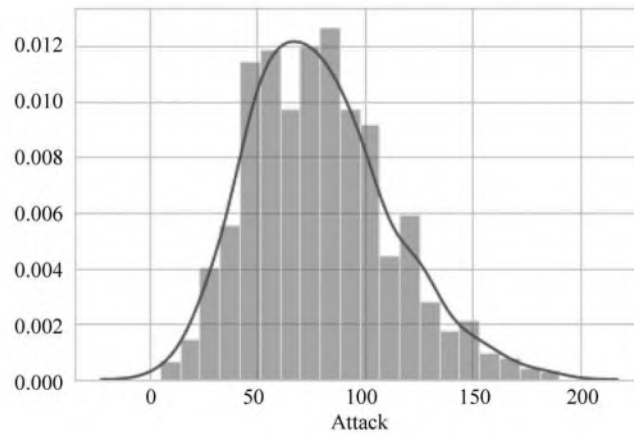


图9.39 叠加概率密度曲线的直方图

如果要可视化精灵某两种能力间的关系，如攻击力与防守力，可以使用如下代码。

```
# 推荐方式
>>> sns.lmplot(x='Attack', y='Defense', data=df)
# 另一种
# sns.lmplot(x=df.Attack, y=df.Defense)
```

此时将得到如图9.40所示的图形，`lmplot()` 函数不仅绘制散点图，同时提供了回归线，以及置信区间。

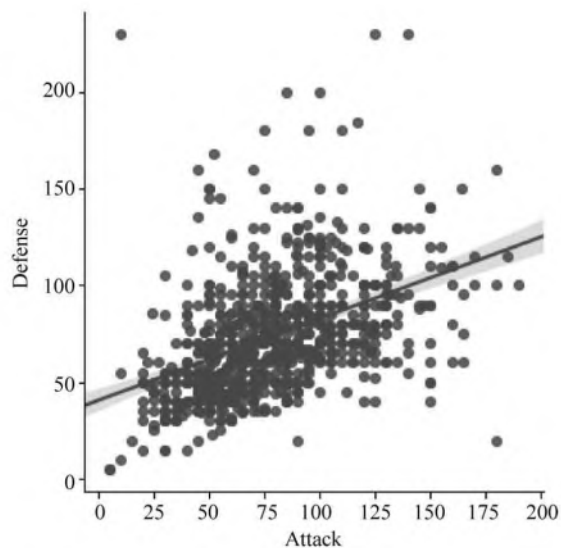


图9.40 攻击力与防守力

如果想按第几代精灵来进行分析，那么可以使用如下代码。

```
>>> sns.lmplot(x = 'Attack', y = 'Defense', data = df,
               fit_reg = False, # 不画回归线
               hue = 'Generation')
```

通过hue参数，数据可视化时可以选择各代精灵采用不同颜色区别，同时fit_reg参数选择不画回归线，因为数据中有6代精灵，而多条回归线会让图形混乱，最终图形如图9.41所示。

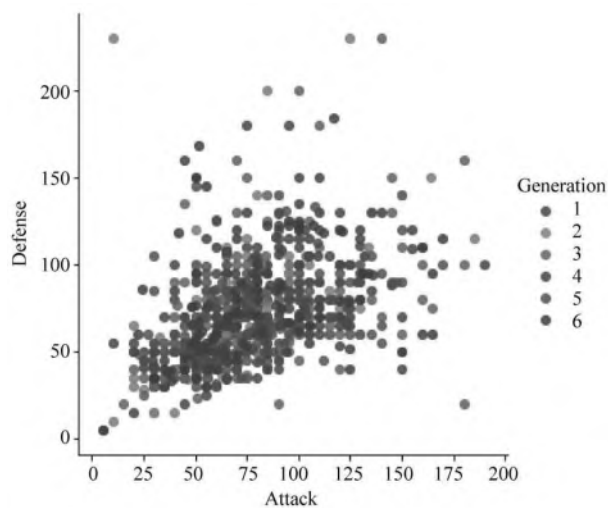


图9.41 不同代精灵的防守力与攻击力

当然，Seaborn也可以与Matplotlib结合。例如，如果修改上面的图形坐标轴范围，可以加上下面的两行代码。

```
>>> plt.ylim(0, None)
>>> plt.xlim(0, 160)
```

不过有的时候不只是想查看精灵的攻击力与防守力间的关系，分析人员还想同时了解攻击力与防守力各自的分布情况，此时就需要用到联合分布图，利用如下代码就可以得到二者的联合分布图，如图9.42所示。

```
>>> sns.jointplot(x='Attack', y='Defense', data=df)
```

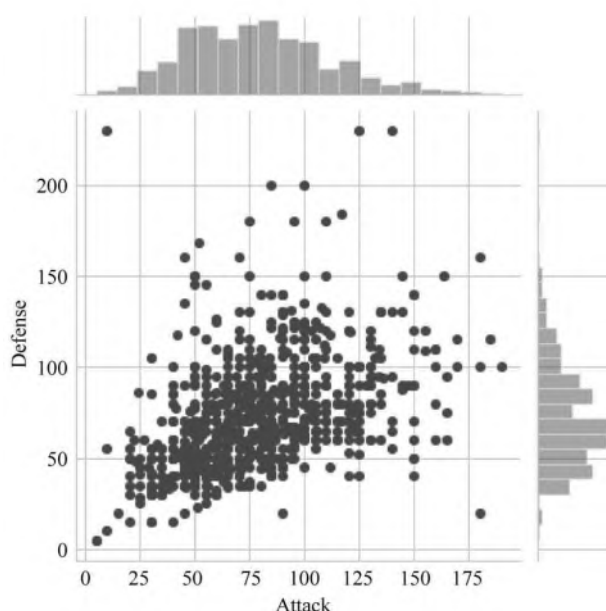


图9.42 联合分布图

如果想查看精灵的攻防、速度等多个属性的分布，联合分布图就不是最佳方案，那么此时可以使用箱线图。只需要在boxplot（）函数中传入对应的DataFrame就可以完成绘图，如下代码将输出图9.43所示的图形。

```
>>> sns.boxplot(data = df)
```

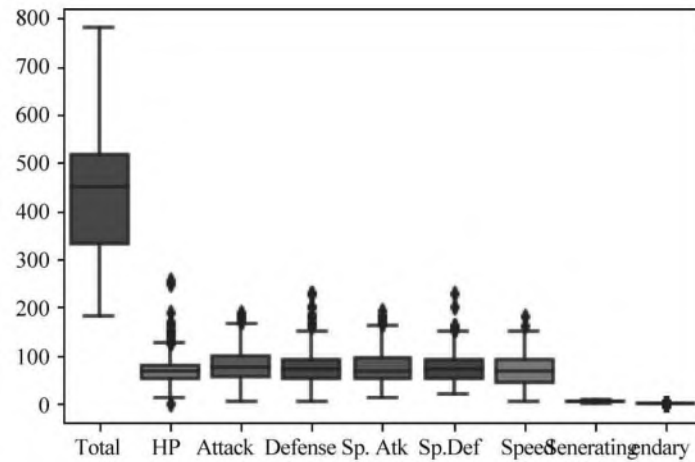


图9.43 查看多个属性分布的箱线图

boxplot() 函数绘图时将自动选择传入数据中的数值列进行箱线图绘制，而对于非数值列则自动忽略。读者应该很容易就注意到上面的图形中X轴标记文本存在重叠，那么此时可以选择调整图形大小，或者旋转X轴标记文本的方法来解决该问题，这里将问题留给读者进行处理。除了使用箱线图，还可以使用violinplot() 函数，这种图形是箱线图的改进，称为小提琴图。除了提供箱线图的信息，它还提供了数据分布信息。例如，如果想查看Type 1类型精灵的各项属性分布，则可以用如下代码，最终得到图9.44。

```
>>> plt.figure(figsize = (12,8))
>>> sns.set_style('whitegrid')
>>> sns.violinplot(x = 'Type 1', y = 'Attack', data = df)
```

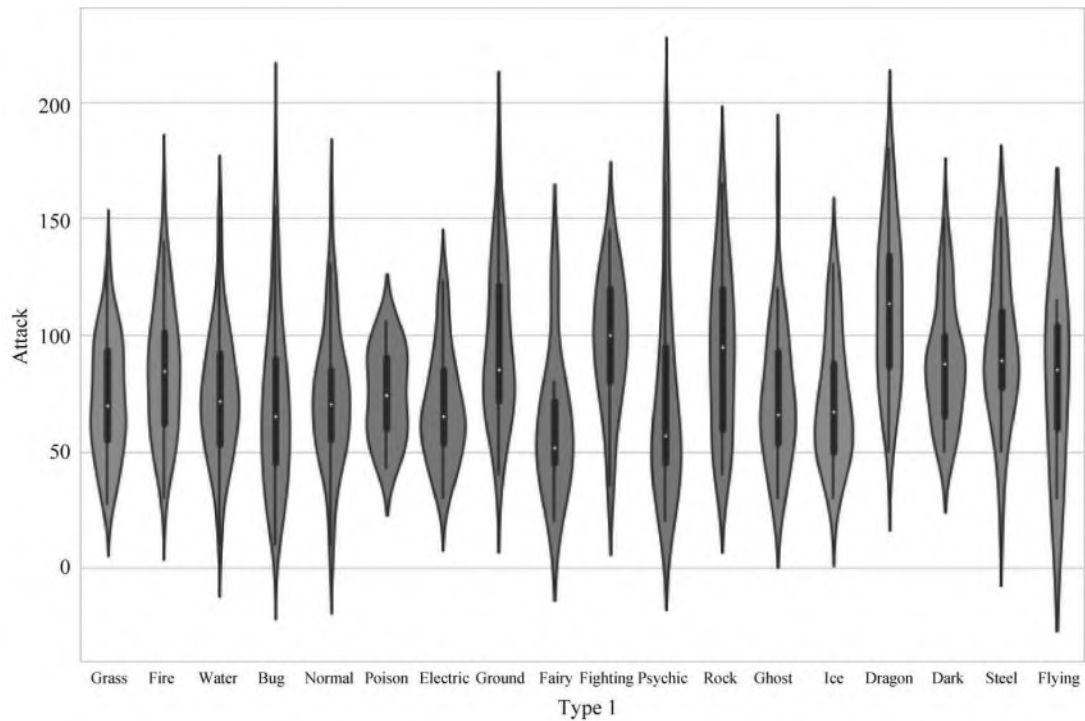


图9.44 小提琴图

对应Seaborn中的绘图函数，使用时还可以指定调色板参数。例如，如下代码将输出图9.45。

```
>>> pkmn_type_colors = ['#78C850', # Grass
                        '#F08030', # Fire
                        '#6890F0', # Water
                        '#A8B820', # Bug
                        '#A8A878', # Normal
                        '#A040A0', # Poison
                        '#F8D030', # Electric
                        '#E0C068', # Ground
                        '#EE99AC', # Fairy
                        '#C03028', # Fighting
                        '#F85888', # Psychic
                        '#B8A038', # Rock
                        '#705898', # Ghost
                        '#98D8D8', # Ice
                        '#7038F8', # Dragon]

>>> plt.figure(figsize=(12,8))
>>> sns.violinplot(x='Type 1', y='Attack',
                  data=df, palette=pkmn_type_colors)
```

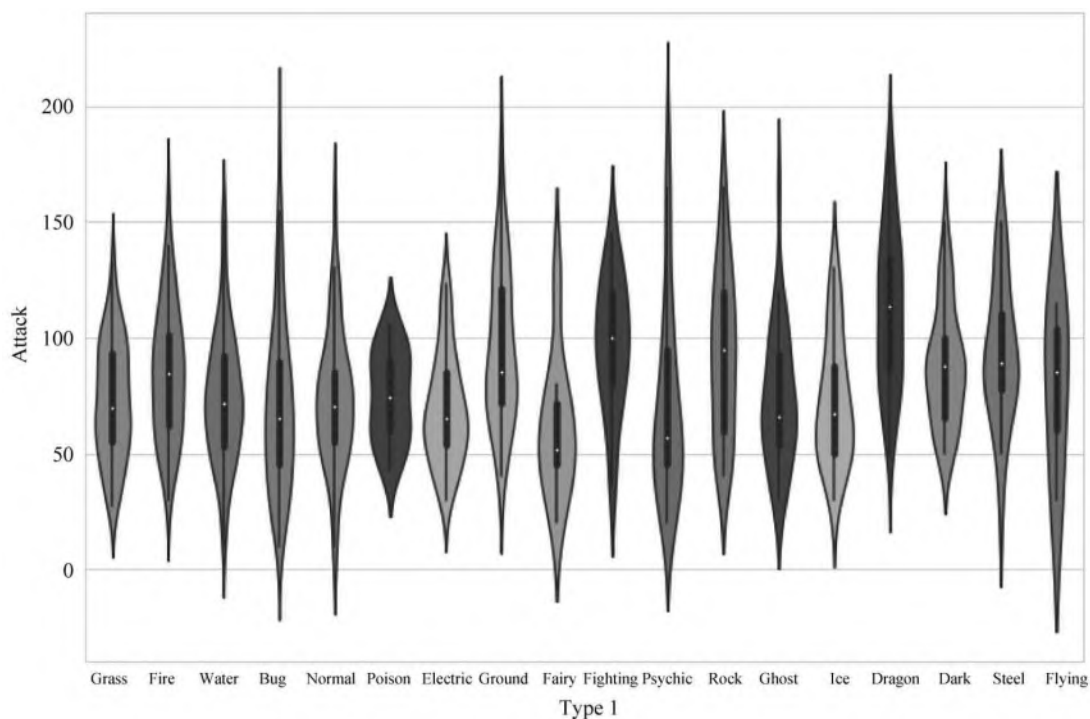



图9.45 使用调色板

Seaborn除了支持上述两种可视化数据分布的图形外，还提供了对 `swarmplot()` 函数的支持，用户还可以将 `violinplot()` 与 `swarmplot()` 函数结合。例如，运行如下代码，得到图9.46和图9.47。

```
#代码段 1
>>> plt.figure(figsize=(12,8))
>>> sns.swarmplot(x='Type 1', y='Attack',
                 data=df,palette=pkmn_type_colors)
#代码段 2
>>> plt.figure(figsize=(12,9))
>>> sns.violinplot(x='Type 1',
                  y='Attack',
                  data=df,
                  inner=None,          # 去掉内部条形
                  palette=pkmn_type_colors)
>>> sns.swarmplot(x='Type 1',
                  y='Attack',
                  data=df,
                  color='k',          # 黑色
                  alpha=0.7)         # 透明
>>> plt.title('Attack by Type')
```

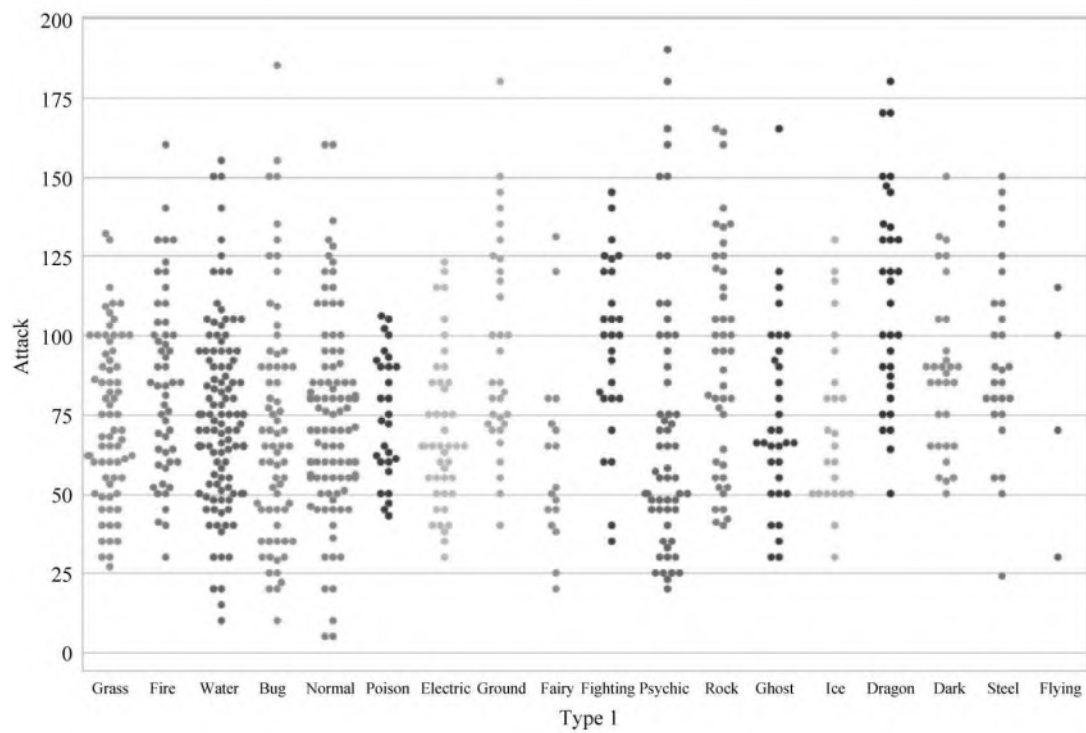


图9.46 swarmplot

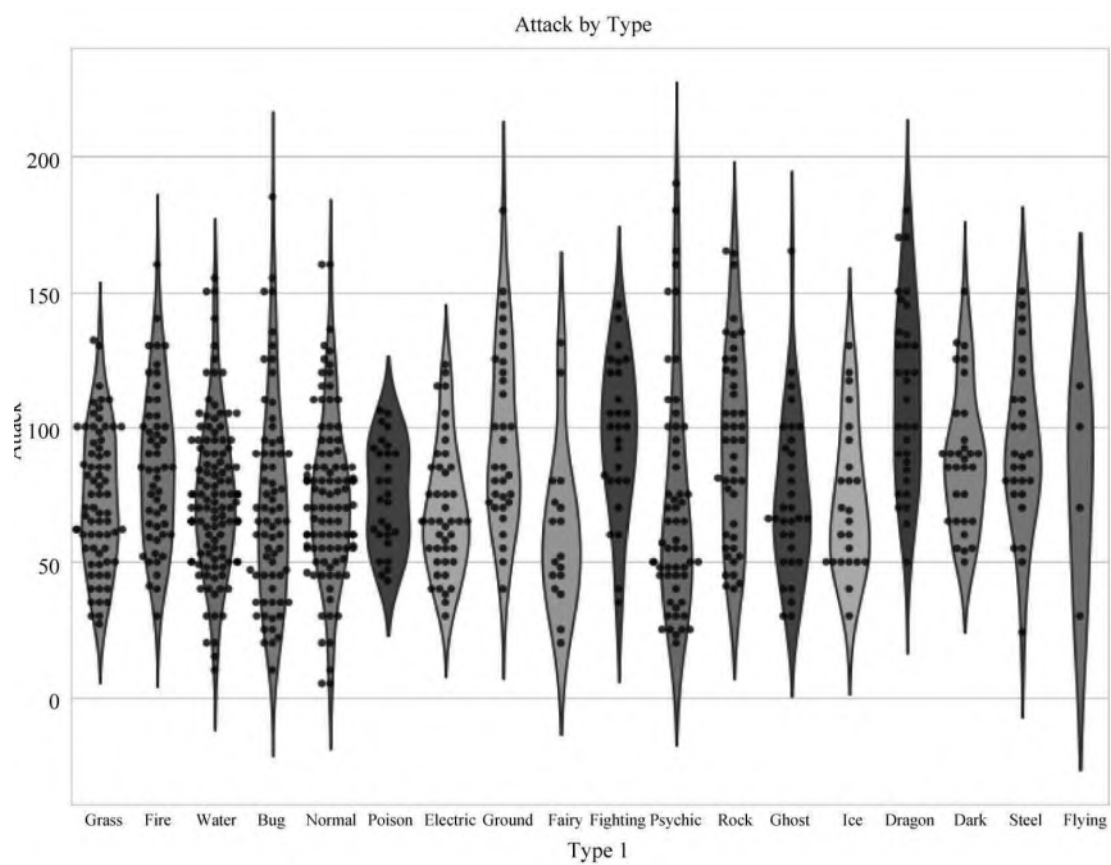


图9.47 swarmplot叠加violinplot

代码段2是在同一个figure中绘制两个图形，因此得到图9.47，需要通过在swarmplot（）函数中指定alpha参数来增加透明度，使得图形不会互相遮挡。掌握了上面的图形后，如果想绘制图9.48所示图形，应该怎么办？

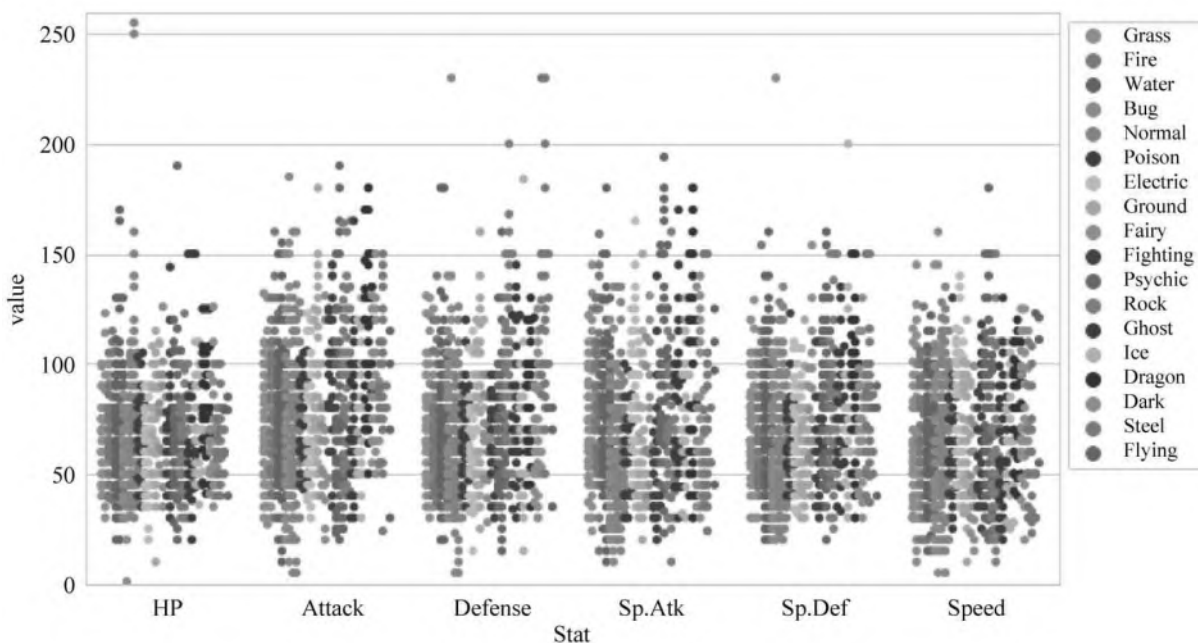


图9.48 读者练习

这里不给出代码，希望读者能自行练习。简单提示如下：上面图形是swarmplot，使用了参数hue，它由‘Type 1’决定，同时每种类型的分布图是并列的，因此dodge参数应该为True；而plt.legend（bbox_to_anchor=（1，1），loc=2）可以将图例放到右边。能够表达数据间关系的除了上述图形，还有一种应用广泛的图形——热力图，例如图9.49很好地将精灵的各项属性间的关系以图形的方式展示了出来，颜色越淡代表了相关性越强，如果读者习惯颜色深代表剪关性强，那么可以通过修改cmap参数来完成，如cmap=‘rocket_r’。

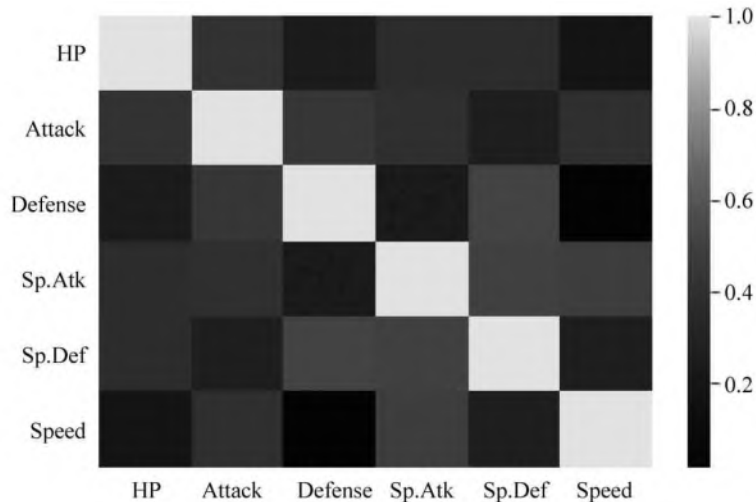


图9.49 各属性间的相关性

实现上面图形的代码也非常简单，具体如下。

```
>>> stats_df = df.drop(['Total', 'Generation', 'Legendary'], axis = 1)
>>> corr = stats_df.corr()
>>> sns.heatmap(corr)
```

在Seaborn中探索数据时，常用到的一种方法就是在数据集的不同子集上绘制同一图的多个实例。这种技术有时被称为“facet”或“grid”绘图。通过这种方式，某种意义上将可视化的维度提高到了三维或四维，因此通过这一可视化方式，数据分析人员可以快速提取复杂数据的大量信息。例如，图9.50能让数据分析人员一眼就看出各代精灵的攻防关系，图9.51则可以让数据分析人员迅速发现是否吸烟、就餐时间是否对就餐人数与总账单间的关系有影响。

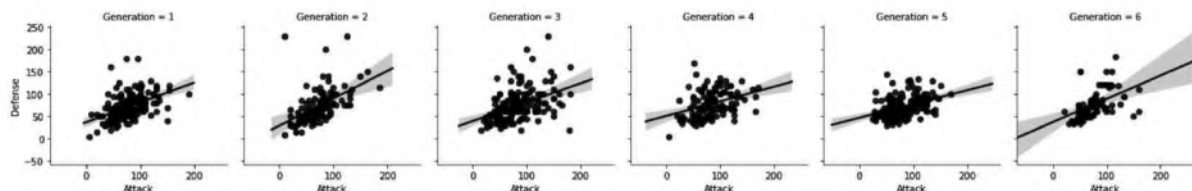


图9.50 各代精灵的攻防对比

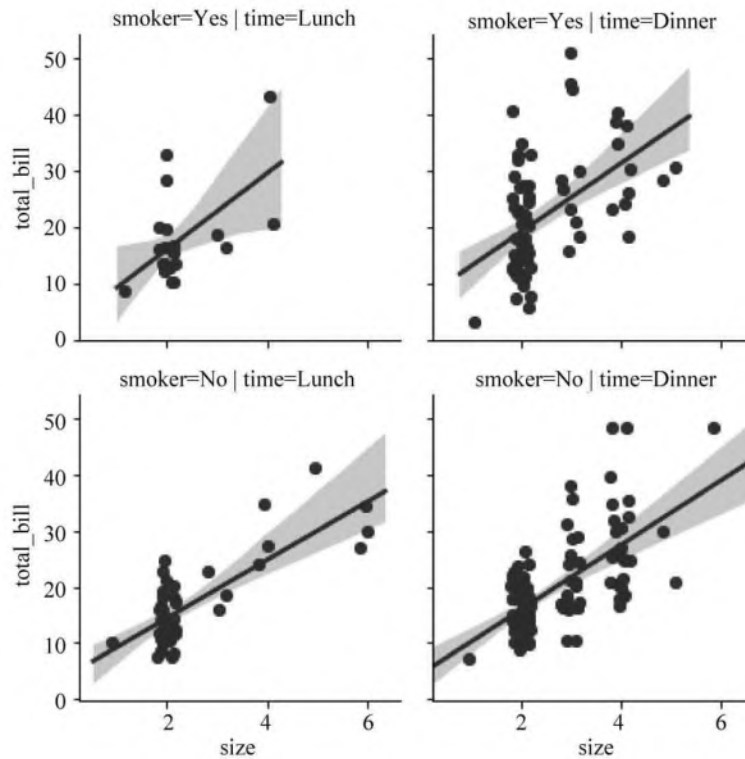


图9.51 就餐人数与账单关系

要实现这样的可视化也非常简单，只需要运行如下代码。

```
# 图 9.50
>>> g = sns.FacetGrid(df, col = "Generation", # row
                      margin_titles = False)
>>> g.map(sns.regplot, "Attack", "Defense", color = "black",
          fit_reg = True, x_jitter = .2);
# 图 9.51
>>> g = sns.FacetGrid(tips, row = "smoker", col = "time",
                      margin_titles = False)
>>> g.map(sns.regplot, "size", "total_bill", color = "blue",
          fit_reg = True, x_jitter = .2);
```

这里 Seaborn 参考了 R 语言中 ggplot 的思想，首先通过 FacetGrid() 函数来指明想要可视化的数据，以何种方式进行布局。这个布局通过参数 row 和 col 来指定，row 给出了按照行来进行分类的变量，col 给出了按照列进行分类的变量，之后 sns.FacetGrid() 函数会返回一个新的 FacetGrid 对象，该对象包含了定义的布局。完成了绘

图布局的定义，接下来需要将数据映射到这个布局，也就是绘制何种图形。以图9.50对应代码为例，可视化的目标是想针对不同Generation分列来观察df数据，因此会得到一个如图9.52所示的1行6列布局对象。上述代码通过`g.map()`函数告诉Seaborn可视化的数据是Attack和Defense这两列，用`regplot`来展示它们之间的关系。而图9.51与图9.50的区别在于不仅按照“smoker”来分行，还按照“time”来分列。

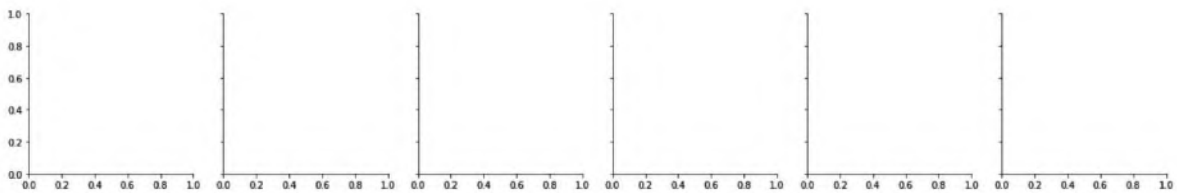


图9.52 FacetGrid

9.4 可视化进阶

9.4.1 其他可视化工具

模式、趋势和相关性在文字叙述性的数据中不容易被察觉，但在可视化图表上却是一目了然。利用前面几节介绍的可视化工具，数据分析人员可以很好地完成这一功能，但有的情况下需要对数据进行动态展示。此时就需要交互式可视化，交互式可视化能够让决策者深入了解细节层次，更深刻地理解趋势变化。这种展示方式的改变，使得用户可以查看分析背后的事实。近年来，不少基于Python的交互数据可视化工具如雨后春笋般出现，典型的有Plotly和Bokeh，以及后起之秀Altair。图9.53、图9.54和图9.55分别给出了一些使用Plotly、Bokeh和Altair绘制的图形，由于篇幅限制，本书不会对其进行介绍，建议读者查看下面的网站进行了解：

- Plotly: <https://plot.ly/javascript/>;
- Bokeh: <http://bokeh.pydata.org/en/latest/>;
- Altair: <https://altair-viz.github.io/>.

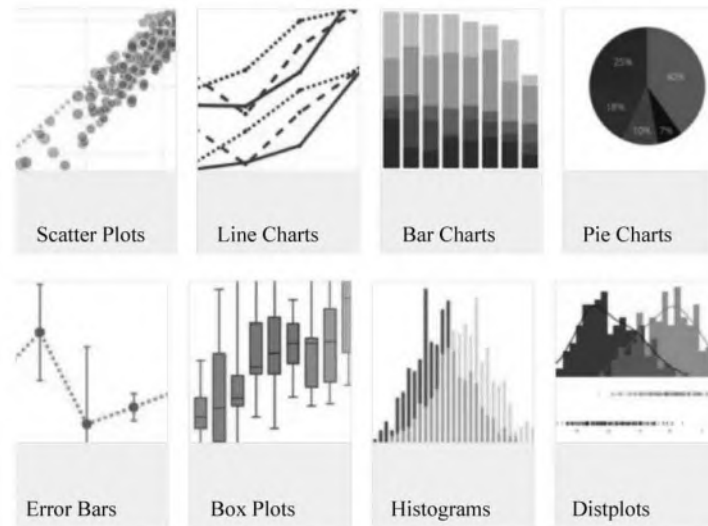


图9.53 Plotly

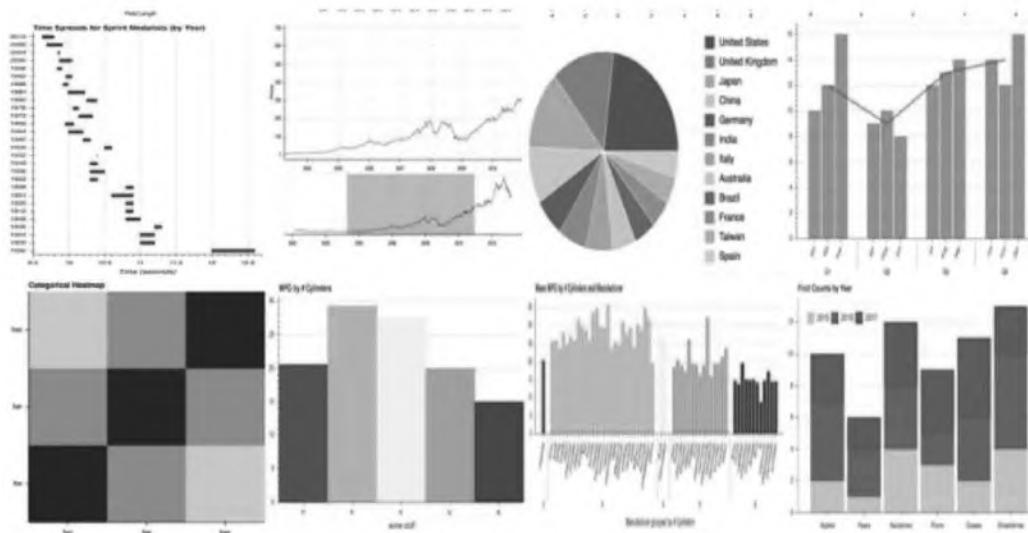


图9.54 Bokeh

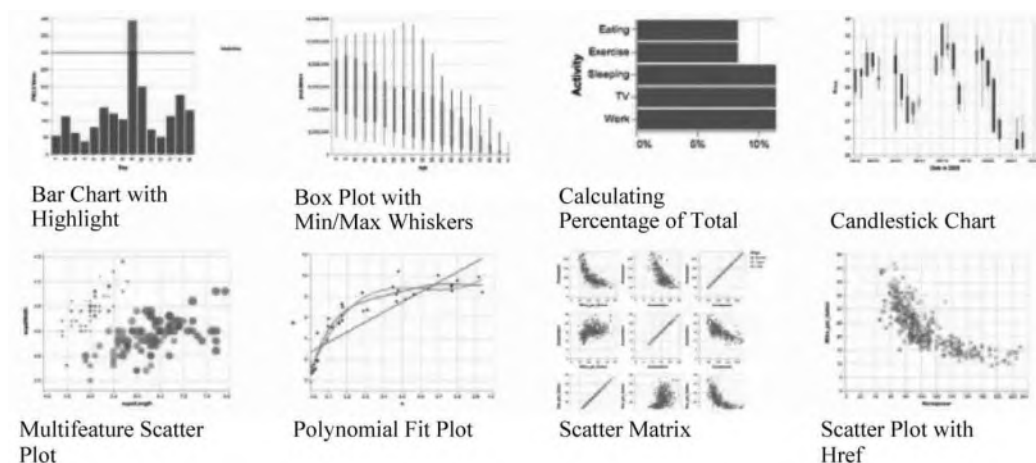


图9.55 Altair

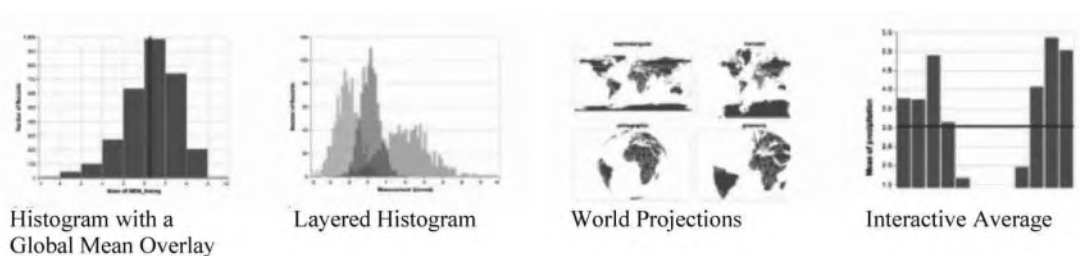


图9.55 （续）

9.4.2 推荐读物

到这里，读者朋友可能已经有能力完成各种高难度的数据可视化了，然而这些都是“术”，可视化的“道”在于知道如何用图形来表达信息、传递观点。因此，要做好图形可视化，这里强烈建议大家阅读以下几本经典书籍。

- Edward Tufte. *The Visual Display of Quantitative Information* [M]. CT, USA: Graphics Pr, 2001.
- Julie Steele. 数据可视化之美[M]. 祝洪凯，李妹芳，译. 北京：机械工业出版社，2011.
- Nathan Yau. 鲜活的数据：数据可视化指南[M]. 向怡宁，译. 北京：人民邮电出版社，2012.

- Nathan Yau. *Data Points : Visualization That Means Something* [M]. NJ, USA: Wiley, 2013.

[\(1\)](#) 本书中部分代码运行结果为输出含有彩色信息的图像，相关插图仅显示灰度信息，谨供读者对照实验结果。读者可在本书配套电子资源中查看彩色图像。

第10章 探索性数据分析——某电商销售数据分析

生命的全部的意义在于无穷地探索尚未知道的东西。

——左拉

通过前面章节的学习，读者已经了解Pandas中的基础数据结构DataFrame和Series，学会如何筛选数据，进行数据清洗与整理，如何进行数据的分组统计，如何进行数据可视化等操作。本章将对某电商的销售数据进行探索性数据分析，以此为例来对前面学习到的知识进行一个总结。本章使用的数据来自英国某在线零售商，该网站主要销售各类礼品，它的顾客大部分为本地或国际批发商。这里要分析的数据是2010年12月到2011年12月的销售数据。

10.1 数据清洗

10.1.1 分析准备

开始分析之前，首先导入需要的库，分析中将使用到Pandas和NumPy，利用Matplotlib和Seaborn进行数据可视化。同时，待分析的数据中包含了时间数据，所以还需要导入datetime库。

```
>>> import pandas as pd
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> import datetime
>>> %matplotlib inline
```

之后需要进行一些基础设置，如Seaborn使用的样式、中文字体，绘图时使用到的调色板等，具体代码如下。

```
>>> import warnings
>>> warnings.filterwarnings('ignore')           # 忽略警告
>>> sns.set_style('whitegrid')
>>> plt.rcParams['font.sans-serif'] = ['SimHei']   # 用来正常显示中文标签
>>> color = sns.color_palette()
```

接下来读入待分析的数据，如图10.1所示。

```
>>> online = pd.read_csv("../data/Online_Retail.csv")
>>> online.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010/12/1 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010/12/1 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010/12/1 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010/12/1 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010/12/1 8:26	3.39	17850.0	United Kingdom

图10.1 读入某电商销售数据

从前5行数据看出，该数据集中包含了购物客户ID、客户来自哪个国家、所购物品的描述、数量、单价以及发票日期、号码等信息。

10.1.2 了解数据

开始数据分析前的第一步就是对待分析的数据有一个大致了解，包括数据类型、数据大小、是否有空值、数据大致分布等。下面首先来看看数据的基本信息，代码如下。

```
>>> online.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
InvoiceNo      541909 non-null object
StockCode      541909 non-null object
Description    540455 non-null object
Quantity       541909 non-null int64
InvoiceDate    541909 non-null object
UnitPrice      541909 non-null float64
CustomerID     406829 non-null float64
Country        541909 non-null object
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

从代码输出可以看出数据有541909行，8列，其中2列为浮点型数据，1列为整型，还有5列是字符串，整个数据使用了33.1MB内存。接下来再看看数据中是否存在缺失值，代码如下。

```
>>> online.isnull().sum().sort_values(ascending=False)
CustomerID      135080
Description      1454
Country           0
UnitPrice        0
InvoiceDate      0
Quantity         0
StockCode        0
InvoiceNo        0
dtype: int64
```

输出结果说明CustomerID列存在大量的缺失数据，几乎25%的CustomerID信息都没有。显然，CustomerID对要进行的数据分析而言是很关键的信息，如果是真实的数据分析，那么需要去探究到底什么原因导致信息的缺失，有无办法补全。此外，Description列也存在一定的数据缺失，但是它的比例不到0.3%，所以基本可以忽略。接下来可以选择看看数据缺失的列到底是哪些，代码如下。

```
>>> online[online.isnull().any(axis=1)].head()
```

运行结果如图10.2所示。

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
622	536414	22139	NaN	56	2010/12/1 11:52	0.00	NaN United Kingdom
1443	536544	21773	DECORATIVE ROSE BATHROOM BOTTLE	1	2010/12/1 14:32	2.51	NaN United Kingdom
1444	536544	21774	DECORATIVE CATS BATHROOM BOTTLE	2	2010/12/1 14:32	2.51	NaN United Kingdom
1445	536544	21786	POLKADOT RAIN HAT	4	2010/12/1 14:32	0.85	NaN United Kingdom
1446	536544	21787	RAIN PONCHO RETROSPOT	2	2010/12/1 14:32	1.66	NaN United Kingdom

图10.2 CustomerID列数据缺失

由于这里选择了不对缺失数据进行补全，因此直接将这部分数据丢弃，代码如下。

```
>>> df_new = online.dropna()
>>> df_new.isnull().sum().sort_values(ascending = False)
Country      0
CustomerID   0
UnitPrice    0
InvoiceDate  0
Quantity     0
Description  0
StockCode    0
InvoiceNo    0
dtype: int64
```

从代码的输出可以看到目前已经没有缺失数据了。其实除了丢弃数据，也可以采用另一种办法，如给所有缺失的CustomerID都赋一个特殊值。在后续的分析中看这些CustomerID对应的交易有没有什么特殊规律，有兴趣的读者可以自行尝试。对数据的行、列有了基本了解后，下一步就可以尝试看看数据的大致分布是什么样的，这可以通过描述性统计完成，代码如下。

```
>>> df_new.describe().round(2)
```

结果如图10.3所示。从describe()函数的输出来看，客户购买礼品中75%的价格在3.75美元以下，购物数量少于12。不过购物数量数据中很明显存在一个异常值，即最大购买数量为80995，最小购买数量

为-80995。由于购买数量为负代表退货，这里最大值和最小值一样，那么很可能是某一用户错误购买后又立刻退货了。通过如下代码可以对此信息进行确认，结果如图10.4和图10.5所示。

	Quantity	UnitPrice	CustomerID
count	406829.00	406829.00	406829.00
mean	12.06	3.46	15287.69
std	248.69	69.32	1713.60
min	-80995.00	0.00	12346.00
25%	2.00	1.25	13953.00
50%	5.00	1.95	15152.00
75%	12.00	3.75	16791.00
max	80995.00	38970.00	18287.00

图10.3 数据分布情况

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
540421	581483	23843	paper craft , little birdie	80995	2011-12-09 09:15:00	2.08	16446.0 United Kingdom

图10.4 信息确认（80995）

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
540422	C581484	23843	paper craft , little birdie	-80995	2011-12-09 09:27:00	2.08	16446.0 United Kingdom

图10.5 信息确认（-80995）

```
>>> df_new[df_new['Quantity'] == 80995]
>>> df_new[df_new['Quantity'] == -80995]
```

对于此问题，下一节进行数据清洗的时候将进行处理。

10.2 数据清洗与整理

10.2.1 数据类型转换与错误数据删除

现在继续上一节内容，将这一组错误下单后又立刻取消的订单删除，具体代码如下。

```
>>> df_new.drop([540421, 540422], axis = 0, inplace = True)
```

在前一节中提到了数据集中有5列为字符串，实际上读者只要仔细思考一下就会发现这有问题，因为InvoiceDate应该是日期数据。所以需要对该列数据类型进行转换，这里使用Pandas中提供的to_datetime()函数来完成变换。此外，Description列中字符串均为大写，为方便阅读，也统一将其转换为小写，代码如下。

```
>>> df_new['InvoiceDate'] = pd.to_datetime(df_new['InvoiceDate'],
      format = '%Y/%m/%d %H: %M')
>>> df_new['Description'] = df_new['Description'].str.lower()
>>> df_new.head()
```

转换结果如图10.6所示。

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	white hanging heart t-light holder	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	white metal lantern	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	cream cupid hearts coat hanger	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	knitted union flag hot water bottle	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	red woolly hottie white heart.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

图10.6 字符串转换为小写

其实如果数据分析前知道某列是日期类型数据，那么也可以在数据读入的时候就指定参数parse_dates=['InvoiceDate']，代码如下。

```
>>> online = pd.read_csv("../data/Online_Retail.csv",
      parse_dates = ['InvoiceDate'])
```

此外，CustomerID列在数据集中显示为浮点数，实际上它应该是整数类型，因此也需要对其进行类型转换，代码如下。

```
>>> df_new['CustomerID'] = df_new['CustomerID'].astype('int64')
```

完成了上述修改后，再次运行如下代码可以发现InvoiceDate列的数据类型确实已经改变。

```
>>> df_new.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 406827 entries, 0 to 541908
Data columns (total 8 columns):
InvoiceNo      406827 non-null object
StockCode      406827 non-null object
Description    406827 non-null object
Quantity       406827 non-null int64
InvoiceDate    406827 non-null datetime64[ns]
UnitPrice      406827 non-null float64
CustomerID     406827 non-null int64
Country        406827 non-null object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 27.9 + MB
```

10.2.2 添加新数据

在分析销售数据时，除了单价、购物数量以外，消费总金额也是很重要的信息，但是原数据集中却没有提供，因此需要自行计算得到，代码如下。

```
>>> df_new['AmountSpent'] = df_new['Quantity'] * df_new['UnitPrice']
>>> df_new = df_new[['InvoiceNo', 'InvoiceDate', 'StockCode',
                    'Description', 'Quantity', 'UnitPrice', 'AmountSpent',
                    'CustomerID', 'Country']]
>>> df_new.head()
```

计算结果如图10.7所示。

	InvoiceNo	InvoiceDate	StockCode	Description	Quantity	UnitPrice	AmountSpent	CustomerID	Country
0	536365	2010-12-01 08:26:00	85123A	white hanging heart t-light holder	6	2.55	15.30	17850	United Kingdom
1	536365	2010-12-01 08:26:00	71053	white metal lantern	6	3.39	20.34	17850	United Kingdom
2	536365	2010-12-01 08:26:00	84406B	cream cupid hearts coat hanger	8	2.75	22.00	17850	United Kingdom
3	536365	2010-12-01 08:26:00	84029G	knitted union flag hot water bottle	6	3.39	20.34	17850	United Kingdom
4	536365	2010-12-01 08:26:00	84029E	red woolly hottie white heart.	6	3.39	20.34	17850	United Kingdom

图10.7 计算消费总金额

上述代码通过 UnitPrice 与 Quantity 计算得到了新的列 AmountSpent。之后对 df_new 中列的顺序进行了重排。在分析的时候除了需要总价，分析人员可能还会通过顾客购物的日期以及具体时间来判断订单产生的趋势等信息，因此需要从 InvoiceDate 中提取上述信息，创建新的列，具体代码如下。

```
>>> df_new.insert(loc = 2, column = 'YearMonth',
                  value = df_new['InvoiceDate'].map(lambda x:
                  100 * x.year + x.month))
>>> df_new.insert(loc = 3, column = 'month',
                  value = df_new['InvoiceDate'].dt.month)
# Monday = 1.....Sunday = 7
>>> df_new.insert(loc = 4, column = 'day',
                  value = (df_new['InvoiceDate'].dt.dayofweek) + 1)
>>> df_new.insert(loc = 5, column = 'hour',
                  value = df_new['InvoiceDate'].dt.hour)
>>> df_new.head()
```

运行结果如图10.8所示。

	InvoiceNo	InvoiceDate	YearMonth	month	day	hour	StockCode	Description	Quantity	UnitPrice	AmountSpent	CustomerID	Country
0	536365	2010-12-01 08:26:00	201012	12	3	8	85123A	white hanging heart t-light holder	6	2.55	15.30	17850	United Kingdom
1	536365	2010-12-01 08:26:00	201012	12	3	8	71053	white metal lantern	6	3.39	20.34	17850	United Kingdom
2	536365	2010-12-01 08:26:00	201012	12	3	8	84406B	cream cupid hearts coat hanger	8	2.75	22.00	17850	United Kingdom
3	536365	2010-12-01 08:26:00	201012	12	3	8	84029G	knitted union flag hot water bottle	6	3.39	20.34	17850	United Kingdom
4	536365	2010-12-01 08:26:00	201012	12	3	8	84029E	red woolly hottie white heart.	6	3.39	20.34	17850	United Kingdom

图10.8 提取具体时间信息

10.3 探索性数据分析

10.3.1 客户分析

由于该电商面对的是批发客户，所以在数据分析时想了解每个客户的订单数是多少，订单量最大的5个客户是谁，每个客户消费金额是多少，消费最多的5个客户是谁。下面首先来看看5个客户的订单数，代码如下。

```
>>> df_new.groupby(by = ['CustomerID', 'Country'],
                    as_index = False)['InvoiceNo'].count().head()
```

	CustomerID	Country	InvoiceNo
0	12346	United Kingdom	2
1	12347	Iceland	182
2	12348	Finland	31
3	12349	Italy	73
4	12350	Norway	17

图10.9 5个客户的订单数

结果如图10.9所示。利用groupby（）函数可以按照CustomerID和Country对数据分组后统计对应的InvoiceNo数目，这样就得到了每个国家的客户的订单数目。运行如下代码，将得到如图10.10所示的客户订单数图。

```
>>> orders = df_new.groupby(by = ['CustomerID', 'Country'],
                            as_index = False)['InvoiceNo'].count()
>>> plt.subplots(figsize = (15,6))
>>> plt.plot(orders['CustomerID'], orders['InvoiceNo'])
>>> plt.xlabel('客户 ID')
>>> plt.ylabel('订单数')
>>> plt.title('客户订单数')
>>> plt.show()
```

从图10.10看出该电商的订单贡献主要来自少数的几个大客户，因此，了解头部客户有助于更好地服务这些大客户。下面的代码先利用

sort_values（）函数对订单数进行排序，之后利用head（）函数对前五大客户进行查看，如果数据分析时想查看更多客户，则可以在head（）函数中输入具体值来完成，结果如图10.11所示。

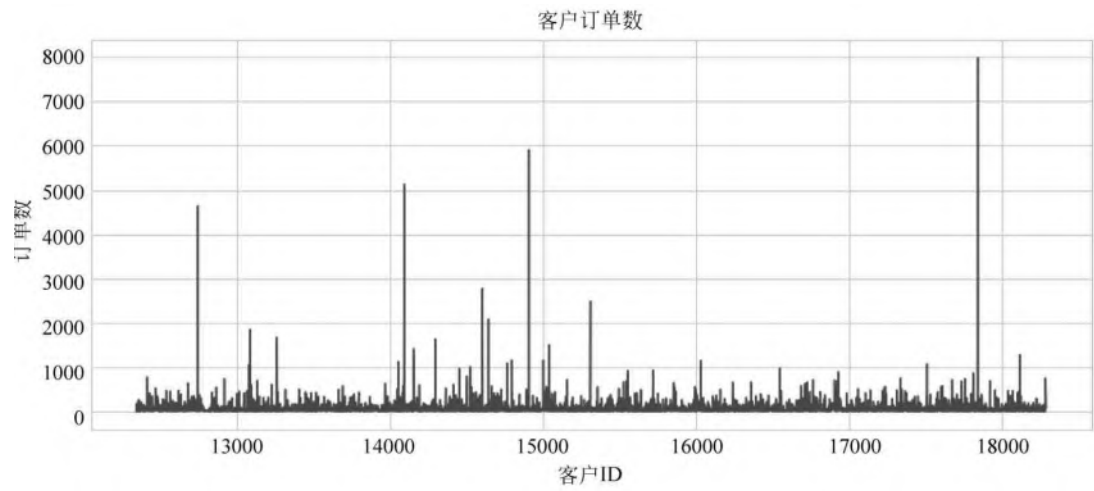


图10.10 客户订单数

```
>>> orders.sort_values(by = 'InvoiceNo', ascending = False).head()
```

	CustomerID	Country	InvoiceNo
4050	17841	United Kingdom	7983
1903	14911	EIRE	5903
1308	14096	United Kingdom	5128
338	12748	United Kingdom	4642
1682	14606	United Kingdom	2782

图10.11 查看前五大客户（订单数）

从前五大客户来看，订单主要来自英国，这个结论也和分析的数据是英国某电商销售数据一致。除了了解客户订单，客户消费的金额也是数据分析时需要关注的，利用如下代码可以完成每客户消费金额分析，如图10.12所示。

```
>>> money_spent = df_new.groupby(by = ['CustomerID', 'Country'],
    as_index = False)['AmountSpent'].sum()

>>> plt.subplots(figsize = (15,6))
>>> plt.plot(money_spent['CustomerID'], money_spent['AmountSpent'])
>>> plt.xlabel('客户 ID')
>>> plt.ylabel('消费金额 (美元)')
>>> plt.title('客户消费金额')
>>> plt.show()
```

客户消费金额也反映出了同样趋势，那么它会不会和客户订单是一致的呢？因此可以利用如不代码输出前五大消费客户来进行对比，结果如图10. 13所示。

```
>>> money_spent.sort_values(by = 'AmountSpent',
    ascending = False).head()
```

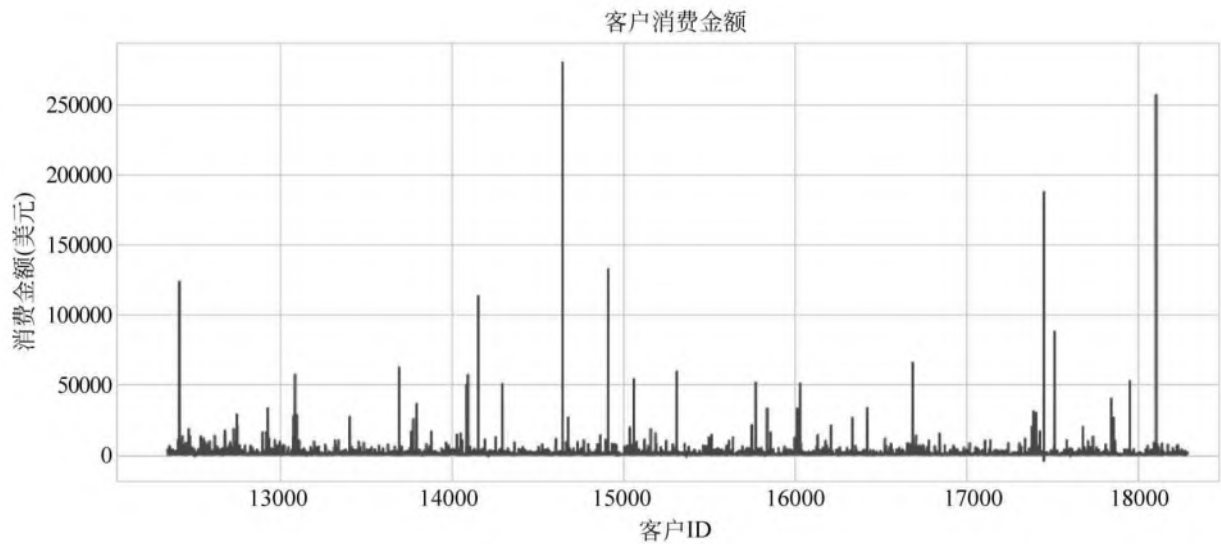


图10. 12 客户消费金额

	CustomerID	Country	AmountSpent
1711	14646	Netherlands	279489.02
4241	18102	United Kingdom	256438.49
3766	17450	United Kingdom	187482.17
1903	14911	EIRE	132572.62
57	12415	Australia	123725.45

图10.13 查看前五大客户（消费金额）

出人意料的是，没有一个消费金额前5名的客户与订单数前5名的客户一致。这是一个很有意思的现象，显然，数据分析人员可以进一步分析造成这一现象的原因。如果是一部分客户订单很多，但是都是买的相对低价的礼品，而另一部分客户订单相对较少而购买的是相对高价的礼品，是否公司可以进行有针对性的活动？该电商网站是低价产品利润高，还是高价产品利润高，据此又可以做什么？这些都是数据分析之后提出的新问题，带着这些问题，数据分析人员又将进行新的探索。

10.3.2 订单趋势分析

了解了客户后，接下来看看订单是否存在某种规律，如圣诞期间订单数是否激增。为了验证这一问题，让我们按月来统计订单，代码如下。

```
>>> ax = df_new.groupby('InvoiceNo')['YearMonth']. \
    unique().value_counts().sort_index(). \
    plot('bar', color = color[0], figsize = (15, 6))
>>> ax.set_xlabel('月', fontsize = 15)
>>> ax.set_ylabel('订单数', fontsize = 15)
>>> ax.set_title('每月订单数(2010.12 - 2011.12)', fontsize = 15)
>>> ax.set_xticklabels(('12/10', '1/11', '2/11', '3/11', '4/11',
    '5/11', '6/11', '7/11', '8/11', '9/11', '10/11', '11/11', '12/11'),
    rotation = 'horizontal', fontsize = 13)
>>> plt.show()
```

上述代码中，第一行代码使用了链式方法，首先利用groupby（）函数按照InvoiceNo分组，之后用unique（）函数选出不同的InvoiceNo，然后按照YearMonth汇总。之后对得到的数据进行绘图，绘图的时候先对index排序，绘制条形图，选用了调色板中的第一个颜色[运行代码sns.palplot（color）可以查看调色板]。这样就得到了

图10.14所示的每月订单数图。从图10.14中看出确实是在11月的时候订单相对较多，而在年初1—4月订单相对较少。

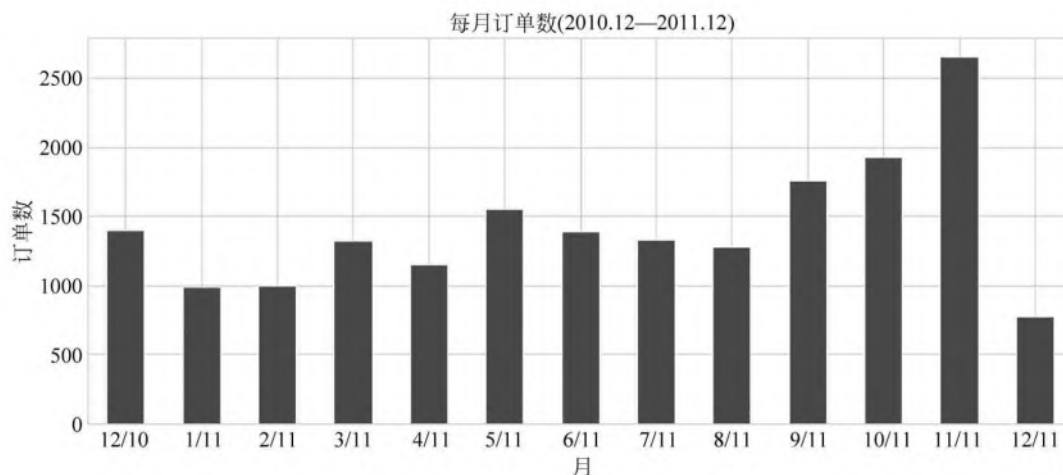


图10.14 每月订单数

按月考察订单存在不同，那么是否每天也有很大差别呢？可以用类似方法进行分析，代码如下。

```
>>> df_new.groupby('InvoiceNo')['day'].unique().\
      value_counts().sort_index()
[1]    2863
[2]    3185
[3]    3455
[4]    4033
[5]    2831
[7]    2169
Name: day, dtype: int64
```

细心的读者可能已经发现上述代码输出有一个有意思的现象，即似乎周六没有订单，原因不得而知，也许周六是休息日。不过还是让我们将上面的信息可视化吧，代码如下。

```

>>> ax = df_new.groupby('InvoiceNo')['day'].unique().\
    value_counts().sort_index().\
    plot('bar',color = color[0],figsize = (15,6))
>>> ax.set_xlabel('',fontsize = 15)
>>> ax.set_ylabel('订单数',fontsize = 15)
>>> ax.set_title('每天订单数',fontsize = 15)
>>> ax.set_xticklabels(('周一','周二','周三','周四','周五','周日'),
    rotation = 'horizontal', fontsize = 15)
plt.show()

```

从每天订单数（图10.15）看，似乎周四订单最多的，周日反而订单最少，这也许是由于该网站的客户主要是批发商，他们的员工周日会休息，所以很少在周末下订单。

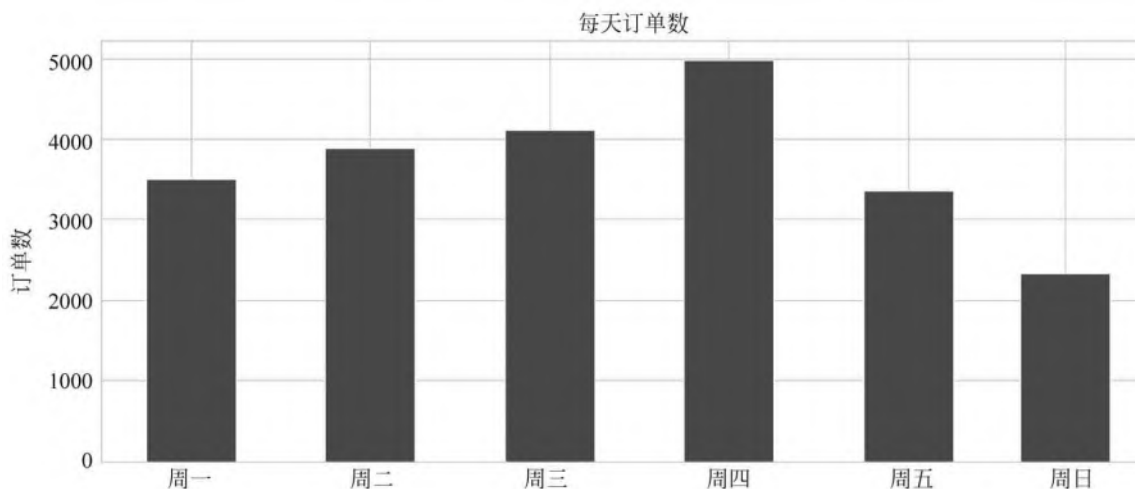


图10.15 每天订单数

通常对电商而言，响应速度是极其关键的指标，那么客户下单最多的时段是否需要加派人手？要回答这一问题，需要按照时间来统计订单数。图10.16即是不同时间的订单数示意图。

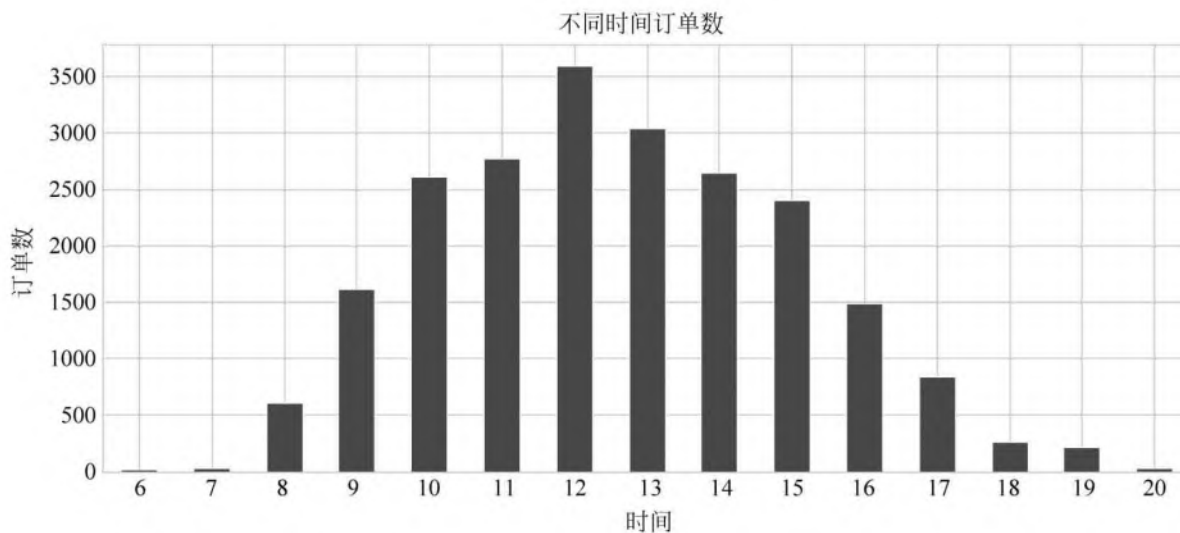


图10.16 不同时间订单数

从图10.16中可以看出订单数基本上符合正态分布，12点是订单最多的时刻。要得到上面的图形，需要运行如下代码。

```
>>> ax = df_new.groupby('InvoiceNo')['hour'].unique().\
    value_counts().iloc[: - 1].sort_index().\
    plot('bar', color = color[0], figsize = (15, 6))
>>> ax.set_xlabel('时间', fontsize = 15)
>>> ax.set_ylabel('订单数', fontsize = 15)
>>> ax.set_title('不同时间订单数', fontsize = 15)
>>> ax.set_xticklabels(range(6, 21), rotation = 'horizontal',
    fontsize = 15)
>>> plt.show()
```

10.3.3 客户国家分析

前面已经提到该电商的客户主要来自英国，那么让我们来看看具体的每个国家客户订单分布情况吧，运行如下代码。


```
>>> group_country_orders = df_new.groupby('Country')['InvoiceNo']. \
    count().sort_values()
>>> plt.subplots(figsize=(15,8))
>>> group_country_orders.plot('barh', fontsize=12, color=color[0])
>>> plt.xlabel('订单数', fontsize=12)
>>> plt.ylabel('国家', fontsize=12)
>>> plt.title('各国订单数', fontsize=12)
>>> plt.show()
```

得到如图10.17所示的各国订单数。

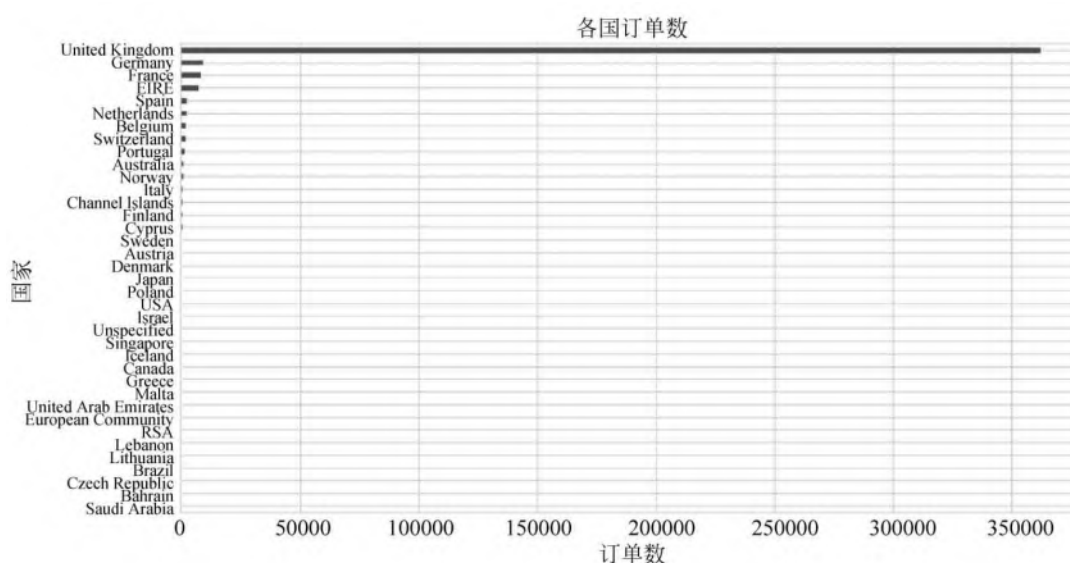


图10.17 各国订单数

图10.17中英国订单占比过高使得可视化时没法对其他国家的订单有更好的了解，所以考虑将英国的订单数据去掉后再进行分析，代码如下。

```

>>> group_country_orders = df_new.groupby('Country')['InvoiceNo'].\\
    count().sort_values()
>>> del group_country_orders['United Kingdom']
>>> plt.subplots(figsize=(15,8))
>>> group_country_orders.plot('barh', fontsize=12, color=color[0])
>>> plt.xlabel('订单数', fontsize=12)
>>> plt.ylabel('国家', fontsize=12)
>>> plt.title('各国订单数', fontsize=12)
>>> plt.show()

```

运行上述代码将得到不含英国的各国订单数，如图10.18所示。

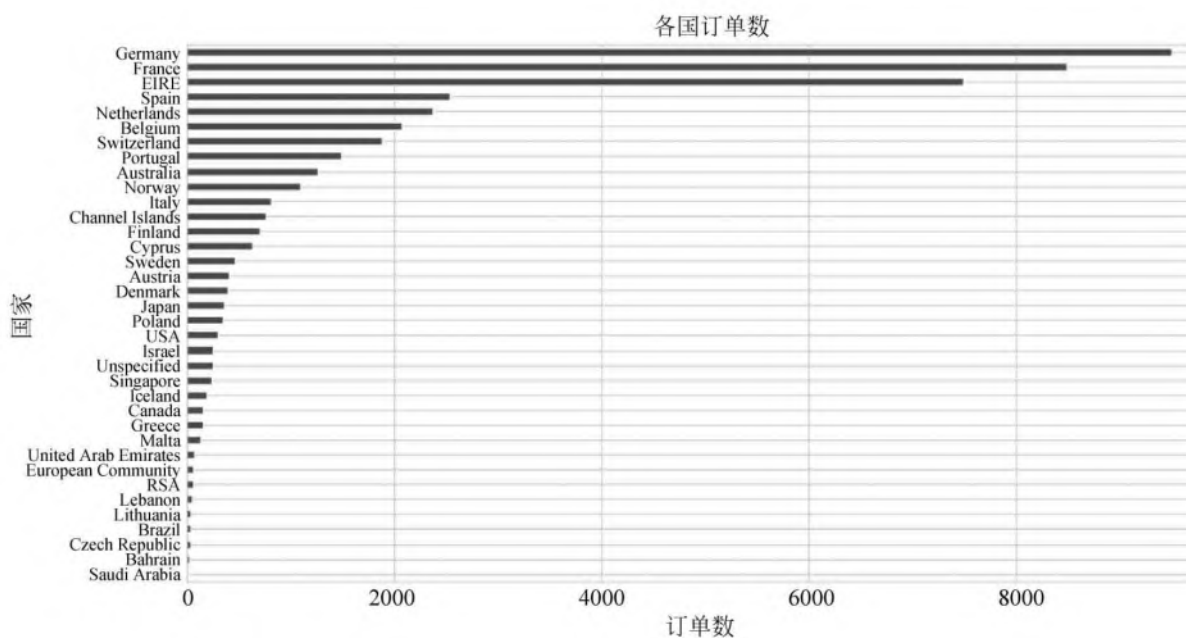


图10.18 各国订单数（不含英国）

从图10.18中看出网站的主要客户还是来自欧盟国家，中东和美洲国家的订单几乎可以忽略不计。与分析订单类似，还可以分析不同国家的消费金额，分别如图10.19和图10.20所示。

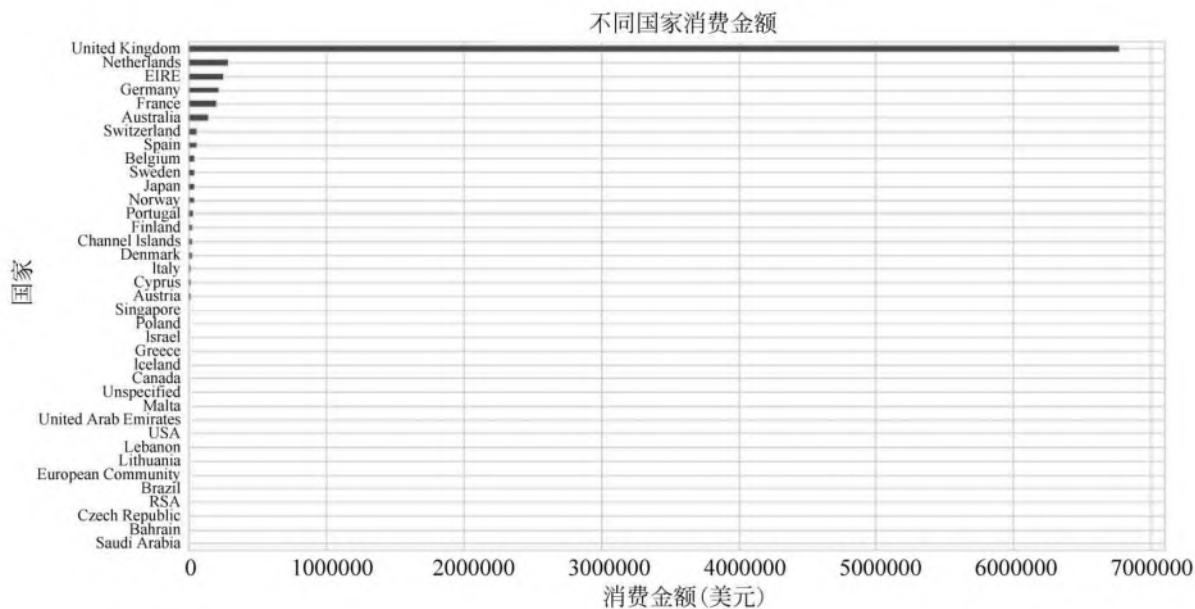


图10.19 不同国家消费金额

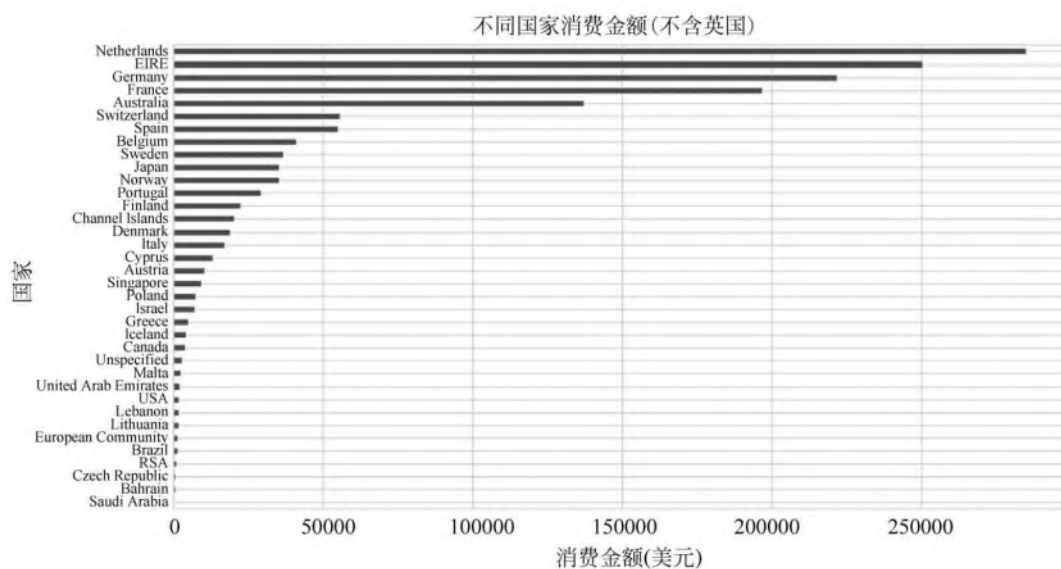


图10.20 不同国家消费金额（不含英国）

10.3.4 留给读者的问题

前面已经对客户、订单以及客户国家等信息进行了分析，这些分析是否就够了呢？是否有必要分析网站的商品售价？如何用盒子图来

查看客户的消费金额的具体分布？或者说读者是否有自己的问题待解答呢？

下篇 Python数据分析实战

第11章 群组分析

仰观宇宙之大，俯察品类之盛，
所以游目骋怀，足以极视听之娱，
信可乐也。

——《兰亭集序》

在移动互联的初期，如何让用户发现你的应用并下载它是各方竞争的焦点。当蓝海变成红海后，真正的成功就不是让别人下载你的应用，产品若没有很好地留存，花费大量时间和资源来挖掘新客户，终将“世事转头空”。对产品运营人而言，需要考虑的问题已经变成：

- 如何提高用户留存率？
- 如何找到用户流失的原因？
- 如何挽回将要流失或已经流失的用户？
- 用户群体有什么特征？
- 重点需要关注的用户群是哪些？

而所有这些都可以通过群组分析（Cohort Analysis）来完成。本章将介绍什么是群组分析，如何进行群组分析，最后用一个案例来演示实际的群组分析。

11.1 群组分析概述

11.1.1 从AARRR到RARRA的转变

2007年1月，iPhone 2G在美国上市；2008年7月，苹果推出iPhone 3G；2008年9月，谷歌发布首款Android手机HTC G1。移动互联网的大幕终于缓缓拉开，各类移动互联创业公司如雨后春笋般诞生。视线的另一角，有一个叫Dave McClure的人在2007年提出了一种创业公司的业务增长模式——海盗指标（AARRR），该模型在过去10年中或多或少已经成为行业标准。一时间，大量公司利用AARRR方法来跟踪产品营销和管理，该方法论已成为企业家创业的增长利器。在AARRR模型中，McClure将用户置于如图11.1所示的漏斗中。

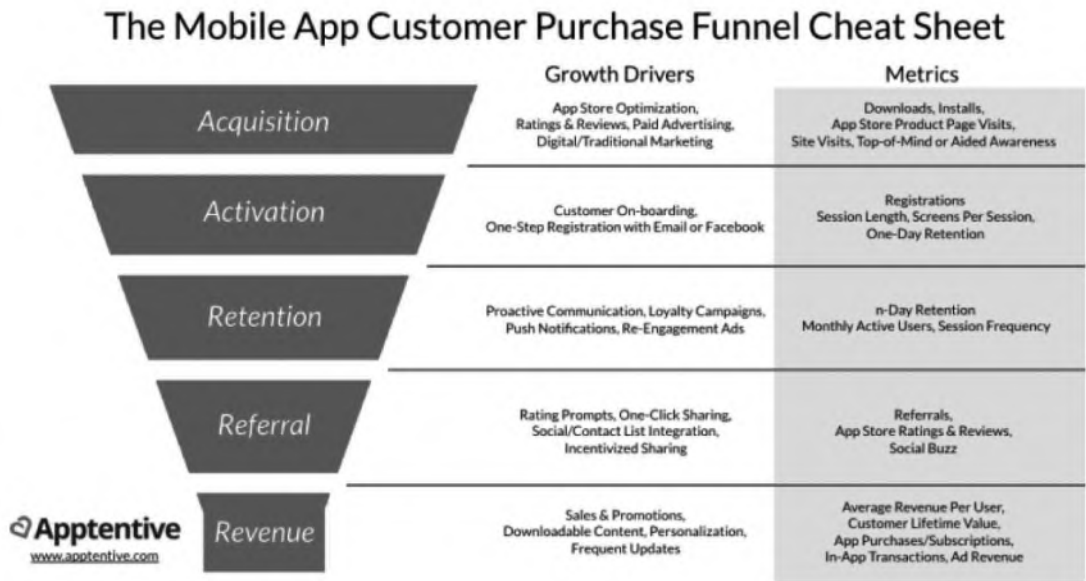


图11.1 AARRR模型

这一漏斗依次代表了：

- 用户获取（Acquisition）；
- 用户激活（Activation）；
- 用户留存（Retention）；
- 用户推荐（Referral）；
- 收入（Revenue）。

AARRR模型从用户的角度，以线性顺序来分析产品的营销。模型认为的产品推广会经历如下阶段。

(1) 首先通过搜索引擎优化 (Search Engine Optimization, SEO)、广告、地推、邮件推广等模式让用户知道你的产品。

(2) 接着通过用户引导去激活他们。

(3) 用户激活后，接下来的工作就是提高用户留存率。

(4) 推动用户分享，推荐你的产品。

(5) 最终完善商业模式，获得收入。

然而在AARRR应用中，人们很快发现了其中的一个缺点。许多应用都会经历这样一种模式：应用发布，媒体广泛关注，用户反响也非常积极，许多用户都在不遗余力地为这款产品叫好。于是成千上万的新用户涌入，然而它们都存在一个大问题——用户流失率太高。最终由于新用户越来越少，老用户慢慢离去，这些应用逐渐消失在人们的视线中。另外，在互联网的初期，获客成本只需要几元或十几元，随着移动互联网的发展，各类应用商店上的应用数都达到了百万以上，此时获客成本增加到了几十元。从2018年的小米招股说明书披露的数据来看，互联网巨头的获客成本目前已经到了人民币100元以上。于是，一个新的模型RARRA应运而生，RARRA模型是Thomas Petit和Gabor Papp对于海盗指标AARRR模型的优化。RARRA模型突出了用户留存的重要性，认为原有的模型中将用户获取放在第一位是错误的，进行用户分析时，我们第一步应该关注的是留存，即R (Retention)，如图11.2所示。



图11.2 RARRA模型

设想一下，如果你的应用在用户安装后，3天内70%以上的用户将不再使用，一个月后90%的客户完全忘记了它，3个月后95%以上的用户已经卸载了它，无论你在用户获取上多么努力都无济于事。当获客不再是增长的王道时，那么用户留存（Retention）就成为你的新起点。通过为用户提供价值，让用户不断使用你的产品，与此同时在用户激活（Activation）时就能感受到产品价值，最终用户推荐（Referral）他人使用，完成用户获取（Acquisition）。这才是更好的增长模式！而要完成用户留存，我们就需要了解用户，他们有什么特征，不同用户群之间有何区别。这一切都可以通过群组分析来完成。

11.1.2 什么是群组分析

Cohort在英文里面是群组的意思，顾名思义，Cohort Analysis就是群组分析，即将用户分成不同的群组进行分析。群组分析的实现非常简单，就是首先做好观察用户的分组。分组先分维度，再分粒度。所谓维度，如果按用户的新增日期分组，时间就是维度；如果按新增用户的渠道来源分组，渠道就是维度。而粒度是指在该维度如何划分，如时间维度是按照月还是天划分；新增渠道维度是按照新增的来

源网站还是来源的具体网址划分，这些都是粒度差异。以下群组就可能是实际数据分析中会遇到的群组：

- 过去30天试用产品的用户；
- 5月的付费用户；
- 通过社交媒体获得的客户；
- 把商品加入了购物车却没有完成付款的用户。

我们可以根据分析需要划分出无数个用户群，之后就可以计算分组留存率，通过对比的差异值逐级锁定，寻找原因。如图11.3中对3个不同群组的用户在12个时间周期的商品购买情况进行的分析，我们发现2010-12这个组的用户购买明显比另外两个组要好。那么是什么原因导致的呢，可以进一步分析这个群组用户有什么特征，与另外的群组有何不同，最终提高用户留存率。

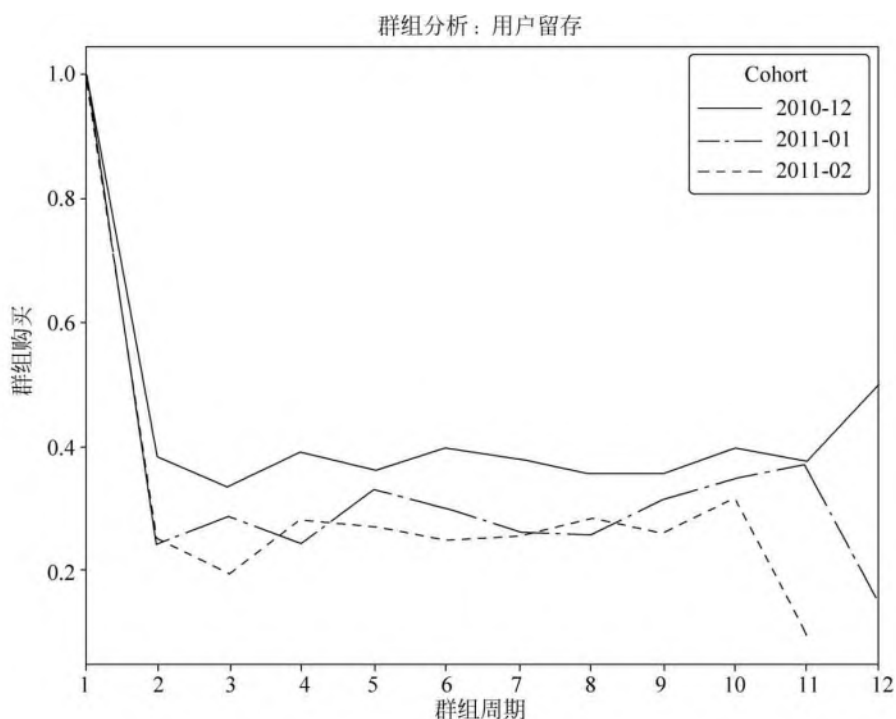


图11.3 用户留存

11.2 群组分析实战

本节仍将利用前面使用过的英国某电商的销售数据进行群组分析。

11.2.1 定义群组以及周期

首先读入数据，进行简单了解，如图11.4所示。

```
>>> online = pd.read_csv("../data/Online_Retail.csv", parse_dates = ['InvoiceDate'])
>>> online.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

图11.4 读入销售数据

这里的数据包含英国某电商网站2010年12月到2011年12月的销售数据，数据包含8列。进一步可以通过info（）函数对数据类型、是否有缺失数据进行了解。

```
>>> online.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
InvoiceNo      541909 non-null object
StockCode      541909 non-null object
Description     540455 non-null object
Quantity       541909 non-null int64
InvoiceDate    541909 non-null datetime64[ns]
UnitPrice      541909 non-null float64
CustomerID     406829 non-null float64
Country        541909 non-null object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1 + MB
```

从代码输出可以看出其中的 InvoiceDate 列为日期类型，CustomerID 列以及 Description 列有些数据是有缺失的，不过这里先不对其进行处理。群组分析的第一步就是思考分析的目标是什么，如果想了解不同时期用户的留存率有无变化，购买量有无变化，那么可以以 InvoiceDate 作为分组依据。因此，时间就是前面提到的维度。游戏、应用推广有的会以天、周作为粒度来观察用户的激活、在线、购买趋势，而对当前问题以月为粒度就可以了。至此，我们可以明确分析目标为按照月来观察用户购买、留存趋势。InvoiceDate 列虽然提供了日期信息，但是它不是以月来度量，因此，分析的第一目标就是对该数据进行重构，转换为以月来区分，之后需要将客户（由 CustomerID 唯一提供）按其首次购买月来分组。要将日期转换为月来度量，可以采用如下方法。

```
>>> def get_month(x):
>>>     return datetime(x.year, x.month, 1)
>>> online['InvoiceDate'].apply(get_month)
0          2010-12-01
1          2010-12-01
2          2010-12-01
...
541907     2011-12-01
541908     2011-12-01
Name: InvoiceDate, Length: 541909, dtype: datetime64[ns]
```

上述代码利用 get_month（）函数将传入日期只取年和月信息，而最后的日都统一为1日，这样同一个月的日期信息就一致了。之后利用在前面章节学习过的 apply（）函数，可以将整个 InvoiceDate 列都进行相同转换，并添加新的 InvoiceMonth 列来保存该信息。变换后的数据如图 11.5 所示。

```
>>> online['InvoiceMonth'] = online['InvoiceDate'].apply(get_month)
>>> online.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceMonth
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	2010-12-01
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	2010-12-01
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01

图11.5 变换后的数据（以月度量日期）

有了用户购买月信息InvoiceMonth后，我们就可以对所有订单先按照客户，再按照购买月进行分组。然而，进行群组分析想要的是按照首次购买月来对用户进行分组，此时就可以利用Pandas中的grouby提供的transform功能来完成这一功能，即以群中的最小购买日期作为首购月，代码如下。

```
>>> grouping = online.groupby('CustomerID')['InvoiceMonth']
>>> online['CohortMonth'] = grouping.transform('min')
>>> online.tail()
```

运行结果如图11.6所示。

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceMonth	CohortMonth
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	12680.0	France	2011-12-01	2011-08-01
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France	2011-12-01	2011-08-01
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France	2011-12-01	2011-08-01
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France	2011-12-01	2011-08-01
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France	2011-12-01	2011-08-01

图11.6 用户首购月

上述代码中的grouping是按照CustomerID分组后的分组对象，利用transform（）函数对InvoiceMonth中信息进行变换，取首次购买月（min函数可以完成这一功能）就可以作为对应该组的CohortMonth。CohortMonth就是群组分析需要的群组信息，而分析目标是了解不同群组随着时间流逝有何变化。因此，还缺少一个时间周期信息，即对应群组在1个月后的购买情况如何，2个月后购买情况如何，等等。该信

息可以通过整合InvoiceDate以及CohortMonth来获得。InvoiceDate中有当前购买日期，而CohortMonth中是用户首购月，两者求差可以得出当前购买日是在用户首次购买后的几个月。为了处理方便，在本书中认为用户在首购月购买计数为第1个统计周期，所以计算差的时候会有加1操作，代码如下。

```
>>> def get_date_int(df, column):
>>>     year = df[column].dt.year
>>>     month = df[column].dt.month
>>>     day = df[column].dt.day
>>>     return year, month, day
>>> cohort_year, cohort_month, _ = get_date_int(online, 'CohortMonth')
>>> invoice_year, invoice_month, _ = get_date_int(online, 'InvoiceDate')
```

上述代码定义的get_date_int（）函数可以取对应日期的年和月，有了这个信息才能进行时间周期计算，最终代码如下。

```
>>> years_diff = invoice_year - cohort_year
>>> months_diff = invoice_month - cohort_month
>>> online['CohortPeriod'] = years_diff * 12 + months_diff + 1
>>> online.tail()
```

运行结果如图11.7所示。

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceMonth	CohortMonth	CohortPeriod	
	541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	12680.0	France	2011-12-01	2011-08-01	5.0
	541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France	2011-12-01	2011-08-01	5.0
	541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France	2011-12-01	2011-08-01	5.0
	541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France	2011-12-01	2011-08-01	5.0
	541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France	2011-12-01	2011-08-01	5.0

图11.7 时间周期计算

11.2.2 群组分析具体过程

CohortMonth和CohorPeriod分别提供了群组以及时间周期信息，接下来数据分析要做的就是统计用户活跃情况了。在开始之前，为了后面可视化时显示美观，还需要对CohortMonth进行美化。CohortMonth名义上是月的群组，但是还有日的信息，如果想只显示月，那么可以简单地利用字符串截取功能对数据进行如下处理。

```
>>> online['Cohort'] = online['CohortMonth'].astype(str)
>>> online['Cohort'] = online['Cohort'].apply(lambda x:x[: - 3])
```

每个Cohort中的活跃用户数可以通过计算该组中不同CustomerID个数获得，代码如下。

```
>>> grouping = online.groupby(['Cohort', 'CohortPeriod'])
>>> cohort_data = grouping['CustomerID'].apply(pd.Series.nunique)
>>> cohort_data = cohort_data.reset_index()
>>> cohort_data.head()
```

	Cohort	CohortPeriod	CustomerID
0	2010-12	1.0	948
1	2010-12	2.0	362
2	2010-12	3.0	317
3	2010-12	4.0	367
4	2010-12	5.0	341

图11.8 活跃用户数

结果如图11.8所示。至此已经完成了所有的数据准备工作，Cohort列提供了群组，CohortPeriod提供了时间周期，CustomerID列提供了活跃用户数。接下来唯一需要做的就是将数据按照时间周期进行展示，即按照不同Cohort来观察不同的CohortPeriod、CustomerID中的数据变化。而这正是pivot（）函数提供的功能，代码如下。

```
>>> cohort_counts = cohort_data.pivot(index = 'Cohort',
                                       columns = 'CohortPeriod',
                                       values = 'CustomerID')
>>> cohort_counts
```

结果如图11.9所从代码输出可以看出，2010-12这个组在第1个时间周期有948个活跃用户，但是在第2个时间周期就只有362个活跃用户了，之后基本没有太大变化，到第12个时间周期，活跃用户数突然出现了增长，第13个时间周期又突然下降。可以看出，群组分析很好地展示这一趋势变化，数据分析人员很自然地会思考开始的用户下降的原因，之后的用户增长是促销还是别的原因导致。除了观察组内趋势，群组分析还可以对比组间的区别。例如，在第2个时间周期，2011-01组的活跃用户数为何下降比2010-12组快。此外，除了分析活跃用户，还可以利用如下代码计算用户留存情况，结果如图11.10所示。

CohortPeriod	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	13.0
Cohort													
2010-12	948.0	362.0	317.0	367.0	341.0	376.0	360.0	336.0	336.0	374.0	354.0	474.0	260.0
2011-01	421.0	101.0	119.0	102.0	138.0	126.0	110.0	108.0	131.0	146.0	155.0	63.0	NaN
2011-02	380.0	94.0	73.0	106.0	102.0	94.0	97.0	107.0	98.0	119.0	35.0	NaN	NaN
2011-03	440.0	84.0	112.0	96.0	102.0	78.0	116.0	105.0	127.0	39.0	NaN	NaN	NaN
2011-04	299.0	68.0	66.0	63.0	62.0	71.0	69.0	78.0	25.0	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2011-08	167.0	42.0	42.0	42.0	23.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09	298.0	89.0	97.0	36.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10	352.0	93.0	46.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11	321.0	43.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12	41.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

13 rows x 13 columns

图11.9 按时间周期展示数据

```
>>> cohort_sizes = cohort_counts.iloc[:,0]
>>> retention = cohort_counts.divide(cohort_sizes, axis = 0)
>>> retention.head()
```


CohortPeriod	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	13.0
Cohort													
2010-12	1.0	0.381857	0.334388	0.387131	0.359705	0.396624	0.379747	0.354430	0.354430	0.394515	0.373418	0.500000	0.274262
2011-01	1.0	0.239905	0.282660	0.242280	0.327791	0.299287	0.261283	0.256532	0.311164	0.346793	0.368171	0.149644	NaN
2011-02	1.0	0.247368	0.192105	0.278947	0.268421	0.247368	0.255263	0.281579	0.257895	0.313158	0.092105	NaN	NaN
2011-03	1.0	0.190909	0.254545	0.218182	0.231818	0.177273	0.263636	0.238636	0.288636	0.088636	NaN	NaN	NaN
2011-04	1.0	0.227425	0.220736	0.210702	0.207358	0.237458	0.230769	0.260870	0.083612	NaN	NaN	NaN	NaN

图11.10 用户留存情况

留存率的计算是与第1个时间周期对比，因此需要首先获得第1个周期内各组的用户数，保存到cohort_size变量中，之后将每个周期的用户总数除以cohort_size，就得到了留存率。为了简单起见，下面的代码将其转化为百分比，同时只取小数点后1位，结果如图11.11所示。

```
>>> retention.round(3) * 100
```

为了更好地分析每个时间周期用户留存的变化，还可以对上面的数据进行可视化。retention数据中，用户所属群作为行，时间周期作为列，而数据可视化是想观察不同时间周期的数据变化，因此，需要对原数据进行一个行列变化，这可以通过.T完成，代码如下。

```
>>> retention.iloc[[0,1,2],:].T.plot(figsize=(10,8),fontsize=15)
>>> plt.title('群组分析: 用户留存',fontsize=15)
>>> plt.xticks(np.arange(1, 12.1, 1))
>>> plt.xlim(1, 12)
>>> plt.ylabel('群组购买',fontsize=15)
>>> plt.xlabel('群组周期',fontsize=15);
```

CohortPeriod	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	13.0
Cohort													
2010-12	100.0	38.2	33.4	38.7	36.0	39.7	38.0	35.4	35.4	39.5	37.3	50.0	27.4
2011-01	100.0	24.0	28.3	24.2	32.8	29.9	26.1	25.7	31.1	34.7	36.8	15.0	NaN
2011-02	100.0	24.7	19.2	27.9	26.8	24.7	25.5	28.2	25.8	31.3	9.2	NaN	NaN
2011-03	100.0	19.1	25.5	21.8	23.2	17.7	26.4	23.9	28.9	8.9	NaN	NaN	NaN
2011-04	100.0	22.7	22.1	21.1	20.7	23.7	23.1	26.1	8.4	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2011-08	100.0	25.1	25.1	25.1	13.8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09	100.0	29.9	32.6	12.1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10	100.0	26.4	13.1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11	100.0	13.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12	100.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

13 rows x 13 columns

图11.11 留存率

除了对趋势进行可视化，还可以绘制留存率的热图，最终得到如图11.12所示的留存率热图，代码如下。

```
>>> plt.figure(figsize = (10, 8))
>>> plt.title('留存率')
>>> sns.heatmap(data = retention,
                annot = True,
                fmt = '.0%',
                vmin = 0.0,
                vmax = 0.5,
                cmap = 'BuGn')
```

从热图中可以很容易地看到不同群组间在各时间周期的留存率存在差异。那么数据分析人员可能就需要考虑进一步分析留存突然增加或减少的原因，有没有办法对其进行改善。虽然群组分析经常用来进行用户留存分析，但是以类似思路还可以分析客户的购买量、生命周期价值等，下面的代码就是对用户的购买量进行分析，结果如图11.13所示。

```

>>> grouping = online.groupby(['Cohort', 'CohortPeriod'])
>>> cohort_data = grouping['Quantity'].mean()
>>> cohort_data = cohort_data.reset_index()
>>> average_quantity = cohort_data.pivot(index = 'Cohort',
                                         columns = 'CohortPeriod',
                                         values = 'Quantity')

>>> average_quantity.round(1)

```

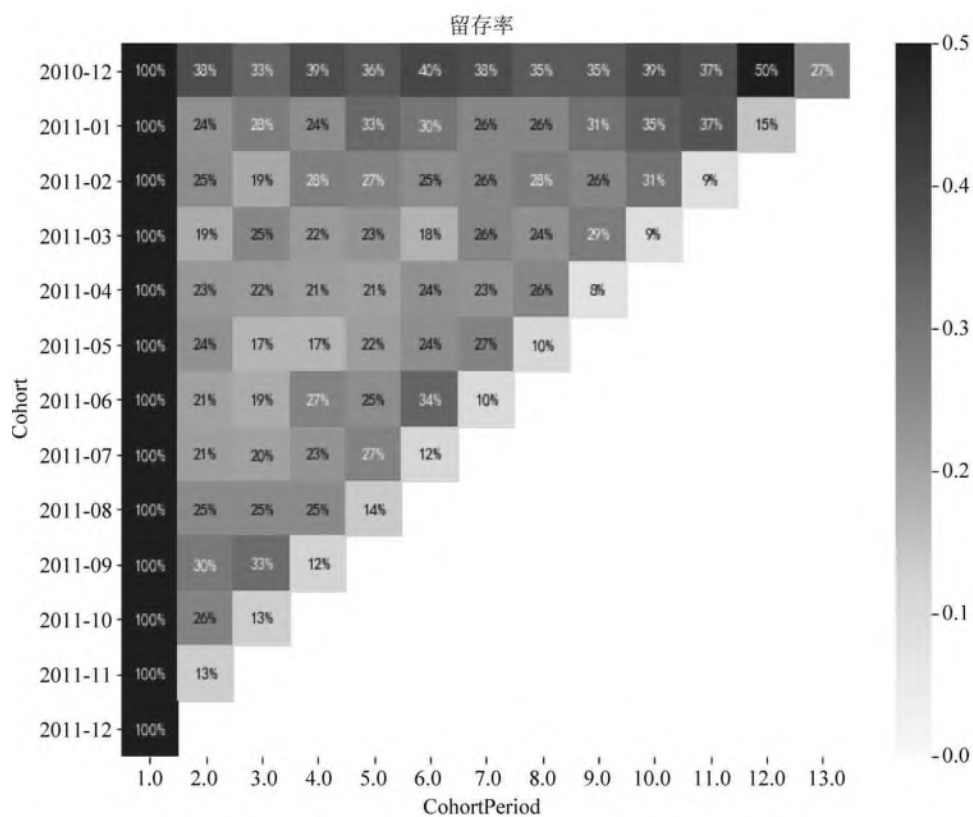


图11.12 留存率热图

CohortPeriod	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	13.0
Cohort													
2010-12	11.0	14.6	15.0	14.8	12.9	14.3	15.2	14.8	16.7	16.7	17.3	12.8	14.8
2011-01	10.0	12.6	12.3	10.9	12.2	14.9	14.2	14.4	11.4	9.9	9.1	9.5	NaN
2011-02	10.8	12.1	18.6	12.0	11.1	11.4	13.3	12.4	10.3	11.9	12.6	NaN	NaN
2011-03	9.8	9.9	12.2	9.5	13.6	12.3	13.2	12.2	10.5	8.9	NaN	NaN	NaN
2011-04	9.8	10.1	9.4	11.6	11.5	8.2	9.7	9.3	7.3	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2011-08	9.9	6.0	5.3	6.0	7.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09	11.9	5.5	7.6	8.8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10	8.4	6.9	8.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11	8.7	9.3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12	14.5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

13 rows x 13 columns

图11.13 用户购买量分析

11.2.3 思考

在待分析的数据中还包含了用户购买商品数量、单价信息，如果网站想利用群组分析对用户生命周期价值进行分析，应该如何做呢？本章的例子是以时间维度进行群组分析，那么实际中读者可能还会遇到基于用户行为进行分组的问题，此时又该如何实现呢？群组分析只是用户分析的一种方法，有时数据分析人员还会对用户进行RFM建模（RFM即为Recency, Frequency, Monetary），这将是下一章讨论的内容。

第12章 利用RFM分析对用户进行分类

顾客是上帝，顾客也可能是魔鬼，
但是最难对付的顾客都有一个特点：
他是你的顾客，他是你的推销对象。

——戈登《推销员成功之道》

顾客是上帝，但是他们也是最善变的，不知道什么时候就离你而去。那我们应该如何去分析顾客，了解他们呢？RFM模型隶属于用户价值模型，有两个方向：一个是基于用户生命周期，也就是时间和用户在产品内的成长路径进行的生命周期模型的搭建；另一个就是基于用户关键行为进行搭建。其中，RFM模型是最典型的基于用户关键行为的模型，它是衡量用户价值和用户创利能力的一个重要的工具和手段，被广泛应用在各个行业中。本章将介绍什么是RFM分析，如何利用RFM分析的结果对客户进行分类。

12.1 RFM分析简介

12.1.1 RFM模型概述

在用户运营领域，有一个叫做RFM的词，相信很多人看到过。那它到底代表什么意思呢？RFM中的R、F、M分别代表了3个英文单词的首字母，各自的意义如下。

- R：最近一次消费（Recency），代表用户距离当前最后一次消费的时间。当然，最近一次消费的时间越短越好，对我们来说更有价值。

- F: 消费频次 (Frequency)，用户在一段时间内，在产品内的消费频次，这里的重点是对“一段时间”的定义。
- M: 消费金额 (Monetary)，代表用户的价值贡献。

有了RFM模型之后，进行客户运营时可以决定发送短信时，对哪些用户加上前缀“尊敬的VIP用户”，对哪些用户加上前缀“好久不见”。该模型也可以帮助企业判断哪些用户有异动，是否有流失的预兆，从而增加相应的运营动作。例如，根据如图12.1所示的RFM模型将用户分为重要价值客户、重要发展客户、一般价值客户等，据此来进行有针对性的运营。

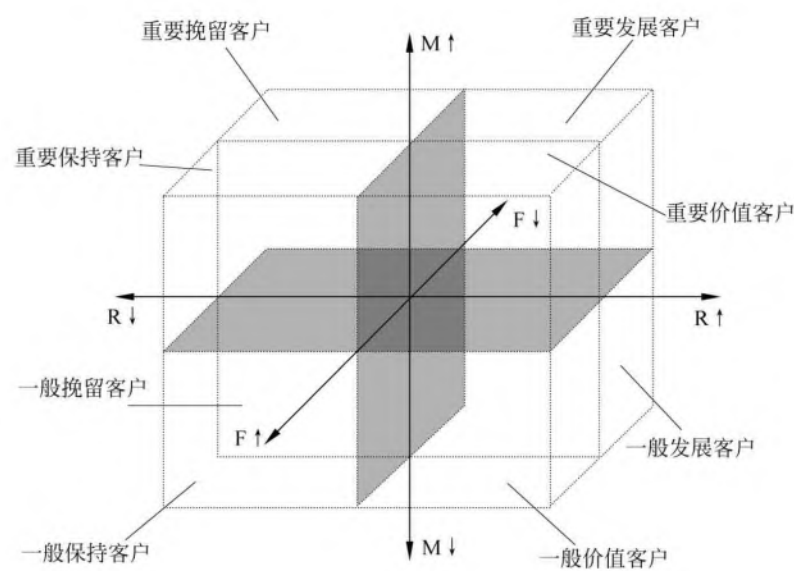


图12.1 RFM模型

上面的模型中，客户的重要与否是根据对应的R、F、M的值决定的。例如，可以分别将R、F、M简单分为高、中、低3个区间，然后根据客户所在区间来对客户进行细分，如表12.1所示。

表12.1 利用RFM模型细分客户

客 户 类 别	R	F	M
重要价值客户	高	高	高
重要挽留客户	低	低	高
一般价值客户	中	低	低
一般发展客户	高	中	中
...	...	...	...

有了RFM模型之后，运营人员就可以基于模型的评分来更好地指导运营，因为RFM告诉了我们：

- 谁是最好的客户？
- 哪些客户正处于流失的边缘？
- 谁有可能转化为更有利可图的客户？
- 谁是不需要关注的无价值客户？
- 必须保留哪些客户？
- 谁是忠实客户？
- 哪些客户最有可能对当前的营销动作做出回应？

12.1.2 理解RFM

1. R——最近一次消费

对不同的企业，R有不同含义。对电商而言，R指的是客户在店铺最近一次消费和当前的时间间隔，理论上R值越小的客户价值越高。而对社交网站、在线视频播放而言，R可能就是最近一次登录时间、最近一次发帖时间、最近一次投资时间、最近一次观看时间。以电商为例，目前网购便利，顾客已经有了更多的购买选择和更低的购买成本，去除地域的限制因素，客户非常容易流失，因此，要提高回购率和留存率，需要时刻警惕R值。

2. F——消费频率

消费频率是客户在固定时间内（如1年、1个月）的购买次数。不同的行业，用户购买频率有很大区别，如用户可能每个月都会购买纸尿裤，而电子产品可能一年内也才消费一次。因此，消费频率取决于产品和行业。我们在构建RFM模型时，有时把F值的时间范围去掉，替换成累计购买次数。影响复购的核心因素是商品，因此对复购不适合做跨类目比较。例如食品类目和美妆类目：食品属于“半标品”，产品的标品化程度越高，客户背叛的难度就越小，越难形成忠实用户；但是相对美妆，食品又属于易耗品，消耗周期短，购买频率高，相对容易产生重复购买。因此跨类目复购并不具有可比性。

3. M——消费金额

M值是RFM模型中最具有价值的指标。大家熟知的“二八定律”（又名“帕累托法则”）曾给出过这样的解释：公司80%的收入来自20%的用户。理论上M值和F值是一样的，都带有时间范围，指的是一段时期（通常是1年或1个月）内的消费金额。有的情况下，我们还会考虑客单价，对于客单价高的用户，我们考虑的是如何提高他的购物频次；而对于客单价低的客户，我们考虑是否可以想办法提高客单价。

12.2 RFM实战

在第11章，我们对某电商的销售数据进行了群组分析，本节将从RFM模型的角度对同一数据的用户进行细分。

12.2.1 R、F、M值的计算

所有的数据分析都是从对目标数据的了解而开始，首先读入数据，代码如下。


```

>>> import pandas as pd
>>> import numpy as np
>>> pd.set_option('display.max_columns', 20)
>>> pd.set_option('display.max_rows', 10)
>>> online = pd.read_csv("../data/Online_Retail.csv", parse_dates = ['InvoiceDate'])
>>> online.head()

```

read_csv () 函数指定了参数parse_dates=['InvoiceDate'], 因此InvoiceDate列将解析为日期格式。通过head () 函数可以对数据集的信息大致了解。其中的InvoiceDate、InvoiceNo、CustomerID将是RFM模型构建的基础。接下来需要考察数据中是否有缺失信息, 代码如下。

```

>>> online.isnull().sum()
InvoiceNo      0
StockCode      0
Description    1454
Quantity      0
InvoiceDate    0
UnitPrice      0
CustomerID    135080
Country        0
dtype: int64

```

从代码输出发现CustomerID有135080行缺失, 而整个数据集一共只有541909条。在现实数据分析中如果发现接近1/4的数据存在缺失, 而该数据又正是需要分析的数据, 就需要考虑什么原因导致数据缺失, 是否有办法补齐。不过对当前的案例而言, 只能采用将有数据缺失的行丢弃的方法, 处理代码如下。

```
>>> mask = online['CustomerID'].isnull()
>>> online_rfm = online[~mask]
>>> online_rfm.isnull().sum()
InvoiceNo      0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

上述代码利用mask掩码，将CustomerID缺失的行过滤掉。而新构建的online_rfm就是需要分析的数据集。处理完缺失数据后，下一步就是计算RFM模型中对应的R、F、M值。对电商而言，R值的计算可以通过当前日期与最近一次购买的间隔来计算获得，F值可以通过统计客户购物InvoiceNo的数目获得。用UnitPrice来代表M值显然不合理，因此，需要将购物单价与购物数量相乘后得到总价，作为M值计算的基础。这里需要在online_rfm数据集中添加新的列来记录每张InvoiceNo对应的总价，方法如下。

```
>>> online_rfm['Total'] = online_rfm['UnitPrice'] * online_rfm['Quantity']
C:\ProgramData\Anaconda3\envs\py37\lib\site-packages\ipykernel_launcher.py: 1:
SettingWithCopyWarning:
  A value is trying to be set on a copy of a slice from a DataFrame.
  Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy
"""Entry point for launching an IPython kernel.
```

在执行这一计算的过程中，读者可能会得到SettingWithCopyWarning的警告，这也是令Pandas初学者经常感到迷惑的一个警告。这个警告实际源于Pandas的设计机制，因为online_rfm是通过布尔筛选的方式过滤CustomerID为缺失的行而得到，此时对它进行赋值就会产生一个警告。一种解决方法是如上面建

议，使用.loc方式赋值，不过这会重复一次之前的数据筛选，而这里由于分析时很明确地知道当前就是要在原DataFrame切片备份上进行，那也可以选择忽略该警告。如果要忽略该警告，可以使用如下代码。

```
>>> import warnings
>>> warnings.simplefilter('ignore')
```

再执行前面的赋值操作就不会有警告了。添加了Total列之后，下一个需要处理的问题就是CustomerID列的数据类型了，通过.dtypes属性可以发现该列为float64类型，而实际上CustomerID的数据应该是为整型，因此需要做一次数据类型的变换，代码如下。

```
>>> online_rfm.dtypes
InvoiceNo      object
StockCode      object
Description    object
Quantity       int64
InvoiceDate    datetime64[ns]
UnitPrice      float64
CustomerID     float64
Country        object
Total          float64
dtype: object
>>> online_rfm['CustomerID'] = online_rfm['CustomerID'].astype(int)
```

至此，所有的数据准备工作都已经完成，接下来可以开始计算R值了。前面已经提到过当前数据集是从2010年12月到2011年12月的数据。在分析时假设当前的日期为2011年12月10日，利用该日期来计算各个用户的最近购物间隔天数。采用如下代码即可完成这一操作。

```
>>> current_date = max(online_rfm.InvoiceDate) + timedelta(days=1)
```

而客户的最近购物日期与当前日期的间隔天数可以通过先以CustomerID作为分组，找出每组中最晚的日期（InvoiceDate），计算它与当前日期的差值来获得。购物频率则通过对每组中的InvoiceNo数

目计数得到，总的消费额则通过对组内Total列汇总得到。这一切通过Pandas中的分组汇聚功能即可实现，代码如下。

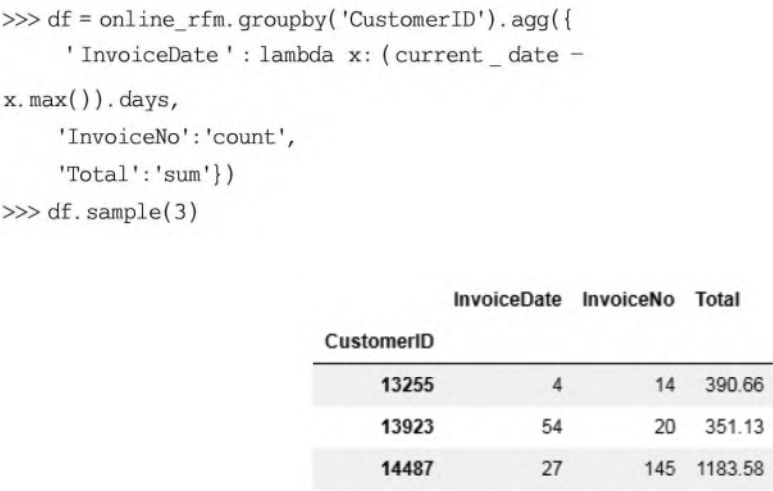


图12.2 RFM计算结果

结果如图12.2所示。上述代码的最后一行采用了sample（）函数来对df进行采样，取其中3行进行显示。通过分组汇聚后得到的新数据集中，InvoiceDate列实际代表了当前日期与最近购物日期的间隔天数（即Recency），InvoiceNo代表了购物频次（即Frequency），Total列代表了总购物金额（即Monetary），因此有必要对上述的列名进行修正，可以采用.rename（）函数实现，结果如图12.3所示。

```
>>> df.rename(columns = {'InvoiceDate': 'Recency',
    'InvoiceNo': 'Frequency',
    'Total': 'Monetary'}, inplace = True)
>>> df.sample(3)
```

虽然现在已经得到了想要的R、F、M值，但是由于这些值分布很广，不利于进行后续的分析，因此在进一步分析前还需要对它们进行再加工。为简单起见，本例将当前的R、F、M划分到4个区间，分别用1，2，3，4代表，值越小表示该指标越差。Recency列代表了最近一次购物的间隔日期，显然该值应该越小越好，采用如下代码处理。

```
>>> r = range(4, 0, -1)
>>> r_quartiles = pd.cut(df['Recency'], 4, labels = r)
```

利用cut（）函数可以自动将数据划分到指定个数区间，而每个区间对应的标签依次为4，3，2，1，这样刚好可以实现对应区间4的是间隔时间最短的。而对Frequency和Monetary列，显然是值越大代表越好，所以它们的标签处理刚好与Recency相反，直接使用range（1，5）来对应，代码如下。

```
>>> f = range(1,5)
>>> m = range(1,5)
>>> f_quartiles = pd.qcut(df['Frequency'], 4, labels = f)
>>> m_quartiles = pd.qcut(df['Monetary'], 4, labels = m)
>>> df = df.assign(F = f_quartiles.values)
>>> df = df.assign(M = m_quartiles.values)
>>> df.sample(5)
```

运行结果如图12.4所示。

	Recency	Frequency	Monetary
CustomerID			
17420	50	30	598.83
14782	320	6	200.10
18256	355	4	-50.10

图12.3 修正列名

	Recency	Frequency	Monetary	R	F	M
CustomerID						
14116	20	72	1382.74	4	3	3
12921	4	741	16389.74	4	4	4
15225	234	23	409.40	2	2	2
17739	12	48	2786.05	4	3	4
16697	241	21	112.75	2	2	1

图12.4 获取对应的R、F、M值

大功告成，现在已经获得了所有客户对应的R、F、M值，接下来就是利用RFM模型对客户进行细分。

12.2.2 利用RFM模型对客户进行细分

RFM模型分析的目的是将客户进行细分，进而制定有针对性的营销/运营策略。本案例中的R、F、M取值各有4种可能，那么理论上一共有 $4^3=64$ 种组合。在进一步分析前，我们先利用如下代码查看组合的分布以及得分情况，结果如图12.5所示。

```
>>> def concat_rfm(df):  
    return str(df['R']) + str(df['F']) + str(df['M'])  
>>> df['RFM_Segment'] = df.apply(concat_rfm, axis = 1)  
>>> df['RFM_Score'] = df[['R', 'F', 'M']].sum(axis = 1)  
>>> df.head()
```

	Recency	Frequency	Monetary	R	F	M	RFM_Segment	RFM_Score
CustomerID								
12346	326	2	0.00	1	1	1	111	3.0
12347	2	182	4310.00	4	4	4	444	12.0
12348	75	31	1797.24	4	2	4	424	10.0
12349	19	73	1757.55	4	3	4	434	11.0
12350	310	17	334.40	1	1	2	112	4.0

图12.5 组合的分布以及得分情况

上述代码首先将R、F、M的值连接在一起构成新的RFM_Segment列，而RFM_Score则是通过将R、F、M这3列求和得到，后面分析中可以基于RFM_Score来将用户分为常见的金牌客户、银牌客户、铜牌客户。进一步分析各细分群组，可以发现优质客户最多，代码如下。

```
>>> df.groupby('RFM_Segment').size().sort_values(ascending = False)[:10]
RFM_Segment
444      760
433      448
422      322
411      311
443      201
434      176
111      167
423      162
211      155
432      152
dtype: int64
```

444这个客户群代表的是最近购物时间距当前日期短，购物频率高，同时购物金额大的群体。通过分组统计可以发现这个群体是最大的，同时433、422这两个客户群也排在了第2和第3位，这说明网站大量客户的购物日期距当前日期都很近，网站的客户留存做得很好。不过从输出也发现411群体也比较多，那么这部分客户购物日期距当前日期近，但是购物频率和购物金额都小，这是否是由于是新客户导致？因此可以筛选出411这个客户群的数据进行进一步分析，由于这里没有客户的注册日期等信息，所以就不再展开。

RFM分析中用得最广泛的就是客户细分，下面基于RFM_Score将客户分为3类。需要说明的是，前面的RFM_Score的计算采用简单求和得到，实际分析中经常考虑赋予不同权重给各指标，为简单起见，本例没有考虑这个问题，运行下面的代码，分类结果如图12.6所示。

```
>>> def segment(df):
    if df['RFM_Score'] >= 10:
        return 'Gold'
    elif (df['RFM_Score'] >= 6) and (df['RFM_Score'] <= 9):
        return 'Silver'
    else:
        return 'Bronze'
>>> df['Segment'] = df.apply(segment, axis = 1)
>>> df.groupby('Segment').agg({
    'Recency': 'mean',
    'Frequency': 'mean',
    'Monetary': ['mean', 'count']}).round(1)
```

Segment	Recency	Frequency	Monetary	
	mean	mean	mean	count
Bronze	260.0	12.8	190.4	698
Gold	30.3	192.1	4053.8	1744
Silver	87.1	32.6	568.5	1930

图12.6 客户分类情况

本例中首先定义了一个segment（）函数，将RFM_Score大于10的归类为金牌客户，得分为6~9的归类为银牌客户，其他客户归类为铜牌客户。之后利用apply（）函数构建了新的Segment列。最后利用分组统计可以观察各组客户Recency、Frequency、Monetary的区别。很明显，金牌客户的平均消费金额最大，达到了4053.8，而银牌和铜牌客户只有568.5和190.4，同时金牌客户平均购物频次达到了192.1，另两组却只有32.6和12.8。通过这一数据可以看出这个公司主要由金牌客户驱动，客户运营时只需要重点关注这些客户就好。

12.2.3 思考

除了案例里面展示的分析，读者还可以对每个RFM_Segment的消费情况进行进一步细分。同时，前面的客户细分只是简单分为3类，实际分析还可以进一步细化该分类方法，以得出哪些客户消费多，但是最

近没有购物；哪些客户消费频次高，但是购物金额不大。类似的问题还有很多，唯一需要强调的是数据分析需要与具体场景结合，它的目标永远是推动运营和销售的改进。所有这些问题都需要基于此而提出。

第13章 购物篮分析

天地与我并生，万物与我为一。

——《老子》

俗话说：“商场如战场。”那是指商人之间的争斗，而这种争斗要通过商品这一道具完成。商品如同它们的主人一样，有不同的个性及命运，而货架就是商品展开“厮杀”的“战场”。有的商品一帆风顺，成功进入客户手中的购物篮，帮助自己的主人完成使命，而有的商品在门店中郁郁寡欢，始终与客户手中的购物篮无缘，最终落得一个被赶下货架，扫地出门的悲惨结局。

不同的商品决定了不同商店的命运，这点很好理解，可是不能让大家理解的是开在相同位置的商店，卖的是同样的商品，甚至销售价格也差不多，为什么别人能够活得好好的，自己的商店却每况愈下，最终落得凄凉倒闭的结局。商店倒闭的元凶很多，不了解客户手中的购物篮，从而失去客户的信赖是致命原因之一。本章将讨论零售业中应用广泛的购物篮分析（Market Basket Analysis），同时也将带领大家完成对前两章分析过的电商销售数据的购物篮分析。

13.1 购物篮分析概述

13.1.1 什么是购物篮分析

顾名思义，购物篮指的就是超级市场中供顾客购物时使用的装商品的篮子，当顾客付款时，这些购物篮内的商品被营业人员通过收款机一一登记结算并记录。所谓的购物篮分析就是通过这些购物篮所显

示的信息来研究顾客的购买行为。购物篮分析的主要目的在于找出什么样的东西应该放在一起，其目标是由顾客的购买行为来了解是什么样的顾客以及这些顾客为什么买这些产品，找出相关的关联（Association）规则，企业借由这些规则的挖掘获得利益并建立竞争优势。举例来说，零售店可通过此分析改变货架上的商品排列或是设计吸引客户的商业套餐，等等。通过购物篮分析挖掘出来的信息可以指导交叉销售、追加销售、商品促销、顾客忠诚度管理、库存管理和折扣计划。购物篮分析技术可以应用在下列问题上。

- 针对信用卡购物，能够预测未来顾客可能购买什么。
- 对于电信与金融服务业，通过购物篮分析能够设计不同的服务组合以扩大利润。
- 保险业能通过购物篮分析侦测出可能不寻常的投保组合并进行预防。
- 对病人而言，在疗程的组合上，购物篮分析能作为这些疗程组合是否会导致并发症的判断依据。

13.1.2 购物篮分析在超市中的应用

购物篮分析在大型超市的运营体系中占据了非常重要的地位。购物篮分析的结果不仅为门店的商品陈列、促销提供了有力的依据，更重要的是，通过它超市可以更充分了解客户的真实需求，并帮助供应商开发新的产品。利用购物篮分析可以完成如下应用。

- 商品配置分析：哪些商品可以一起购买，关联商品如何陈列、促销。
- 客户需求分析：分析顾客的购买习惯，包括顾客购买商品的时间、地点等。
- 销售趋势分析：利用数据仓库对品种和库存的趋势进行分析，选定需要补充的商品，研究顾客购买趋势，分析季节性购买模

式，确定降价商品。

- 帮助供应商改进老产品及开发新品：通过购物篮分析，根据客户的需求，开发新的产品或改进老产品及产品包装。

具体而言，超市根据商品在购物篮中出现的数量，可以分析商品包装和规格的机会点，为商品优化提供支持；而通过跟踪购物篮的变化，超市可以细分顾客消费行为及分析其对销售的影响；分析商品的销售趋势可以帮助制定有针对性的产品销售计划，提升顾客忠诚度。例如，超市通过商品一级品类的购物篮分析，发现用户经常购买的是水果和乳品，这两个品种十分契合下午茶场景，进而考虑超市周围是办公区，那么借此就可以理解超市的用户群体中，办公室白领和比较喜欢下午茶的人群比例有多少；同时通过分析发现果蔬和乳品是另一大受欢迎的组合，这两个品种十分契合居家生活场景，对这一数据的分析可以了解在超市的顾客群体中，做饭居家的用户比例有多少。有了这些数据佐证，超市一方面可以告诉运营部门在这几类品种上不断扩品和扩类，另一方面可以指导运营，在合适的时间（下午茶时间和下班买菜时间）进行合适的品类秒杀和促销。另外可以确定，如果要发放优惠券撬动销售额，那么发放什么品类优惠券合适；如果要进行促销，那么促销品中加入哪款商品能促进该促销品的销售；如果要制定下午茶活动，或者是菜市场活动，选哪些品类比较合适。

而通过对顾客在购买某一目标产品时出现在同一购物篮里的关联最紧密的商品以及相关购买金额、数量、出现概率的分析，可以为运营部门在交叉陈列、销售、商品促销或开发复合包装产品时提供依据。知名的大型连锁超市沃尔玛利用购物篮分析获得丰厚收益的故事很多，本书在这里简单介绍几个公开的案例。

第一个案例是，沃尔玛的采购人员在对一种礼品包装的婴儿护肤品进行购物篮分析时发现，该礼品的购买者基本都是一些商务卡客户，进一步了解才知道，商品都是作为礼品买来送人的，而不是原先

预想的“母亲”客户买给自己的孩子。因此，该商品的购买目的才得以明确，这样的购买目的信息为商品的进一步改进提供极大的帮助。

第二个案例是，通过对购物篮分析，沃尔玛发现，很多客户在购买沐浴用品时都会同时购买沐浴露一类商品。这条信息提示，可以针对这种需求，将毛巾、沐浴球、洗澡用品与沐浴露等沐浴主题商品进行捆绑销售或进行相关沐浴用品主题陈列。

第三个案例是，美国著名饮料制造商Welch's有一种专门为情人节定制的果汁饮料，但是如何展示、陈列这种情人节专用饮料始终是个难题。通过购物篮分析发现，这种商品与情人节专用的糖果（如巧克力）、贺卡具有商品关联关系。因此，这种饮料在情人节前可与情人节专用季节性通道的糖果、贺卡放在一起，并成为情人节商品整体规划的一部分。

13.1.3 购物篮分析实现

介绍购物篮分析的实现原理之前，首先需要理解几个定义。

1. 项集

购物篮也称为事务数据集，它包含属于同一个项集的项集合。在一篮子商品中的一件消费品即为一项（Item），若干项的集合为项集（Items），如{婴儿奶粉，尿片}构成一个二元项集。

2. 关联规则

X为先决条件，Y为相应的关联结果，用于表示数据内隐含的关联性，如关联规则：尿片→婴儿奶粉[支持度=3%，置信度=80%]。

3. 支持度

支持度（Support）是指在所有项集中，{X, Y}出现的可能性，即项集中同时含有X和Y的概率。假设全部交易中同时购买了婴儿奶粉和尿片的概率是 $3/100=0.03$ ，那么{尿片→婴儿奶粉}的支持度为3%。支持度指标作为建立强关联规则的第一个门槛，衡量了所考察关联规则在“量”上的多少。

4. 置信度

置信度（Confidence）表示在先决条件X发生的条件下，关联结果Y发生的概率：

$$\text{Confidence}(X \rightarrow Y) = \text{Support}(X, Y) / \text{Support}(X)$$

如果在1000条交易记录中，100个购买了尿片的顾客中有80个又购买了婴儿奶粉，即{尿片→婴儿奶粉}的置信度为 $(80/1000) / (100/1000) = 0.8$ 。这是生成强关联规则的第二个门槛，衡量了所考察的关联规则在“质”上的可靠性。

5. 提升度

提升度（Lift）表示“使用X的用户中同时使用Y的比例”与“使用Y的用户比例”的比值：

$$\begin{aligned} \text{Lift}(X \rightarrow Y) &= (\text{Support}(X, Y) / \text{Support}(X)) / \text{Support}(Y) \\ &= \text{Confidence}(X \rightarrow Y) / \text{Support}(Y) \end{aligned}$$

交易记录中，有100个顾客买了尿片，有200个顾客买了婴儿奶粉，有80个顾客同时购买了尿片和婴儿奶粉，那么{尿片→奶粉}的提升度为 $((80/1000) / (100/1000)) / (200/1000) = 0.8/0.2=4$ 。

该指标与置信度同样用于衡量规则的可靠性，可以看作是置信度的一种互补指标。

6. 出错率

出错率（Conviction）的意义在于度量规则预测错误的概率，表示X出现而Y不出现的概率：

$$\text{Conviction} (X \rightarrow Y) = (1 - \text{Support} (Y)) / (1 - \text{Confidence} (X \rightarrow Y))$$

那么{尿布→婴儿奶粉}的出错率为 $(1 - 200/1000) / (1 - 0.8) = 0.8/0.2 = 4$

以上各指标中，支持度是一种重要的度量，因为支持度很低的规则可能只是偶然出现，低支持度的规则多半也是无意义的。因此，支持度通常用来删去那些无意义的规则。置信度度量的是通过规则进行推理的可靠性。对于给定的规则 $X \rightarrow Y$ ，置信度越高，Y在包含X的事物中出现的可能性就越大，即Y在给定X下的条件概率越大。提升度反映了关联规则中A与B的相关性。如果提升度等于1，说明A和B没有任何关联；如果小于1，说明A和B是排斥的；只有大于1，才认为A和B是有关联的，但是在具体的应用之中，通常认为提升度大于3才算作值得认可的关联。一个大的提升度值是一个重要的指标，它表明一个规则是很重要的，并反映了商品之间的真实联系。选择哪个指标作为衡量指标，取决于数据分析的目标。如果数据分析是为了提高销量，那么选择考虑将支持度与置信度结合使用；如果数据分析目标是用于随机推荐，则考虑提升度会更有价值。

理解了上述指标后，基于前面的定义可以很方便地生成关联规则。规则生成可以简单地分为两步。

(1) 找频繁项集：在Apriori算法中，一个频繁项集的所有子集必须也是频繁的，即如果{婴儿奶粉，尿片}是频繁集，那么{婴儿奶粉}和{尿片}也得是频繁集，也就是说想要进入后续的规则整理，该商品被采购频率必须大于或等于一个阈值（即Apriori函数里的support参数）。 n 个项，可以产生 $2^n - 1$ 个项集，指定最小支持度就可以过滤掉非频繁项集，这样既能减轻计算负荷，又能提高预测质量。

(2) 找出频繁项集的规则： n 个项总共可以产生 $3^n - 2^{n+1} + 1$ 条规则，指定最小置信度过滤掉弱规则。经过上一步的过滤，剩余的项集已能满足最低支持度。计算各项之间的置信度作为候选规则，将这些候选规则与最小置信度阈值相比较，不能满足最小置信度的规则将被消除。

13.2 购物篮分析案例

13.2.1 Mlxtend库中Apriori算法使用介绍

购物篮分析最关键的就是构建频繁项集，之后根据它来构建规则。开源的Mlxtend库已经提供了相应的算法实现，首先导入相应的库，代码如下。

```
>>> from mlxtend.preprocessing import TransactionEncoder
>>> from mlxtend.frequent_patterns import apriori
>>> from mlxtend.frequent_patterns import association_rules
```

以上3行代码分别提供了交易记录编码转换、Apriori算法、关联规则的实现函数。对于如下的交易记录，首先需要对其进行编码转换，如图13.1所示。


```
>>> dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', \
                'Yogurt'], ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', \
                'Eggs', 'Yogurt'], ['Milk', 'Apple', 'Kidney Beans', \
                'Eggs'], ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', \
                'Yogurt'], ['Corn', 'Onion', 'Onion', 'Kidney Beans', \
                'Ice cream', 'Eggs']]
>>> te = TransactionEncoder()
>>> te_ary = te.fit(dataset).transform(dataset)
>>> df = pd.DataFrame(te_ary, columns = te.columns_)
>>> df
```

	Apple	Corn	Dill	Eggs	Ice cream	Kidney Beans	Milk	Nutmeg	Onion	Unicorn	Yogurt
0	False	False	False	True	False	True	True	True	True	False	True
1	False	False	True	True	False	True	False	True	True	False	True
2	True	False	False	True	False	True	True	False	False	False	False
3	False	True	False	False	False	True	True	False	False	True	True
4	False	True	False	True	True	True	False	False	True	False	False

图13.1 交易记录编码转换

TransactionEncoder（）函数首先生成了一个编码转换对象，之后将其转换为Apriori算法需要的格式。交易记录按规定格式编码后，就可以利用apriori（）函数计算频繁项集了，代码如下。

```
>>> apriori(df, min_support = 0.6)
      support  itemsets
0    0.8      (3)
1    1.0      (5)
...  ...      ...
9    0.6    (10, 5)
10   0.6    (8, 3, 5)
11  rows × 2 columns
```

默认情况下，apriori（）函数会返回项的列索引，如果想查看具体项的名称，那么可以指定use_colnames=True参数，代码如下。

```

apriori(df, min_support = 0.6, use_colnames = True)
      support  itemsets
0    0.8      (Eggs)
1    1.0      (Kidney Beans)
...    ...      ...
9    0.6      (Yogurt, Kidney Beans)
10   0.6      (Eggs, Onion, Kidney Beans)
11  rows × 2 columns

```

这里输出的结果和前面一段代码其实是一样的，只是这里给出了每项的具体名称。获得了频繁项集后，通常分析人员会根据支持度来选择与过滤，代码如下。

```

>>> frequent_itemsets = apriori(df, min_support = 0.6, use_colnames = True)
>>> frequent_itemsets['length'] = frequent_itemsets['itemsets'].\
      apply(lambda x: len(x))
>>> frequent_itemsets
      support  itemsets      length
0    0.8      (Eggs)          1
1    1.0      (Kidney Beans)      1
...    ...      ...          ...
9    0.6      (Yogurt, Kidney Beans)  2
10   0.6      (Eggs, Onion, Kidney Beans)  3
11  rows × 3 columns

```

可以通过如下代码来选择频繁项集中频繁项长度大于或等于2、支持度大于或等于0.8的频繁项集。

```

>>> frequent_itemsets[ (frequent_itemsets['length'] == 2) &\
      (frequent_itemsets['support'] >= 0.8) ]
      support  itemsets      length
5    0.8      (Kidney Beans, Eggs)  2

```

13.2.2 在线销售数据购物篮分析

本节将利用Mlxtend库提供的功能来对之前分析过的英国某电商网站的销售数据进行购物篮分析，该数据来自UCI机器学习数据集网站。首先导入数据，如图13.2所示。

```
>>> online = pd.read_csv('../data/Online_Retail.csv',\
    parse_dates = ['InvoiceDate'])
>>> online.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

图13.2 导入销售数据

数据集中一共有541909条交易记录，其中Description列提供了用户购物的描述，该列即是要分析的项。通常在进行购物篮分析前会对交易中经常出现的购物项进行一个简单的探索性分析。例如，图13.3展示了交易中出现最频繁的购物项，利用如下代码即可生成该图。

```
>>> online['des'] = online['Description'].str[:15]
>>> plt.figure(figsize=(8,4),dpi=120)
>>> color = plt.cm.inferno(np.linspace(0,1,20))
>>> online['des'].value_counts().head(20).plot.bar(color = color)
>>> plt.title('前 20 最频繁购买项')
>>> plt.ylabel('计数')
>>> plt.xlabel('购物项')
>>> plt.show()
```

也可以利用Squarify库提供的矩阵树图对其进行可视化，得到图13.4。

从上面的探索性分析得知ALARM CLOCK BAK、PLASTERS IN TIN等是顾客购买频率最高的商品，显然欧洲人购买礼物的习惯和国内还是有很大区别的。现在对数据有了一定了解，接下来就需要生成交易记录，因此首先要对Description列进行处理，去掉空格，去掉Description列为空的行，代码如下。

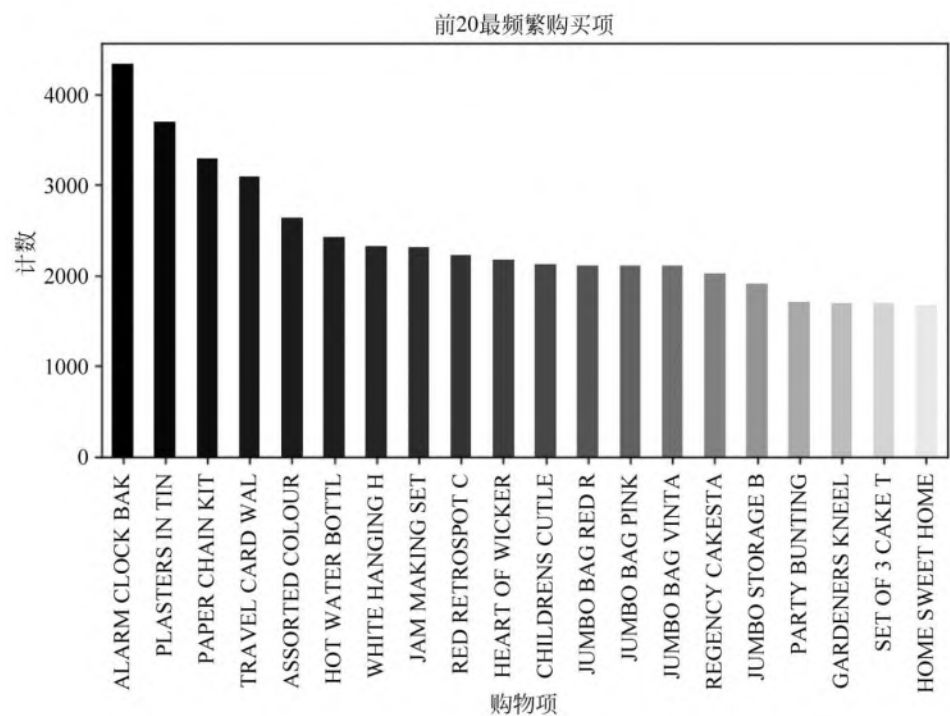


图13.3 频繁购买项



图13.4 矩阵树图

```
>>> online['Description'] = online['Description'].str.strip()
>>> online.dropna(axis = 0, subset = ['Description'], inplace = True)
```

此外，InvoiceNo列中带有“C”字符的都是取消了的购物记录，因此需要将这一部分记录去掉，代码如下。

```
>>> online['InvoiceNo'] = online['InvoiceNo'].astype('str')
>>> online = online[~online['InvoiceNo'].str.contains('C')]
```

完成了上述准备工作后，需要利用Description以及InvoiceNo列来生成交易记录，即一条交易的InvoiceNo对应到多个Description列。此时可以采用按此两列分组后再unstack（）的方法达到这一效果，代码如下。

```
>>> basket = (online.groupby(['InvoiceNo', 'Description'])\
                ['Quantity'].sum().unstack().reset_index().fillna(0)\
                .set_index('InvoiceNo'))
>>> basket
```

运行结果如图13. 5所示。

Description	4 PURPLE FLOCK DINNER CANDLES	50'S CHRISTMAS GIFT BAG LARGE	DOLLY GIRL BEAKER	I LOVE LONDON MINI BACKPACK	I LOVE LONDON MINI RUCKSACK	...	wrongly marked carton 22804	wrongly marked. 23343 in box	wrongly sold (22719) barcode	wrongly sold as sets	wrongly sold sets
InvoiceNo											
536365	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
536366	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...
C581568	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
C581569	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0

24446 rows x 4223 columns

图13. 5 生成交易记录

理论上，有了上述交易记录后就可以准备生成频繁项集了，不过由于Apriori算法要求交易记录中记录的内容是有没有购买，而这里记录的是数量，因此需要将其进行转换。此外，如果读者阅读UCI网站上的数据说明，可以发现其中一项购物项POSTAGE实际是邮费，所以需要将其去除。具体代码如下。

```
>>> def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1
>>> basket_sets = basket.applymap(encode_units)
>>> basket_sets.drop('POSTAGE', inplace = True, axis = 1)
```

现在得到了最终的交易记录，接下来就可以利用它生成频繁项集了。因为分析中感兴趣的频繁项集至少要有两项才具有意义，因此需要用下面的代码过滤掉这些只有1项的频繁项集，结果如图13.6和图13.7所示。

```
>>> frequent_itemsets = apriori(basket_sets, min_support = 0.02, \
    use_colnames = True)
>>> frequent_itemsets['length'] = frequent_itemsets['itemsets']. \
    apply(lambda x: len(x))
>>> frequent_itemsets
>>> frequent_itemsets[frequent_itemsets['length'] >= 2].head(20)
```

如果想查看支持度大于或等于0.03、项集长度大于1的频繁项集，则可以运行如下代码，结果如图13.8所示。

	support	itemsets	length
0	0.039066	(6 RIBBONS RUSTIC CHARM)	1
1	0.025280	(60 CAKE CASES VINTAGE CHRISTMAS)	1
...	...	...	...
228	0.022049	(WOODEN FRAME ANTIQUE WHITE , WOODEN PICTURE F...	2
229	0.022171	(PINK REGENCY TEACUP AND SAUCER, GREEN REGENCY...	3

230 rows x 3 columns

图13.6 生成频繁项集

	support	itemsets	length
192	0.026180	(ALARM CLOCK BAKELIKE GREEN, ALARM CLOCK BAKEL...	2
193	0.021353	(CHARLOTTE BAG PINK POLKADOT, RED RETROSPOT CH...	2
...	...	...	...
228	0.022049	(WOODEN FRAME ANTIQUE WHITE , WOODEN PICTURE F...	2
229	0.022171	(PINK REGENCY TEACUP AND SAUCER, GREEN REGENCY...	3

38 rows × 3 columns

图13.7 过滤频繁项集

```
>>> frequent_itemsets[ (frequent_itemsets['length'] == 2) &\
                        (frequent_itemsets['support'] >= 0.03) ]
```

	support	itemsets	length
199	0.031416	(GREEN REGENCY TEACUP AND SAUCER, ROSES REGENC...	2
203	0.033748	(JUMBO BAG RED RETROSPOT, JUMBO BAG PINK POLKA...	2

图13.8 支持度大于或等于0.03、项集长度大于1的频繁项集

从输出结果看，这里有两条频繁项集满足这一规则。如果分析目标为进行随机推荐，那么可以考虑运行如下代码查看提升度，结果如图13.9所示。

```
>>> rules = association_rules(frequent_itemsets, metric = "lift", \
                             min_threshold = 3)
>>> rules
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(ALARM CLOCK BAKELIKE GREEN)	(ALARM CLOCK BAKELIKE RED)	0.040088	0.042993	0.026180	0.653061	15.190043	0.024457	2.758433
1	(ALARM CLOCK BAKELIKE RED)	(ALARM CLOCK BAKELIKE GREEN)	0.042993	0.040088	0.026180	0.608944	15.190043	0.024457	2.454665
...	...	...	...	...	...	...	...	...	...
78	(GREEN REGENCY TEACUP AND SAUCER)	(PINK REGENCY TEACUP AND SAUCER, ROSES REGENCY...	0.041520	0.024503	0.022171	0.533990	21.792860	0.021154	2.093297
79	(ROSES REGENCY TEACUP AND SAUCER)	(PINK REGENCY TEACUP AND SAUCER, GREEN REGENCY...	0.043606	0.025894	0.022171	0.508443	19.635691	0.021042	1.981674

80 rows × 9 columns

图13.9 查看提升度

输出结果显示一共有80条规则满足要求，此时就需要和业务专家一起来研究哪些规则可以应用于网站，例如第一条规则实际是两种不同颜色的闹钟，数据分析人员可能无法理解为什么二者会经常同时出现。当然如果80条规则太多，还可以进一步过滤，比如添加置信度指标进行过滤，代码如下。

```
>>> rules[(rules['lift'] >= 6)&(rules['confidence'] >= 0.8)]
>>> rules
```

如图13.10所示，添加了置信度指标过滤后，最后的规则就只有3条。

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
10	(PINK REGENCY TEACUP AND SAUCER)	(GREEN REGENCY TEACUP AND SAUCER)	0.031334	0.041520	0.025894	0.826371	19.902916	0.024593	5.520268
74	(PINK REGENCY TEACUP AND SAUCER, GREEN REGENCY...	(ROSES REGENCY TEACUP AND SAUCER)	0.025894	0.043606	0.022171	0.856240	19.635691	0.021042	6.652717
75	(PINK REGENCY TEACUP AND SAUCER, ROSES REGENCY...	(GREEN REGENCY TEACUP AND SAUCER)	0.024503	0.041520	0.022171	0.904841	21.792860	0.021154	10.072447

图13.10 添加置信度进行过滤

13.3 留给读者的思考

虽然购物篮分析属于数据分析师的工作，但是通过前面的案例分析可以发现，理解业务对数据分析是至关重要的。数据分析永远服务于业务目标，因此懂一些业务知识将对数据分析师的工作大有帮助。

第14章 概率分布

在终极的分析中，一切知识都是历史；

在抽象的意义下，一切科学都是数学；

在理性的世界里，所有的判断都是统计学。

——*C. R. Rao*

数据分析中使用的许多统计工具和技术都基于概率。所谓概率就是事件发生的可能性，其取值范围为从0（事件从未发生）到1（事件总是发生）。而分析数据时，通常会将数据集中的变量作为随机变量，即变量的值无法预先确定，仅以一定的可能性（概率）取值的量。概率分布则描述了随机变量是如何分布的，它告诉我们随机变量最有可能的值和不太可能的值。而在统计学中，存在一系列具有不同形状的精确定义的概率分布，数据分析时经常用它们来模拟不同类型的随机事件。本章将讨论一些常见问题的概率分布以及如何在Python中使用它们。

14.1 随机数

在讨论概率分布之前，首先来看看Python中的随机数，因为它是后续研究概率分布的基础。自从计算机发明后，便产生了一种全新的解决问题的方式：使用计算机对现实世界进行统计模拟。使用统计模拟，首先要产生随机数，在Python中，`numpy.random`模块提供了丰富的随机数生成函数。例如，可以用如下代码生成0到1之间的任意随机数。

```
>>> np.random.random(size = 5)
array([0.96617957, 0.2342853, 0.84400828, 0.83488971, 0.84322257])
```

如下代码可以生成一定范围内的随机整数。

```
>>> np.random.randint(1, 10, size = 5)
array([9, 5, 4, 4, 9])
```

不过需要注意的是，计算机生成的随机数其实是伪随机数，是由一定的方法计算出来的。因此，可以按下面方法指定随机数生成的种子，这样的好处是以后重复计算时，能保证得到相同的模拟结果。例如，如下代码中seed=1的时候，两次得到的随机数都是相同的。

```
>>> np.random.seed(1)
>>> print('seed = 1 :', np.random.randint(1, 10, size = 5))
>>> np.random.seed(2)
>>> print('seed = 2 :', np.random.randint(1, 10, size = 5))
>>> np.random.seed(1)
>>> print('seed = 1 :', np.random.randint(1, 10, size = 5))
seed = 1 : [6 9 6 1 1]
seed = 2 : [9 9 7 3 9]
seed = 1 : [6 9 6 1 1]
```

在NumPy中，不仅可以生成上述简单的随机数，还可以按照一定的统计分布生成相应的随机数。例如，`np.random.binomial()`、`np.random.poisson()`、`np.random.exponential()`和`np.random.normal()`分别对应二项分布、泊松分布、指数分布和正态分布的随机数生成函数。如下代码则随机生成了一个符合均匀分布的随机数。

```
>>> np.random.uniform(0,10)
6.531200998466402
```

除了按照以上各种方式生成随机数，还可以从指定序列中随机抽取数据。例如，如下代码实现从列表[2, 4, 6, 9]中随机抽取一个数。

```
>>> random.choice([2,4,6,9])
9
```

14.2 常见的概率分布

14.2.1 均匀分布

在概率论和统计学中，均匀分布又称矩形分布，它是对称概率分布，在相同长度间隔的分布概率是等可能的。均匀分布由两个参数 a 和 b 定义，它们是数轴上的最小值和最大值，通常缩写为 $U(a, b)$ 。

如下代码即可模拟100000个符合区间[0, 10]均匀分布的值。

```
>>> uniform_data = stats.uniform.rvs(size = 100000, \
                                     loc = 0, scale = 10)
```

利用如下代码，则可以得到其概率密度，如图1 4. 1所示。

```
>>> pd.DataFrame(uniform_data).plot(kind = "density",
                                   figsize = (6, 3),
                                   xlim = (-1, 11),
                                   fontsize = 14,
                                   legend = None);
```

上述代码在区间[0, 10]内生成了100000个符合均匀分布的数据点。从概率密度图也可以发现，对均匀分布而言，概率密度基本上是水平的，这意味着任何给定值具有相同的发生概率。而概率密度曲线下的面积总和始终等于1。

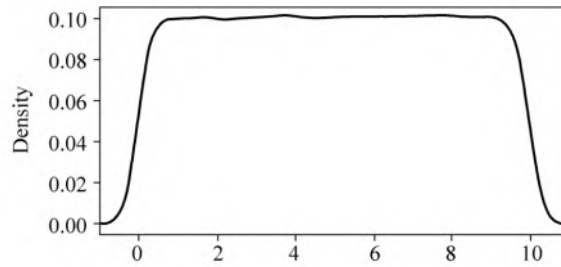


图14.1 均匀分布概率密度图

SciPy 中常用的概率函数有 4 类，分别以 `stats.distribution.rvs()`、`stats.distribution.cdf()`、`stats.distribution.ppf()`、`stats.distribution.pdf()` 形式存在。其中，`stats.distribution.rvs()` 代表根据指定的分布生成随机数。`distribution` 指明分布类型，如均匀分布用 `uniform`。`stats.distribution.rvs()` 中的参数则取决于分布类型，对均匀分布而言，需要指定起点、终点以及生成多少个随机数。而 `stats.distribution.cdf()` 代表累积概率分布函数，用于确定从某个分布中抽取某观察值小于指定值的概率。几何意义上，`stats.distribution.cdf()` 就是 X 轴上某个值左侧的分布密度曲线面积。在上面的均匀分布中，观察到的值为 $0 \sim 2.5$ 的概率为 25%，也就是有 75% 的可能性将落在 $2.5 \sim 10$ 的范围内，可以用 `stats.distribution.cdf()` 函数计算如下。

```
>>> stats.uniform.cdf(x = 2.5, loc = 0, scale = 10)
0.25
```

`stats.distribution.ppf()` 的作用与 `stats.distribution.cdf()` 函数恰好相反，是根据概率求变量（也称为分位点函数）。例如，著名的庞加莱面包案中，想知道 90% 的情况下买到的面包会小于多少克，就可以用这个方法。针对当前的均匀分布，运行如下代码将得到输出 4.0。

```
>>> stats.uniform.ppf(q = 0.4, loc = 0, scale = 10)
4.0
```

最后，`stats.distribution.pdf()` 为概率密度函数，它计算给定 x 值的概率密度（分布的高度）。由于均匀分布是平坦的，因此该范围内的所有 x 值将具有相同的概率密度，超出范围的 x 值的概率密度为0，示例代码如下。

```
>>> for x in range(-1,12,3):
    print("{}处概率密度: ".format(x),\
          stats.uniform.pdf(x, loc = 0, scale = 10))
-1 处概率密度: 0.0
2 处概率密度: 0.1
5 处概率密度: 0.1
8 处概率密度: 0.1
11 处概率密度: 0.0
```

14.2.2 正态分布

正态分布（Normal Distribution）也称“常态分布”，又名高斯分布（Gaussian Distribution），最早由棣莫弗在求二项分布的渐近公式中得到。正态分布曲线像一只倒扣的钟，两头低，中间高，左右对称。大部分数据集中在平均值，小部分在两端。正态分布是自然界最常见的分布，不管是人的身高、手臂长度、肺活量，还是我们的考试成绩，都符合正态分布。神奇数字黄金分割比例同样适用于正态分布。正态分布的期望值 μ 决定了其位置，标准差 σ 决定了分布的幅度。图14.2表示的正态分布曲线中，正负1个标准差的范围内包含了68%的数据。

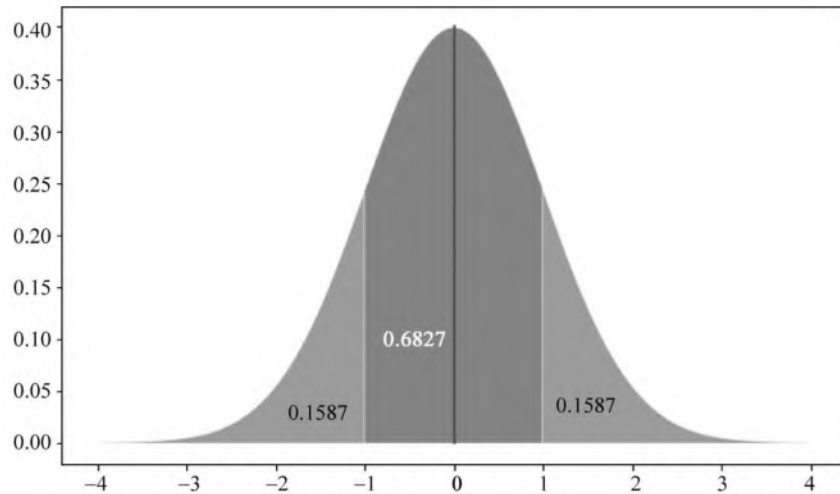


图14.2 正态分布曲线

要完成这一计算，可以利用`stats.distribution.cdf()`函数计算累积概率分布。计算与绘图代码分别如下。

```
# 正负 1 个标准差的累积概率分布
>>> prob_under_minus1 = stats.norm.cdf(x = -1, loc = 0, scale = 1)
>>> prob_over_1 = 1 - stats.norm.cdf(x = 1, loc = 0, scale = 1)
>>> between_prob = 1 - (prob_under_minus1 + prob_over_1)
>>> print(prob_under_minus1, prob_over_1, between_prob)
0.15865525393145707 0.15865525393145707 0.6826894921370859
# 绘图
>>> plt.figure(figsize = (10,6))
>>> plt.fill_between(x = np.arange(-4, -1, 0.01), \
                    y1 = stats.norm.pdf(np.arange(-4, -1, 0.01)), \
                    facecolor = 'red', alpha = 0.35)
>>> plt.fill_between(x = np.arange(1, 4, 0.01), \
                    y1 = stats.norm.pdf(np.arange(1, 4, 0.01)), \
                    facecolor = 'red', alpha = 0.35)
>>> plt.fill_between(x = np.arange(-1, 1, 0.01), \
                    y1 = stats.norm.pdf(np.arange(-1, 1, 0.01)), \
                    facecolor = 'blue', alpha = 0.35)
>>> plt.plot((0,0), (0,0.4))
>>> plt.text(x = -1.8, y = 0.03, s = round(prob_under_minus1, 4))
>>> plt.text(x = -0.5, y = 0.1, s = round(between_prob, 4))
>>> plt.text(x = 1.4, y = 0.03, s = round(prob_over_1, 4));
```

14.2.3 二项分布

二项分布是由伯努利提出的概念。重复 n 次独立的伯努利试验，在每次试验中只有两种可能的结果，而且两种结果发生与否互相对立，并且相互独立，与其他各次试验结果无关，事件发生与否的概率在每一次独立试验中都保持不变，则这一系列试验总称为 n 重伯努利试验。当试验次数为1时，二项分布服从0-1分布。二项分布是一个离散分布，所以使用概率质量函数（Probability Mass Function, PMF）来表示 k 次成功的概率。以投硬币为例，如下代码可以模拟10000次投掷公平硬币，每次投10枚硬币的结果，其可视化如图14.3所示。

```
>>> fair_coin_flips = stats.binom.rvs(n=10, \
                                     p=0.5, \
                                     size=10000)
>>> print(pd.crosstab(index="counts", columns=fair_coin_flips))
>>> pd.DataFrame(fair_coin_flips).hist(range=(-0.5,10.5), bins=11);
```

col_0	0	1	2	3	4	5	6	7	8	9	10	
row_0	counts	12	102	428	1216	2030	2435	2115	1125	448	79	10

从图14.3所示直方图中可以发现该分布基本上是对称的，成功和失败的可能各占50%，这与正态分布类似。

如果投掷的硬币是非公平硬币呢？例如正面概率是80%，那么可以通过指定参数 $p=0.8$ 模拟，代码如下。

```
>>> biased_coin_flips = stats.binom.rvs(n=10, \
                                       p=0.8, \
                                       size=10000)
>>> print(pd.crosstab(index="counts", columns=biased_coin_flips))
>>> pd.DataFrame(biased_coin_flips).hist(range=(-0.5,10.5), bins=11);
```

col_0	3	4	5	6	7	8	9	10	
row_0	counts	9	50	276	893	1972	3078	2662	1060

可视化结果如图14.4所示。

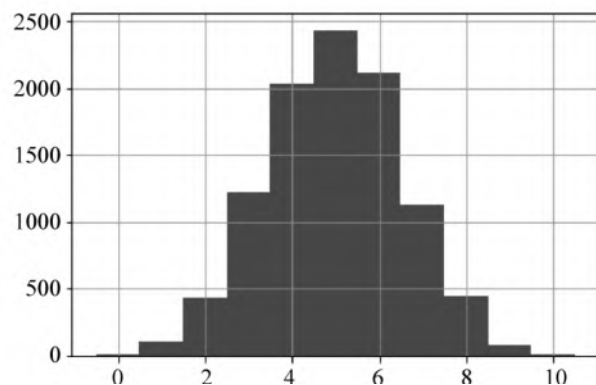


图14.3 公平硬币

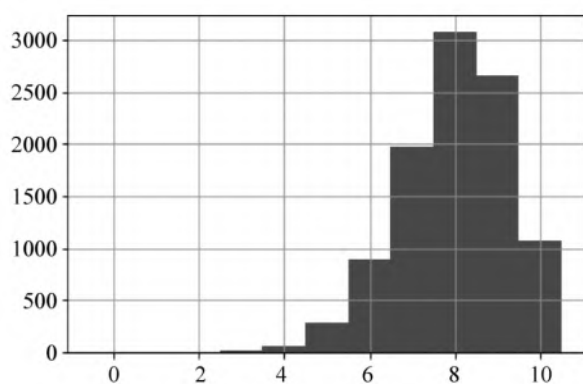


图14.4 不公平硬币

如果想知道试验中正面有5个或更少的概率是多少，又应该如何计算呢？此时可以用累积概率分布函数`stats.distribution.cdf()`计算，代码如下。

```
>>> stats.binom.cdf(k=5,n=10,p=0.8)
0.032793497599999964
```

其中， $k=5$ 代表成功5次或以下， $n=10$ 代表投掷10次。同理，如果想知道正面大于或等于9次的概率是多少，只需要用1减去正面小于或等于8次的累积概率分布就可以，代码如下。

```
>>> 1 - stats.binom.cdf(k=8,n=10,p=0.8)
0.37580963840000003
```


对于连续概率分布，使用密度函数`stats.distribution.pdf()`来检查指定值的概率密度，而对离散分布则使用`stats.distribution.pmf()`概率质量函数。例如，想获得成功5次的概率则使用如下代码。

```
>>> stats.binom.pmf(k = 5, n = 10, p = 0.5)
0.24609375000000025
```

这与前面模拟10000次投掷硬币的结果十分接近，其中5次正面朝上刚好2435次，概率密度为 $2435/10000=0.2435$ 。

14.2.4 泊松分布

泊松分布用于描述单位时间内随机事件发生次数的概率分布，它也是离散分布。例如等公交车，假设这些公交车的到来是独立且随机的（当然这不是现实），前后车之间没有关系，那么在1小时中到来的公交车数量就符合泊松分布。日常生活中大量事件是有固定频率的，如某医院平均每小时出生10个婴儿，某客服中心每分钟接到10个电话，某网站平均每分钟有10000次访问。它们的特点是可以预估这些事件的总数，但是没法知道具体的发生时间。下面来模拟一个每时间单元发生一次的泊松分布，代码如下。其可视化如图14.5所示。

```
>>> np.random.seed(12)
>>> arrival_rate_1 = stats.poisson.rvs(size = 10000, mu = 1)
>>> print(pd.crosstab(index = "counts", columns = arrival_rate_1))
>>> pd.DataFrame(arrival_rate_1).hist(range = (-0.5, \
    max(arrival_rate_1) + 0.5), \
    bins = max(arrival_rate_1) + 1, figsize = (10, 6));
```

从图14.5可以看出，当事件发生频率（ μ ）低的时候很少出现事件同时发生的情况，此时分布显著左偏。

不过如果提高事件发生频率，如将 μ 提高到10，此时分布就变得更加对称，如图14.6所示。

```
>>> np.random.seed(12)
>>> arrival_rate_1 = stats.poisson.rvs(size = 10000, mu = 10)
>>> print(pd.crosstab(index = "counts", columns = arrival_rate_1))
>>> pd.DataFrame(arrival_rate_1).hist(range = (-0.5, \
max(arrival_rate_1) + 0.5), \
bins = max(arrival_rate_1) + 1, figsize = (10, 6));
```

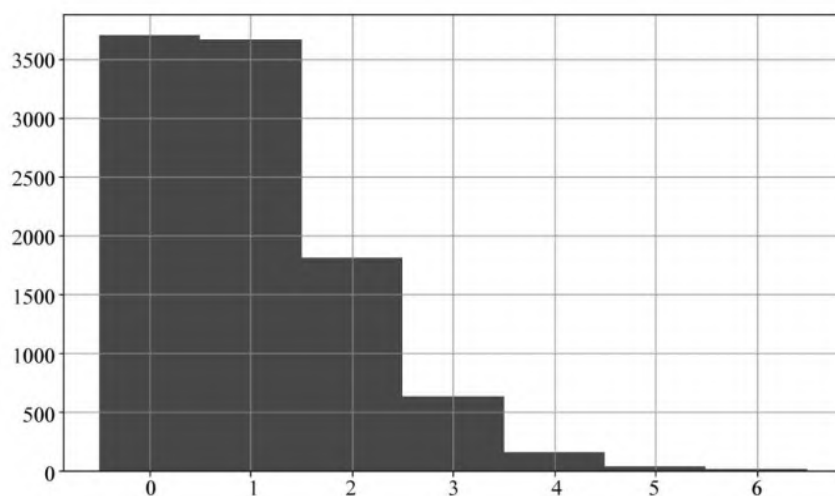


图14.5 泊松分布 ($\mu=1$)

接下来再看一个生活中的例子，假设某家销售零食的网店平均每周卖出30件坚果零食，请问该网店的最佳库存量应该是多少？对于此问题，假定不存在季节以及促销因素，那么可以近似认为，该问题满足以下3个条件：顾客购买坚果是小概率事件；购买坚果的顾客之间是独立的，即不会互相依赖或影响；顾客购买坚果的概率是稳定的。在统计学上，如果某类事件满足上述3个条件，就称它服从泊松分布。因此，可以用泊松分布的累积概率分布进行计算，代码如下。

```
>>> rv = stats.poisson(30)
>>> [(i, rv.cdf(i)) for i in range(35, 45)]
[(35, 0.8426165255696685),
 (36, 0.8803733589751277),
 (37, 0.9109870076822568),
 (38, 0.935155677714201),
 (39, 0.9537469623541579),
 (40, 0.9676904258341258),
 (41, 0.9778929600877606),
 (42, 0.9851804845546427),
 (43, 0.9902648039501418),
 (44, 0.993731385356164)]
```

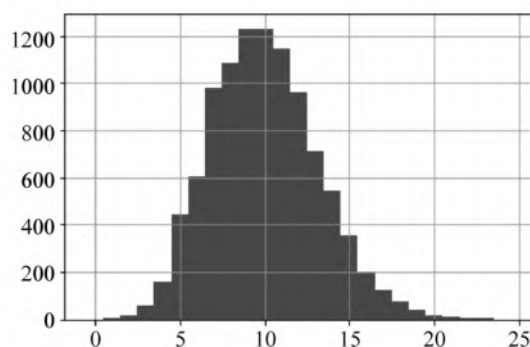


图14.6 泊松分布 ($\mu=10$)

从输出可以看出只要存货为39，那么网店有95%的概率不会缺货，如果商家想降低库存，那么就需要承担更大的缺货概率。

14.2.5 几何分布与指数分布

几何分布 (Geometric Distribution) 是离散型概率分布，其中一种定义为：在 n 次伯努利试验中，试验 k 次才得到第一次成功的概率。具体而言就是前 $k-1$ 次皆失败，第 k 次成功的概率。下面利用几何分布来模拟一下投掷公平硬币，要多少次试验才能出现正面朝上，运行如下代码，将得到如图14.7所示的结果。

```
>>> np.random.seed(12)
>>> flips_till_heads = stats.geom.rvs(size=10000, p=0.5)
>>> print(pd.crosstab(index="counts", columns=flips_till_heads))
>>> pd.DataFrame(flips_till_heads).hist(range=(-0.5,\
      max(flips_till_heads)+0.5), bins=max(flips_till_heads)+1);
col_0      1      2      3      4      5      6      7      8      9     10     11     12     15
row_0
counts  5052  2485  1212  629  329  141  69  38  14  16  10   4   1
```

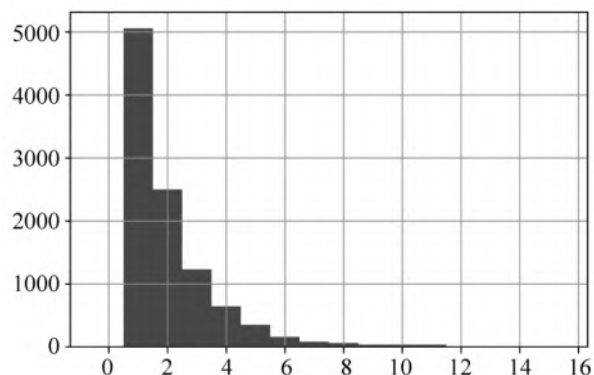


图14.7 投掷硬币直方图

图14.7的结果与预期非常一致，大多数情况下只需要投1次或2次就可以成功，要投5次以上才出现正面朝上的概率极低。模拟的10000次试验中，最多是15次才成功。下面用`stats.distribution.cdf()`函数来检查一下至少5次才成功的概率，代码如下。

```
>>> first_five = stats.geom.cdf(k=5, p=0.5) # 在前5次就成功的概率
>>> 1 - first_five
0.03125
```

很显然，0.03125这个概率很低，这也与前面的模拟试验一致。如果想知道刚好2次成功的概率，此时则可以用`stats.distribution.pmf()`函数计算，代码如下。

```
>>> stats.geom.pmf(k=2, p=0.5)
0.25
```

下面再来看一下指数分布，在概率理论和统计学中，指数分布（也称为负指数分布）是描述泊松过程中的事件之间的时间的概率分布，即事件以恒定平均速率连续且独立地发生的过程。它是几何分布的连续模拟，具有无记忆的关键性质。除了用于分析泊松过程外，还可以在其他各种环境中找到指数分布。指数分布用来描述独立随机事件发生的时间间隔，以等公交车为例，两辆车到来的时间间隔就符合指数分布。假设公交到达时间服从参数 $\lambda=6$ 的指数分布，那么现在在公交站15分钟以内有公交车到达的概率就是：

```
>>> prob_1 = stats.expon.cdf(x = 0.25, scale = 1/6) # scale = 1/lambda
>>> prob_1
0.7768698398515702
```

14.3 点估计与置信区间

统计推断是分析样本数据以从中了解总体的过程。在数据分析中，经常对某些总体的特征感兴趣，但收集整个数据总体可能不可行。例如，经济普查中对收入的调查，对每个人进行调查并不可行。现实场景中的经济普查都是对一部分人进行普查，如抽样10000人，并使用这些数据来推断总体。

14.3.1 点估计

点估计正是基于样本数据的总体参数估计方法。例如，如果想知道人群的平均年龄，那么可以对登记人群进行调查，然后使用他们的平均年龄的点估计作为总体的估计。这里样本的平均值称为样本平均值，样本平均值通常与总体平均值不完全相同。这种差异可以由许多因素造成，包括调查设计不佳、抽样方法有偏见以及随机从总体中抽取样本所固有的误差。下面通过生成一个人群年龄数据的总体，然后从中抽取样本以估算均值来说明这一问题，代码如下。

```
>>> np.random.seed(1)
>>> population_ages1 = stats.poisson.rvs(loc = 18, mu = 35, size = 150000)
>>> population_ages2 = stats.poisson.rvs(loc = 18, mu = 10, size = 100000)
>>> population_ages = np.concatenate((population_ages1, \
    population_ages2))
>>> population_ages.mean()
42.98932
```

上述代码模拟生成了250000个年龄数据作为总体，其平均值为42.98932，这里使用了泊松分布来生成随机数，估计具体试验，不同读者可以选择不同的分布函数。接下来随机抽样500个样本，计算样本均值，代码如下。

```
>>> np.random.seed(2)
>>> sample_ages = np.random.choice(a = population_ages, size = 500)
>>> print(sample_ages.mean())
>>> population_ages.mean() - sample_ages.mean()
42.904
0.08531999999999584
```

从输出结果可以得知，基于500个样本的抽样得出的点估计与真实的总体平均年龄非常接近，差别仅为0.085左右。这也说明通过相对小的样本也可以得出较准确的对总体的估计。

14.3.2 抽样分布与中心极限定理

许多统计过程都假定数据遵循正态分布，因为正态分布具有良好的属性，如对称性，并且大多数数据都聚集在一个平均值标准偏差内。不幸的是，现实世界中的数据通常不是正态分布的。而样本的分布趋向于反映总体的分布，这意味着从具有偏斜分布的人群中获取的样本也将趋于偏斜。以前面创建的年龄数据为例，将其可视化，如图14.8所示。

```
>>> pd.DataFrame(population_ages).hist(bins = 58,  
                                       range = (17.5,75.5),  
                                       figsize = (10,6))  
>>> print(stats.skew(population_ages));
```

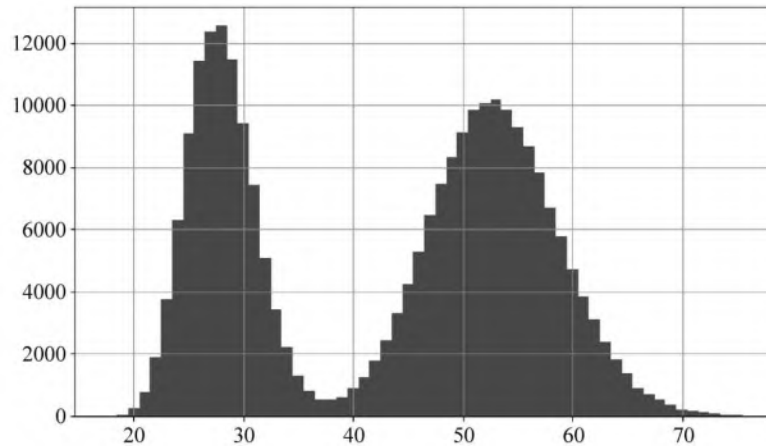


图14.8 年龄数据直方图

虽然该分布偏度不大，但图14.8中数据显然不是正态分布，而是有两个峰。那么抽样也应该与总体的形状和偏斜大致相同，将样本数据可视化，如图14.9所示。

```
>>> pd.DataFrame(sample_ages).hist(bins = 58,  
                                   range = (17.5,75.5),  
                                   figsize = (10,6));  
>>> print(stats.skew(sample_ages))
```

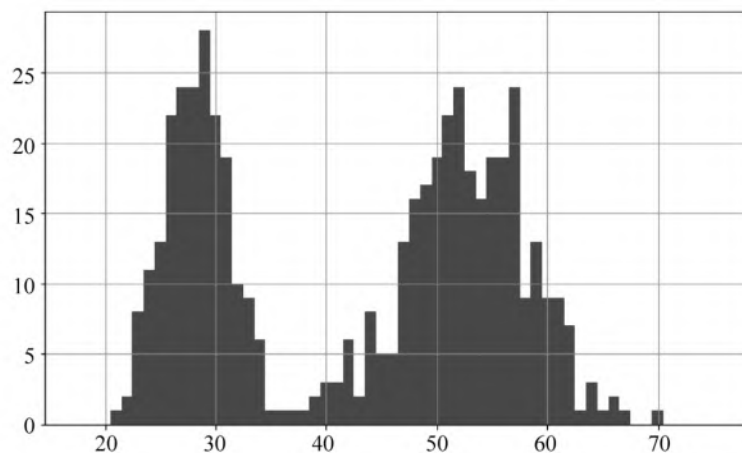


图14.9 样本年龄直方图

显然，样本的分布形状与总体的形状大致相同。由此表明不能将假定正态分布应用于此数据集，因为从上面的图形看出该分布不是正态分布。那非正态分布的数据应该怎么进行参数估计呢？幸运的是，有数学家提出了中心极限定理。中心极限定理是概率论最重要的理论之一，它是许多统计分析方法的基础。简单来说，中心极限定理指的是给定一个任意分布的总体，每次从这些总体中随机抽取 n 个抽样，一共抽取 m 次，然后把这 m 组抽样分别求出平均值，最终得到的平均值的分布接近正态分布。为了说明这一点，如下代码模拟了从总体中抽取200个样本来创建样本分布，然后对平均值进行200个点估计，结果可视化如图14.10所示。

```
>>> np.random.seed(10)
>>> point_estimates = []
>>> for x in range(200):
    sample = np.random.choice(a = population_ages, size = 500)
    point_estimates.append(sample.mean())
>>> pd.DataFrame(point_estimates).plot(kind = "density", \
    figsize = (10,6), \
    xlim = (41,45));
```

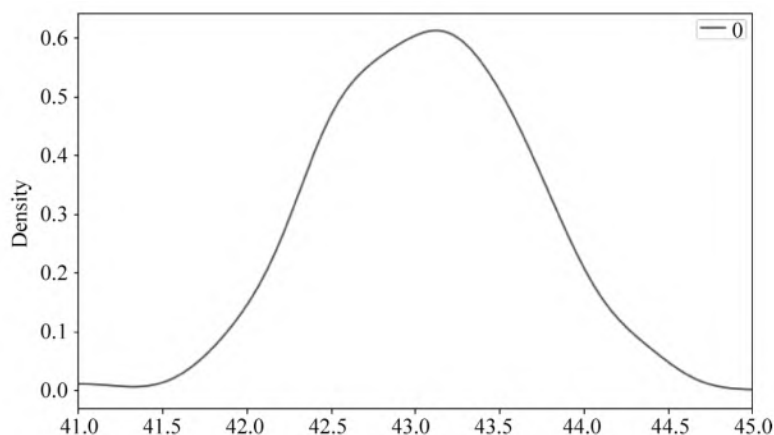


图14.10 抽样分布

从图14.10可以看出，尽管样本是从非正态分布的总体中抽取，但是抽样分布却基本上符合正态分布。同时，抽样分布的均值也和总体的均值基本一致，抽样次数越多，最终估计的参数就越接近总体。针对当前讨论的问题，两者之差只有0.0728左右。

```
>>> population_ages.mean() - np.array(point_estimates).mean()  
- 0.07279000000000048
```

14.3.3 置信区间

点估计可以让我们大致了解总体参数（如均值），但估计是容易出错的，并且采取多个样本来获得改进的估计值可能并不可行。因此，统计学家们又引入了置信区间（Confidence Interval），一个概率样本的置信区间是对这个样本的某个总体参数的区间估计。置信区间展现的是这个参数的真实值有一定概率落在测量结果的周围的程度。置信区间给出的是被测量参数的测量值的可信程度，即前面所要求的“一定概率”。这个概率被称为置信度或置信水平。如果想拥有一个95%的机会通过点估计和相应的方法来捕获真实的总体参数置信区间，那么可以将置信度设置为95%。较高的置信度会得到范围更广的置信区间。在已经得到点估计的情况下，将其加上和减去边际误差就可以得到置信区间。如下代码对置信区间的计算进行了展示。

```

>>> np.random.seed(10)
sample_size = 1000
>>> sample = np.random.choice(a = population_ages, size = sample_size)
>>> sample_mean = sample.mean()
>>> z_critical = stats.norm.ppf(q = 0.975) # z - critical value 双尾
>>> print("z - critical value:", z_critical)
>>> pop_stdev = population_ages.std()
>>> margin_of_error = z_critical * (pop_stdev/math.sqrt(sample_size))
>>> confidence_interval = (sample_mean - margin_of_error, \
                           sample_mean + margin_of_error)
>>> print("Confidence interval:", confidence_interval)
z - critical value: 1.959963984540054
Confidence interval: (41.827654507799416, 43.46634549220058)

```

上述代码中的sample_mean就是点估计，而边际误差通过首先计算关键值 $z_{critical}$ ，然后由 $z_{critical} * (pop_stdev/math.sqrt(sample_size))$ 计算得到。那么应该如何理解上面的置信区间呢？图14.11提供了很好的说明，置信区间代表了有多大概率将总体均值包含在区间之内，代码如下。

```

>>> np.random.seed(12)
>>> sample_size = 1000
>>> intervals = []
>>> sample_means = []
>>> for sample in range(25):
    sample = np.random.choice(a = population_ages, size = sample_size)
    sample_mean = sample.mean()
    sample_means.append(sample_mean)
    z_critical = stats.norm.ppf(q = 0.975)
    pop_stdev = population_ages.std()
    stats.norm.ppf(q = 0.025)
    margin_of_error = z_critical * \
        (pop_stdev/math.sqrt(sample_size))
    confidence_interval = (sample_mean - margin_of_error, \
        sample_mean + margin_of_error)
    intervals.append(confidence_interval)

>>> plt.figure(figsize = (10,6))
>>> plt.errorbar(x = np.arange(0.1, 25, 1),
    y = sample_means,
    yerr = [(top - bot)/2 for top, bot in intervals],
    fmt = 'o')
>>> plt.hlines(xmin = 0, xmax = 25,
    y = 43.0023,
    linewidth = 2.0,
    color = "red");

```

在不知道总体标准差的情况下，通常采用样本标准差来代替总体标准差，同时采用 t 分布来计算置信区间。因此，使用 `stats.t.ppf()` 函数来计算 `t_critical`，并用 `sample_stdev` 代替总体标准差，代码如下。

```

>>> np.random.seed(10)
>>> sample_size = 25
>>> sample = np.random.choice(a = population_ages, size = sample_size)
>>> sample_mean = sample.mean()
>>> t_critical = stats.t.ppf(q = 0.975, df = 24)
>>> print("t - critical value:", t_critical)
>>> sample_stdev = sample.std(ddof = 1)
>>> sigma = sample_stdev/math.sqrt(sample_size)
>>> margin_of_error = t_critical * sigma
>>> confidence_interval = (sample_mean - margin_of_error, \
    sample_mean + margin_of_error)
>>> print("Confidence interval:", confidence_interval)
t - critical value: 2.0638985616280205
Confidence interval: (39.95797358704254, 51.72202641295747)

```

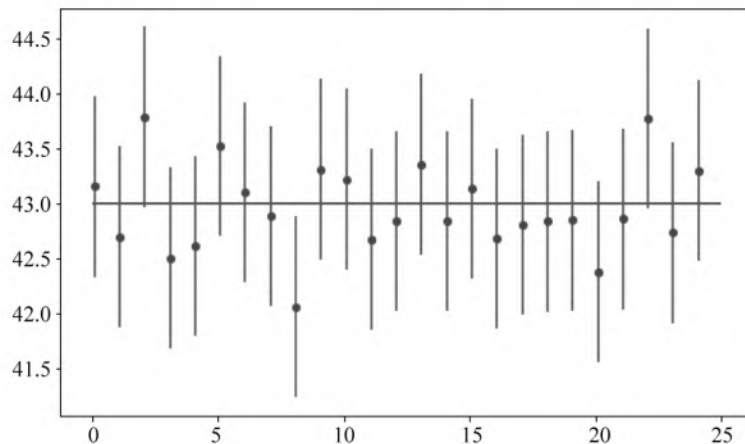


图14.11 置信区间

用样本标准偏差代替总体标准偏差，最终结果是得到较宽的置信区间。

14.4 留给读者的思考

理解概率论，无论是对数据分析还有将来的机器学习都大有帮助，这里推荐读者进一步学习以下书籍。

- 陈希儒. 概率论与数理统计[M]. 合肥：中国科学技术大学出版社，2009.

- Thomas Haslwanter. *An Introduction to Statistics with Python : With Applications in the Life Sciences* [M]. Basel, Switzerland: Springer, 2016.
- Allen B. Downey. *Think Stats* [M]. 2nd Edition. CA, USA: O'Reilly Media, 2014.
- Allen B. Downey. *Think Bayes* [M]. CA, USA: O'Reilly Media, 2013.

第15章 假设检验

正如一个法庭宣告某一判决为“无罪”而不为“清白”，统计检验的结论也应为“不拒绝”而不为“接受”。

——*Jan Kmenta*

有些人看起来是好人，实际是坏人；有些人看起来是坏人，实际是好人。好人还是坏人，不但与他实际是好人还是坏人有关，还与我们的眼睛有关。所谓“甲之蜜糖，乙之砒霜也”，判断一个事物的好坏，不但和我们的判断标准有关，还和判断方法有关。统计学则是利用假设检验（又称统计检验）这一框架来进行判断。本章将介绍假设检验的基本原理，同时也将讨论在Python中如何进行一些常用的检验。

15.1 假设检验概述

15.1.1 初识假设检验

假设检验是数理统计学中根据一定假设条件由样本推断总体的一种方法。事先对总体参数或分布形式做出某种假设，然后利用样本信息来判断原假设是否成立，采用逻辑上的反证法，依据统计上的小概率原理得出结论。所谓反证法，就是如果要证明一个结论是正确的，那么先假设这个结论是错误的，然后以这个结论是错误的为前提条件进行推理，推理出来的结果与假设条件矛盾，这个时候就说明这个假设是错误的，也就是这个结论是正确的。

前面一段关于假设检验的论述非常拗口，下面用一个真实的例子——女士品茶来说明。女士品茶这个故事最早出现在统计学家Fisher的著作*The Design of Experiment*中。书中提到，有位女士一次喝茶时提出了一个有趣的观点：把茶加到奶里和把奶加到茶里，最后得到的奶茶的味道是不一样的。大部分人都觉得这位女士在瞎说，但是Fisher教授提出了要用科学的方法去证明到底一样还是不一样（牛人就是牛人，所以遇到问题要多想想别人为什么那么说，而不是不经思考就拒绝）。接下来，我们具体看一下Fisher的实验方法：他调配出了8杯其他条件一模一样而仅仅是倒茶、倒奶顺序相反的茶，其中每类各4杯。然后他让女士品尝之后告诉他哪4杯是先加奶的，剩下的就都是先加茶的了。

在分析实验结果的时候，Fisher运用了这样的逻辑：首先假设女士没有这个能力（这个假设被称为原假设或空假设），如果女士很准确地鉴别了这8杯茶，那就说明在原假设成立的情况下，发生了非常反常的现象（小概率事件），以至于说明原假设是令人怀疑的。从统计学上来说，如果在原假设成立的前提下，发生了非常小概率的事件，那就有理由怀疑原假设的真实性。我们把上面这个过程叫做假设检验。

15.1.2 假设检验的步骤

了解了假设检验的思想后，本节将讨论假设检验的具体步骤。

1. 提出空假设和备择假设

空假设（ H_0 ）一般是要推翻的论点，备择假设（ H_1 ）则是要证明的论点。以上面的女士品茶为例， H_0 和 H_1 分别如下。

- H_0 ：把茶加到奶里和把奶加到茶里得到的奶茶是一样的。

- H1: 把茶加到奶里和把奶加到茶里得到的奶茶是不一样的。

2. 构造检验统计量

检验统计量是根据样本观测结果计算得到的样本统计量，并以此对空假设和备择假设做出决策。具体而言，假设总体 $X \sim N(\mu, \sigma^2)$ ， $X_1, X_2, \dots, X_n$ 为取自该总体 X 的样本， n 为样本总量， $\bar{X}$ 为样本均值， S^2 为样本方差。那么有：

$$\text{统计量 } U = \frac{\bar{X} - \mu}{\sigma} \sqrt{n} \sim N(0, 1)$$

$$\text{统计量 } T = \frac{\bar{X} - \mu}{S} \sqrt{n} \sim t(n-1)$$

$$\text{统计量 } \chi^2 = \frac{(n-1)S^2}{\sigma^2} \sim \chi^2(n-1)$$

这3种统计量及其对应分布分别为 z 检验 (z -test)、 t 检验 (t -test) 和卡方检验。

- z 检验一般用于大样本（即样本容量大于30）平均值差异性检验。它是用标准正态分布的理论来推断差异发生的概率，从而比较两个平均数的差异是否显著。
- t 检验主要用于样本含量较小（如 $n < 30$ ），总体标准差 σ 未知的正态分布。 t 检验是用 t 分布理论来推论差异发生的概率，从而比较两个平均数的差异是否显著。
- 卡方检验是统计样本的实际观测值与理论推断值之间的偏离程度，实际观测值与理论推断值之间的偏离程度决定了卡方值的大小。卡方值越大，二者偏差越大；反之，二者偏差越小；若两个值完全相等时，卡方值就为0，表明理论值完全符合。实际运用中可以参考图15.1来完成不同类型检验的选择。

3. 根据要求的显著性水平求临界值和拒绝域

还记得前面提到的反常事件（小概率事件）吗？如果小概率事件发生了，就表示空假设是错误的，可是具体多小的概率才算是小概率呢？一般这个概率为0.05，也就是5%，如果一件事情发生的概率小于或等于5%，我们就认为这是一个小概率事件，0.05就是显著性水平，用 α 表示。显著性水平把概率分布分为两个区间：拒绝区间和接受区间。最后计算出来的结果落在拒绝区间，就可以拒绝空假设；如果落在了接受区间，就需要接受空假设， $1-\alpha$ 则称为置信水平（置信度）。

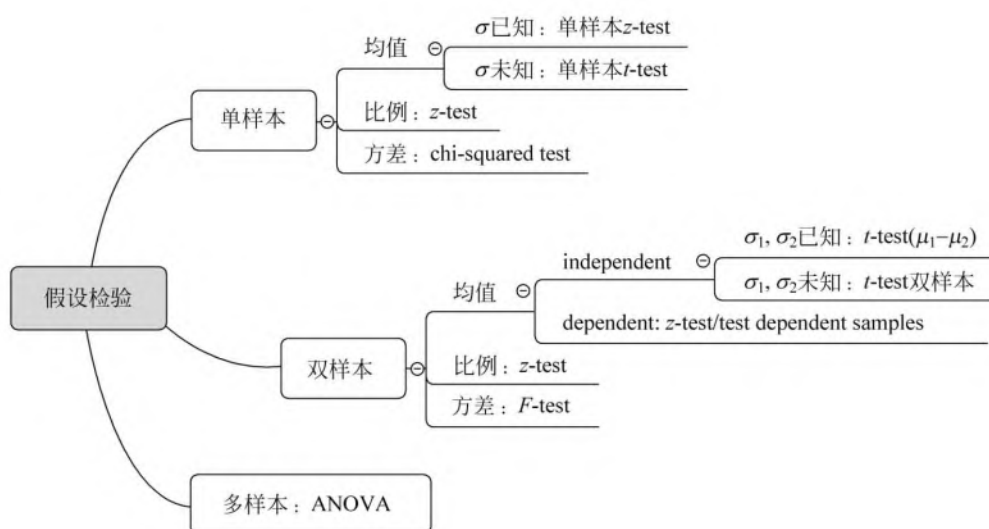


图15.1 假设检验选择

4. 计算检验统计量

根据前面选择的检验统计量类型，计算对应的检验统计量的值。除此之外，我们还可以根据样本量得出 p 值， p 值就是实际样本中小概率事件的具体概率值。

5. 决策

比较计算出来的检验统计量与临界值和拒绝域，如果值落在了拒绝域内，那就要拒绝空假设，否则接受空假设。比较计算出来的 p 值和显著性水平 α 值，如果 $p \leq \alpha$ ，则拒绝空假设，否则接受空假设。

15.1.3 假设检验中的 I 类错误与 II 类错误

前面探讨了假设检验的基本思想，本节将介绍假设检验中的两类错误。假设检验的最终目的是去伪存真，那么它对应的两类错误就是弃真存伪。第一类错误叫做弃真错误，通俗一点说就是漏诊，就是本来生病了（假设是正确的），但是没有检测出来，所以给拒绝掉了；第二类错误是存伪错误，通俗一点就是误诊，就是本来没病（假设是错误的），结果诊断说生病了（假设是正确的），所以就把假设给接受了。

为了更形象地说明这两类错误，以图15.2为例，正常情况下，该实例的假设检验应该为：

- H_0 ：没有怀孕；
- H_1 ：怀孕了。

左图：这类错误为弃真错误，也就是空假设为没有怀孕，但是检验的结果落在拒绝域，因而拒绝没有怀孕的空假设，认定左图里的男士怀孕了，而事实上图里面的男士根本不可能怀孕，这就犯了第一类错误——弃真。右图：这类错误为存伪错误，也就是空假设为没有怀孕，检验结果落在接受域，所以接受没有怀孕的空假设，认定右图中的女士没有怀孕，而事实上图片里的女士是怀孕的，这就犯了第二类错误——存伪。



图15.2 假设检验中的两类错误

15.2 Python中的假设检验

15.2.1 单样本 t -test

单样本 t -test 属于参数检验，用于比较样本数据与一个特定数值之间的差异情况。例如，想比较成都的工资水平与全国工资水平是否存在显著差异，首先构造数据如下。

```
>>> import numpy as np
>>> import pandas as pd
>>> import scipy.stats as stats
>>> import math
>>> np.random.seed(1)
>>> population_salary1 = stats.poisson.rvs(loc = 1300, mu = 3500, \
    size = 150000)
>>> population_salary2 = stats.poisson.rvs(loc = 1300, mu = 2000, \
    size = 100000)
>>> population_salary = np.concatenate((population_salary1, \
    population_salary2))
>>> cd_salary1 = stats.poisson.rvs(loc = 1300, mu = 3200, size = 30)
>>> cd_salary2 = stats.poisson.rvs(loc = 1300, mu = 1700, size = 20)
>>> cd_salary = np.concatenate((cd_salary1, cd_salary2))
>>> print(population_salary.mean())
4199.829132
>>> print(cd_salary.mean())
3891.94
```

模拟数据中成都的平均工资为3891.94元，全国的平均工资为4199.8元，现在想知道二者有无显著差异，就可以用stats模块中的ttest_1samp（）函数进行假设检验，代码如下。

```
>>> stats.ttest_1samp(a = cd_salary, popmean = population_salary.mean())
Ttest_1sampResult(statistic = -2.897004249304246,
pvalue = 0.00561748480065776)
```

上述代码的输出结果中： $p = 0.0056$ ，显然此时在显著性水平为0.05时应该拒绝空假设——成都的工资水平和全国工资水平没有显著差异。此外，也可以用stats.t.ppf（）函数验证，代码如下。

```
>>> print(stats.t.ppf(q = 0.025, df = 49))
>>> print(stats.t.ppf(q = 0.975, df = 49))
-2.0095752344892093
2.009575234489209
```

显然 t 值在该区间以外，即95%的置信区间以外，通过t.interval（）函数也可以对此进行验证。

```
>>> sigma = cd_salary.std()/math.sqrt(50) # 样本 std/样本数
>>> stats.t.interval(0.95, df = 49, \
                    loc = cd_salary.mean(), \
                    scale = sigma)
(3680.5119838234955, 4103.368016176504)
```

上述代码输出的成都平均工资区间为3681~4103元，与前面的判断一致。

15.2.2 双样本 t -test

双样本 t -test是根据样本数据对两个样本来自的两个独立总体的均值是否有显著差异进行判断。它需要满足以下3个条件。

- 随机抽样，所有观测应该随机地从目标总体中抽出。

- 正态分布，每个样本来自的总体必须满足正态分布。
- 方差齐性，均数比较时，要求两总体方差相等。

还是以上面的工资数据为例，现在构造一个重庆的工资数据样本如下。

```
>>> np.random.seed(2)
>>> cq_salary1 = stats.poisson.rvs(loc = 1300, mu = 3100, size = 30)
>>> cq_salary2 = stats.poisson.rvs(loc = 1300, mu = 1100, size = 20)
>>> cq_salary = np.concatenate((cq_salary1, cq_salary2))
>>> print(cq_salary.mean())
3600.78
```

现在样本中成都的平均工资为3891.94元，重庆的平均工资为3600.78元，请问两者有显著差别吗？从数字上看有差别，但是需要用双样本t-test来确认一下，代码如下。

```
>>> stats.ttest_ind(a = cd_salary, b = cq_salary, equal_var = False)
Ttest_indResult(statistic = 1.6628806744879354,
pvalue = 0.09975252036818426)
```

输出结果中 $p = 0.09975$ ，如果显著性水平为0.05，此时无法拒绝空假设。实际上也就是说是有9.975%的可能看到成都和重庆的工资数据会有这样的差异。

15.2.3 配对 t -test

配对 t -test在医疗行业应用广泛，例如在医疗试验中：配对的两个受试对象分别接受两种不同的处理；同一受试对象接受两种不同的处理；同一受试对象处理前后的结果进行比较（即自身配对）；同一对象的两个部位给予不同的处理。下面以某种减肥药的试验为例进行说明，假设有如下的受试者体重数据。

```

>>> np.random.seed(3)
>>> before = stats.norm.rvs(scale=10, loc=60, size=100)
>>> after = before + stats.norm.rvs(scale=5, loc=-1.25, size=100)
>>> weight_df = pd.DataFrame({"weight_before":before,\
                               "weight_after":after, "weight_change":after - before})
>>> weight_df.describe()

```

	weight_before	weight_after	weight_change
count	100.000000	100.000000	100.000000
mean	58.913629	57.498908	-1.414721
std	10.693536	10.468844	4.418131
min	30.842622	32.167345	-11.401968
25 %	51.399515	50.033280	-4.529386
50 %	58.210360	55.887337	-1.687446
75 %	66.769302	64.518491	1.550781
max	81.581493	88.057492	12.130562

从描述性统计结果看，吃了该药后体重平均要减少1.41千克左右。这是减肥药有效，还是只是偶然事件呢？利用配对 t -test 验证如下。

```

>>> stats.ttest_rel(before, after)
Ttest_relResult(statistic = 3.2020800238221665,
pvalue = 0.0018350355143263457)

```

从配对 t -test 结果来看， $p = 0.0018$ ，应该拒绝空假设，也就是说减肥药是有效的。

15.2.4 卡方检验

卡方检验属于非参数检验的范畴，主要是比较两个及两个以上样本率（构成比）以及两个分类变量的关联性分析。其根本思想就是比较理论频数和实际频数的吻合程度或拟合优度问题。上面的描述非常晦涩难懂，这里用几个简单的卡方检验的例子来说明，例如：

- 男性和女性的线上生鲜食品购买习惯有无差别？
- 不吃早饭对体重下降有无影响？
- 疗效是否与不同药品有关？

- 不同广告渠道获得的用户，次日留存率是否有差别？

以上问题都可以利用卡方检验来回答，这里以最后一个问题为例来说明卡方检验的应用。假设某移动应用推广使用了3个不同渠道，分别是门户网站的在线广告、微信推广、微博推广。现在想判断以上3个不同渠道获得的用户的留存率是否有差别。基于此问题构造如下数据。

```
>>> data = pd.read_csv('../data/reguser.txt', sep = '\t')
>>> data.head()
   type  day  reg  stay
0  weibo day1 2504  752
1  weibo day2 2718  701
2  weibo day3 2538  692
3  weibo day4 2207  540
4  weibo day5 2004  46
```

以上数据是每天该应用的新注册人数和次日继续使用用户数。因此，可以计算用户流失数据如下。

```
>>> data['lost'] = data['reg'] - data['stay']
>>> data.head()
>>> group = data.groupby('type')
>>> observed = group.agg(np.sum)
>>> observed
      reg  stay  lost
type
webad  11570  3173  8397
weibo  15113  3901 11212
wechat 18244  4899 13345
```

现在想知道来自不同渠道的用户的次日留存是否相互独立，利用卡方检验，代码如下。

```
>>> stats.chi2_contingency(observed = observed[['stay', 'lost']])
(9.359095286322784,
 0.0092832122619623,
 2,
 array([[3083.39328244, 8486.60671756],
        [4027.59919425, 11085.40080575],
        [4862.00752332, 13381.99247668]]))
```

输出结果中， $p = 0.00928$ ，因此应该拒绝空假设，即用户的渠道类别影响了次日留存情况。计算不同渠道的留存率如下。

```
>>> observed['stay']/observed['reg']
type
webad      0.274244
weibo      0.258122
wechat     0.268527
dtype: float64
```

似乎通过门户网站获取的用户有更高的留存率，据此，营销部门可以在门户网站相对增加推广。

15.3 留给读者的思考

统计检验属于一门专门的学科，本章的介绍只是一点皮毛。感兴趣的读者可以进一步阅读专业的概率论与统计的相关书籍以加深理解。这里向读者推荐以下两本书籍：

- David Freedman, Robert Pisani, Roger Purves, et al. 统计学[M]. 魏宗舒, 施锡铨, 林举干, 等译. 北京: 中国统计出版社, 1997.
- John A. Rice. 数理统计与数据分析[M]. 田金方, 译. 北京: 机械工业出版社, 2011.

第16章 一名数据分析师的游戏上线之旅

距离已经消失，
要么创新，要么死亡。

——托马斯·彼得斯

移动应用间的竞争越来越激烈，每天都有新的应用上线。你所在的公司也不例外，项目组经过了5个月的紧张开发，终于完成了一款新的移动游戏应用，明天就准备上线了。作为一名数据分析师，你将遇到哪些问题呢？应用第一天上线后会不会因为大量用户涌入，导致应用启动时间比要求的3秒长？次日留存率是否达到目标？应该在游戏的哪一个关卡引入微信分享提示？游戏内购定价是1.99元还是0.99元更好？本章将开始一名数据分析师的游戏上线之旅！

16.1 游戏启动时间是否超过目标

管理团队为了随时了解公司的游戏软件上线情况，一开始就设计了一些简单的运营指标来跟踪游戏运营情况。其中一个指标就是：启动游戏后欢迎界面平均等待时间不能超过3秒。测试团队经过了大量测试，已经验证无论是在iOS还是Android平台，欢迎界面等待时间都不超过3秒。游戏上线后也一切运行正常，然而不妙的是上线6小时后，游戏的上一小时平均启动时间开始超过了管理层设定的3秒平均值。紧张的项目经理跑来问你——团队的数据分析师，这是否可能是用户过多导致启动延迟？

16.1.1 启动时间是否超过3秒

要回答此问题，首先来看一下模拟的游戏运营数据，代码如下。

```
# 导入库
>>> import numpy as np
>>> import pandas as pd
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> import scipy.stats as stats
>>> import statsmodels.stats.weightstats as wstats
>>> from matplotlib.ticker import FuncFormatter
>>> %matplotlib inline
# 绘图设置
>>> plt.style.use('fivethirtyeight')
>>> plt.rcParams['font.sans-serif'] = ['SimHei']
>>> plt.rcParams['axes.unicode_minus'] = False
# 生成模拟数据
>>> observation_hours = 7          # 游戏已经上线 7 小时
>>> expected_installs = 60         # 预期每小时有 60 人安装
>>> loading_times = []
>>> loading_times_averages = []
>>> np.random.seed(12)            # 随机种子
>>> for ix, installs in enumerate(np.random.poisson(\
    lam = expected_installs, size = observation_hours)):
    # 1 小时内每用户的启动时间
    loading_times.append(np.random.gamma(\
        shape = 3, scale = .95, size = installs))
    # 该小时的平均启动时间
    loading_times_averages.append(loading_times[ix].mean())
```

上述代码模拟了7个小时的数据，假定每小时装机数是符合参数为60的泊松分布，而启动时间则是符合参数分别为3和0.95的伽马分布。利用如下代码，将上面数据进行可视化。如图16.1所示，前6小时一切正常，每小时的平均启动时间都在3秒以内，但进入第7小时，似乎突然启动时间就超过了目标值。

```

>>> fig, ax = plt.subplots(figsize = (10,3.4))
>>> plt.plot(loading_times_averages, marker = 'o')
>>> plt.title('平均启动时间', fontdict = {'size':18})
>>> plt.plot([0,observation_hours - 1],[3,3], '-- ', color = 'black')
>>> plt.xlabel('游戏已上线时间', fontsize = 14)
>>> plt.ylabel('启动时间(秒)', fontsize = 14)
>>> plt.ylim(2.3,3.5)

>>> fig, ax = plt.subplots(figsize = (10,3.4))
>>> plt.plot([len(x) for x in loading_times], marker = 'o')
>>> plt.title('每小时装机数', fontsize = 18)
>>> #plt.title('每小时装机', fontdict = {'size':18})
>>> plt.xlabel('游戏已上线时间', fontdict = {'size':14})
>>> plt.ylabel('装机数', fontsize = 14);

```

当下需要考察的是最后一小时的数据，对其启动时间可视化（见图16.2），代码如下。

```

>>> fig, ax = plt.subplots(figsize = (10,4))
>>> plt.bar(range(loading_times[6].shape[0]),
           loading_times[6], align = 'center')
>>> plt.plot([-1,100],[3,3], '-- ', color = '0.3')
>>> plt.xlim(-1,61)
>>> plt.title('过去一小时用户启动时间', fontsize = 16)
>>> plt.ylabel('秒', fontsize = 14)
>>> plt.xticks([]);

```

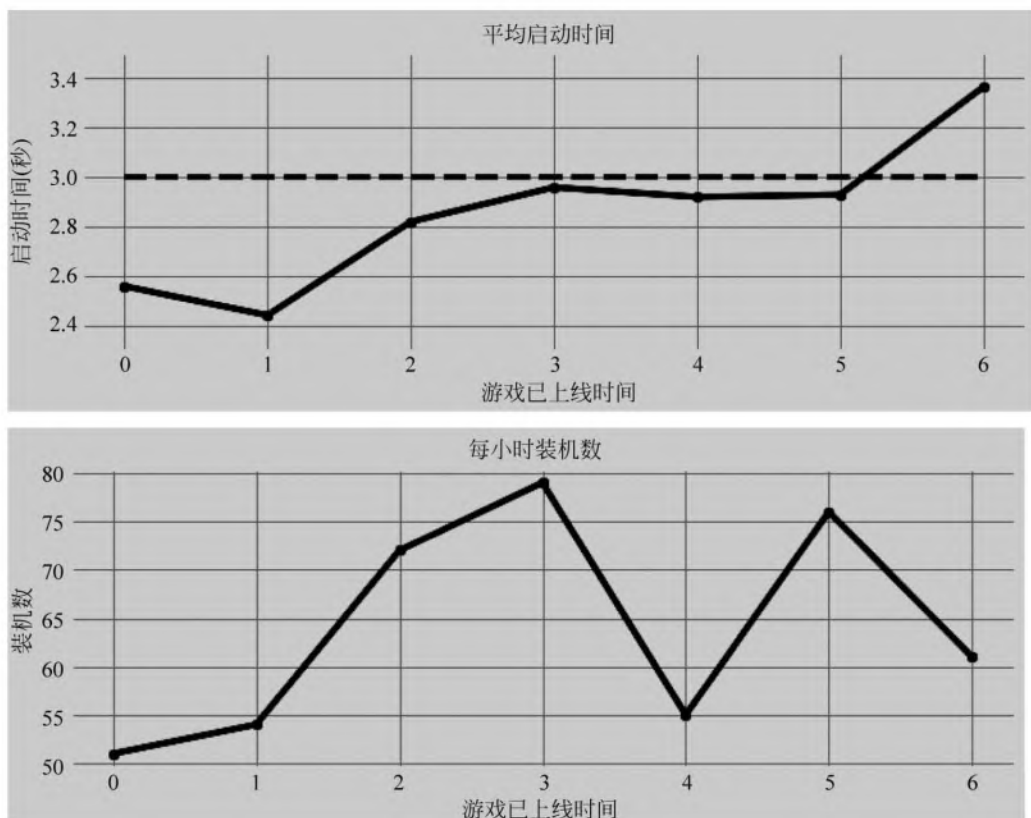


图16.1 游戏平均启动时间与装机数

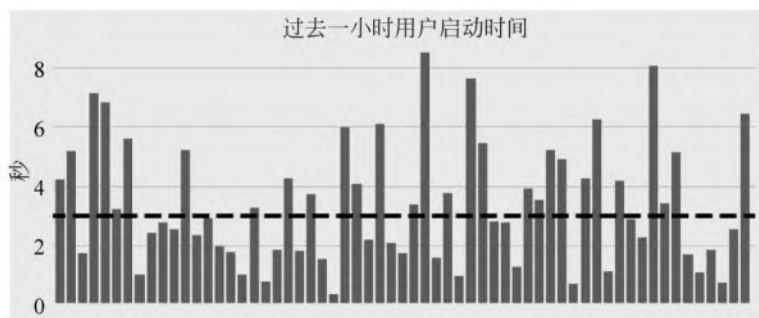


图16.2 过去一小时启动时间

虽然确实有许多用户的启动时间在3秒以上，但是管理层关心的问题是：过去一小时的问题是否具有普适性。作为项目经理，对这一异常感到紧张是很正常的事，然而作为一名数据分析师，对此提出的问题却应该是：最近一小时的平均启动时间是否来自平均启动时间不超

过3秒的启动时间分布？很显然，这是一个统计检验问题，那么该采用何种假设检验呢？上一章的图15.1给出了答案。

根据图15.1得知当前要做的是针对单样本均值的 t -test，具体假设如下。

- 空假设：前一小时总体的观察平均值小于或等于3秒。
- 备择假设：前一小时总体的观察平均值大于3秒。
- 检验：单样本，单尾 t -test， $\alpha = 0.05$ 。

因此，采用 t -test 验证如下。

```
>>> import scipy.stats as stats
>>> t, p = stats.ttest_1samp(loading_times[6], popmean=3.0)
>>> print('t-statistic = {t} \np-value = {p}'.format(t=t, p=p/2))
t-statistic = 1.38005787853313
p-value = 0.08634481804336265
```

SciPy 中的 stats 模块提供了单样本 t -test 函数。利用 `stats.ttest_1samp()` 函数，输入样本值和总体均值即可得到 $t=1.38$ ， $p=0.086$ 。很显然，根据检验结果无法拒绝空假设，那么前一个小时总体的观察平均值小于或等于3秒就应该是正确的。此时你应该可以去拍拍项目经理的肩膀，放心地说：“老大，目前的情况应该很正常，出现启动时间大于3秒应该是偶然事件。”当然前面是直接利用 SciPy 中的函数实现 t 值和 p 值的计算，读者也可以通过如下的公式手工计算 t 值后，查 t 分布表得到 p 值。

$$t = \frac{\bar{x} - \mu_0}{SE} = \frac{\bar{x} - \mu_0}{s / \sqrt{n}}$$

其中， $\bar{x}$ 为样本平均数； μ_0 为总体平均数；SE为测量标准误差； s 为样本标准差； n 为样本数。

16.1.2 构造启动时间监测图

项目经理已经很满意地离开，但是如果每次出现启动时间大于3秒，他就很紧张地到你这里来确认是不是偶然事件，显然不是问题的解决之道。那应该如何来处理这一问题呢？在前面章节读者已经学习过置信区间的概念，回忆一下置信区间CI的计算公式：

$$CI = \bar{x} \pm t_{\text{critical}} \cdot SE$$

其中 $\bar{x}$ 为样本平均数； t_{critical} 为样本平均数与总体平均数的离差统计量这；SE为测量标准误差。

根据此公式我们可以为启动时间构造一个置信区间（Confidence Level=95%），当平均启动时间在置信区间之外时才应该特别关注。以第7小时的启动时间为例，可以计算置信区间如下。

```
>>> import scipy.stats as stats
>>> x = loading_times[6]
>>> SE = stats.sem(x) # Standard Error
>>> t_c = stats.t.ppf(0.975, df = len(x) - 1) # t - critical
>>> CI = x.mean() - t_c * SE, x.mean() + t_c * SE
>>> CI
(2.837226159651679, 3.887130030235891)
```

有兴趣的读者可以自行验证是不是所有的第7小时启动时间都在2.837~3.887秒这个区间之内。类似地，可以利用如下代码计算游戏自从上线以来每个小时的平均启动时间的置信区间，结果如图16.3所示。

```

>>> loading_performance = pd.DataFrame(loading_times_averages,
    columns = ['loadingtime'])
>>> loading_performance['installs'] = [len(x) for x in loading_times]
>>> loading_performance['SE'] = [stats.sem(x) for x in loading_times]
>>> loading_performance['t_critical'] = stats.t.ppf(0.975,
    df = loading_performance['installs'] - 1)
>>> loading_performance['CI_low'] = \
    loading_performance['loadingtime'] - \
    loading_performance['t_critical'] * loading_performance['SE']
>>> loading_performance['CI_high'] = \
    loading_performance['loadingtime'] + \
    loading_performance['t_critical'] * loading_performance['SE']
>>> loading_performance

```

	loadingtime	installs	SE	t_critical	CI_low	CI_high
0	2.559482	51	0.196072	2.008559	2.165660	2.953305
1	2.443319	54	0.183372	2.005746	2.075521	2.811118
2	2.820219	72	0.198834	1.993943	2.423755	3.216683
3	2.958101	79	0.175907	1.990847	2.607897	3.308305
4	2.919596	55	0.257128	2.004879	2.404086	3.435105
5	2.928829	76	0.190776	1.992102	2.548783	3.308875
6	3.362178	61	0.262437	2.000298	2.837226	3.887130

图16.3 每小时的平均启动时间的置信区间

同时可以将上面的数据用如下代码可视化，如图16.4所示。这段代码使用了区域填充图来绘制启动时间的置信区间，关于 `plt.fill_between()` 函数的使用在第9章已经有介绍，这里不再讨论。

```

>>> fig, ax = plt.subplots(figsize = (10,3.4))
>>> plt.plot(loading_times_averages, marker = 'o')
>>> plt.title('平均启动时间', fontsize = 16)
>>> plt.plot([0, observation_hours - 1], [3,3], '-- ', color = '0.3')
>>> plt.fill_between(loading_performance.index, \
    loading_performance['CI_low'], loading_performance['CI_high'], \
    where = loading_performance['CI_low'] \
    < loading_performance['CI_high'], \
    alpha = 0.2, interpolate = True, linewidth = 0.0)
>>> plt.xlabel('游戏已上线时间', fontsize = 14)
>>> plt.ylabel('启动时间(秒)', fontsize = 14)
>>> plt.ylim(2,4)
>>> plt.subplots(figsize = (10,2))
>>> plt.plot(loading_performance['installs'], marker = 'o')
>>> plt.ylim(0, loading_performance['installs'].max() * 1.1)
>>> plt.title('每小时装机数', fontsize = 14)
>>> plt.xlabel('游戏已上线时间', fontsize = 14)
>>> plt.ylabel('装机数', fontsize = 14);

```

显然，图16.4很好地展示了游戏在前7小时平均启动时间指标，一切尽在掌握中！接下来数据分析师可以考虑为项目经理建立如图16.5所示的仪表盘，供项目经理随时监控启动时间是否异常。

利用仪表盘，运营团队可以很方便地对装机数和平均启动时间进行监控。图16.5模拟了后面24小时的游戏运营情况，它由如下代码生成。

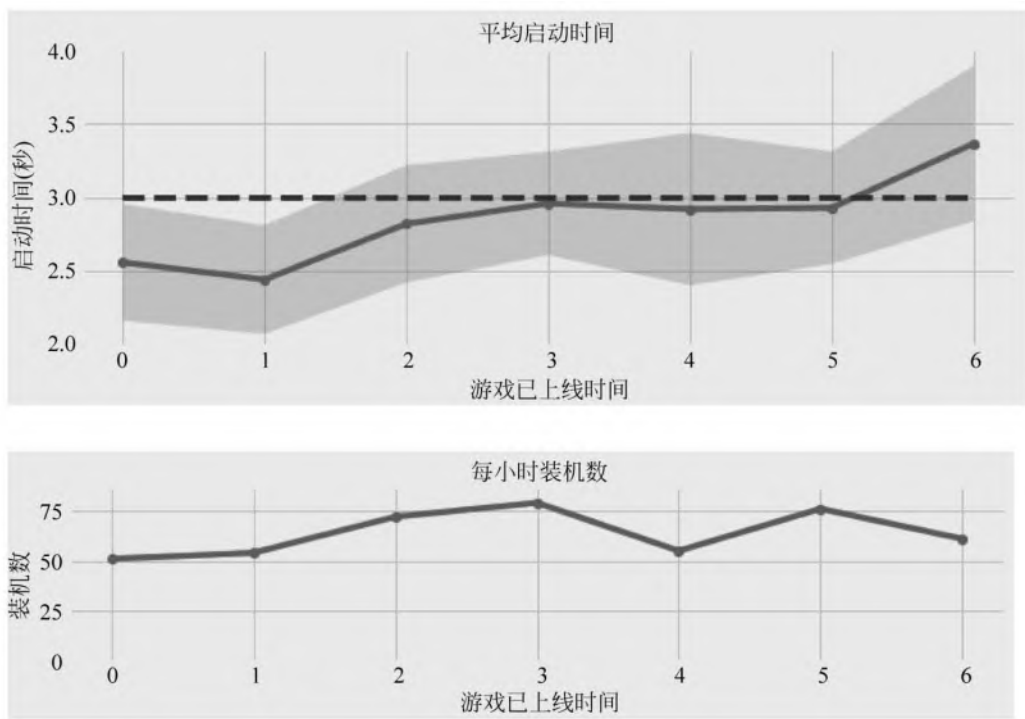


图16.4 带置信区间的前7小时平均启动时间和装机数

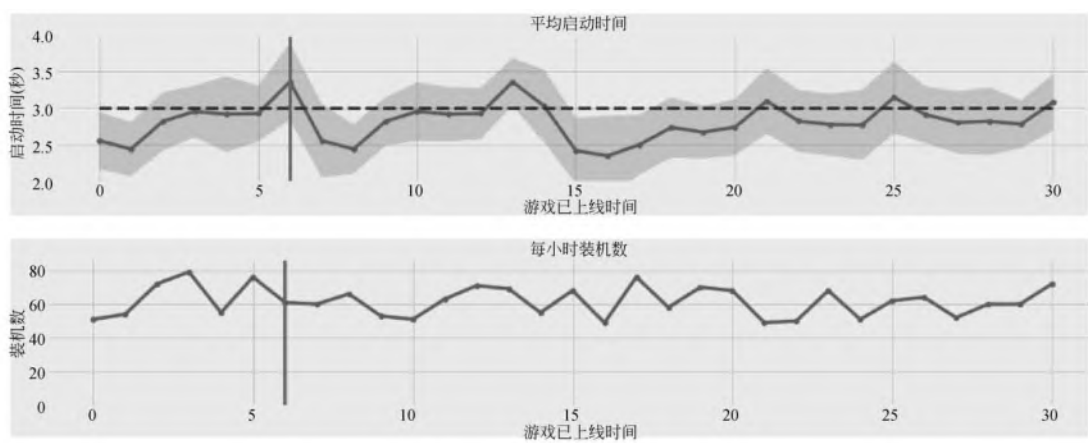


图16.5 启动时间仪表盘

```

>>> observation_hours = 24
>>> expected_installs = 60
>>> np.random.seed(23)
>>> loading_times2 = list(loading_times)
>>> loading_times_averages2 = list(loading_times_averages)

>>> for ix, installs in enumerate(np.random.poisson(\
    lam = expected_installs, size = observation_hours)):
    loading_times2.append(np.random.gamma(shape = 3, \
        scale = .95, size = installs))

    loading_times_averages2.append(loading_times2[ix].mean())

>>> loading_performance = pd.DataFrame(loading_times_averages2,
    columns = ['loadingtime'])
>>> loading_performance['installs'] = [len(x) for x in loading_times2]
>>> loading_performance['SE'] = [standard_error(x) for x \
    in loading_times2]
>>> loading_performance['t_critical'] = stats.t.ppf(0.975, \
    df = loading_performance['installs'] - 1)
>>> loading_performance['CI_low'] = \
    loading_performance['loadingtime'] - \
    loading_performance['t_critical'] * loading_performance['SE']
>>> loading_performance['CI_high'] = \
    loading_performance['loadingtime'] + \
    loading_performance['t_critical'] * loading_performance['SE']
# 仪表盘
>>> fig, ax = plt.subplots(figsize = (20,3))
>>> plt.plot(loading_performance['loadingtime'], marker = 'o')
>>> plt.title('平均启动时间', fontsize = 16)
>>> plt.plot([0, len(loading_performance) - 1], [3, 3], '--', color = '0.3')
>>> plt.fill_between(loading_performance.index, \
    loading_performance['CI_low'], loading_performance['CI_high'], \
        where = loading_performance['CI_low'] < \
        loading_performance['CI_high'],
        alpha = 0.2, interpolate = True, linewidth = 0.0)
>>> plt.plot([6, 6], [0, 5], '- ')
>>> plt.xlabel('游戏已上线时间', fontsize = 14)
>>> plt.ylabel('启动时间(秒)', fontsize = 14)
>>> plt.ylim(2, 4)

>>> fig, ax = plt.subplots(figsize = (20,3))
>>> plt.plot(loading_performance['installs'], marker = 'o')
>>> plt.ylim(0, loading_performance['installs'].max() * 1.1)
>>> plt.title('每小时装机数', fontsize = 14)
>>> plt.xlabel('游戏已上线时间', fontsize = 14)
>>> plt.ylabel('装机数', fontsize = 14);
>>> plt.plot([6, 6], [0, 500], '- ');

```

现在你可以放心地下班了，因为启动时间仪表盘会自动提示项目经理当前的问题是否需要采取行动。

16.2 次日留存率是否大于30%

现在游戏已经成功上线1天，项目经理又开始计算次日留存率指标，如图16.6所示。

很不幸，游戏的次日留存率只有27.6%，离30%的目标还有一定差距。模拟数据代码与可视化代码如下。

```
# 模拟数据
>>> installs = 448
>>> returned = 123
>>> p = returned/installs
>>> print('装机数 = {} \n 次日留存 = {} \n 次日留存率 = {:.2f}% \n \
        目标 >= 30% '.format(installs, returned, 100 * p))
# 可视化
>>> fig, ax = plt.subplots(figsize=(8,4))
>>> plt.bar([1],[p], align='center', width=.8)
>>> plt.plot([0,100],[.3,.3], '--', color='0.3')
>>> plt.xticks(range(1,10))
>>> plt.xlim(0,5)
>>> plt.ylim(0,.40)
>>> ax.set_xticklabels(['%d Sep' % d for d in range(6,10)])
>>> ax.yaxis.set_major_formatter(FuncFormatter(lambda x,\
        pos=0: '%0.0f% %' % (100.0 * x)))
>>> plt.title('次日留存率', fontsize=16)
>>> ax.text(1, p * .95, '%0.2f% %' % (100 * p), ha='center', \
        va='top', \
        fontdict={'size':15, 'weight':'bold', 'color':(0.9,.9,.9)})
```

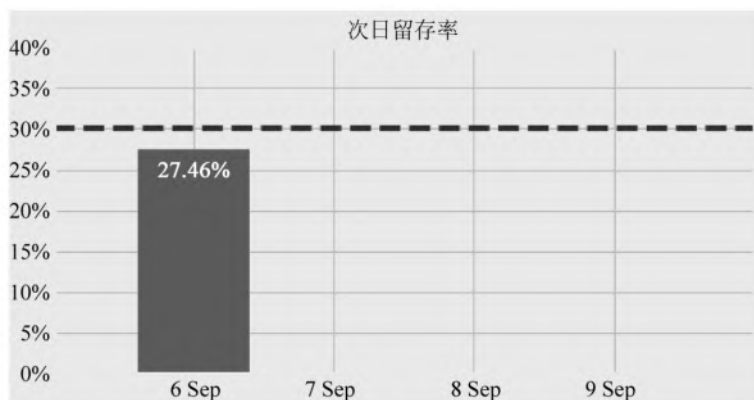


图16.6 游戏的次日留存率

上述代码读者应该已经非常熟悉，这里不过多解释。在第9章中已经提到过，axes对象可以使用各种绘图函数，所以这里使用了ax变量来进行图形属性设置。其中，ax.text()函数中字体的指定通过fontdict这个字典来一次完成多个值的设置，这也是数据可视化经常用到的一种属性设置方式。

回到游戏的次日留存率问题，现在运营团队是否应该为第一天上线后的次日留存率小于30%而担心？很显然，这又是一个统计检验问题，只不过这次遇到的问题是单样本比例问题。参考图15.1可知应该采用 z -test来解决此问题，利用statsmodels中的proportions_ztest()函数，代码如下。

```
>>> from statsmodels.stats.proportion import proportions_ztest
>>> z,p = proportions_ztest(returned, installs, value=.3, \
    alternative='smaller', prop_var=.3)
>>> print('z-stat = {z} \n p-value = {p}'.format(z=z,p=p))
z-stat = -1.175320190850308
p-value = 0.11993331980861971
```

$p > 0.05$ ，显然无法拒绝空假设，也就是无法认为次日留存率小于30%。如果计算置信区间，可以发现95%的置信区间取值是23.32%~31.59%，因此，当前游戏的次日留存率还在合理范围，代码如下。

```
>>> from statsmodels.stats.proportion import proportion_confint
>>> ci_low, ci_upp = proportion_confint(returned, installs)
>>> print('95 % Confidence Interval = ( {0:.2f} % , \
      :.2f} % )'.format(100 * ci_low, 100 * ci_upp))
95 % Confidence Interval = (23.32 % , 31.59 % )
```

相信作为数据分析师，读者应该知道如何回答项目经理了吧？是否有必要也为项目经理构建一个次日留存率的仪表盘呢？这项工作就留给读者完成了。

16.3 应该在游戏第几关加入关联微信提示

16.3.1 A/B测试

经过多日数据跟踪，启动时间和次日留存率等各项指标一切正常。现在项目经理打算着手解决游戏设计之初就激烈讨论的一个问题：应该在游戏第几关加入微信关联页面？在微信当道的今天，游戏产品经理希望用户尽可能地关联他们的微信账号，这样运营团队就可以更好地了解用户以及他们的好友。因此，产品经理想对两种不同的微信登录的页面进行测试，一种是在游戏第一关时加入微信关联页面，另一种是在游戏第二关加入。需要说明的是，该游戏前3关难度都不大，根据运营团队的估算，几乎90%以上用户都会通过前面3关，因此不会存在通过第一关和第二关的用户有很大差别的情况。为了对以上两种策略进行比较，项目组精心设计了如下实验：将用户随机分组，控制组2501人，测试组2141人，控制组在第一关弹出关联微信页面，测试组在第二关弹出关联微信页面。最终测试结果如下。

```

>>> control_installs = 2501
>>> control_connected = 1104
>>> test_installs = 2141
>>> test_connected = 1076

>>> print('{}: 装机数 = {} \t 关联微信 = {} \t 比例 = {}'\
        .format('A', control_installs, control_connected, \
        control_connected/control_installs))
>>> print('{}: 装机数 = {} \t 关联微信 = {} \t 比例 = {}'\
        .format('B', test_installs, test_connected, \
        test_connected/test_installs))
A: 装机数 = 2501    关联微信 = 1104    比例 = 0.4414234306277489
B: 装机数 = 2141    关联微信 = 1076    比例 = 0.5025688930406352

```

将结果可视化，如图16.7所示。现在产品经理想知道哪种策略更好，很明显，测试组中有50.26%的用户关联了微信，而控制组只有44.14%的用户进行了此操作。但是产品经理担心这只是偶然因素，正如游戏上线时游戏的启动时间超过3秒一样。请问数据分析师将如何回答他呢？

```

>>> fig, ax = plt.subplots(figsize=(8,4))
>>> x = [0,1]
>>> y = [control_connected/control_installs, \
        test_connected/test_installs]
>>> ax.bar(x, y, align='center', width=.8)
>>> ax.set_xticks(x)
>>> ax.set_xticklabels(['控制', '测试'])
>>> plt.xlim(-.5,1.5)
>>> plt.ylim(0, .69)
>>> for xx, yy in zip(x,y):
    ax.text(xx, yy*.7, '%0.2f%%' % (100*yy), ha='center', va='bottom', \
            fontdict={'size':15, 'weight':'bold', 'color':(0.9,.9,.9)})
>>> ax.yaxis.set_major_formatter(FuncFormatter(lambda x, \
        pos = 0: '%0.0f%%' % (100.0*x)))
>>> plt.title('哪种策略更好?', fontdict={'size':16});

```

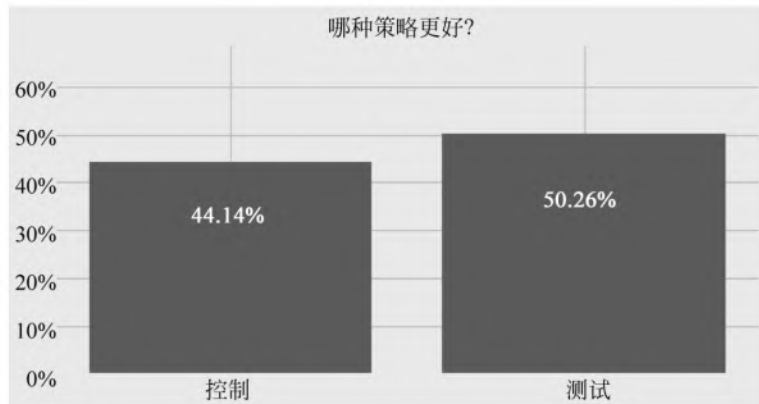


图16.7 关联页面选择

与前面的问题处理方式一样，首先参考图15.1，根据描述，目前需要处理的是双样本，比例问题。因此，应该采用 z -test来回答此问题，利用statsmodels中的proportions_ztest()函数计算如下。

```
>>> from statsmodels.stats.proportion import proportions_ztest
>>> count = np.array([control_connected, test_connected])
>>> nobs = np.array([control_installs, test_installs])
>>> z, p = proportions_ztest(count, nobs, value = 0, \
    alternative = 'two-sided')
>>> print('z-stat = {z} \n p-value = {p}'.format(z = z, p = p))
z-stat = -4.161114920415351
p-value = 3.1669765828788065e-05
```

p 远远小于0.05，那么可以拒绝空假设，即两种关联策略没有区别。因此，数据分析师应该告诉产品经理在第二关引入该页面有更好的效果。不过产品经理不仅关心哪种策略更好，他还关心采用这一策略游戏的微信关联指标能提高多少，将其转化为统计语言实际就是策略二的关联比例减去策略一关联比例的置信区间。要解决此问题，首先需要计算出标准误差（Standard Error）和 z -test中的 $z_{critical}$ 值，之后再利用置信区间计算公式就可以得到答案，代码如下。

```

>>> def compute_standard_error_prop_two_samples(\
    x1, n1, x2, n2, alpha = 0.05):
    p1 = x1/n1
    p2 = x2/n2
    se = p1 * (1 - p1)/n1 + p2 * (1 - p2)/n2
    return np.sqrt(se)

>>> def zconf_interval_two_samples(x1, n1, x2, n2, alpha = 0.05):
    p1 = x1/n1
    p2 = x2/n2
    se = compute_standard_error_prop_two_samples(x1, n1, x2, n2)
    z_critical = stats.norm.ppf(1 - 0.5 * alpha)
    return p2 - p1 - z_critical * se, p2 - p1 + z_critical * se

>>> ci_low, ci_upp = zconf_interval_two_samples(control_connected,\
    control_installs, test_connected, test_installs)
>>> print('95 % Confidence Interval = ( {0:.2f} % , {1:.2f} % )'\
    .format(100 * ci_low, 100 * ci_upp))
95 % Confidence Interval = (3.24 % , 8.99 %)

```

在 $\alpha = 0.05$ 的情况下，置信区间为3.24%~8.99%。从统计学角度，就是说 $(p_2 - p_1)$ 有95%的可能落在这个区间，因此可以认为第二种策略下微信关联指标的提升在3.24%以上。

16.3.2 贝叶斯解决方案

了解概率论的读者看到这里可能会说：“前面都是采用的频率学派的解决方法。”如果用贝叶斯的解决方法该怎么做呢？其实可以用PyMC3这个包来完成，开源书籍 *Probabilistic Programming & Bayesian Methods for Hackers* 以PyMC3为例详细介绍了概率编程以及贝叶斯方法的应用，其中的第2章也给出了贝叶斯A/B测试的例子。由于PyMC3以及贝叶斯理论涉及太多新内容，本书篇幅有限，这里仅给出代码，不进行详细说明。


```

>>> import pymc3 as pm
>>> control = [1] * control_connected + [0] * (control_installs - \
    control_connected)
>>> treatment = [1] * test_connected + [0] * (test_installs - test_connected)
>>> control = np.asarray(control)
>>> treatment = np.asarray(treatment)
>>> start = {}
>>> start['p_C'] = (control).sum()/len(control)
>>> start['p_T'] = (treatment).sum()/len(treatment)
# 模型
>>> with pm.Model() as model:
    p_C = pm.Uniform('p_C', 0.00001, .9)
    p_T = pm.Uniform('p_T', 0.00001, .9)
    # 模拟
    uplift = pm.Deterministic('uplift', p_T - p_C)
    obs_C = pm.Bernoulli("obs_C", p_C, observed = control)
    obs_T = pm.Bernoulli("obs_T", p_T, observed = treatment)
    # Inference
    step = pm.NUTS()
    trace = pm.sample(10000, step, start = start, progressbar = True)
>>> pm.stats.hpd(trace['uplift'], alpha = 0.05)
array([0.03264087, 0.09001931])

```

从输出结果可以看出策略二的提升在0.032~0.09之间，与前面得出的结果类似。从图16.8所示的Traceplot图也可以看出，10000次采样得到的提升基本上符合均值为0.06的正态分布。

```

>>> pm.traceplot(trace, ['uplift']);

```

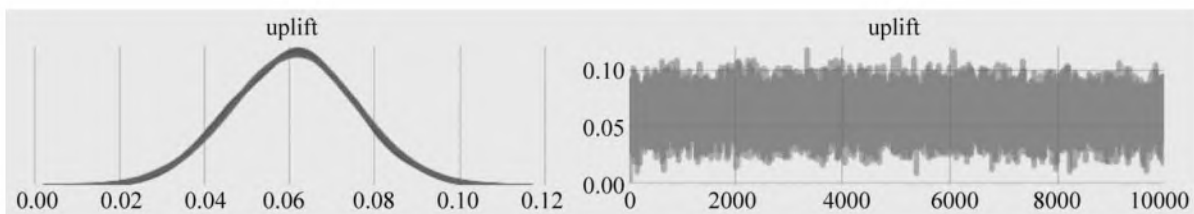


图16.8 Traceplot

虽然本书对概率编程以及贝叶斯方法没有进行深入介绍，但是强烈建议有兴趣的读者去看看前面提到的书籍，以及掌握PyMC3包，相信它一定会对读者的学习和工作大有帮助。

16.4 如何定价

目前游戏中用户在通关失败后可以花1.99元获得新的生命进行尝试。但是运营推广部门认为1.99元太贵了，这样大量玩家都不会成为付费用户。他们建议降价到0.99元，这样虽然价格低了，但是整体上游戏收入将提高。因此，设计如下的A/B测试：

- 只对新用户进行测试；
- 运行2个月；
- 统计每个用户前30天的总收入；
- 空假设：0.99元组收入更高。

而具体度量指标则包括：

- 每用户平均收入（Average Revenue Per User，ARPU）值在前30天的区别；
- 转化率（多少比例用户至少购买一次）。

假设经过30天后，得到如下数据。

```
>>> conversion_a = 0.015
>>> conversion_b = 0.013
>>> installs_a = 30000
>>> installs_b = 30000
>>> payers_a = int(installs_a * conversion_a)
>>> payers_b = int(installs_b * conversion_b)
>>> print('A组付费用户数：', payers_a)
>>> print('B组付费用户数：', payers_b)
A组付费用户数：450
B组付费用户数：390
```

如果仅从图16.9来看，显然0.99元组有更好的转化率。

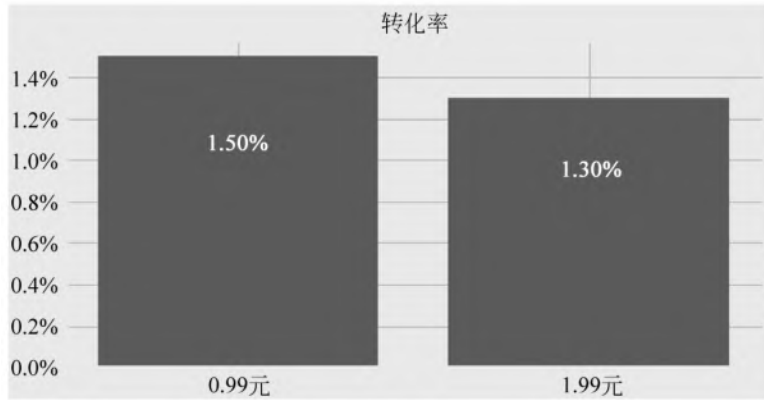


图16.9 转化率

不过这也需要进行统计检验来验证，代码如下。

```
>>> from statsmodels.stats.proportion import proportions_ztest
>>> count = np.array([payers_a, payers_b])
>>> nobs = np.array([installs_a, installs_b])
>>> z, p = proportions_ztest(count, nobs, value = 0, \
                             alternative = 'two - sided')
>>> print('z - stat = {z} \n p - value = {p}'.format(z = z, p = p))
z - stat = 2.0848420114464243
p - value = 0.03708364384057536
```

$p = 0.037$ ，可以认为具有统计显著性，因此拒绝空假设。确实0.99元组的表现比1.99元组要好。接下来再来看ARPU数据，生成模拟数据如下。

```

>>> np.random.seed(8)
>>> revenues_a = np.clip(np.random.gamma(.7, 10, payers_a).\
    astype(np.int),1,1000) * np.clip(np.random.gamma(1, 25, \
    payers_a).round(2), .99, 99)
>>> revenues_b = np.clip(np.random.gamma(.7, 13, payers_b).\
    astype(np.int),1,1000) * np.clip(np.random.gamma(1, 30, \
    payers_b).round(2), 2.99, 99)
>>> print('A: ARPU = {}, min = {}, max = {}'.format(revenues_a.mean(),
    revenues_a.min(), revenues_a.max()))
>>> print('B: ARPU = {}, min = {}, max = {}'.format(revenues_b.mean(),
    revenues_b.min(), revenues_b.max()))
>>> pd.set_option('display.precision', 2)
>>> pd.DataFrame(data = revenues_a, columns = ['0.99']).\
    describe().join(pd.DataFrame(data = revenues_b,
    columns = ['1.99']).describe())
A: ARPU = 151.86206666666666, min = 0.99, max = 3960.0
B: ARPU = 214.2176923076923, min = 2.99, max = 3647.7999999999997

```

	0.99	1.99
count	450.00	390.00
mean	151.86	214.22
std	325.31	372.41
min	0.99	2.99
25 %	20.82	26.52
50 %	55.31	69.44
75 %	147.27	231.28
max	3960.00	3647.80

注意，这里的代码生成的是任意模拟数据，读者关注的重点不是数据的准确性，关键应在于理解后面的计算逻辑。基于此数据，利用如下代码可视化，如图16.10所示。

```

>>> x = [0,1]
>>> y = [revenues_a.mean(), revenues_b.mean()]
>>> fig, ax = plt.subplots(figsize = (8,4))
>>> ax.bar(x, y, align = 'center', width = .8)
>>> ax.set_xticks(x)
>>> ax.set_xticklabels(['0.99 元', '1.99 元'])
>>> plt.xlim(-.5,1.5)
>>> for xx, yy in zip(x,y):
    ax.text(xx, yy * .7, '%0.2f' % (yy), ha = 'center', va = 'bottom', \
        fontdict = {'size':18, 'weight':'bold', 'color':(0.9,.9,.9)})
>>> ax.yaxis.set_major_formatter(FuncFormatter(lambda x, \
    pos = 0: '%0.02f' % (x)))
>>> plt.title('付费用户 ARPU', fontdict = {'size':16})

```

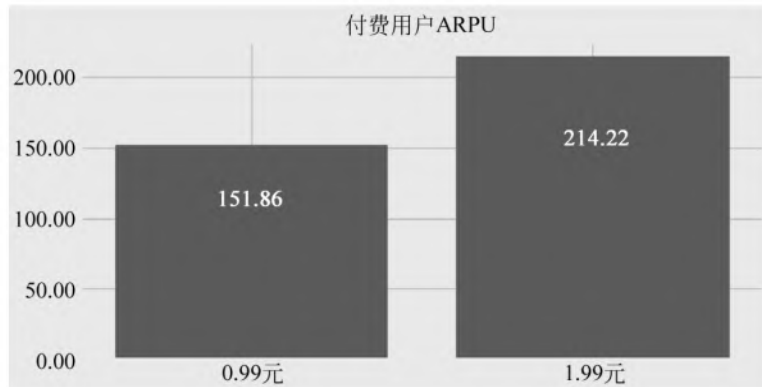


图16.10 付费用户ARPU

从ARPU角度看，1.99元组具有更高的付费用户ARPU。如果将所有用户考虑进来计算ARPU，得到的数据如下。

```
>>> revenues_a = np.concatenate([np.zeros(installs_a - payers_a), \
    revenues_a])
>>> revenues_b = np.concatenate([np.zeros(installs_b - payers_b), \
    revenues_b])
>>> print('A: ARPU = {}, min = {}, max = {}'.format(\
    revenues_a.mean(), revenues_a.min(), revenues_a.max()))
>>> print('B: ARPU = {}, min = {}, max = {}'.format(\
    revenues_b.mean(), revenues_b.min(), revenues_b.max()))
>>> print('ARPU 差别: ', revenues_b.mean() - revenues_a.mean())
>>> print('ARPU 提升比例: ', (revenues_b.mean() - \
    revenues_a.mean())/revenues_b.mean())
A: ARPU = 0.7669801346801346, min = 0.0, max = 3960.0
B: ARPU = 0.9363920645595157, min = 0.0, max = 3647.7999999999997
ARPU 差别: 0.16941192987938114
ARPU 提升比例: 0.18091986924202896
```

利用双样本独立 t -test，计算结果如下。

```
>>> import scipy.stats as stats
>>> t, p = stats.ttest_ind(revenues_a, revenues_b, axis=0, \
    equal_var=False)
>>> print('t-stat = {t} \n p-value = {p}'.format(t=t, p=p/2))
t-stat = -1.337055975602803
p-value = 0.09060474681783709
```

由于 $p = 0.09$ ，无法拒绝空假设。那么这是否是说明1.99元组消费更高只是偶然因素，游戏应该对所有人降价？从统计学角度回答此问题很简单，但是如果从商业角度也许就变得更复杂了。如果查看模拟的数据，可以发现少数用户消费在1000元以上，这些就是所谓的VIP游戏玩家了，如果将这一部分用户去掉后进行观察，将得到如下结果。

```
>>> revenues_a_trimmed = np.minimum(revenues_a, 1000)
>>> revenues_b_trimmed = np.minimum(revenues_b, 1000)
>>> print('A: ARPU = {}'.format(revenues_a_trimmed.mean()))
>>> print('B: ARPU = {}'.format(revenues_b_trimmed.mean()))
>>> print('ARPU 差别: ', revenues_b_trimmed.mean() \
        - revenues_a_trimmed.mean())
A: ARPU = 2.0011756666666667
B: ARPU = 2.4898616666666666
ARPU 差别: 0.48868599999999995
```

此时重新计算 p 值，代码如下。

```
>>> t, p = stats.ttest_ind(revenues_a_trimmed, revenues_b_trimmed, \
        axis=0, equal_var=False)
>>> print('t-stat = {t} \n p-value = {p}'.format(t=t, p=p/2))
t-stat = -1.7854914296667377
p-value = 0.03709347599105856
```

此时 $p = 0.037$ ，是否意味着应该拒绝空假设？其实仅凭这些数据我们是无法知道定价是如何影响这些客户的，不过商业决策始终是有风险的，也许更好的方式是继续采用1.99元的定价，之后用更激进的定价策略，进行新一轮的A/B测试。

16.5 留给读者的思考

A/B测试是一门专门的学科，其核心并不是简单的统计检验。在测试时需要考虑的问题很多：什么时候开始A/B测试，做哪些A/B测试，

需要运行多久，怎么衡量A/B测试是否成功，等等。如果要从事A/B测试相关的分析，建议读者进一步阅读A/B测试相关书籍。

第17章 利用数据分析找工作

在前一个半世纪所有伟大发明的背后，绝不仅仅是技术本身的长期进步，同时还有思维方式的改变。

——路易·芒福德

数据时代来临，我们需要用新的视角来解读这个世界。数据分析不仅可以用于企业改进运营，提升效率，也可以用来帮助我们做出日常决策，如找工作、买房。本章将对拉勾网2019年8月发布的数据分析职位进行分析，以说明目前数据分析职位的需求情况。

17.1 设定分析目标

17.1.1 问题定义

所有的数据分析都源于某个待解决的问题，那么分析数据分析相关的职位的目的是什么呢？作为在校学生，可能看到数据分析目前正流行，未来想从事这一职业，因此想了解：

- 数据分析相关职位主要在哪些行业有需求？
- 哪些城市对这些职位有大量需求？
- 都是哪些公司在招聘？
- 这些公司目前发展怎么样？
- 职位待遇如何？
- 对学历有什么要求？
- 对工作经验要求高吗？
- 技能要求如何？

类似的问题还可以提出很多，如果企业想发布一项数据分析的招聘需求，那么作为职位发布者可能想了解：

- 市场上这个行业的人才紧缺吗？
- 其他公司的薪酬如何？
- 平均招聘周期多长？
- 都有哪些公司对这类人才有大量需求？

17.1.2 获取数据

对要解决的问题有了清晰定义后，接下来就需要考虑当前有哪些可用数据了。很明显，目前我们手里没有任何数据，那么就需要思考如何去获得数据。假设读者是一名求职者，那么很容易就想到可以从招聘网站了解这些数据，如智联招聘、前程无忧，还有互联网行业经常使用的拉勾网，等等。图17.1是在拉勾网搜索数据分析职位返回的页面。



图17.1 拉勾网职位

从这个页面可以看出，通过该网站可以得到招聘公司、职位所在城市、工作经验要求、学历、薪酬等信息，关于职位的技能要求则需要进一步单击该职位后获得。显然，如果能获取所有数据分析职位的信息，那么应该可以回答上一小节提出的问题。

17.2 准备分析数据

本节将以拉勾网的数据分析职位为例进行讲解，关于数据获取部分，可以通过数据爬虫来获得，具体实现过程本书就不再讨论，有兴趣的读者可以自行学习网络爬虫相关知识。现在假设已经获得了这一数据。

17.2.1 数据准备

首先导入需要的Python模块，完成一些初始的设置。

```
>>> import numpy as np
>>> import pandas as pd
>>> import random
>>> import re
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> pd.set_option('display.max_columns', 10)
>>> pd.set_option('display.max_rows', 10)
>>> %matplotlib inline
# 用来正常显示中文标签
>>> plt.rcParams['font.sans-serif'] = ['SimHei']
# 当坐标轴有负号的时候可以显示负号
>>> plt.rcParams['axes.unicode_minus'] = False
```

接下来需要读入待分析的数据，并对其有一个初步了解，如图17.2所示。

```
>>> lagou = pd.read_csv('lagou_data.csv')
>>> lagou.head()
```

Unnamed: 0	adWord	appShow	approve	businessZones	...	skillLables	stationname	subwayline	thirdType	workYear
0	0	0	0	1	[张江] ...	[SQL, '数据分析']	金科路	2号线	数据分析	3-5年
1	1	0	0	1	['四惠', '十里堡', '甘露园'] ...	['数据分析']	四惠东	1号线	数据分析	不限
2	2	0	0	1	NaN ...	['数据分析', '数据库']	中山公园	1号线	数据分析	1-3年
3	3	0	0	1	NaN ...	['数据挖掘', '数据分析', '算法']	安华桥	10号线	数据挖掘	1-3年
4	4	0	0	1	['学院路', '北太平庄', '牡丹园'] ...	['BI', '数据分析']	西土城	10号线	数据分析	3-5年

5 rows x 50 columns

图17.2 读入待分析职位数据

通过查看数据的前5行大致可以看出该数据提供了公司所在位置、工作年限要求等信息。进一步观察可以发现该数据一共有50列，但是由于前面使用了代码`pd.set_option('display.max_columns', 10)`来设置仅展示10列，所以其余列在Jupyter中显示时都用省略号代替，因此这里对数据的了解并不全面。当然在数据分析时也可以取消这一限制，不过更好的方法是用下面的代码来了解列的信息。

```
>>> lagou.columns
```

```
Index(['Unnamed: 0', 'adWord', 'appShow', 'approve', 'busniessZones', 'city',
      'companyFullName', 'companyId', 'companyLabelList', 'companyLogo',
      'companyShortName', 'companySize', 'createTime', 'deliver',
      'district', 'education', 'explain', 'financeStage', 'firstType',
      'formatCreatTime', 'gradeDescription', 'hitags', 'imstate',
      'industryField', 'industryLables', 'isHotHire', 'isSchoolJob',
      'jobNature', 'lastLogin', 'latitude', 'linestation', 'longitude',
      'pcShow', 'plus', 'positionAdvantage', 'positionId',
      'positionLables', 'positionName', 'promotionScoreExplain',
      'publisherId', 'resumeProcessDay', 'resumeProcessRate', 'salary',
      'score', 'secondType', 'skillLables', 'stationname', 'subwayline',
      'thireType', 'workYear'],
      dtype = 'object')
```

上述代码输出了所有的列名，其中有些信息是数据分析中不需要的，因此对数据进行精简处理，只选择需要的列来进行分析，如图17.3所示。

```
>>> columns = ['positionId', 'city', 'companyFullName', 'companySize',
               'district', 'education', 'financeStage', 'industryField',
               'jobNature', 'latitude', 'longitude', 'salary', 'skillLables',
               'workYear']
>>> lagou = lagou[columns]
>>> lagou.head()
```

	positionId	city	companyFullName	companySize	district	...	latitude	longitude	salary	skillLables	workYear
0	6199988	上海	上海麦路科技有限公司	50-150人	浦东新区	...	31.200891	121.602909	15k-30k	['SQL', '数据分析']	3-5年
1	6247003	北京	掌阅科技股份有限公司	500-2000人	朝阳区	...	39.905841	116.514211	15k-30k	['数据分析']	不限
2	5880792	厦门	厦门多快好省网络科技有限公司	50-150人	思明区	...	24.449090	118.079900	8k-16k	['数据分析', '数据库']	1-3年
3	6240762	北京	联洋国融(北京)科技有限公司	50-150人	西城区	...	39.969851	116.383333	15k-30k	['数据挖掘', '数据分析', '算法']	1-3年
4	6247059	北京	北京小川在线网络技术有限公司	150-500人	海淀区	...	39.980092	116.368056	20k-40k	['BI', '数据分析']	3-5年

5 rows x 14 columns

图17.3 选择需要的列

现在基本上已经完成了对要分析的数据的筛选，接下来需要对数据清洗以便后续分析。

17.2.2 数据清洗

对待分析数据进行简单浏览，很容易就可以发现有些列中的数据只适合人工查看，不适合数据分析。例如，companySize列提供了公司的人数范围，虽然方便阅读，但是对于数据分析却不方便，因此可以考虑处理为两列数据或只取最小数据，类似地，还有salary、workYear等数据，都需要进行类似处理。首先用如下代码处理companySize数据。

```
>>> companysize = lagou['companySize'].str.extract('(\d* )-?')
>>> lagou['size'] = companysize
>>> lagou['size'].head()
0      50
1     500
2      50
3      50
4     150
Name: size, dtype: object
```

上述代码采取了取最小公司规模人数的方法，通过正则表达式提取companySize列的前半部分数字，之后将取到的数值作为新的列添加到原数据集中。细心的读者可能会注意到这里的lagou['size']列实际上是字符串类型，因此还需要将其转化为整型，代码如下。

```
>>> lagou['size'] = lagou['size'].astype(int)
```

接下来再处理salary列，该列以类似“10k-20k”这种格式来表示工资范围。利用正则表达式可以分别提取出最低工资和最高工资，之后再利用最低、最高工资计算平均工资，代码如下。

```
>>> lagou['lowerSalary'] = lagou['salary'].str.\
    extract('(\d* )k-').astype(int)
>>> lagou['upperSalary'] = lagou['salary'].str.\
    extract('k-(\d* )').astype(int)
>>>> lagou['avgSalary'] = (lagou['lowerSalary'] + lagou['upperSalary'])/2
```

workYear列采用了两种数据格式来记录工作年限要求，一种是类似“3-5年”这种格式，还有一种用“不限”或“应届毕业生”来代表没有工作年限要求。因此，需要先统一将第二种格式改为第一种格式，之后再利用正则表达式提取最低工作年限要求，代码如下。

```
>>> lagou['workYear'] = lagou['workYear'].str.\
    replace('不限', '0 - ')
>>> lagou['workYear'] = lagou['workYear'].str.\
    replace('应届毕业生', '0 - ')
>>> lagou['workYears'] = lagou['workYear'].str.\
    extract('(\d* )').astype(int)
```

现在基本已经完成了数据的清理工作，使用head（）函数再次查看数据，如图17.4所示。

```
>>> lagou.head()
```

	positionId	city	companyFullName	companySize	district	...	size	lowerSalary	upperSalary	avgSalary	workYears
0	6199988	上海	上海麦路科技有限公司	50-150人	浦东新区	...	50	15	30	22.5	3
1	6247003	北京	掌阅科技股份有限公司	500-2000人	朝阳区	...	500	15	30	22.5	0
2	5880792	厦门	厦门多快好省网络科技有限公司	50-150人	思明区	...	50	8	16	12.0	1
3	6240762	北京	联洋国融（北京）科技有限公司	50-150人	西城区	...	50	15	30	22.5	1
4	6247059	北京	北京小川在线网络技术有限公司	150-500人	海淀区	...	150	20	40	30.0	3

5 rows x 19 columns

图17.4 职位数据清洗结果

17.3 开始数据分析

17.3.1 职位来自哪里

根据前面数据中提供的信息，职位来自哪里这个问题有3个层面的意思。其一是哪些行业对这类职位有需求；其二是哪些城市对这类职位有需求；其三是对这些职位有需求的公司规模如何，都处在什么阶段。首先来回答哪些行业对这类职位有需求的问题。通过前面的数据准备工作，读者应该已经注意到数据集中industryField列给出了公司所在行业的信息。因此，通过groupby操作，利用如下代码就可以回答该问题，结果如图17.5所示。

```
>>> lagou.groupby('industryField').count()
```

	positionId	city	companyFullName	companySize	district	...	size	lowerSalary	upperSalary	avgSalary	workYears
industryField											
人工智能	3	3	3	3	3	...	3	3	3	3	3
人工智能,物联网	1	1	1	1	1	...	1	1	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...
金融,电商	1	1	1	1	1	...	1	1	1	1	1
金融,移动互联网	1	1	1	1	1	...	1	1	1	1	1

70 rows x 18 columns

图17.5 公司所在行业信息

然而，如果仔细查看代码的输出结果，就会发现industryField中的信息似乎和我们期待的结果有所不同。该列中包含了多个行业信息，如“金融，电商”“金融，移动互联网”。对于这种情况，我们的理解是一个公司可以从属于多个子行业。更好的统计方法是，如果某一职位属于两个行业，那么分别在两个行业进行统计。因此，需要先对industryField列进行拆分，方法如下。

```
>>> industrySplits = lagou['industryField'].str.split(',', expand = True)
>>> industrySplits
```

	0	1
0	移动互联网	None
1	文娱 内容	None
...	...	...
433	企业服务	数据服务
434	企业服务	金融
435	rows x 2	columns

上述代码将原来的industryField列分成了两列，每列包含一个子行业信息，之后再利用stack（）函数将两列合并如下。

```
>>> industrySplits.stack()
0    0    移动互联网
1    0    文娱 | 内容
...
434  0    企业服务
      1    金融
Length: 631, dtype: object
```

合并后的数据中增加了一级索引，索引的取值就是industrySplits的列名。该信息在最终的统计中是不需要的，因此可以直接去掉，代码如下。

```
>>> lagou_industry = industrySplits.stack().reset_index(\
    level = 1, drop = True)
>>> lagou_industry.head()
0    移动互联网
1    文娱 | 内容
2    金融
3    数据服务
4    文娱 | 内容
dtype: object
```

利用上面得到的lagou_industry就可以完成各行业对数据分析类职位需求的统计，代码如下。

```
>>> lagou_industry = lagou_industry.groupby(lagou_industry.values).\
    count().reset_index()
>>> lagou_industry.columns = ['industryField', 'number']
>>> lagou_industry
   industryField  number
0   人工智能         7
1   企业服务        35
...  ...         ...
21  软件开发         1
22   金融         93
23 rows × 2 columns
```

接下来将该信息以可视化的方式进行展示，代码如下。


```

>>> fig = plt.figure(figsize = (8,6),dpi = 120)
>>> ax = fig.add_subplot(1, 1, 1)
>>> sns.barplot('industryField', 'number',
               data = lagou_industry,color = 'royalblue')
>>> x = np.arange(len(lagou_industry))
>>> y = lagou_industry.number
>>> for industry, jobs in zip(x,y):
        plt.text(industry, jobs + 2, '{}'.format(jobs),
                 ha = 'center', va = 'bottom', fontsize = 8)
>>> plt.xticks(fontsize = 8)
>>> props = {"rotation" : 45}
>>> plt.setp(ax.get_xticklabels(), ** props)
>>> plt.yticks([])
>>> plt.xlabel('')
>>> plt.ylabel('')
>>> plt.title('数据分析需求行业分布',size = 10)
>>> sns.despine(left = True)
>>> plt.show()

```

上述代码选择了使用条形图来展示信息，同时将绘图子区域保存到了ax变量中，因为后面修改X轴的刻度标记时需要用到该信息。之后使用plt.text()函数在每个条形上面添加了该条形代表的具体数值，当然也可以使用代码plt.text(industry, jobs+2, '{:.1f}%'.format(jobs/sum(lagou_industry.number)*100),ha='center',va='bottom',fontsize=10)以百分比的形式进行展示。由于X轴的刻度标记文字较长，会存在重叠现象，因此选择了对其进行旋转的方式，代码plt.setp(ax.get_xticklabels(),**props)完成了这一功能。最后，sns.despine()函数设置不显示坐标轴的左边框架，得到如图17.6所示的条形图。

从图17.6看出移动互联网、金融、数据服务、电商等行业是对数据分析职位需求最大的行业，很可能是由于这些行业有大量运营相关工作，因此对数据分析的需求较大。而人工智能、广告营销、物联网、医疗健康等行业相对需求较少，也许这些行业对机器学习会有更

大的需求。感兴趣的读者可以试着去获取机器学习相关的职位数据，然后来验证一下。

下面再来看看对数据分析职位需求最多的城市有哪些。首先通过分组统计获取职位需求最大的10个城市，代码如下。

```
>>> lagou_city = lagou.groupby('city')['positionId'].count()\n      .sort_values(ascending = False)
```

这段代码按照城市分组对每个城市职位数进行了汇总，之后按照职位数进行排序。获得了每个城市的职位数后，采用类似前面各行业数据分析职位需求的可视化代码来对城市职位需求进行可视化，得到图17.7，具体代码如下。

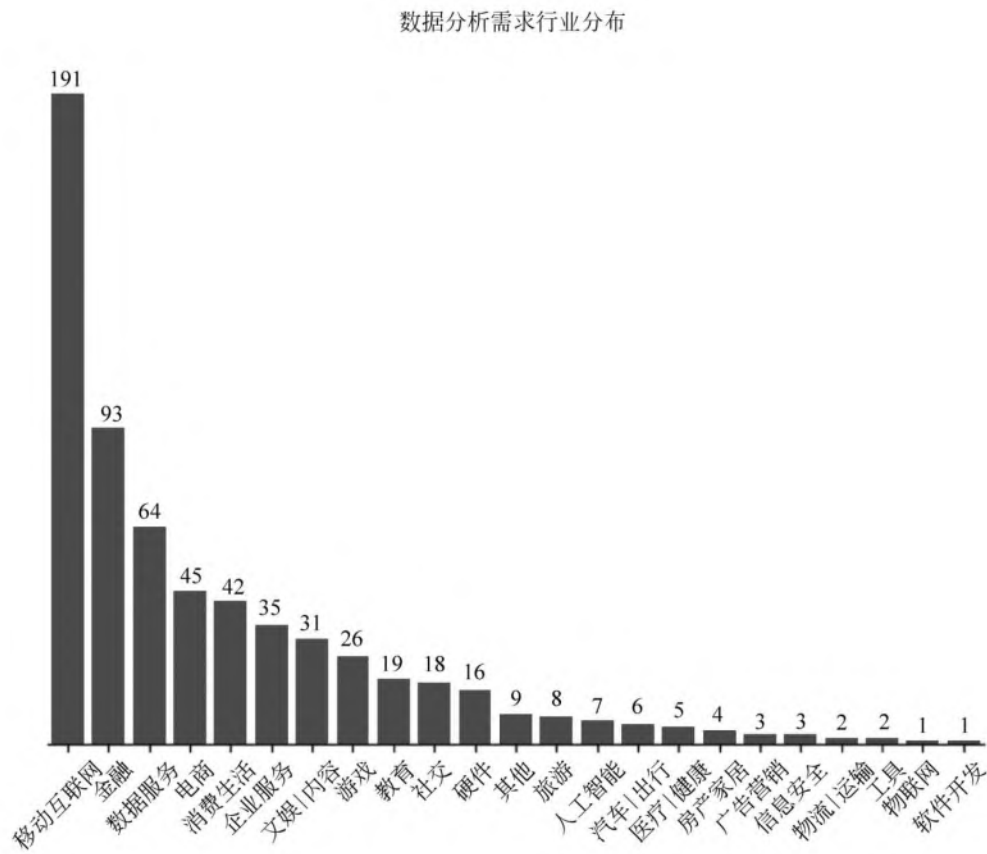


图17.6 各行业数据分析职位需求

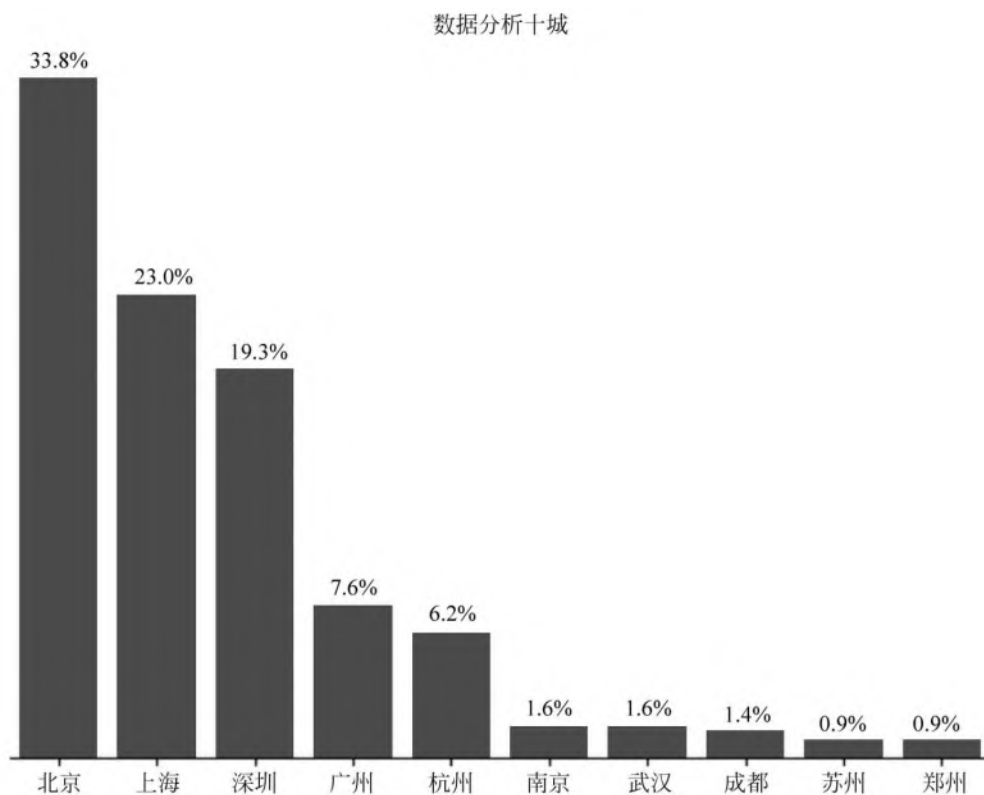


图17.7 数据分析十城

```
>>> fig = plt.figure(figsize = (8,6),dpi = 120)
>>> sns.barplot(lagou_city[:10].index,lagou_city[:10],
               color = 'royalblue')
>>> x = np.arange(0,10)
>>> y = lagou_city.values
>>> for x_loc,jobs in zip(x,y):
    plt.text(x_loc, jobs + 2,
             '{:.1f} %'.format(jobs/sum(lagou_city) * 100),
             ha = 'center', va = 'bottom',fontsize = 8)
>>> plt.xticks(fontsize = 8)
>>> plt.yticks([])
>>> plt.xlabel('')
>>> plt.ylabel('')
>>> plt.title('数据分析十城',size = 8)
>>> sns.despine(left = True)
>>> plt.show()
```

毫无疑问，北上广深占据了前4位，超过80%的职位都来自以上城市。如果读者想从事数据分析相关的职位，显然应该考虑以上城市。

对数据分析职位需求的行业和城市有了了解后，接下来再来分析提供职位的都是些什么类型的公司。下面了解公司所处财务阶段，利用如下代码，可以得到图17.8的输出。

```
>>> financeStage = lagou.groupby('financeStage')['positionId'].count()
>>> financeStage = financeStage.sort_values(ascending = True)
>>> plt.figure(figsize = (8,6), dpi = 120)
>>> sns.barplot(x = financeStage.index, y = financeStage.values)
>>> plt.xlabel("财务阶段")
```

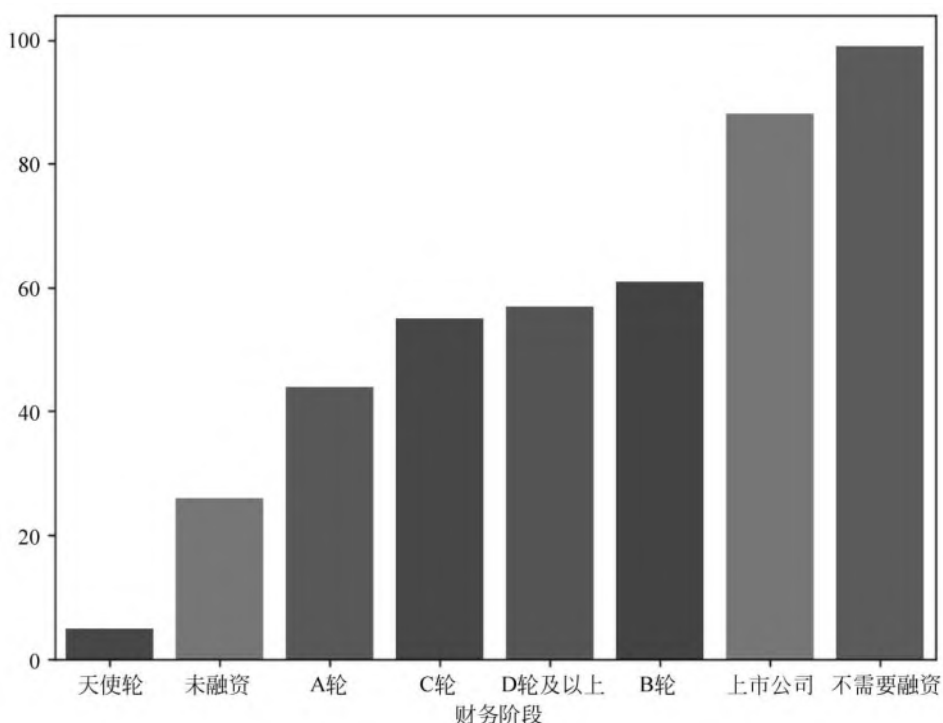


图17.8 公司所处财务阶段

从图17.8看出，招聘公司中，各类存在不同融资阶段的创业公司占据了主要地位。从某种意义上这也验证了数据分析是当下的一大热点。除了分析公司所处财务阶段，还可以分析公司规模，这部分就留给读者来完成。

17.3.2 职位薪酬如何

作为求职者，最关心的肯定就是薪酬了。那么数据分析职位的平均薪酬如何呢？通过describe（）函数可以快速查看描述性统计值。

```
>>> lagou['avgSalary'].describe()
count      435.000000
mean        19.086207
std          7.422723
min          1.000000
25 %        15.000000
50 %        19.500000
75 %        22.500000
max         52.500000
Name: avgSalary, dtype: float64
```

从描述性统计看，该职位平均月薪在1.9万元左右，其中75%在2.25万元以下，最高工资是5.25万元。从这个统计上看，数据分析职位的薪酬还是不错的，那么具体工资分布情况如何呢？考虑将工资分为7000元以下、7000元到1.5万元、1.5万元到2.5万元、2.5万元到5万元、5万元以上5个区间来进行考察，代码如下。

```
>>> bins = [0,7,15,25,50,100]
>>> cats = pd.cut(lagou['avgSalary'],bins,right=False)
>>> salary_data = pd.value_counts(cats)
>>> salary_data
[15, 25)      252
[7, 15)       99
[25, 50)       73
[0, 7)         9
[50, 100)       2
Name: avgSalary, dtype: int64
```

上述代码中pd.cut（）函数可以根据输入的bins将数据分割到对应区间。从输出结果看，大部分职位月薪在1.5万元到2.5万元。将结果可视化，如图17.9所示，代码如下。

```

>>> salary_data = salary_data.sort_index()
>>> fig = plt.figure(figsize = (8,6), dpi = 120)
>>> sns.barplot(salary_data.index, salary_data, color = 'royalblue')
>>> x = np.arange(len(salary_data))
>>> y = salary_data.values
>>> for x_loc, jobs in zip(x, y):
>>>     plt.text(x_loc, jobs + 2,
>>>               '{:.1f} %'.format(jobs/sum(salary_data) * 100),
>>>               ha = 'center', va = 'bottom', fontsize = 8)
>>> plt.xticks(fontsize = 8)
>>> plt.yticks([])
>>> plt.xlabel('单位:千元', fontsize = 15)
>>> plt.ylabel('')
>>> plt.title('数据分析岗位平均月薪', size = 8)
>>> sns.despine(left = True)
>>> plt.show()

```

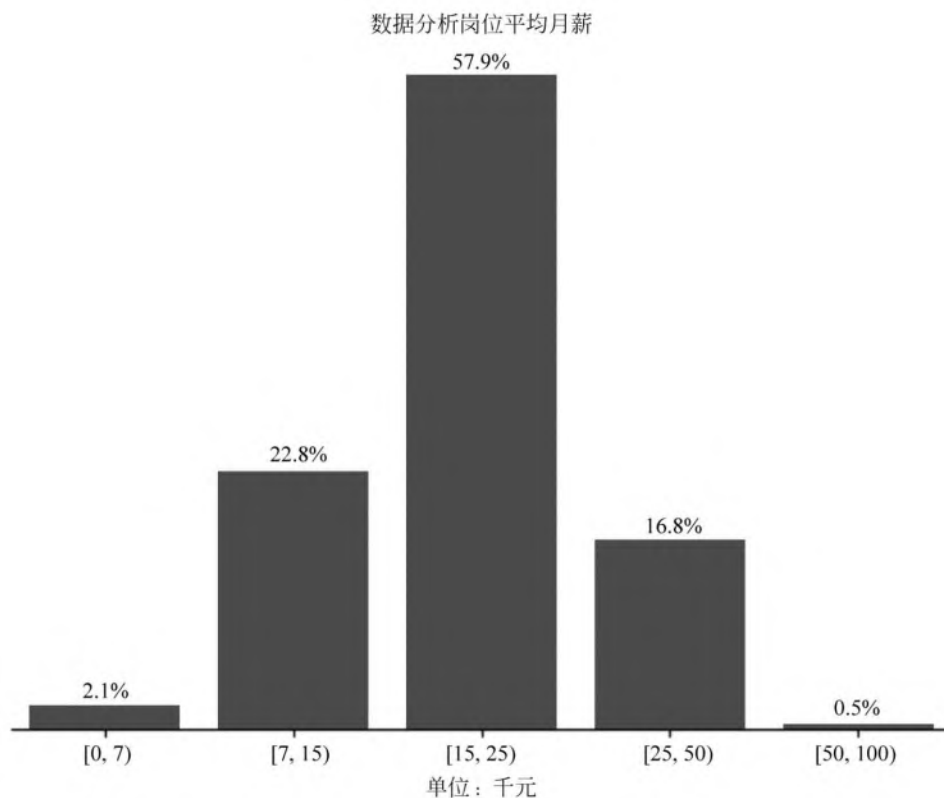


图17.9 数据分析岗位薪酬

除了用条形图来展示数据分析职位的薪酬分布以外，还可以使用分布图对其进行更进一步分析，如图17.10所示，输出该图的代码如

下。

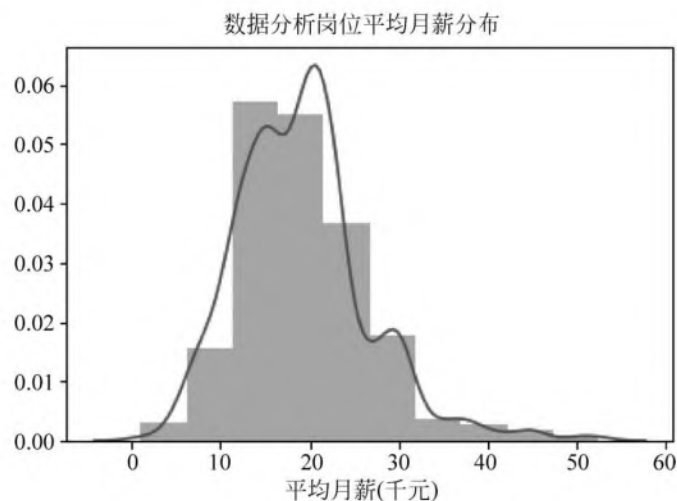


图17.10 数据分析岗位薪酬分布

```
>>> sns.distplot(lagou['avgSalary'], bins = 10)
>>> plt.xlabel('平均月薪(千元)', fontsize = 15)
>>> plt.title('数据分析岗位平均月薪分布', size = 15)
```

从分布图看，该职位的薪酬分布是基本符合正态分布的，只有极少数的人才拿到了较高薪酬。

17.3.3 岗位要求

经过前面的分析，读者对数据分析职位需求的行业、城市和薪酬等已经有了了解，那么这些职位对求职者有什么要求呢？下面从3个方面进行分析：学历、工作经验、技能。首先来看职位对学历的要求。要实现这一统计，代码很简单，只需要使用Seaborn中的countplot（）函数就可以了，运行如下代码，将得到如图17.11所示的图形。

```
>>> sns.countplot(lagou['education'])
>>> plt.xlabel('学历要求', fontsize = 10)
>>> plt.title('职位数', size = 10)
```

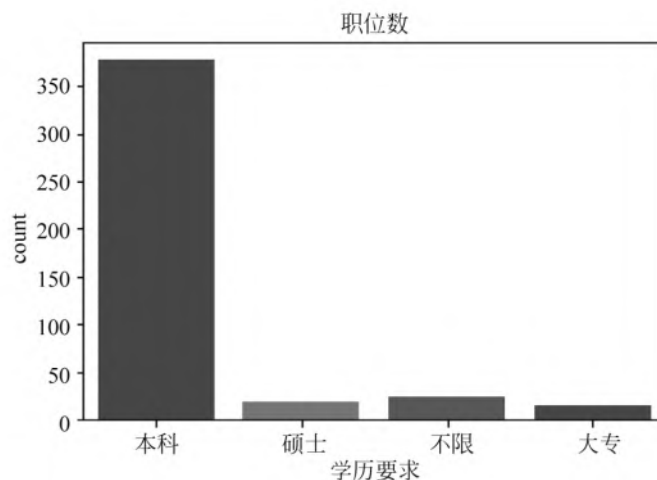


图17.11 学历要求

从图17.11看出绝大多数职位对学历的要求都是本科，看来有个大学文凭还是很重要的。了解了学历要求后，用同样方法可以得到工作经验要求，如图17.12所示。

生成图17.12的代码非常简单，基本上与生成图17.11的代码完全一样，这段代码留给读者自行完成。以上都是单独分析职位对工作经验以及学历的要求，那么工作经验和学历是如何共同影响薪酬的呢？要回答这一问题，可以利用Seaborn中FaceGrid图进行分析。具体代码如下。

```
>>> g = sns.FacetGrid(lagou, col = "education", # row
                        margin_titles = False)
>>> g.map(sns.regplot, 'workYears', 'avgSalary')
```

上述代码将不同学历情况下工资与工作经验间的关系进行了可视化，如图17.13所示。

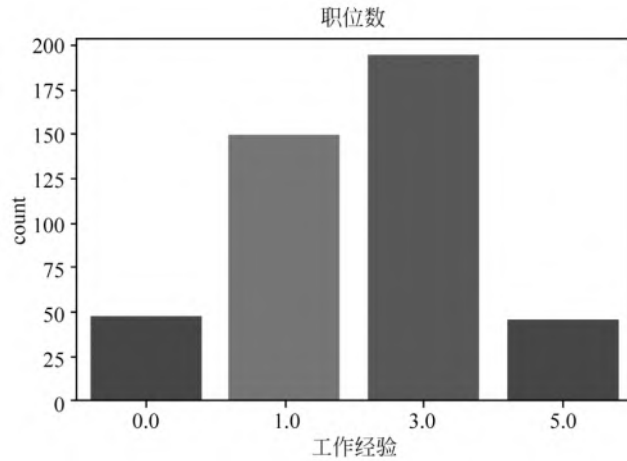


图17.12 工作经验要求

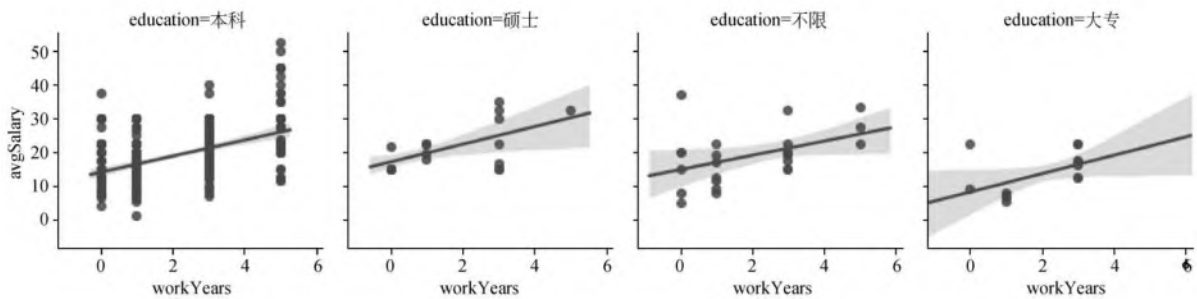
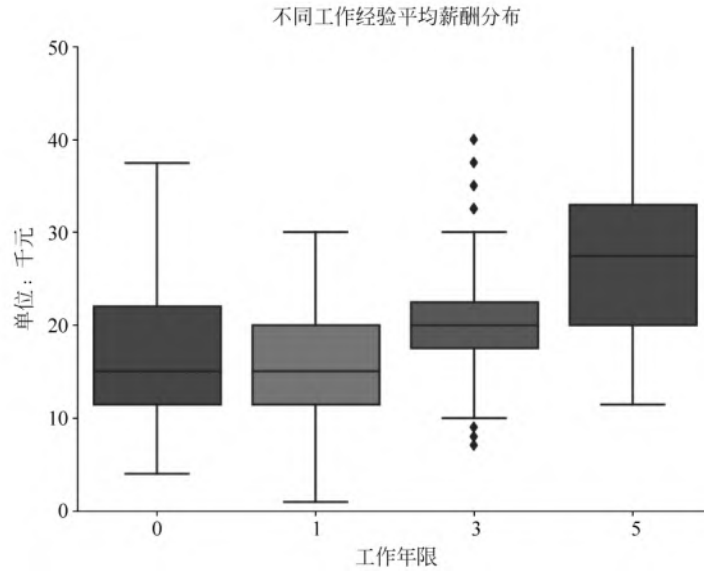


图17.13 不同学历对工资与工作经验间关系的影响

从图17.13可以看出学历和工资还是很明显存在一定的正比关系，同时随着工作经验的增加，学历在其中起的作用随之减小。这也充分说明了学历只是敲门砖。当然也可以用箱线图来进一步考察工作经验对薪酬的影响，例如，从图17.14可以看出对于5年及以上工作经验的人，薪酬明显有一个大的增长。而毕业生和只有一年工作经验的人，工资几乎没有大的差别，所以刚工作就跳槽并不一定是增长工资的好方式。



要得到图17.14，需要使用boxplot（）函数，这里将这段代码留给读者完成。最后让我们来看一下数据分析职位的技能要求，这里使用词云对其进行展示，如图17.15所示。

图17.15 数据分析技能要求

显然，数据分析是该职位最基本的技能要求，此外，数据运营、可视化、数据库、数据挖掘等也是数据分析职位需要的技能。要得到图17.15，首先需要有技能列表，已有数据集中的skillLables列已经提供了该信息，只需要将其转换为列表即可，代码如下。

```
>>> skills = lagou['skillLables'].values.tolist()
>>> skills = [c.replace('[', '') for c in skills]
>>> skills = [c.replace(']', '') for c in skills]
>>> skills = [c.replace("\'", '') for c in skills]
```

上述代码首先将skillLables列的数据转换为列表，之后去掉了列表中的一些无用字符。有了这个技能列表，就可以利用Python中的三方库WordCloud来生成词云（使用pip install wordcloud进行安装），具体代码如下。

```
>>> from wordcloud import WordCloud, STOPWORDS
>>> sw = set(STOPWORDS)
>>> font = r'C:\\Windows\\fonts\\simhei.ttf'
>>> skills_list = " ".join(skills)
>>> wc = WordCloud(width = 800, height = 600,
                    background_color = 'white', font_path = font, max_words = 50,
                    collocations = False, stopwords = sw).generate(skills_list)
>>> fig = plt.figure(figsize = (10,5), dpi = 150)
>>> plt.imshow(wc)
>>> plt.axis("off")
>>> plt.show()
```

上述代码中最关键的函数就是WordCloud（），该函数指定了词云的宽、高、背景色、字体、停用词等。WordCloud库还支持根据某个输入图像来输出词云，感兴趣的读者可以进一步阅读其说明文档，这里不再进行讨论。

17.3.4 思考

通过前面的分析，我们已经对数据分析相关职位有了一定了解，但是这些是否就能解答本章一开始提出的全部问题？或者读者是否可以提出一些新的问题呢？如果能拿到更多数据，能否分析从事数据分析的创业公司都分布在各个城市的哪个区域？最佳跳槽时机是工作几年以后？各位读者又有什么其他问题呢？

第18章 用数据解读成都房价

安得广厦千万间，大庇天下寒士俱欢颜。

——杜甫《茅屋为秋风所破歌》

无论你愿意还是不愿意，解决住房可能是每一个年轻人都不得不面临的问题。既然我们学习了数据分析，那么可不可以用数据分析来帮助了解一个城市的房价高低，判断自己应该在哪个区域买房，某个区域的房产是否有价值洼地呢？本章将用数据来对成都的房价进行解读。

18.1 设定分析目标

18.1.1 问题定义

数据分析的问题需要具体而明确，简单的一句房价分析包含了太多内容。例如，分析中关心的是新房还是二手房，普通住宅还是别墅；从买房者还是开发商角度分析；从投资还是自住角度分析。对这些问题的回答决定了数据分析的重点。这里先假定分析的视角为成都的一名普通买房者，目前想买一套二手房。那么本次数据分析要回答的问题就包括：

- 目前成都的二手房市场平均房价是多少？
- 成都各区二手房价如何？
- 别墅和普通住宅的房价如何？
- 买装修房划算吗？
- 二手房主要分布在哪些区域？

- 小户型与大户型的价格差异如何？
- 能否以热力图的形式将房价在成都地图上显示出来？

18.1.2 获取数据

针对前面的问题，需要考虑从哪里能收集到这些数据。我们的第一反应就是房产网站，这里以链家为例。从网站的成都页面可以看到如图18.1所示的各项信息。

如果进入具体的某一套房产页面，还可以查看更详细的信息，如图18.2所示。



图18.1 链家二手房页面

90.4万 10132元/平米 首付及贷款情况请咨询经纪人			基本信息	
3室2厅	东南	89.23平米	基本属性	房屋户型
低楼层/共32层	平层/毛坯	2018年建/板塔结合		3室2厅1厨2卫
必看！房 诚心卖，真心买				建筑面积
				89.23㎡
				套内面积
				暂无数据
				房屋朝向
				东南
				装修情况
				毛坯
				配备电梯
				有
			所在楼层	低楼层 (共32层)
			户型结构	平层
			建筑类型	板塔结合
			建筑结构	钢筋混凝土
			梯户比例	两梯六户
			产权年限	70年
小区名称 中铁丽景书香 地图 所在区域 温江 温江大学城 绕城高速至第二绕城高速 看房时间 提前预约随时可看 链家编号 106103125055 举报				

图18.2 具体房产信息

很自然地，我们可以考虑利用爬虫来爬取成都的二手房信息，之后利用这些数据就可以对成都的二手房进行自己的解读了。

18.2 解读成都二手房

这里已经提前利用爬虫爬取了2019年7月链家成都发布的大部分二手房数据，一共有7万多条。下面就利用这一数据来解读一下成都的二手房市场。

18.2.1 数据准备

首先导入需要的Python模块，完成一些初始的设置。

```
>>> import pandas as pd
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> pd.set_option('display.max_columns', 10)
>>> pd.set_option('display.max_rows', 10)
>>> %matplotlib inline
# 用来正常显示中文标签
>>> plt.rcParams['font.sans-serif'] = ['SimHei']
# 当坐标轴有负号的时候可以显示负号
>>> plt.rcParams['axes.unicode_minus'] = False
```

提前爬取的房价数据已经以 UTF-8 格式保存在本地 `lianjia_house.csv` 中，图18.3显示了该文件的前几行。



图18.3 爬取的房价信息

接下来需要读入已经爬取下来的房价数据，对该数据有一个初步了解，如图18.4所示。

```
>>> house_data = pd.read_csv('../data/lianjia_house.csv',
    encoding = "utf-8",
    engine = 'python',
    header = None,
    names = ['district', 'area', 'des', 'price', 'detail'],
    usecols = [2, 3, 4, 5, 6])
>>> house_data.head()
```

	district	area	des	price	detail
0	成华	八里小区	此房满两年 无增值税 居家装修 保养好 可拎包入住	112万	中铁二局玛塞城 2室1厅 60.75平米 西北 精装
1	成华	八里小区	八里小区套二顶楼带花园, 户型方正采光好, 对中庭不吵	77.9万	国光苑 2室1厅 78平米 东南 简装
2	成华	八里小区	府青惠园, 居家套二, 客厅带阳台, 位置安静	122万	府青惠园 2室0厅 70.22平米 南 简装
3	成华	八里小区	安静不临高架, 带60平方平台, 居住舒适度好	176万	浅水半岛一期 2室2厅 73平米 东南 简装
4	成华	八里小区	东2.5环, 7号线, 电梯小跃层, 视野很棒	73.5万	春熙苑二期 1室1厅 53.71平米 南 简装

图18.4 读入房价数据

由于爬取下来的文件是以 UTF-8 格式保存，所以在读取时需要使用 `encoding=utf-8` 参数，同时指定 `engine='python'`。根据图18.3可知该文件是没有文件头的，每列的标题需要自行添加（通过 `names` 参数实现）。此外，我们还发现需要的数据只存在于中间5列，因此只需要读入这几列就可以了（通过 `usecols` 参数实现）。从数据的前5行输出可

知数据包含了本次数据分析中需要的二手房区域、小区名称、价格、面积、房屋状态等信息，但是如果显示房价热力图，还需要获取对应的经纬度信息。另外，数据中detail列包含了多项信息：小区名、房屋户型、面积、朝向、装修状态等，后续数据分析需要将以上信息拆分到多列。

首先处理二手房所在小区的经纬度信息，要得到这个信息，首先需要小区的地址，其中小区在成都哪个区，具体区域已经在数据集中的district和area列给出，而小区名称包含在detail列内。通过对前面5行数据的观察，可以发现detail列中不同信息的分隔采用了“|”，因此可以利用字符串函数split（）将这些信息拆分出来，如图18.5所示。

```
>>> house_detail = house_data['detail'].str.split('|', expand = True)
>>> house_detail.head()
```

	0	1	2	3	4	5
0	中铁二局玛塞城	2室1厅	60.75平米	西北	精装	None
1	国光苑	2室1厅	78平米	东南	简装	None
2	府青惠园	2室0厅	70.22平米	南	简装	None
3	浅水半岛一期	2室2厅	73平米	东南	简装	None
4	春熙苑二期	1室1厅	53.71平米	南	简装	None

图18.5 信息拆分

拆分后的数据中第0列即是小区名称，其他各列包含了户型、面积等信息。显然我们需要对house_detail数据的各列进行重新命名，让它们有更明确的意义，不过这里晚些再来处理这个问题，先利用获得的小区名称来构造小区的位置，如图18.6所示。

```
>>> house_data['name'] = house_data['district'] +
    house_data['area'] + house_detail[0]
>>> house_data.head()
```

	district	area	des	price	detail	name
0	成华	八里小区	此房满两年 无增值税 居家装修 保养好 可拎包入住	112万	中铁二局玛赛城 2室1厅 60.75平米 西北 精装	成华八里小区中铁二局玛赛城
1	成华	八里小区	八里小区套二顶楼带花园，户型方正采光好，对中庭不吵	77.9万	国光苑 2室1厅 78平米 东南 简装	成华八里小区国光苑
2	成华	八里小区	府青惠园，居家套二，客厅带阳台，位置安静	122万	府青惠园 2室0厅 70.22平米 南 简装	成华八里小区府青惠园
3	成华	八里小区	安静不临高架，带60平方平台，居住舒适度好	176万	浅水半岛一期 2室2厅 73平米 东南 简装	成华八里小区浅水半岛一期
4	成华	八里小区	东2.5环，7号线，电梯小跃层，视野很棒	73.5万	春熙苑二期 1室1厅 53.71平米 南 简装	成华八里小区春熙苑二期

图18.6 构造小区位置

运行代码`house_data.shape`，可以得到输出（79970，7），说明爬取的数据一共有79970条。通过上述代码将小区的位置与名称信息合并到`name`列，不过仅利用该名称还是没法取得经纬度，因为这里缺少了城市信息。前面已经提到这是成都的二手房，因此可以手动添加新列`city`，其取值均为“成都”，如图18.7所示。

	city	name
0	成都	成华八里小区中铁二局玛赛城
1	成都	成华八里小区国光苑
2	成都	成华八里小区府青惠园
3	成都	成华八里小区浅水半岛一期
4	成都	成华八里小区春熙苑二期

图18.7 添加城市信息

```
>>> house_data['city'] = '成都'
>>> house_data[['city', 'name']].head()
```

将`city`与`name`合并就可以得到所有楼盘的地址。

```
>>> addr = house_data['city'] + house_data['name']
>>> addr.head(3)
```

有了具体的楼盘地址，此时就可以想办法获得经纬度，通过地址获取经纬度的方法很多，这里采用百度提供的一个API来获取，首先定义获取经纬度函数，代码如下。

```

>>> import requests
>>> import json
>>> def getLocation_json(addr):
    url = 'http://api.map.baidu.com/geocoder?address = % s&output = json&key = Nku3ztTbOD
9MI8mfMlR9ZUol0wduESvB' % (addr)
    json_resp = requests.get(url)
    json_result = json_resp.json()           # 转化为 dict 类型
    lng = json_result['result']['location']['lng']   # 经度
    lat = json_result['result']['location']['lat']   # 纬度
    lng_lat = [lng, lat]
    return lng_lat

```

getLocation_json () 函数根据输入的地址，访问api.map.baidu.com之后，将返回的响应转换为JSON格式，最后从该JSON对象中取出经纬度。例如，运行如下代码将得到成都成华八里小区中铁二局玛塞城的经纬度。

```

>>> getLocation_json('成都成华八里小区中铁二局玛塞城')
[104.108344, 30.689665]

```

如果想得到所有楼盘的经纬度，只需要遍历一下全部的楼盘就可以了，代码如下。

```

>>> import time
>>> longitude = list()
>>> latitude = list()
>>> length = len(addr)
>>> for i in range(0,length):
    lon,lati = getLocation_json(addr[i])
    if lon == -1:
        break
    longitude.append(lon)
    latitude.append(lati)
    if i!= 0 and i % 20 == 0:
        time.sleep(0.5)
    if i!= 0 and i % 500 == 0:
        print("Now fetch row: ",i+1)

```

不过在实际分析时如果运行上述代码就会出错。这是由于网站限制，可能在调用了几百次API后就会遇到网站无法访问的错误，这显然是服务端为了保护自己采取的措施。对于这种情况可以考虑采用访问一定次数后就休息一段时间的方法（`time.sleep()`），或者也可以分段获取，每次取1000条，保存到本地，之后再将所有数据连接起来，代码如下。

```
>>> import os
>>> if os.path.exists('../data/location.csv'):
    location_df = pd.read_csv('../data/location.csv')
    else:
        location_df = pd.DataFrame()
>>> lon_lat = {
    "lon":longitude,
    "lat":longitude}
>>> lon_lat_df = pd.DataFrame(lon_lat)
>>> location_df = pd.concat([location_df,lon_lat_df],axis = 0)
>>> location_df.to_csv('../data/location.csv',index = False)
```

假设之前获取的经纬度信息保存在location.csv中，每次获取新的地址前，预先将location.csv读入到location_df，之后将新获取的经纬度lon_lat_df与location_df合并，最后再次保存为location.csv。这似乎看起来是一个不错的方法，不过细心的读者可能会想到虽然二手房记录有79970条，但是楼盘肯定不可能有这么多，也就是说数据中有许多相同楼盘，那么多次获取经纬度就是重复的工作。所以更好的方法实际上是利用`addr.unique()`获得不同楼盘信息，之后再得到这些楼盘的经纬度，保存到一个DataFrame中。将该DataFrame与之前的house_data合并就可以得到需要的数据，代码如下。

```
>>> addr.unique()
array(['成都成华八里小区中铁二局玛塞城 ', ...,
      '成都新都犀浦朗基少帅府 '], dtype = object)
```

假设现在已经完成了所有地址的获取，并将其保存到了location.csv中，下一步工作就是把该数据整合到house_data中，代码如下。

```
>>> location_df = pd.read_csv('../data/location.csv')
>>> house_df = pd.concat([house_data, location_df], axis = 1)
>>> house_df.head()
```

整合结果如图18.8所示。

	district	area	des	price	detail	name	lon	lat
0	成华	八里小区	此房满两年 无增值税 居家装修 保养好 可拎包入住	112万	中铁二局玛塞城 2室1厅 60.75平米 西北 精装	成华八里小区中铁二局玛塞城	104.108344	30.689665
1	成华	八里小区	八里小区套二顶楼带花园，户型方正采光好，对中庭不吵	77.9万	国光苑 2室1厅 78平米 东南 简装	成华八里小区国光苑	104.116106	30.694133
2	成华	八里小区	府青惠园，居家套二，客厅带阳台，位置安静	122万	府青惠园 2室0厅 70.22平米 南 简装	成华八里小区府青惠园	104.107503	30.686327
3	成华	八里小区	安静不临高架，带60平方平台，居住舒适度好	176万	浅水半岛一期 2室2厅 73平米 东南 简装	成华八里小区浅水半岛一期	104.112553	30.688031
4	成华	八里小区	东2.5环，7号线，电梯小跃层，视野很棒	73.5万	春熙苑二期 1室1厅 53.71平米 南 简装	成华八里小区春熙苑二期	104.129863	30.686791

图18.8 二手房数据整合结果

18.2.2 列名调整

数据已经就位，下一步需要做的工作就是对数据进行清洗。前面已经提到house_detail的列名需要处理，所以先来看一下house_detail，具体信息如图18.5所示。

初看house_detail数据的前5行，我们的第一印象是似乎第6列没有什么用（注意：第6列的名称为5），都是空值。那么来具体看看第6列为空到底是什么原因，运行如下代码，得到如图18.9所示的结果。

```
>>> house_detail[house_detail[5].notnull()].head()
```

	0	1	2	3	4	5
1317	万科魅力之城三期	联排别墅	4室2厅	183.4平米	东	精装
1395	万科魅力之城三期	联排别墅	3室2厅	130.8平米	南北	精装
3020	多元总部国际1号	独栋别墅	6室2厅	662.17平米	东	其他
8399	嘉裕第六洲	联排别墅	4室2厅	178.04平米	东南	毛坯
8421	高尔国际2期	独栋别墅	5室3厅	479平米	南	毛坯

图18.9 查看第6列非空的数据

原来第6列不都是空值，对于别墅，该列的信息是房屋装修状态；而如果房屋是普通住宅，则第6列是空值，房屋的装修状态在第5列。基于此，我们将这两种不同的房产类型分别处理保存在villa和apartment两个不同的DataFrame中。

```
>>> villa = house_detail[house_detail[5].notnull()]
>>> apartment = house_detail[house_detail[5].isnull()]
>>> apartment = apartment[[0,1,2,3,4]]
```

接下来将villa和apartment中各列数据存在的空白字符都去掉，代码如下。

```
>>> for i in range(0,5):
    apartment[i] = apartment[i].str.strip()
>>> for i in range(0,6):
    villa[i] = villa[i].str.strip()
```

最后再给apartment和villa的列赋予有意义的名称，重命名结果如图18.10和图18.11所示。

```
>>> apartment.columns = ['house_name', 'rooms', 'space',
    'orientation', 'status']
>>> apartment.head()
>>> villa.columns = ['house_name', 'type', 'rooms', 'space',
    'orientation', 'status']
>>> villa.head()
```


	house_name	rooms	space	orientation	status
0	中铁二局玛塞城	2室1厅	60.75平米	西北	精装
1	国光苑	2室1厅	78平米	东南	简装
2	府青惠园	2室0厅	70.22平米	南	简装
3	浅水半岛一期	2室2厅	73平米	东南	简装
4	春熙苑二期	1室1厅	53.71平米	南	简装

图18.10 重命名列（普通住宅）

到了这里，似乎数据已经处理完毕，但是只需要再仔细查看一下apartment数据，就会发现预期的rooms列中有异常信息，运行如下代码查看。

```
>>> apartment[apartment['rooms'] == '车位'].head()
```

	house_name	type	rooms	space	orientation	status
1317	万科魅力之城三期	联排别墅	4室2厅	183.4平米	东	精装
1395	万科魅力之城三期	联排别墅	3室2厅	130.8平米	南北	精装
3020	多元总部国际1号	独栋别墅	6室2厅	662.17平米	东	其他
8399	嘉裕第六洲	联排别墅	4室2厅	178.04平米	东南	毛坯
8421	高尔国际2期	独栋别墅	5室3厅	479平米	南	毛坯

图18.11 重命名列（别墅）

如图18.12所示，apartment数据中有些是车位的信息，因此需要将车位信息放到另一个数据集parking中，代码如下。

```
>>> parking = apartment[apartment['rooms'] == '车位']
>>> apartment = apartment[apartment['rooms'] != '车位']
```

	house_name	rooms	space	orientation	status
607	浅水半岛一期	车位	28.57平米	东 东南 南 西南 西	None
628	上行东方	车位	26.41平米	东	None
629	蓝光COCO时代	车位	31.83平米	东	None
630	上行东方	车位	27.9平米	南	None
632	浅水半岛三期	车位	28平米	东 东南 西南 南 西	None

图18.12 车位信息

现在最初的数据集被分成了3部分：普通房产、别墅、车位。利用如下代码可以查看各数据集的大小，parking、apartment和villa 3个数据集的行数加起来刚好是79970。

```
>>> parking.shape, apartment.shape, villa.shape, house_detail.shape
((3705, 5), (74799, 5), (1466, 6), (79970, 6))
```

因为本次分析的目标是分析成都房价，所以我们应该把别墅和普通住宅放到一起来分析。为了区分别墅与普通住宅，我们给apartment数据集加上一列type，取值为“普通住宅”，代码如下。

```
>>> apartment['type'] = '普通住宅'
>>> apartment = apartment[['house_name', 'type', 'rooms',
                             'space', 'orientation', 'status']]
>>> apartment_villa = pd.concat([apartment, villa], axis = 0)
>>> apartment_villa.sample(5)
```

结果如图18.13所示。之后再将apartment与villa合并，代码如下。

```
>>> house_df = pd.concat([house_df[['district', 'area',
                                     'price', 'lon', 'lat']], apartment_villa], axis = 1)
>>> house_df.sample(5)
```


	house_name	type	rooms	space	orientation	status
65525	花样年花样城	普通住宅	3室2厅	119.08平米	东	简装
29510	顶峰水岸汇景	普通住宅	2室2厅	82.38平米	东	精装
69301	龙锦慧苑	普通住宅	4室2厅	155平米	东	简装
51804	欧洲印象	普通住宅	2室2厅	58平米	东	精装
49034	天府豪庭	普通住宅	3室2厅	101.08平米	南	简装

图18.13 添加“普通住宅”列

合并结果如图18.14所示。

	district	area	price	lon	lat	house_name	type	rooms	space	orientation	status
41775	青羊	八宝街	120万	104.060388	30.680234	西岸锦景	普通住宅	1室1厅	58.97平米	南	简装
38796	郫都	红光	90万	103.971056	30.798798	晶宝塞纳国际	普通住宅	3室2厅	83平米	东南	精装
11093	高新	市一医院	216万	104.059022	30.602937	南城都汇汇雅园二期	普通住宅	2室2厅	89.37平米	南	精装
70767	武侯	华西	136万	104.072522	30.647360	人民南路三段16号	普通住宅	2室1厅	80平米	东	精装
6596	成华	万年场	146.5万	104.140998	30.652351	鲁能公馆	普通住宅	2室1厅	61平米	南	精装

图18.14 房产数据合并结果

实际的数据清洗过程经常不是一帆风顺的，这一工作可能要持续很久，所以经常将工作成果保存是一个良好习惯，下面的代码将处理后的房产数据保存到本地。

```
>>> house_df.to_csv('../data/lianjia_final.csv', index = False)
>>> parking.to_csv('../data/lianjia_parking.csv', index = False)
```

18.2.3 数据类型转换

将前一小节处理后的数据再次读入，如图18.15所示，我们发现其中的price、space、rooms等信息都是字符串，因此需要进一步处理这些列的数据。

```
>>> df = pd.read_csv('../data/lianjia_final.csv')
>>> df.head()
```

	district	area	price	lon	lat	...	type	rooms	space	orientation	status
0	成华	八里小区	112万	104.108344	30.689665	...	普通住宅	2室1厅	60.75平米	西北	精装
1	成华	八里小区	77.9万	104.116106	30.694133	...	普通住宅	2室1厅	78平米	东南	简装
2	成华	八里小区	122万	104.107503	30.686327	...	普通住宅	2室0厅	70.22平米	南	简装
3	成华	八里小区	176万	104.112553	30.688031	...	普通住宅	2室2厅	73平米	东南	简装
4	成华	八里小区	73.5万	104.129863	30.686791	...	普通住宅	1室1厅	53.71平米	南	简装

5 rows x 11 columns

图18.15 读入处理后的数据

如果只考虑前5行数据，我们的第一反应是直接提取price、space、rooms列中的数字信息就可以完成数据类型的转换。现实总是不尽如人意，如果直接提取price列中的数字转换为浮点数，我们会碰到无法将NaN转换为float类型的错误，其实这是由于数据爬虫爬取的数据中存在错误而导致。输入如下代码，得到图18.16的结果。

```
>>> df.iloc[5726:5730,:]
```

	district	area	price	lon	lat	...	type	rooms	space	orientation	status
5726	成华	驷马桥	41万	104.102753	30.706117	...	普通住宅	1室1厅	36.21平米	东南	简装
5727	成华	驷马桥	150万	104.101956	30.704810	...	普通住宅	3室1厅	86.87平米	东	精装
5728	成华	驷马桥	总价低户型好通风采光俱佳	104.098105	30.701102	...	普通住宅	NaN	NaN	NaN	NaN
5729	成华	驷马桥	168万	104.098871	30.705401	...	普通住宅	2室1厅	88平米	东	毛坯

4 rows x 11 columns

图18.16 查看5726~5730行

可以发现5728行的price信息存在错误，同时该行的space、orientation等信息为空。如果是真实的数据分析项目，那么我们需要考虑是不是爬虫有问题，需要修正该信息。或者考虑到当前一共有7万多条数据，可以利用如下代码将错误信息行删除，结果如图18.17所示。

```
>>> df.drop([5728], inplace = True)
>>> df.iloc[5726:5730,:]
```

	district	area	price	lon	lat	...	type	rooms	space	orientation	status
5726	成华	驷马桥	41万	104.102753	30.706117	...	普通住宅	1室1厅	36.21平米	东南	简装
5727	成华	驷马桥	150万	104.101956	30.704810	...	普通住宅	3室1厅	86.87平米	东	精装
5729	成华	驷马桥	168万	104.098871	30.705401	...	普通住宅	2室1厅	88平米	东	毛坯
5730	成华	驷马桥	190万	104.104989	30.721520	...	普通住宅	3室2厅	124平米	东	精装

4 rows x 11 columns

图18.17 删除错误信息行

对5728行处理完毕后，如果继续进行数据清洗，会发现65332行也存在问题，因此采用同样方法对其进行处理，代码如下。

```
>>> df.drop([65332], inplace = True)
```

此外，读者应该已经注意到space列存在空值的情况。分析房价的时候，因为我们需要根据面积和总价来计算房价，所以对这类数据也需要丢弃。如下代码即可完成这一工作。

```
>>> df = df[df['space'].notnull()]
```

处理完错误数据后，我们可以再次尝试提取之前提到的数字信息，代码如下。

```
>>> df['price'] = df['price'].str.extract("(\d+\.\?\d+)").astype(float)
>>> df['space'] = df['space'].str.extract("(\d+\.\?\d+)").astype(float)
```

读到这里，有的读者可能就会有疑问了。作者是怎么知道5728行有问题的？实际上数据清洗是一个不断迭代的过程，在真实的分析中，我们是先执行前面的提取数字和类型转换代码，这时就会遇到类型转换错误。那么要解决这一问题，我们可能就会只执行 `df['price'].str.extract("(\d+\.\?\d+)")` 这段代码，之后在其返回的结果中利用 `isnull()` 函数来查看哪一行数据为空，这样就会得到前面提到的5728和65332行。完成了price、space信息的提取，我

们发现房型信息中给出的是几房几厅，分析中需要将该信息分为两部分，分别记录房间与厅，代码如下。

```
>>> rooms = df['rooms'].str.extract("(\\d+ )室(\\d+ )")
>>> df['livingRooms'] = rooms[0].astype(int)
>>> df['halls'] = rooms[1].astype(int)
>>> df.head()
```

结果如图18.18所示。

	district	area	price	lon	lat	...	space	orientation	status	livingRooms	halls
0	成华	八里小区	112.0	104.108344	30.689665	...	60.75	西北	精装	2	1
1	成华	八里小区	77.9	104.116106	30.694133	...	78.00	东南	简装	2	1
2	成华	八里小区	122.0	104.107503	30.686327	...	70.22	南	简装	2	0
3	成华	八里小区	176.0	104.112553	30.688031	...	73.00	东南	简装	2	2
4	成华	八里小区	73.5	104.129863	30.686791	...	53.71	南	简装	1	1

5 rows x 13 columns

图18.18 拆分房型信息

大功告成，目前似乎已经获得了房产的相关数据了。处理后的数据只有76263行（运行代码df.shape进行查看），这是由于我们丢弃了一些错误数据。现在来检查一下最终的数据类型，代码如下。

```
>>> df.dtypes
district      object
area          object
price         float64
lon           float64
lat           float64
house_name    object
type          object
rooms         object
space         float64
orientation   object
status        object
livingRooms   int32
halls         int32
dtype: object
```

房价、面积等数据类型已经是浮点型，房间与厅的数目为整型。不过我们似乎还缺少每平方米价格数据，该数据可以通过总价除以面积得到。

```
>>> df['priceSquare'] = (df['price']/df['space']).round(4) * 10000
```

最终得到的数据如下，如图18.19所示。

```
>>> df = df[['house_name', 'type', 'district', 'area',  
            'space', 'price', 'priceSquare', 'lon', 'lat',  
            'orientation', 'status', 'livingRooms', 'halls']]  
>>> df.sample(5)
```

	house_name	type	district	area	space	price	priceSquare	lon	lat	orientation	status	livingRooms	halls
15640	龙湖时代天街	普通住宅	高新西	高新西	48.94	40.0	8173.0	103.912523	30.777052	东南	简装	1	0
67130	供电局家属院	普通住宅	温江	温江老城	103.92	73.0	7025.0	103.863573	30.688487	东	简装	3	2
46849	成都青羊万达广场	普通住宅	青羊	外光华	32.07	42.0	13096.0	103.978276	30.681982	东	毛坯	1	0
27478	交大智能二期	普通住宅	金牛	九里堤	177.90	316.0	17763.0	104.056654	30.709609	东南	精装	4	2
52576	润扬书院阁	普通住宅	双流	航空港	53.76	69.0	12835.0	103.996624	30.584570	北	精装	1	1

图18.19 最终房价数据

18.2.4 数据解读

1. 综合房价信息

到一个城市买房，可能最先想了解的就是这个城市的房价水平。而这里我们想知道的显然是平均房价以及价格分布。输入如下代码即可达成目标，结果如图18.20所示。

```
>>> df.describe()
```

	space	price	priceSquare	lon	lat	livingRooms	halls
count	76263.000000	76261.000000	76261.000000	76263.000000	76263.000000	76263.000000	76263.000000
mean	99.050024	166.404885	15945.723201	104.054618	30.650110	2.495168	1.490093
std	57.658514	176.405092	6447.867738	0.214618	0.128884	1.010487	0.623891
min	8.800000	9.200000	2298.000000	103.562053	22.235979	0.000000	0.000000
25%	67.980000	90.000000	11667.000000	103.984520	30.596707	2.000000	1.000000
50%	87.700000	128.000000	14754.000000	104.053707	30.658761	2.000000	2.000000
75%	118.000000	185.000000	18785.000000	104.109579	30.700240	3.000000	2.000000
max	1814.220000	12000.000000	146341.000000	113.968541	31.007646	20.000000	7.000000

图18.20 综合房价信息

从代码的输出可以看出成都的平均房价大约为15945元 / 平方米，最贵的是14.6万元 / 平方米，最便宜的是2298元 / 平方米。在成都买一套房平均要166万元，大部分二手房都是3房或3房以下，平均100平方米左右。进一步分析发现，75%的二手房都是118平方米左右，售价185万元以下。不过，正如在统计收入时我们经常被平均一样，房价是否也存在被平均的情况呢？在数据准备时我们已经知道上面的数据包含了别墅与普通住宅，那么是否别墅拉高了平均价呢？房价按不同类型区分，价格又是多少呢？因此可用如下代码进行分组统计。

```
>>> df.groupby('type')['price'].mean()
type
双拼别墅      785.483871
叠拼别墅      505.806154
普通住宅      155.649478
独栋别墅     1410.901961
联排别墅       568.957377
Name: price, dtype: float64

>>> df.groupby('type')['priceSquare'].mean()
type
双拼别墅      24414.387097
叠拼别墅      21087.732308
普通住宅      15811.870031
独栋别墅      33041.596078
联排别墅      19972.660782
Name: priceSquare, dtype: float64
```


按照房产类型统计，普通住宅平均156万元一套；而别墅里面最贵的是独栋别墅，平均一套大约1411万元，最便宜的是叠拼别墅，平均一套506万元左右。普通住宅平均价格是15811元 / 平方米，基本受别墅影响不大，虽然叠拼别墅平均总价低些，但是在平均价格上，联排别墅似乎更低些，为19973元 / 平方米。对成都的房价，现在我们已经有了一个大致的了解，读者朋友们，您是准备买别墅还是普通住宅呢？买在哪个区域呢？让我们继续来为您解读。

2. 成都各区域房价

对成都房价有了大致了解后，接下来我们关注的就是各区域的价格情况，不同房型价格是多少，不同面积房产价格是多少。在前面的综合房价信息中，我们看到有的房产只有厅，而有的房间居然有20间。我们具体来看看这些livingrooms=0的房子吧，如图18.21所示。

```
>>> df[df['livingRooms'] == 0]
```

	house_name	type	district	area	space	price	priceSquare	lon	lat	orientation	status	livingRooms	halls
1732	和泓东28	普通住宅	成华	东郊记忆	34.92	42.8	12257.0	104.134746	30.666928	东南	简装	0	1
35655	壹中心	普通住宅	龙泉驿	龙泉驿城区	46.00	60.0	13043.0	104.248241	30.573475	东南	毛坯	0	1
35689	壹中心	普通住宅	龙泉驿	龙泉驿城区	46.00	58.0	12609.0	104.248241	30.573475	南北	精装	0	2
40255	一里阳光	普通住宅	郫都	郫县万达	30.26	31.0	10245.0	103.906297	30.811277	西南	毛坯	0	0
43953	左右	普通住宅	青羊	光华泡小	49.41	55.0	11131.0	104.016875	30.669354	东	简装	0	0
47978	天祥广场	普通住宅	青羊	西南财大	193.80	380.0	19608.0	104.027469	30.667302	南北	简装	0	0

图18.21 livingrooms=0

对这些房子大致进行一下浏览就可以发现，很可能这些数据都存在错误，如最后一条天祥广场的房子面积为193.8平方米，怎么可能是0室0厅呢？因此，这些数据如果不能修正的话，也应该剔除后再进行分析，这一工作就留给读者朋友了。接下来再继续看看livingrooms≥10的房产吧，如图18.22所示。

```
>>> df[df['livingRooms']>=10]
```

	house_name	type	district	area	space	price	priceSquare	lon	lat	orientation	status	livingRooms	halls
33994	中粮御岭湾	独栋别墅	龙泉驿	阳光城	389.00	3500.0	89974.0	104.314249	30.608306	南	精装	20	5
33998	中粮御岭湾	普通住宅	龙泉驿	阳光城	496.79	1500.0	30194.0	104.314249	30.608306	东南	精装	10	3

图18.22 livingrooms $\geq$ 10

原来这一豪宅在龙泉的中粮御岭湾，不过这里似乎应该有些疑问，389平方米，有20间房，价格要3500万元，而另一套房497平方米只要1500万元。是否是网站数据有误？我们先不纠结于此，还是看看成都各区域的房产价格吧。老成都以前有着“东穷北匪，南富西贵”的说法，那么房价是否在以上各区域有所不同呢？要回答此问题，输入如下代码，结果如图18.23所示。

```
>>> price_by_district = df.groupby(['district', 'type'])['price',  
    'priceSquare'].mean()  
>>> price_by_district.reset_index(inplace = True)  
>>> price_by_district[price_by_district['type'] == '普通住宅']\  
    .sort_values('priceSquare', ascending = False)
```


	district	type	price	priceSquare
62	高新	普通住宅	236.935818	21236.565036
51	锦江	普通住宅	194.030554	21185.851520
58	青羊	普通住宅	171.954882	18411.852449
7	天府新区	普通住宅	226.430212	17711.145318
29	武侯	普通住宅	171.179658	17391.948034
19	成华	普通住宅	139.487899	16682.231293
2	双流	普通住宅	160.475643	15904.005413
47	金牛	普通住宅	143.245017	15504.820833
68	龙泉驿	普通住宅	132.791846	13422.082816
12	天府新区南区	普通住宅	186.004986	13272.528090
24	新都	普通住宅	115.911606	12640.606921
64	高新西	普通住宅	108.070754	12328.160065
41	郫都	普通住宅	113.140513	12220.637277
34	温江	普通住宅	124.685721	11582.472485
44	都江堰	普通住宅	156.962963	10666.407407
37	简阳	普通住宅	108.452086	10008.688525
17	彭州	普通住宅	89.718890	8472.589823
54	青白江	普通住宅	77.868085	7072.680851

图18.23 区域房价（普通住宅）

从输出结果来看，南边的成都高新区和西边的青羊区位列平均房价第一和第三，分别约为21237元 / 平方米和18412元 / 平方米，而城市中心的锦江区位列第二，平均21186元 / 平方米。此外，成都的新贵——南边的天府新区，平均价格也达到了17711元 / 平方米，东边的成华区和北边的金牛区价格要比上述区域相对低些。而新并入成都的彭州、青白江、简阳目前价格都较低。此外，温江与郫都的价格也只是在12000元 / 平方米左右。接下来再看看别墅的情况，如图18.24所示。

```
>>> price_by_district[price_by_district['type']!= '普通住宅']\
    .sort_values(['priceSquare'],ascending = False)
```

别墅与普通住宅稍有不同，其中金牛区和青羊区占据了前3位，南边的天府新区、高新区紧随其后，彭州、青白江仍位于最后。

	district	type	price	priceSquare
48	金牛	独栋别墅	2400.000000	49133.000000
56	青羊	双拼别墅	1623.333333	45571.833333
59	青羊	独栋别墅	1733.333333	43098.666667
27	武侯	双拼别墅	1800.000000	41470.500000
8	天府新区	独栋别墅	1684.765625	41310.500000
63	高新	联排别墅	1501.500000	39847.300000
13	天府新区南区	独栋别墅	1723.967033	38162.648352
...	...	...	...	...
46	都江堰	联排别墅	248.000000	13929.000000
14	天府新区南区	联排别墅	334.513483	13496.213483
16	彭州	叠拼别墅	437.000000	13025.500000
67	龙泉驿	叠拼别墅	280.857143	12857.952381
15	彭州	双拼别墅	290.000000	8869.000000
55	青白江	联排别墅	225.000000	8588.000000
18	彭州	独栋别墅	281.333333	8483.000000

53 rows x 4 columns

图18.24 区域房价（别墅）

3. 房屋面积与价格的关系

很明显，房价肯定与房屋的大小户型有关，输入如下代码，可以得到如图18.25所示的图形。

```
>>> import seaborn as sns
>>> sns.lmplot('space', 'priceSquare', data = df[df['type'] == '普通住宅'])
```

显然，整体上房价和面积还是存在正比关系，不过也存在一些异常点，有兴趣的读者可以进一步对此进行分析，说不定还能买到性价比极高的房产哦！

4. 房价与装修的关系

很多人说买装修好的二手房会比较划算，实际情况是不是如此呢？运行如下代码，来看看各区不同装修状态的房产价格如何，如图18.26所示。

```
>>> district_status_group = apartment_df.\
    groupby(['district', 'status']).mean()
>>> district_status_vis = district_status_group.\
    reset_index()[['district', 'status', 'priceSquare']]
>>> district_status_vis.sample(10)
```

从图18.26可以看出，基本上没有装修的房产和已装修的房产价格差距很小，说明前面说法确实有一定道理。不过对任何事情我们都应该大胆假设，小心求证。会不会是装修的房子房龄相对更老，而毛坯房大多是刚买没有几年的房呢？因为爬取的数据里面没有房龄信息，显然无法直接得出这一结论。

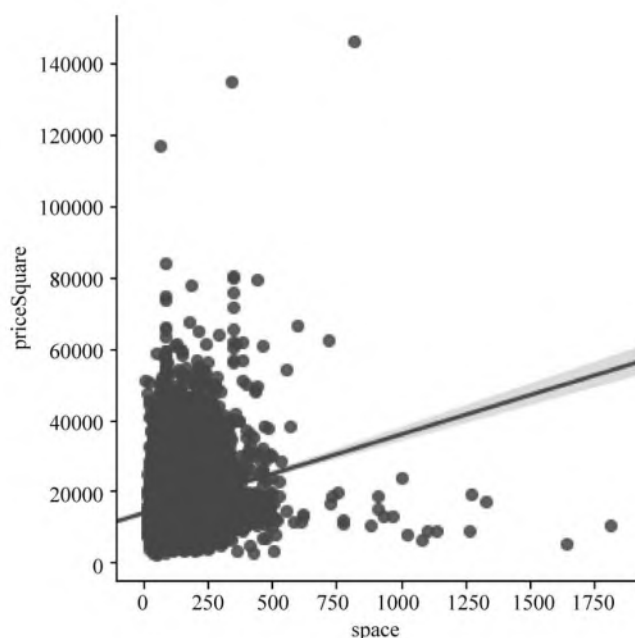


图18.25 房价与面积的关系

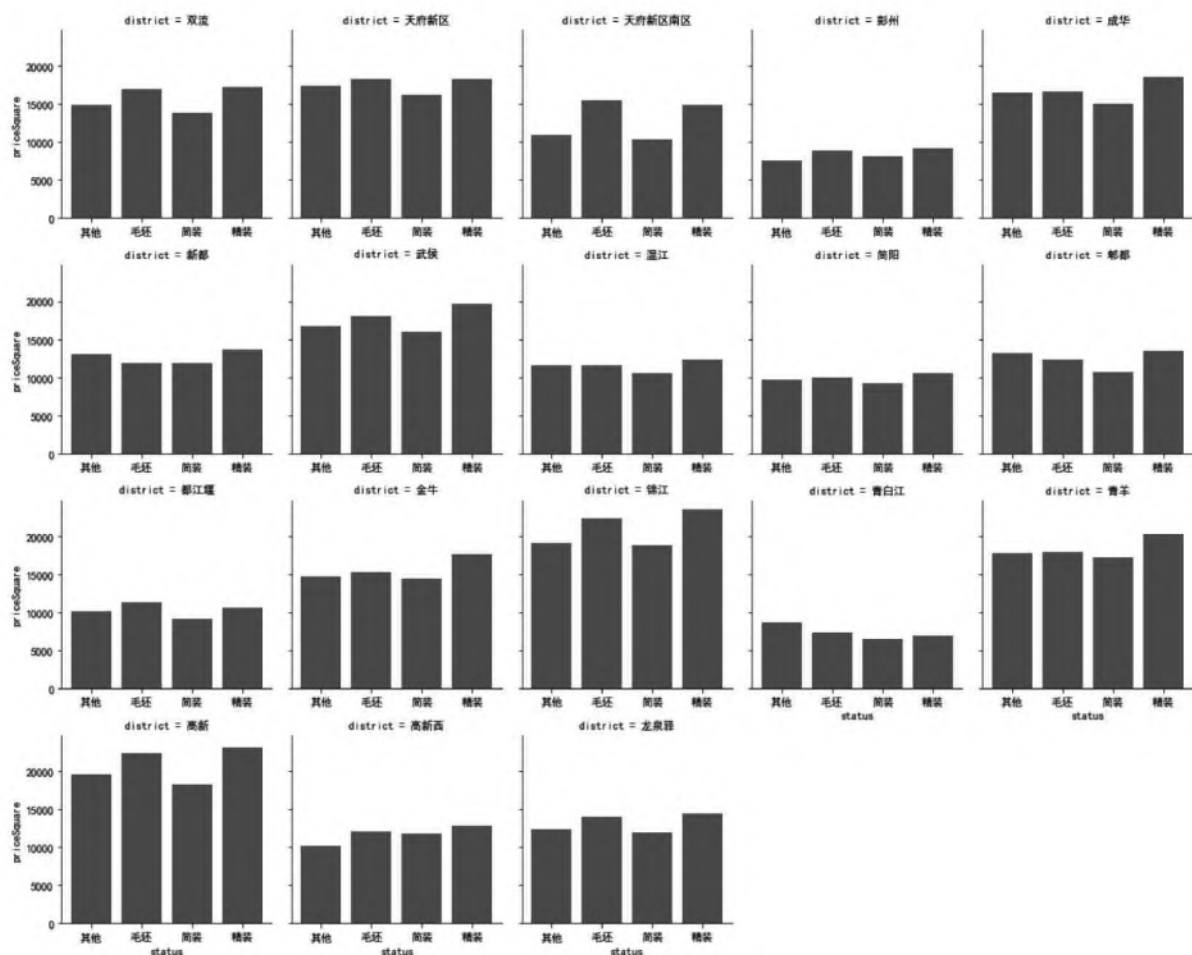


图18.26 各区不同类型房产均价

5. 成都房价热力图

通过前面的分析，我们对成都房价应该已经有了基本了解。如果想要进一步将房产位置和价格结合起来分析，就需要使用热力图来完成这一工作。首先运行如下代码来看看成都的二手房主要来自哪些区域。

```

>>> import folium
>>> from folium import plugins
>>> price_heatmap = folium.Map(location=cd_coords, zoom_start=11)
>>> price_heatmap.add_child(plugins.HeatMap(\
    [[row["lat"],row["lon"]] for name, row in apartment_df.iterrows()])))
>>> price_heatmap.save("../data/price_heatmap.html")
>>> price_heatmap

```

运行上述代码可以得到如图18.27所示的热力图。

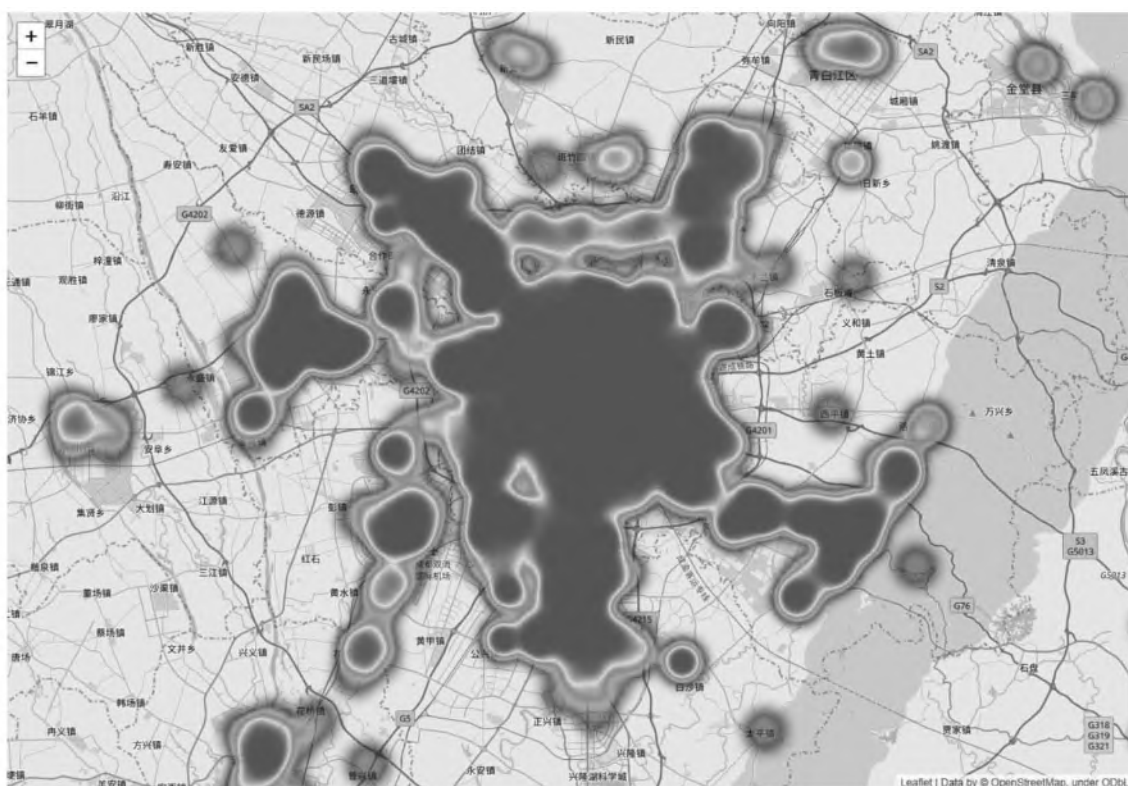


图18.27 二手房分布热力图

从二手房分布热力图可以看出成都二手房交易主要还是在主城区，之后是温江、双流、郫县、龙泉以及高新区。同时，北边三环路到绕城高速之间似乎没有什么二手房，也许未来这一区域会有大量新楼盘。接下来让我们看看房价的热力图，首先定义一个绘图辅助函数，代码如下。

```

>>> def map_points(df, lat_col = 'lat', lon_col = 'lon', zoom_start = 11, \
    plot_points = False, pt_radius = 15, \
    draw_heatmap = False, heat_map_weights_col = None, \
    heat_map_weights_normalize = True, heat_map_radius = 15):

    middle_lat = df[lat_col].median()
    middle_lon = df[lon_col].median()
    curr_map = folium.Map(location=[middle_lat, middle_lon],
        zoom_start = zoom_start)

    # add points to map
    if plot_points:
        for _, row in df.iterrows():
            folium.CircleMarker([row[lat_col], row[lon_col]],
                radius = pt_radius,
                popup = row['name'],
                fill_color = "#3db7e4",
            ).add_to(curr_map)

    # add heatmap
    if draw_heatmap:
        # convert to (n, 2) or (n, 3) matrix format
        if heat_map_weights_col is None:
            cols_to_pull = [lat_col, lon_col]
        else:
            # normalize
            if heat_map_weights_normalize:
                df[heat_map_weights_col] = \
                    df[heat_map_weights_col] /\
                    df[heat_map_weights_col].sum()
            cols_to_pull = [lat_col, lon_col, heat_map_weights_col]
        lat_lon = df[cols_to_pull].values
        curr_map.add_child(plugins.HeatMap(lat_lon,
            radius = heat_map_radius))
    curr_map.save('../data/priceSquare_heatmap.html')

    return curr_map

```

map_points () 函数可以针对输入数据绘制基于 heat_map_weights_col 值的热力图，运行如下代码，将得到如图18.28所示的热力图。

```
>>> apartment_df_geo = apartment_df.dropna()
>>> map_points(apartment_df_geo, plot_points=False, draw_heatmap=True,
               heat_map_weights_normalize=False, \
               heat_map_weights_col='priceSquare', heat_map_radius=9)
```

如图18.29所示，进一步放大热力图可以发现，南边的玉林区域房价存在区域高点，西边永陵路区域、东北方向红星路附近都存在区域高点。有意思的是，传统的富人区桐梓林却没有多少二手房，原因要么是数据不够，要么就是该区域二手房源确实很少。

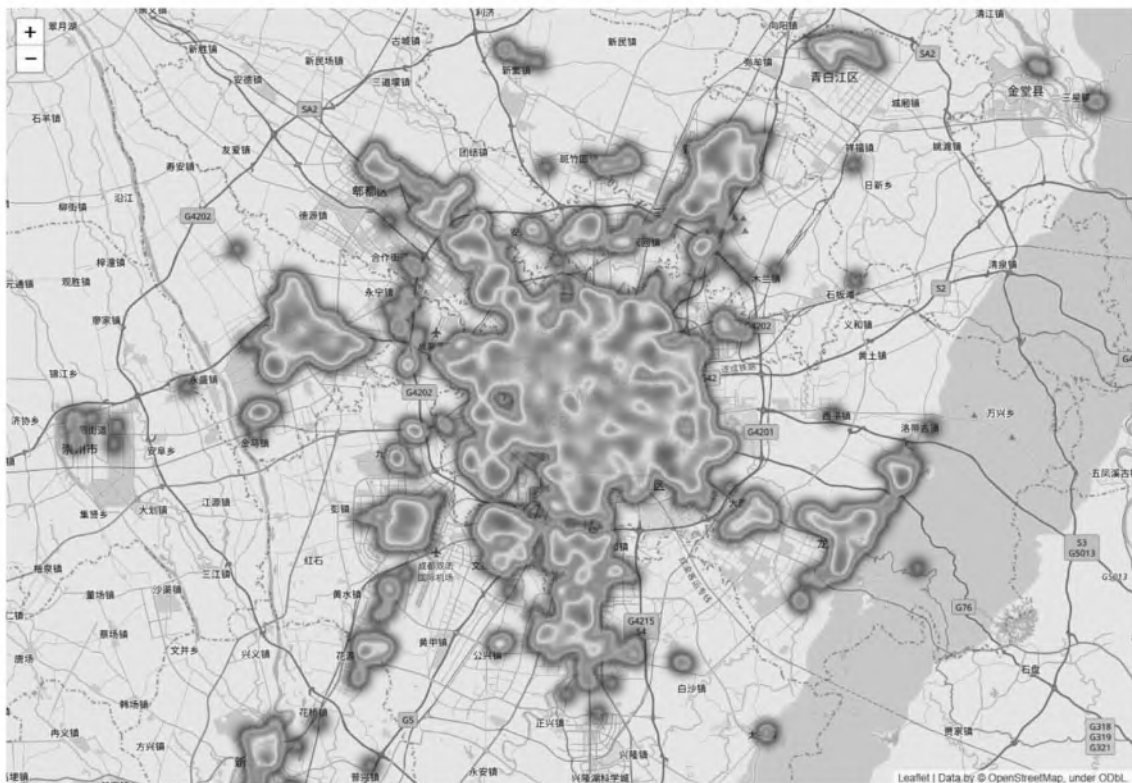


图18.28 房价热力图

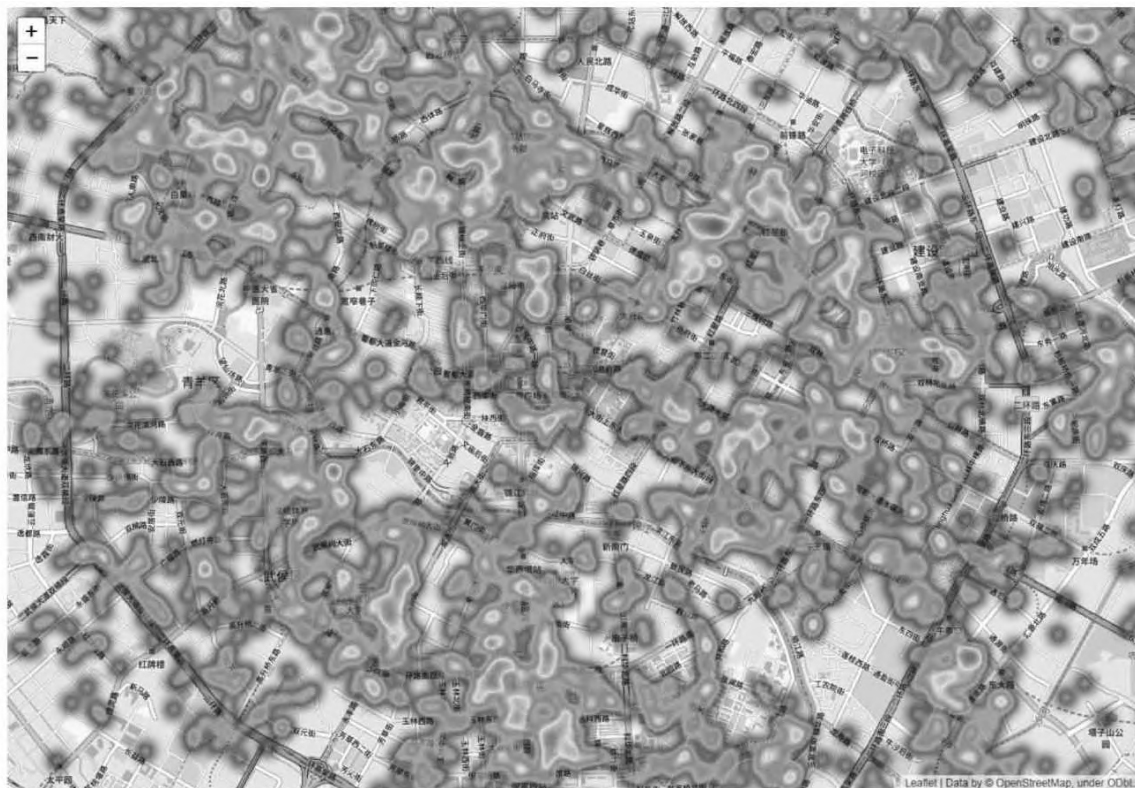


图18.29 放大房价热力图

18.2.5 思考

现在我们对成都房价已经算是有了初步了解，这些是不是就够了呢？在分析的过程中会不会有新的问题产生呢？例如，投资者可能会关注买房的区域内二手房是否好卖；如果能结合租房数据，还可以查看这里的租售比如何，某个区域的交通与房价的关系，学区房的影响，等等。数据分析是一个永无止境的过程，我们不断提出问题，寻找答案，之后再次检视问题，提出新问题。以上这些问题，就留给读者朋友吧！

第19章 时间序列分析

时间是人能消费的最有价值的东西。

——狄奥佛拉斯塔

生活中我们会碰到大量与时间有关的数据，如投资者每天关注的股票价格、每月的经济数据；普通老百姓每天关注的气温；企业、零售商关注的每日销售数据和库存等。这些数据都称为时间序列数据（Time Series Data），即在不同时间上收集到的数据，用于描述现象随时间变化的情况。正所谓“时间从来不语，却回答了所有的问题”，这类数据反映了事物、现象等随时间的变化状态或程度。本章将为大家介绍如何在利用Python进行时间序列分析。

19.1 认识时间序列数据

19.1.1 读入时间序列数据

作为一名个人投资者，最关心的事情肯定是自己购买的股票每天的收盘价如何。而某一股票每天的收盘价就是一个典型的时间序列数据。例如，招商银行的每日收盘价如下。

```
>>> df = pd.read_csv('../data/stock/600036.xls', \
                        usecols = ['date', 'close'], parse_dates = ['date'], sep = '\t')
>>> df.head()
```

	date	close
0	2014-01-20	5.97
1	2014-01-21	6.03
2	2014-01-22	6.20
3	2014-01-23	6.02
4	2014-01-24	5.93

上面的数据中按照时间顺序记录了每天招商银行的收盘价，这就是我们所说的时间序列。对于时间序列，有时也会选择在读入数据时将时间数据所在列作为其索引。例如，对上述数据，使用如下代码读入时将时间列（date）作为索引。

```
>>> df = pd.read_csv('../data/stock/600036.xls', \
                      usecols = ['date', 'close'], \
                      parse_dates = ['date'], index_col = 'date', sep = '\t')
>>> df.head()
              close
date
2014-01-20    5.97
2014-01-21    6.03
2014-01-22    6.20
2014-01-23    6.02
2014-01-24    5.93
```

很多时候，对于时间序列，某一时间不只是观察某一个变量的值，我们还会观察多个变量的值。例如，对于股票信息，会同时观察一天的开盘价、收盘价、最高价、最低价、成交量，如图19.1所示。

```
>>> df = pd.read_csv('../data/stock/600036.xls', \
                      parse_dates = ['date'], index_col = 'date', sep = '\t')
>>> df.head()
```

	open	high	low	close	vol
date					
2014-01-20	6.01	6.10	5.92	5.97	27537862
2014-01-21	6.04	6.18	5.98	6.03	25159216
2014-01-22	6.06	6.28	5.98	6.20	50123656
2014-01-23	6.17	6.21	5.99	6.02	38634112
2014-01-24	5.98	6.06	5.89	5.93	52264196

图19.1 多个变量信息

19.1.2 时间序列数据的可视化

对于时间序列数据，其可视化也非常简单，通常将时间作为X轴，将观测数据作为Y轴。例如前面的招商银行每日收盘价，可以用如下代码可视化，得到图19.2。

```
>>> def plot_df(df, x, y, title="", xlabel='日期', ylabel='股价', dpi=100):  
    plt.figure(figsize=(16,5), dpi=dpi)  
    plt.plot(x, y, color='red')  
    plt.gca().set(title=title, xlabel=xlabel, ylabel=ylabel)  
    plt.show()  
>>> plot_df(df, x=df.index, y=df.close, title='招商银行收盘价')
```

从图19.2可以看出，招商银行的股价从2014年初的6元左右，涨到了2019年10月的35元左右。5年时间，基本上有6倍的回报，从投资的角度看，长期持有这类股票是最好的策略。不过从时间序列数据分析的角度看，这一时间序列数据存在着明显的趋势性。下面再来看一个澳大利亚糖尿病药品的月销售数据，如图19.3所示。

```
# 数据来源于 fpp pacakge in R.  
>>> df_drug = pd.read_csv('../data/a10.csv', \  
    parse_dates=['date'], index_col='date')  
>>> plot_df(df_drug, x=df_drug.index, y=df_drug.value, \  
    title='澳大利亚 1992 - 2008 糖尿病药品月销售额', ylabel='销售额')
```



图19.2 招商银行收盘价

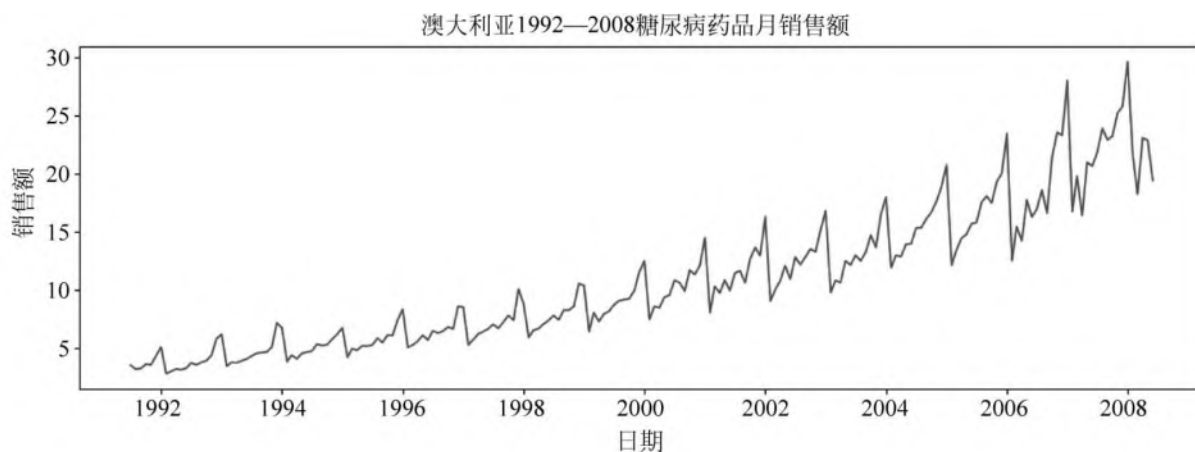


图19.3 澳大利亚糖尿病药品销售数据

从图19.3可以清晰地看出，该数据与前面招商银行的股价数据类似，也存在强烈的趋势性，即逐年增长。不过除了趋势性特征以外，这一时间序列数据还存在另一特征——季节性。从图19.3看出糖尿病药品的销售随着季节变化总是存在相同的波峰和波谷。为了进一步观察这类特征，可以按照月份来对比每年的销售数据，代码如下。

```

>>> df_drug.reset_index(inplace = True)
# 提取年和月信息
>>> df_drug['year'] = [d.year for d in df_drug['date']]
>>> df_drug['month'] = [d.strftime('%b') for d in df_drug['date']]
>>> years = df_drug['year'].unique()
# 用不同颜色用来绘制每年销售
>>> np.random.seed(100)
>>> mycolors = np.random.choice(list(mpl.colors.XKCD_COLORS.keys()),\
                                len(years), replace = False)
# 绘图
>>> plt.figure(figsize = (16,12), dpi = 80)
>>> for i, y in enumerate(years):
    if i > 0:
        plt.plot('month', 'value', \
                  data = df_drug.loc[df_drug.year == y, :], \
                  color = mycolors[i], label = y)
        plt.text(df_drug.loc[df_drug['year'] == y, :].shape[0] - .9, \
                  df_drug.loc[df_drug['year'] == y, \
                  value'][-1:].values[0], y, \
                  fontsize = 12, color = mycolors[i])
>>> plt.gca().set(xlim = (-0.3, 11), ylim = (2, 30), \
                  ylabel = '药品销售', xlabel = '月')
>>> plt.yticks(fontsize = 14, alpha = .7)
>>> plt.title("药品销售季节性数据", fontsize = 20)
>>> plt.show()

```

上述代码首先将索引还原成列date，之后利用date信息创建了新的year和month列。紧接着在后面的绘图代码中，因为1991年数据不全，所以代码略过了该年（ $i > 0$ ）。之后通过代码`df_drug.loc[df_drug.year==y, :]`过滤出一年的数据来绘制折线图，并利用`plt.text()`函数将year信息在右侧轴标记出来，最终得到图19.4。

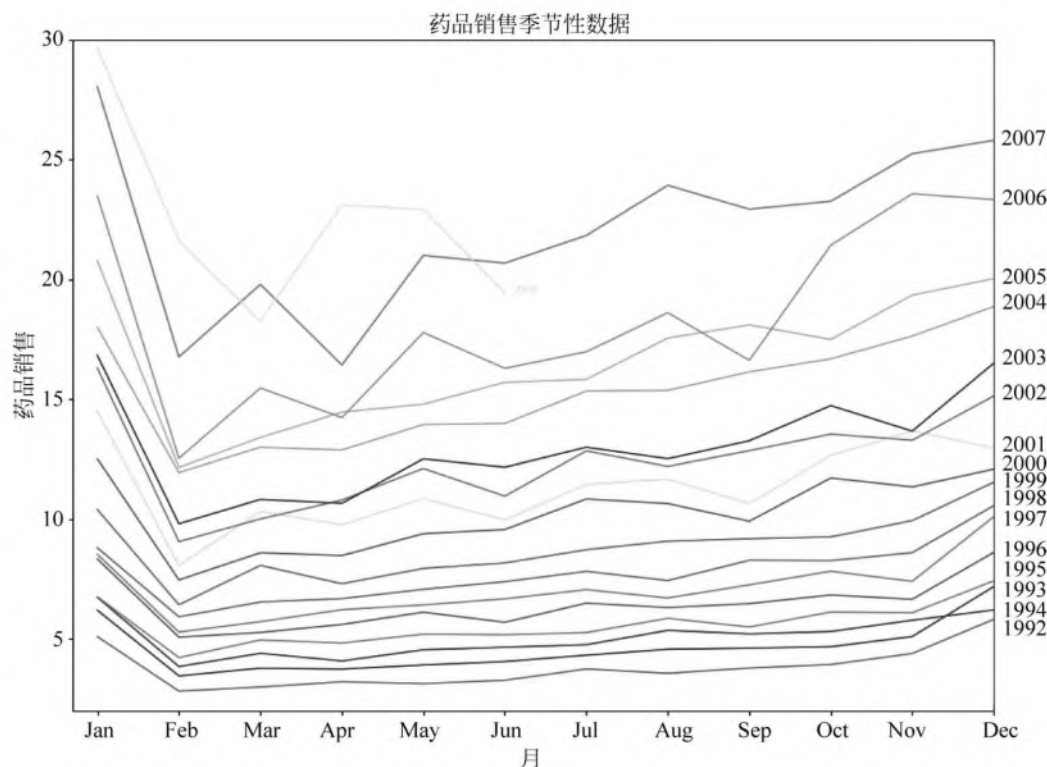


图19.4 药品销售按月对比

从图19.4明显地看到数据存在季节性特征，即1月销售突然增加后立刻回落，而在12月又存在一次异常增加。这很可能是假期因素导致的药品销售的季节性变化。利用如图19.5所示的箱线图，可以很好地将该时间序列数据的季节性与趋势性特征完美展示出来。

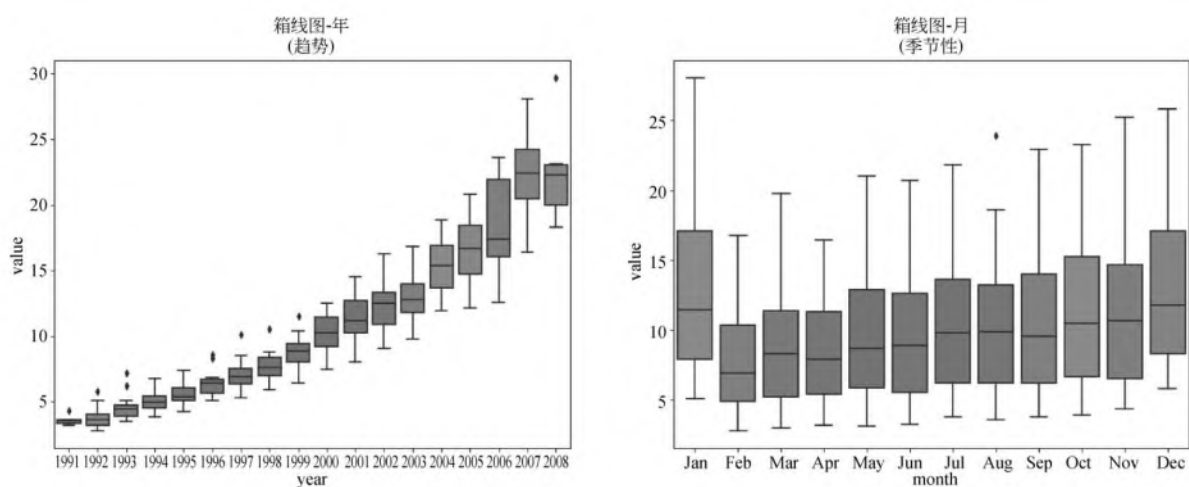


图19.5 药品销售的季节性与趋势性特征

从左侧图形可以看出药品销售逐年上升的趋势非常明显，而从右侧图形可以看出药品销售在1月与12月的季节性特征也非常明显。有兴趣的读者可以用如下代码得到上面的图形。

```
>>> fig, axes = plt.subplots(1, 2, figsize=(20,7), dpi=80)
>>> sns.boxplot(x='year', y='value', data=df_drug, ax=axes[0])
# 去掉了 1991/2018
>>> sns.boxplot(x='month', y='value', \
    data=df_drug.loc[~df_drug['year'].isin([1991, 2008]), :])
>>> axes[0].set_title('箱线图 - 年\n(趋势)', fontsize=18);
>>> axes[1].set_title('箱线图 - 月\n(季节性)', fontsize=18)
>>> plt.show()
```

19.2 时间序列数据的分解

19.2.1 认识时间序列数据中的模式

通过前面的时间序列可视化，读者应该已经观察到了时间序列中的季节性特征和趋势性特征。实际上一个时间序列往往是以下几类变化形式的叠加或耦合：长期趋势（Secular Trend, T）、季节波动（Seasonal Variation, S）、循环波动（Cyclical Variation, C）、不规则波动（Irregular Variation, I）。其中，长期趋势是时间序列在长时期内呈现出来的持续向上或持续向下的变动；季节波动是指时间序列在一年内重复出现的周期性波动，它是诸如气候条件、生产条件、节假日或人们的风俗习惯等各种因素影响的结果；循环波动则是时间序列呈现出的非固定长度的周期性变动，循环波动的周期可能会持续一段时间，但与长期趋势不同，它不是朝着单一方向的持续变动，而是涨落相同的交替波动；不规则波动是指时间序列中除去长期趋势、季节变动和周期波动之后的随机波动，不规则波动通常总是夹杂在时间序列中，致使时间序列产生一种波浪形或震荡式的变

动。不过时间序列并不一定总是同时包含长期趋势与季节性波动，图19.6给出了3个不同的例子，分别是只有长期趋势的时间序列、只有季节性波动的时间序列以及同时具有季节性与趋势特征的时间序列。

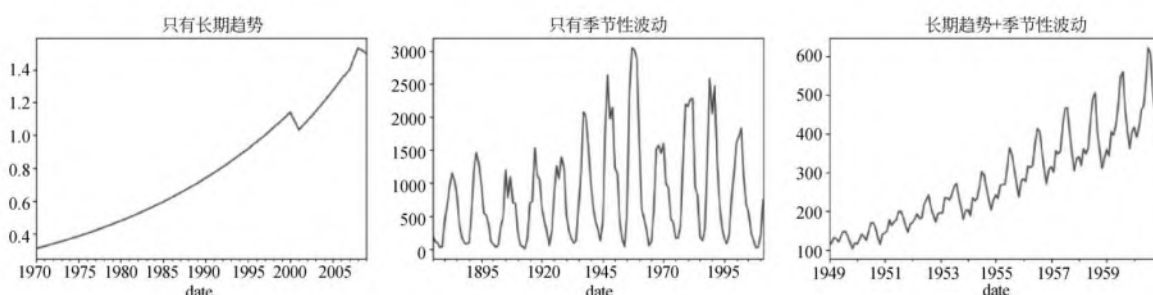


图19.6 时间序列中的模式

如果以数学模型来表示，可以将任意时间序列 y_t 表示为如下的函数：

$$y_t = f(T_t, S_t, C_t, I_t)$$

更具体地，可以用加法模型或乘法模型对其进行分解（另外还有混合模型，本书不作讲解）。

- 加法模型： $y_t = T_t + S_t + C_t + I_t$
- 乘法模型： $y_t = T_t \times S_t \times C_t \times I_t$

19.2.2 Python中进行时间序列数据的分解

Python中的三方包statsmodels提供了经典的时间序列数据分解方式的实现，利用它可以将时间序列数据以加法或乘法模型进行分解，如图19.7和图19.8所示。

模型分解代码如下。


```

>>> from statsmodels.tsa.seasonal import seasonal_decompose
>>> df = pd.read_csv('../data/a10.csv', parse_dates = ['date'], \
    index_col = 'date')
# 乘法
>>> result_mul = seasonal_decompose(df['value'], \
    model = 'multiplicative', extrapolate_trend = 'freq')
# 加法
>>> result_add = seasonal_decompose(df['value'], \
    model = 'additive', extrapolate_trend = 'freq')
# 绘图
>>> plt.rcParams.update({'figure.figsize': (10,10)})
>>> result_mul.plot().suptitle('乘法模型分解', fontsize = 15)
>>> result_add.plot().suptitle('加法模型分解', fontsize = 15)
>>> plt.show()

```

其中，seasonal_decompose（）函数完成了所有的模型分解工作，通过model参数可以选择使用加法模型还是乘法模型，extrapolate_trend参数指明了数据的插值方式，读者可以通过帮助更详细地了解其使用方法。如果读者仔细观察图19.7与图19.8中的residuals图（实际上residuals就是前面提到的I），就会发现加法模型中的residuals还是存在一定的固定模式，而在乘法模型中则相对随机。因此，对于此数据，乘法模型显然更适合。如果想查看具体的分解值，那么可以通过result_mul对象得到，代码如下。

```

# Values = (Seasonal * Trend * Resid)
>>> df_reconstructed = pd.concat([result_mul.seasonal, \
    result_mul.trend, result_mul.resid, \
    result_mul.observed], axis = 1)
>>> df_reconstructed.columns = ['seas', 'trend', \
    'resid', 'actual_values']
>>> df_reconstructed.head()

```

	seas	trend	resid	actual_values
date				
1991-07-01	0.987845	3.060085	1.166629	3.526591
1991-08-01	0.990481	3.124765	1.027745	3.180891
1991-09-01	0.987476	3.189445	1.032615	3.252221
1991-10-01	1.048329	3.254125	1.058513	3.611003
1991-11-01	1.074527	3.318805	0.999923	3.565869

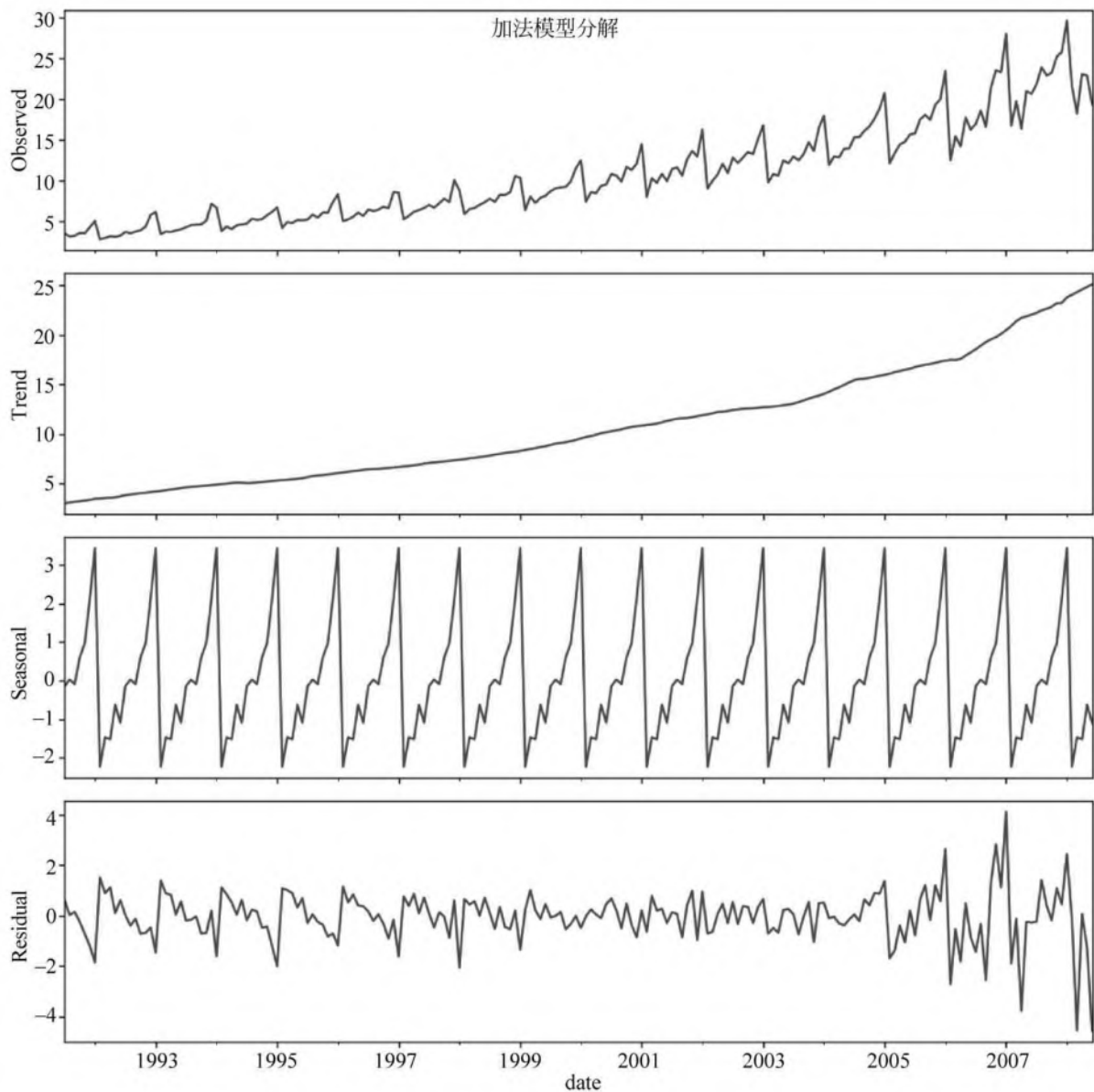


图19.7 加法模型

如果将上面输出中的seas、trend、resid列相乘，就可以得到与actual_values列一样的值。

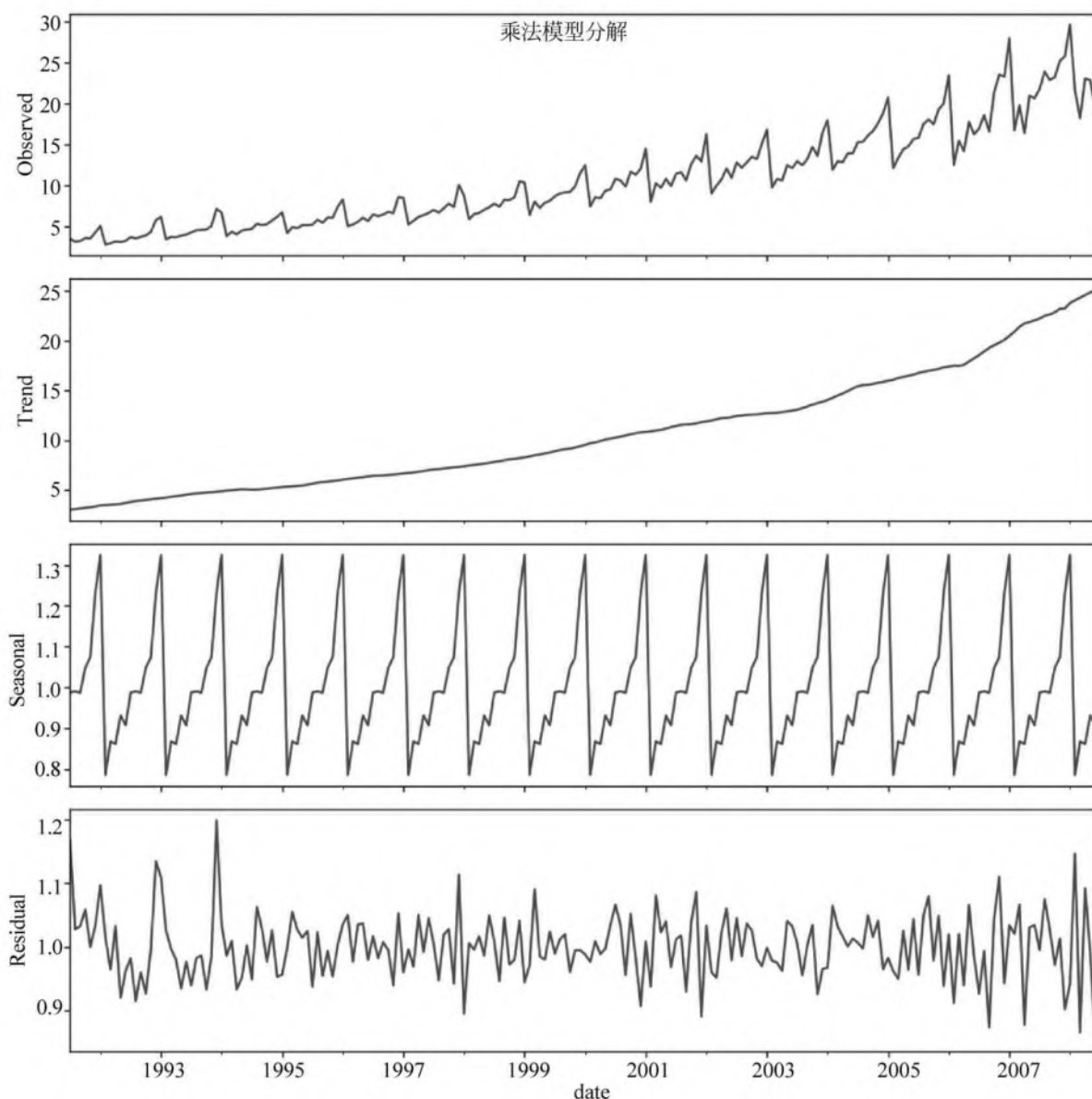


图19.8 乘法模型

19.3 时间序列的平稳性

19.3.1 认识平稳与非平稳时间序列

平稳性 (Stationary) 可以说是时间序列的内部逻辑，也就是说每一期的序列值与前几期之间存在一种一致的结构性的变化关系，只有

这样，我们才能建立模型去分析并预测。其根本原因在于统计学或计量经济学是从数量规律的角度研究问题，如果事物本身的变化毫无规律，这时候还要用统计或计量去分析，那就毫无意义了。图19.9是来自 R 语言的 TSTutorial (Stationary Time Series TSTutorial v1.2.1) 中一些关于平稳与非平稳时间序列的示例。

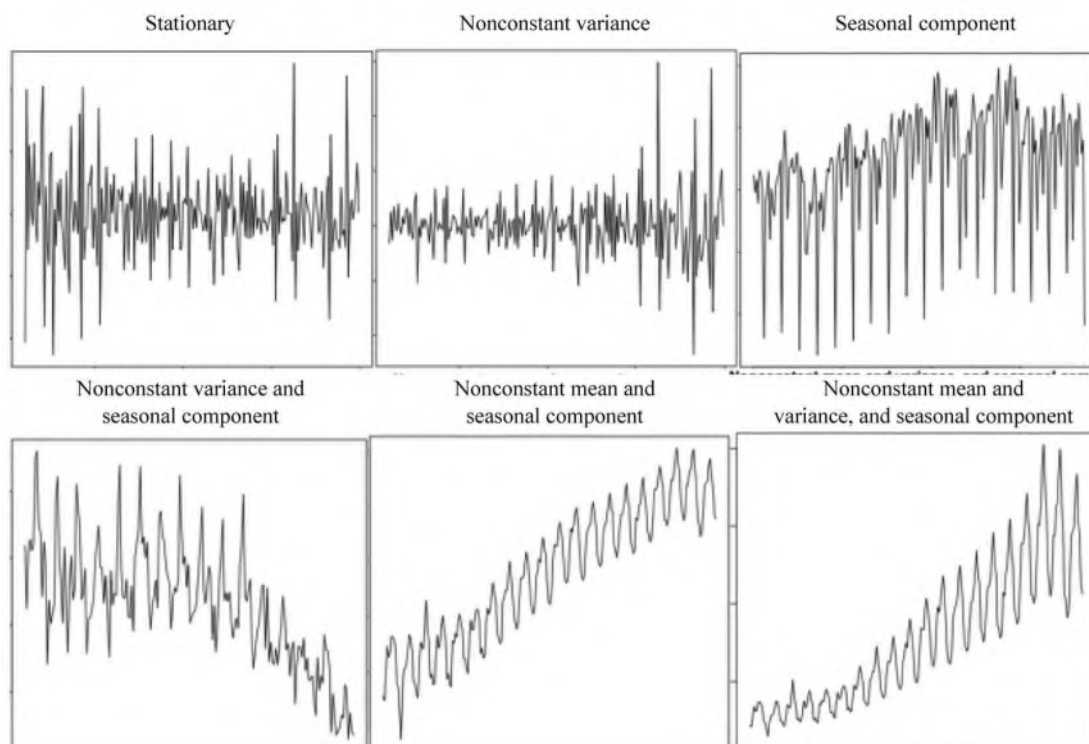


图19.9 平稳与非平稳时间序列

从图19.9中可以大致看出平稳与非平稳时间序列的区别。不过从数学意义上我们需要更严格的定义，所谓平稳就是时间序列的行为并不随时间改变。基于此，可以进一步定义强平稳与弱平稳。对于所有可能的 n ，所有可能的 $t_1, t_2, \dots, t_n$ 和所有可能的 k ，当 $z_{t_1}, z_{t_2}, \dots, z_{t_n}$ 的联合分布与 $z_{t_1-k}, z_{t_2-k}, \dots, z_{t_n-k}$ 相同时，称其为强平稳。但是，由于强平稳的条件经常很难满足，因此通常假定的平稳性是较弱的方式，即弱平稳。弱平稳只需要时间序列 r_t 满足均值为常数，同时 $\text{Cov}(r_t, r_{t-l}) = \gamma_l$ ，其中 γ_l 只与

I 有关。由于本书篇幅有限，这里不对时间序列的平稳性进行进一步讨论，读者只需要明白时间序列分析的基础是要求其平稳性。

19.3.2 如何让时间序列平稳

既然时间序列的平稳性很重要，那么如何让其平稳呢？理论上几乎所有的时间序列通过一系列合适的变换，都可以变为平稳时间序列。典型的方法有：

- 差分，如一阶或更多；
- 进行对数变换去除指数趋势；
- 求 n 次方根；
- 将上述方法组合。

虽然通过可视化的方式，我们可以对时间序列的平稳性进行判断，又或者可以通过计算均值、方差、协方差来判断，不过更多的时候，我们会使用单位根检验（Unit Root Tests）的方式来进行判断。典型的方法有：

- Augmented Dickey Fuller Test (ADF Test) ；
- Kwiatkowski-Phillips-Schmidt-Shin Test (KPSS Test) ；
- Philips Perron Test (PP Test) 。

下面以statsmodels包中提供的ADF检验与KPSS检验为例来进行讲解。

```

>>> from statsmodels.tsa.stattools import adfuller, kpss
>>> df = pd.read_csv('../data/a10.csv', parse_dates = ['date'])
# ADF Test
>>> result = adfuller(df['value'].values, autolag = 'AIC')
>>> print('ADF Statistic:', result[0])
>>> print('p-value:', result[1])
>>> for key, value in result[4].items():
    print('Critical Values:', key, value)
# KPSS Test
>>> result = kpss(df['value'].values, regression = 'c')
>>> print('\nKPSS Statistic: %f' % result[0])
vprint('p-value: %f' % result[1])
>>> for key, value in result[3].items():
    print('Critical Values:', key, value)
ADF Statistic: 3.14518568930673
p-value: 1.0
Critical Values: 1 % -3.465620397124192
Critical Values: 5 % -2.8770397560752436
Critical Values: 10 % -2.5750324547306476

KPSS Statistic: 1.313675
p-value: 0.010000
Critical Values: 10 % 0.347
Critical Values: 5 % 0.463
Critical Values: 2.5 % 0.574
Critical Values: 1 % 0.739

```

对于ADF检验，其空假设是非平稳，因此只有 $p < 0.05$ 时，才判断时间序列为平稳。而KPSS检验则是用于判断趋势的平稳性，其空假设是序列平稳，因此 $p < 0.05$ 代表拒绝空假设。上面的数据显然是非平稳的，可以对其进行一阶或二阶差分后再次检验，代码如下。

```

# 一阶差分
>>> df['value_df1'] = df['value'].diff(1)
>>> result = adfuller(df['value_df1'][1:].values, autolag = 'AIC')
>>> print('ADF Statistic:', result[0])
>>> print('p - value:', result[1])
# 二阶差分
>>> df['value_df2'] = df['value_df1'].diff(1)
>>> result = adfuller(df['value_df2'][2:].values, autolag = 'AIC')
>>> print('ADF Statistic:', result[0])
>>> print('p - value:', result[1])
ADF Statistic: - 2.495172147449673
p - value: 0.11665341686470398
ADF Statistic: - 10.292304706517216
p - value: 3.543916358531434e - 18

```

显然经过二阶差分后，基本上数据已经符合平稳性要求。上述代码中的差分由diff（）函数完成。该函数是用来将数据进行某种移动之后与原数据进行比较得出差异数据，这一过程有两个步骤，首先会执行df.shift（），然后再将该数据与原数据作差，即df-df.shift（）。

19.4 利用ARIMA模型分析家具销售

19.4.1 ARIMA模型简介

时间序列分析中一类常用的模型就是ARIMA模型，其实ARIMA并不是一个特定的模型，而是一类模型的总称。它的3个参数（ p ， d ， q ）分别表示自相关（ p 阶AR模型）， d 次差分，滑动平均（ q 阶MA模型）。在 $p=d=0$ 时，ARIMA模型即MA（ q ）模型；在 $d=q=0$ 时，ARIMA模型即AR（ p ）模型。

具体而言，AR模型代表Auto Regressive（自回归），描述当前值和历史值之间的关系。AR（ p ）中的 p 表示滞后 p 阶的AR模型，即当前时间点与前 p 个时间点的关系。当 $p=1$ 时，表示当前时间点与前一个时间点的关系，通常可以通过偏相关函数（Partial Auto

Correlation Function, PACF) 图, 观察判断 p 参数的选取。I代表差分, 目的是将不平稳的时间序列转化为平稳的时间序列, 或弱平稳时间序列。差分的方法就是求差, 如一阶差分就是时间序列 $r_1, r_2, \dots, r_n$ 减去 $r_2, r_3, \dots, r_{n-1}$ 来获取差值序列, 利用获得的差值序列再进行差分, 就是二阶差分。I(d) 中的 d 表示做多少阶的差分。而MA即Moving Average, 是滑动平均的意思, 它描述了自回归部分的误差累计。其中MA(q) 中的 q 表示前 q 个时间点的时间差, 可以通过自相关函数(Auto Correlation Function, ACF) 图, 观察判断 q 参数的选取。将AR模型与MA模型结合就得到了ARMA模型, 而3个模型结合就得到了ARIMA模型, 即ARIMA(p, d, q) 代表了 p 阶自回归滞后项, q 阶滑动平均滞后项, d 阶差分。

19.4.2 数据准备

首先查看一下待分析的数据, 这是一个家具销售的数据集。利用如下代码完成数据准备, 并可视化数据, 如图19.10所示。

```
>>> df = pd.read_excel("../data/Superstore.xls")
>>> furniture = df.loc[df['Category'] == 'Furniture']
>>> cols = ['Row ID', 'Order ID', 'Ship Date', 'Ship Mode', \
           'Customer ID', 'Customer Name', 'Segment', 'Country', 'City', \
           'State', 'Postal Code', 'Region', 'Product ID', 'Category', \
           'Sub-Category', 'Product Name', 'Quantity', 'Discount', \
           'Profit']
>>> furniture.drop(cols, axis=1, inplace=True)
>>> furniture = furniture.sort_values('Order Date')
>>> furniture = furniture.groupby('Order Date')['Sales']. \
    sum().reset_index()
>>> furniture = furniture.set_index('Order Date')
>>> y = furniture['Sales'].resample('MS').mean()
>>> y.plot(figsize=(15, 6))
>>> plt.show()
```

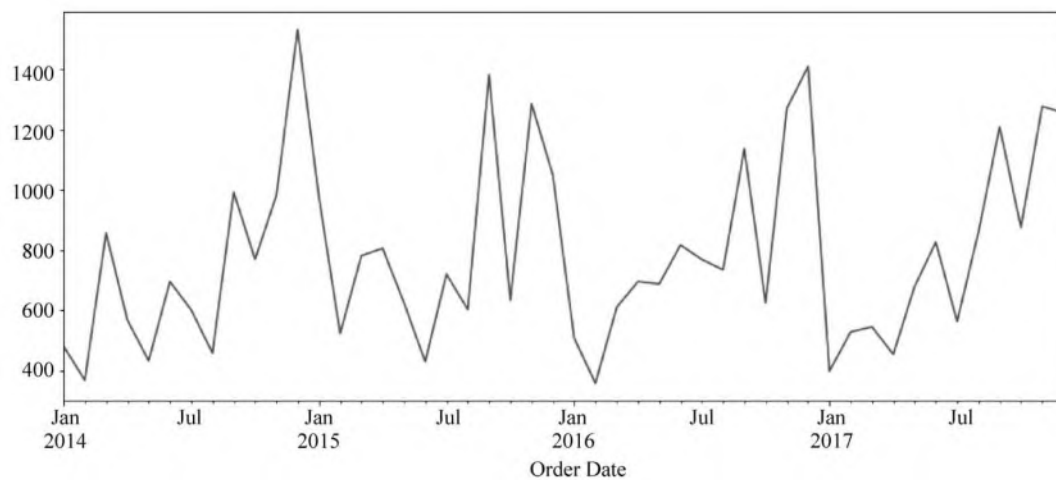



图19.10 家具销售数据可视化

从图19.10可以看出，该数据存在明显的季节性特征。将该时间序列数据分解，得到图19.11，代码如下。

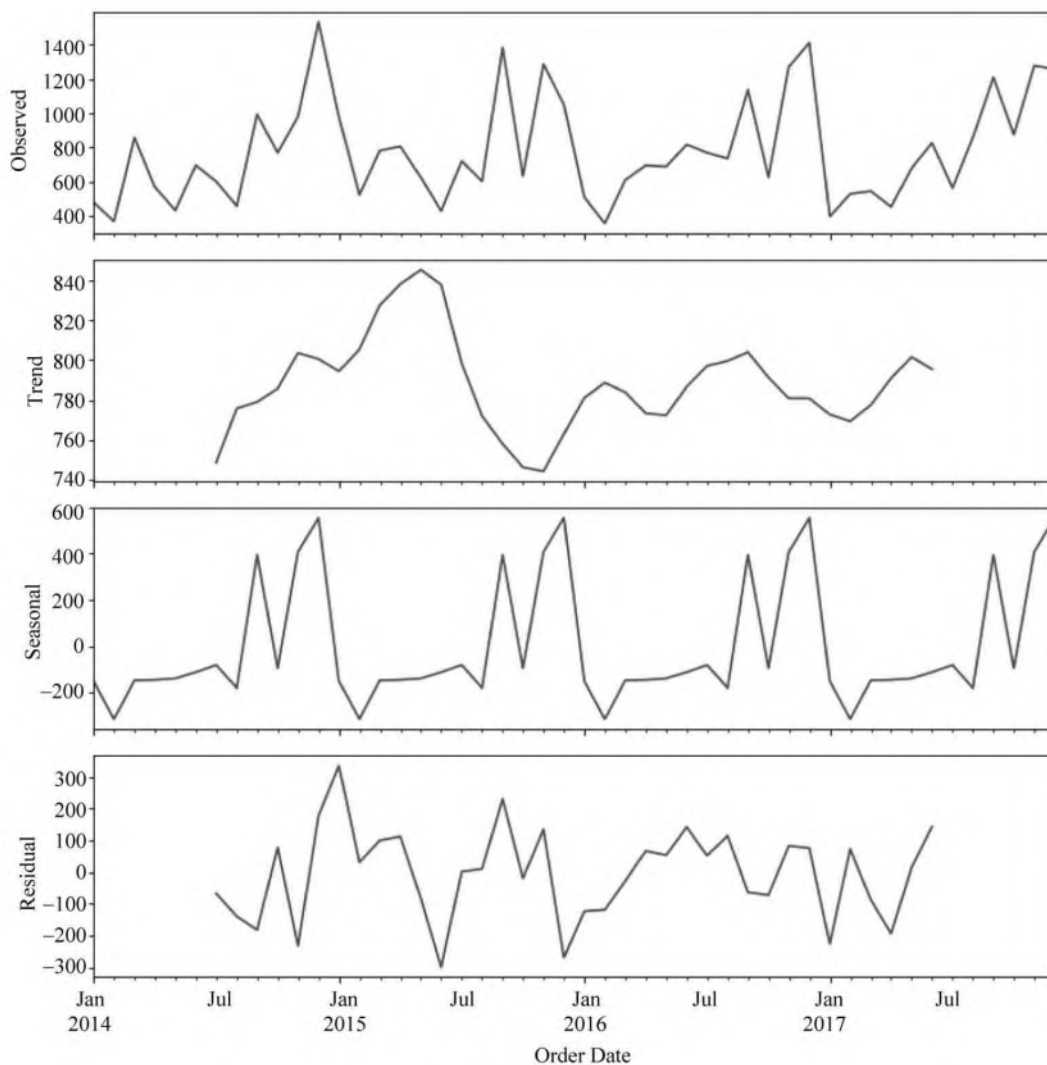


图19.11 数据分解

```
>>> decomposition = sm.tsa.seasonal_decompose(y, model = 'additive')
>>> plt.rcParams.update({'figure.figsize': (10,10)})
>>> fig = decomposition.plot()
>>> plt.show()
```

19.4.3 ARIMA模型中的参数

建立ARIMA模型一般有3个阶段，分别是模型识别和定阶、参数估计和模型检验。ARIMA模型的识别问题和定阶问题，主要是确定 p ， d ， q 这3个参数。要确定参数 p 和 q ，就需要用到自相关函数（ACF）

与偏自相关函数（PACF）。ACF描述的是时间序列观测值与其过去的观测值之间的线性相关性，计算公式如下。

$$ACF(k) = \rho_k = \frac{\text{Cov}(y_t, y_{t-k})}{\text{Var}(y_t)}$$

其中， k 代表滞后期数。而PACF描述的是在给定中间观测值的条件下，时间序列观测值预期与过去的观测值之间的线性相关性。举例说明，假设 $k=3$ ，那么我们描述的是 y_t 和 y_{t-3} 之间的相关性，但是这个相关性还受到 y_{t-1} 和 y_{t-2} 的影响。PACF剔除了这个影响，而ACF包含这个影响。了解了ACF与PACF后，就需要利用`plot_acf()` 和 `plot_pacf()` 中的拖尾与截尾来选择 p 和 q 。拖尾与截尾的定义如下：

- AR(p) 模型的偏自相关函数PACF在 p 阶之后应为零，称其具有截尾性；
- AR(p) 模型的自相关函数ACF不能在某一步之后为零（截尾），而是按指数衰减（或呈正弦波形式），称其具有拖尾性；
- MA(q) 模型的自相关函数ACF在 q 阶之后应为零，称其具有截尾性；
- MA(q) 模型的偏自相关函数PACF不能在某一步之后为零（截尾），而是按指数衰减（或呈正弦波形式），称其具有拖尾性。

在实际判断中，如果样本自相关系数（或偏自相关系数）在最初的 d 阶明显大于2倍标准差范围，而后几乎95%的样本自相关（偏自相关）系数都落在2倍标准差范围以内，而且由非零自相关（偏自相关）系数衰减为小值波动的过程非常突然，这时通常视为自相关（偏自相关）系数截尾。如果有超过5%的样本相关系数落在2倍标准差范围以外，或者是由显著非零的相关函数衰减为小值波动的过程比较缓慢或非常连续，这时通常视为相关系数不截尾。对本案例的数据利用如下代码对ACF和PACF进行可视化，得到图19.12。

```

>>> from statsmodels.tsa.stattools import acf, pacf
>>> from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
>>> fig = plt.figure(figsize = (12,8))
>>> ax1 = fig.add_subplot(211)
>>> fig = plot_acf(y.diff()[1:], lags = 20, ax = ax1)
>>> ax1.xaxis.set_ticks_position('bottom')
>>> fig.tight_layout()
>>> ax2 = fig.add_subplot(212)
>>> fig = plot_pacf(y.diff()[1:], lags = 20, ax = ax2)
>>> ax2.xaxis.set_ticks_position('bottom')
>>> fig.tight_layout()
>>> plt.show()

```

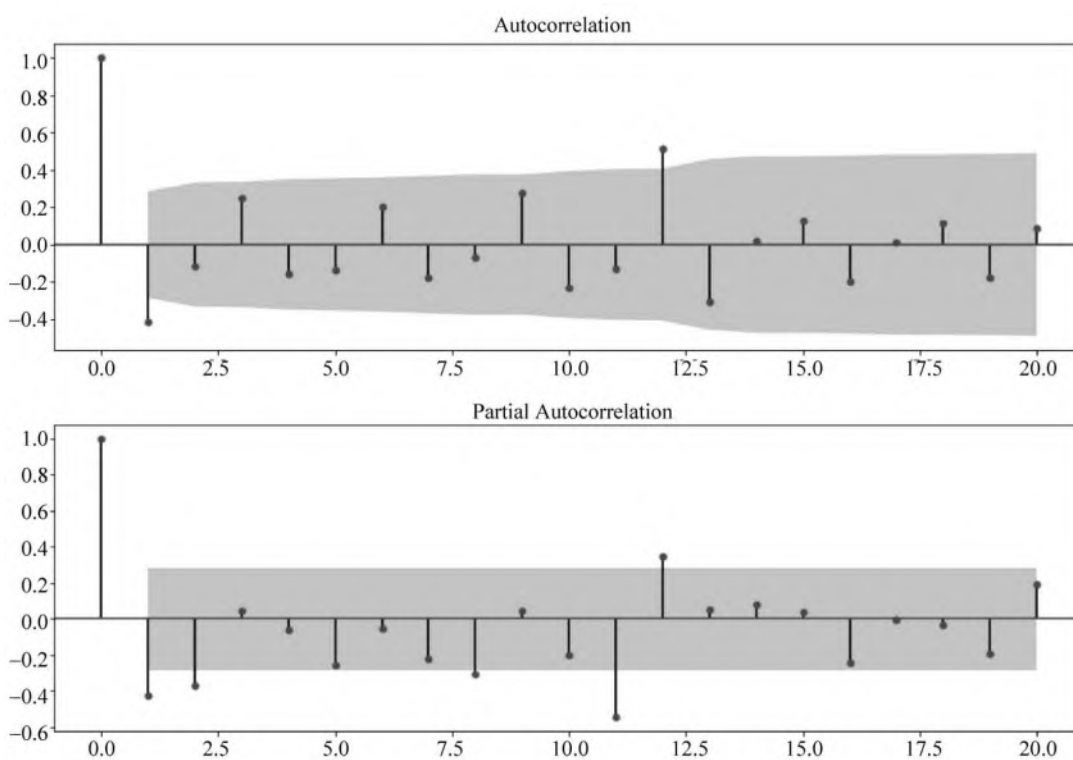


图19.12 ACF和PACF图

从图19.12大致可以判断 $p = 1$ ， $q = 1$ 可能是一个不错的选择。不过通过观察法不一定具有严谨性，这时考虑采用AIC（Akaike Information Criterion）、BIC（Bayesian Information Criterion）等指标作为判断标准。这里以AIC为例，首先构建各种 p ， q ， d 的组合，之后通过最小AIC来得到最佳组合，代码如下。

```

>>> import itertools
>>> p = d = q = range(0, 2)
>>> pdq = list(itertools.product(p, d, q))
>>> seasonal_pdq = [(x[0], x[1], x[2], 12) for x in \
    list(itertools.product(p, d, q))]
>>> best_param = ()
>>> min_aic = 0
>>> for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(y, \
                order = param, seasonal_order = param_seasonal, \
                enforce_stationarity = False, \
                enforce_invertibility = False)
            results = mod.fit()
            if min_aic == 0:
                min_aic = results.aic
            elif results.aic < min_aic :
                best_param = (param, param_seasonal)
                min_aic = results.aic
        except:
            continue
>>> print("最优参数 p,d,q: ", best_param, "最佳 aic:", min_aic)
最优参数 p,d,q: ((1, 1, 1), (1, 1, 0, 12)) 最佳 aic: 297.7875439553055

```

从输出结果看，基本和前面的判断一致， $p = q = 1$ ，差分 $d = 1$ 。接下来需要查看模型参数，同时画出模型诊断图，如图19.13和图19.14所示。

```

>>> mod = sm.tsa.statespace.SARIMAX(y, order = (1, 1, 1),
    seasonal_order = ( 1, 1, 0, 12),
    enforce_stationarity = False,
    enforce_invertibility = False )
>>> results = mod.fit()
>>> print(results.summary().tables[1])
>>> results.plot_diagnostics(figsize = (16, 8))
>>> plt.show()

```

	coef	std err	z	P> z	[0.025	0.975]
ar.L1	0.0146	0.342	0.043	0.966	-0.655	0.684
ma.L1	-1.0000	0.360	-2.781	0.005	-1.705	-0.295
ar.S.L12	-0.0253	0.042	-0.609	0.543	-0.107	0.056
sigma2	2.958e+04	1.22e-05	2.43e+09	0.000	2.96e+04	2.96e+04

图19.13 查看模型参数

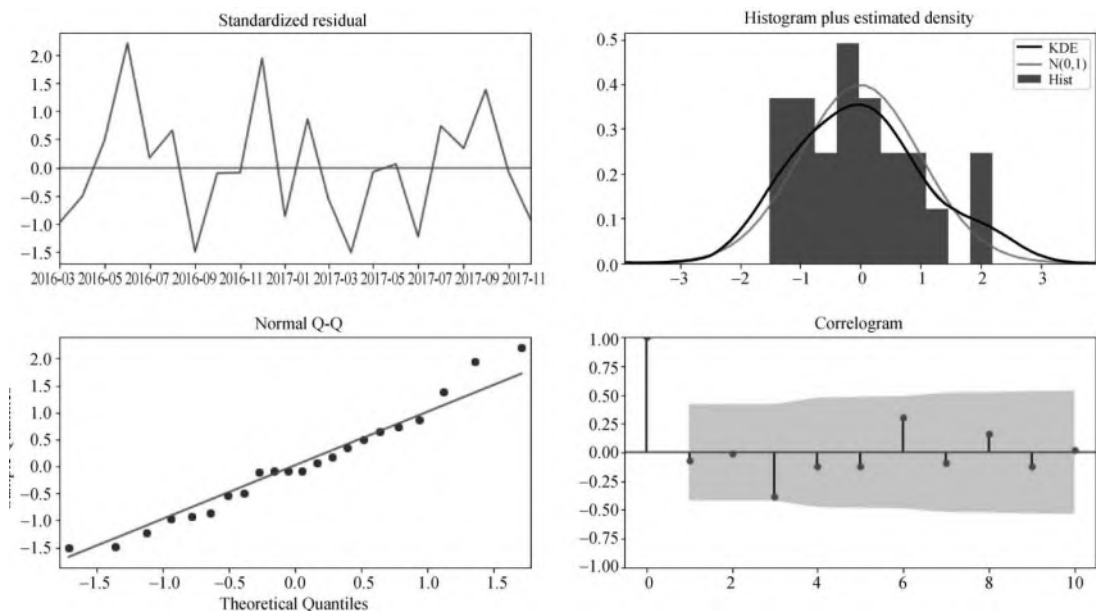


图19.14 ARIMA模型诊断

从输出结果来看，模型虽然不够完美，但是残差（Residuals）基本还是符合正态分布的。假设选用此模型，接下来就可以利用它来进行预测了。例如，如下代码完成了2017年1月1日后的销售数据的预测，其可视化结果如图19.15所示。

```

>>> pred = results.get_prediction(start = \
    pd.to_datetime('2017-01-01'), dynamic = False)
>>> pred_ci = pred.conf_int()
>>> ax = y['2014:'].plot(label = '实际值')
>>> pred.predicted_mean.plot(ax = ax, label = '预测值', alpha = .7, \
    figsize = (14, 7))
>>> ax.fill_between(pred_ci.index, pred_ci.iloc[:, 0], \
    pred_ci.iloc[:, 1], color = 'k', alpha = .2)
>>> ax.set_xlabel('日期')
>>> ax.set_ylabel('家具销售')
>>> plt.legend()
>>> plt.show()

```

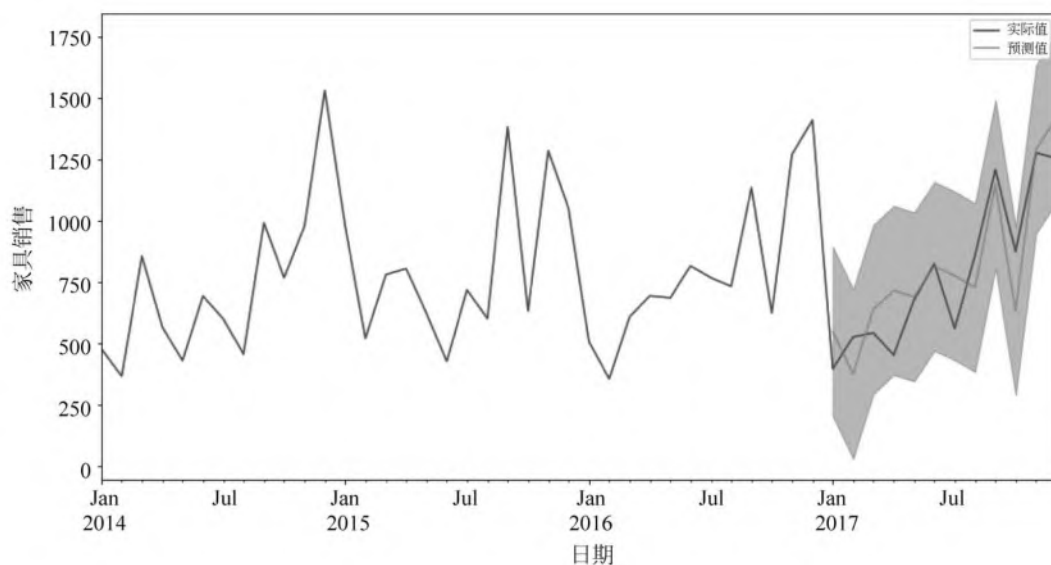


图19.15 销售预测

从预测结果来看，我们对后一时间周期数据的预测结果还是不错的。

19.5 留给读者的思考

时间序列分析是一门专门的学科，本章只是给出了几个简单的示例。这里向读者推荐两本书进行更系统的学习。第一本是Ruey S. Tsay的《金融时间序列分析》，这是一本关于金融时间序列分析的经

典书籍。第二本是Springer出版社的 *Time Series Analysis With Applications in R*，书中提供了大量基于R语言的示例。

第20章 股票数据分析

不要懵懵懂懂地随意买股票，要在投资前扎实地做一些功课，才能成功。

——威廉·欧奈尔

股票数据是时间序列数据中一类特殊的数据，它记录了股票每天或每个特定时间区间的开盘价、收盘价、最高价、最低价、成交量等数据。人们总是期望通过分析股票时间序列而获利。我们学习的数据分析技术也可以用于股票分析中，本章将为大家介绍Python数据分析在股票投资中的应用。

20.1 股票收益分析

20.1.1 获取股票数据

要分析股票收益，首先需要获取股票数据。这里使用开源的Tushare库来获取股票数据（Tushare Pro稳定性更好，不过因为需要注册，本书不使用它作为示例）。例如，使用如下代码可以获取招商银行2018年1月1日至2019年6月30日的股票数据。

```

>>> import pandas as pd
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import matplotlib as mpl
>>> import seaborn as sns
>>> import datetime
>>> from datetime import datetime, date
>>> import tushare as ts
>>> pd.set_option('display.max_columns', 10)
>>> pd.set_option('display.max_rows', 10)
>>> %matplotlib inline
# 招商银行
>>> zsyh = ts.get_hist_data('600036', start = '2018-01-01', \
    end = '2019-06-30')
>>> zsyh.sort_index(inplace = True)

```

其中，`ts.get_hist_data()` 函数用来获取指定股票的数据，而 `zsyh.sort_index(inplace=True)` 则将数据按照时间顺序进行排列。获取了股票数据后，可以对其收盘价可视化，如图20.1所示。

```

>>> plt.figure(figsize = (8,6))
>>> zsyh['close'].plot()

```

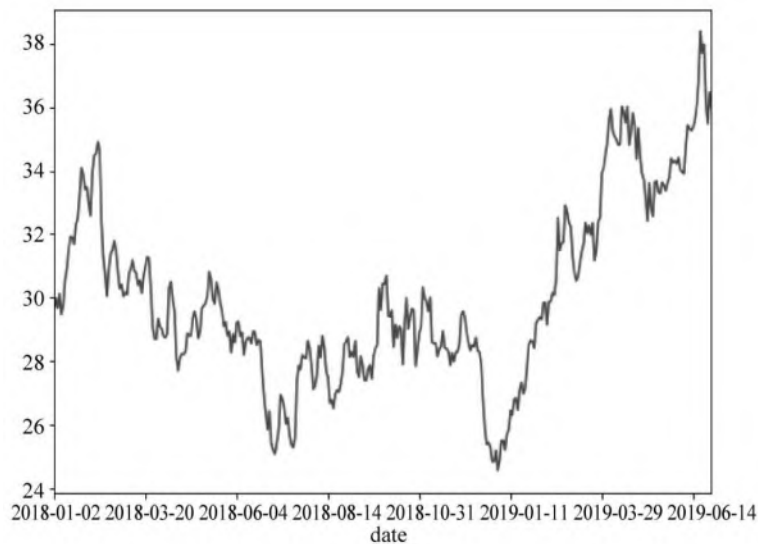


图20.1 招商银行收盘价可视化

20.1.2 计算每日收益

相对于股价，投资者更关心的是股票的每日收益率，因此我们需要用当日收盘价减去前一日收盘价来进行计算，代码如下。

```
>>> zsyh_pc = round((zsyh['close']/ zsyh['close'].shift(1) - 1),4)
>>> zsyh_pc.head()
```

上述代码中 shift (1) 将数据后移，因此 zsyh['close']/zsyh['close'].shift(1) 得到的就是当日收盘价与昨日收盘价的比值，减去1后就得到了每天的涨跌幅。不过Pandas中其实已经提供了收益计算函数pct_change()，利用如下代码可以得到与前面一样的结果。

```
>>> zsyh['pct'] = zsyh['close'].pct_change()
```

相对于每日收益，长期投资者可能更关心累积收益率，此时可以用如下代码进行计算。

```
>>> cum_daily_return = (1 + zsyh['pct']).cumprod()
>>> cum_daily_return.head()
date
2018-01-02    1.000000
2018-01-03    1.011812
2018-01-04    1.001350
2018-01-05    1.016537
2018-01-08    0.994938
Name: pct, dtype: float64
```

cumprod() 函数完成了累积的工作，最后将上面的数据可视化，得到图20.2。

```
>>> cum_daily_return.plot(figsize=(8,6))
```

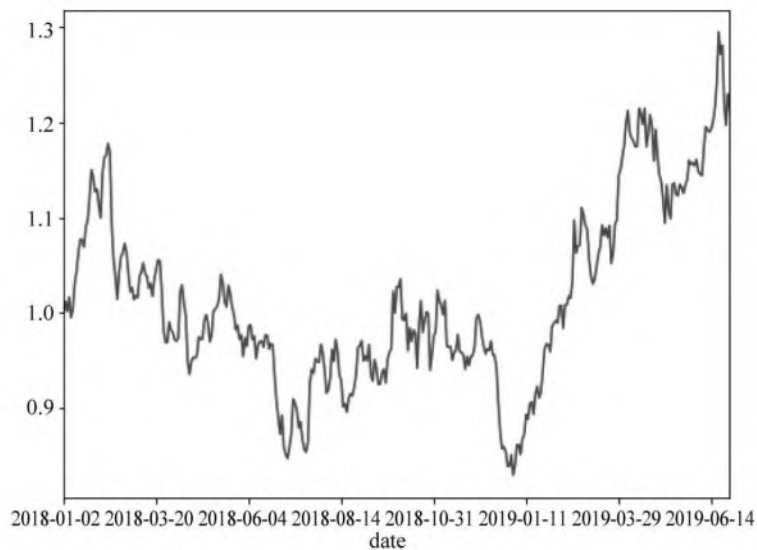


图20.2 招商银行累积收益图

从图20.2可以看出，这段时间如果长期持有招商银行还是有不错收益的，达到了20%左右。

20.1.3 多只股票收益比较

采用类似的方法，可以对多只股票的收益进行比较，首先来获取股票数据，代码如下。

```

# 招商银行
>>> zsyh = ts.get_hist_data('600036', start = '2018-01-01', \
                             end = '2019-06-30')
# 贵州茅台
>>> gzmt = ts.get_hist_data('600519', start = '2018-01-01', \
                             end = '2019-06-30')
# 上海汽车
>>> shqc = ts.get_hist_data('600104', start = '2018-01-01', \
                             end = '2019-06-30')
# 中国平安
>>> zgpa = ts.get_hist_data('601318', start = '2018-01-01', \
                             end = '2019-06-30')
# 中信证券
>>> zxzq = ts.get_hist_data('600030', start = '2018-01-01', \
                             end = '2019-06-30')
# 保利地产
>>> bldc = ts.get_hist_data('600048', start = '2018-01-01', \
                             end = '2019-06-30')

```

接下来将以上股票的收盘价合并到一个新的DataFrame中，如图20.3所示。

```

>>> zsyh['id'] = "招商银行"
>>> gzmt['id'] = "贵州茅台"
>>> zgpa['id'] = "中国平安"
>>> bldc['id'] = "保利地产"
>>> zxzq['id'] = "中信证券"
>>> shqc['id'] = "上海汽车"
>>> stocks = stocks.append([gzmt[['id', 'close']], zxzq[['id', 'close']], \
                             shqc[['id', 'close']], bldc[['id', 'close']], \
                             zgpa[['id', 'close']]], sort = True)
>>> stocks = stocks.reset_index()
>>> stocks_c = stocks.pivot("date", "id", "close")
>>> stocks_c.head()

```

	id	上海汽车	中信证券	中国平安	保利地产	招商银行	贵州茅台
date							
	2018-01-02	31.77	18.45	72.62	14.61	29.63	704.00
	2018-01-03	31.55	18.63	71.00	14.50	29.98	715.70
	2018-01-04	31.60	18.68	71.19	14.50	29.67	737.02
	2018-01-05	31.93	18.88	70.86	15.29	30.12	738.50
	2018-01-08	32.41	19.57	69.98	15.60	29.48	752.02

图20.3 多只股票数据

最后，将多只股票收益率可视化，如图20.4所示，可以看出贵州茅台、招商银行、中国平安、中信证券在这一年多里有不错的收益，而保利地产和上海汽车则是负收益。

```
>>> stocks_pct = stocks_c.pct_change()  
>>> stocks_pct.fillna(0,inplace=True)  
>>> cum_daily_return = (1 + stocks_pct).cumprod()  
>>> cum_daily_return.plot(figsize=(8,6))
```

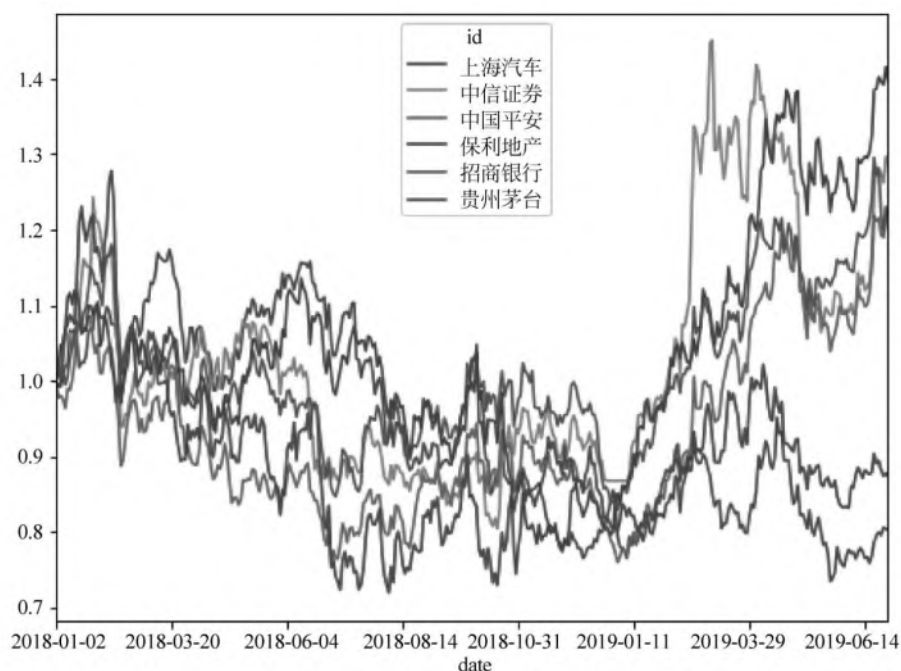


图20.4 多只股票收益率对比

尽管股票累积收益各有不同，如果观察图20.5所示的每日收益，就会发现各只股票的每日收益基本都符合均值为0的正态分布。某种意义上，这也说明了股价每日涨跌大多数情况下其实是随机的。

```
>>> stocks_pct.hist(bins=20, sharex=True, figsize=(12,8));
```

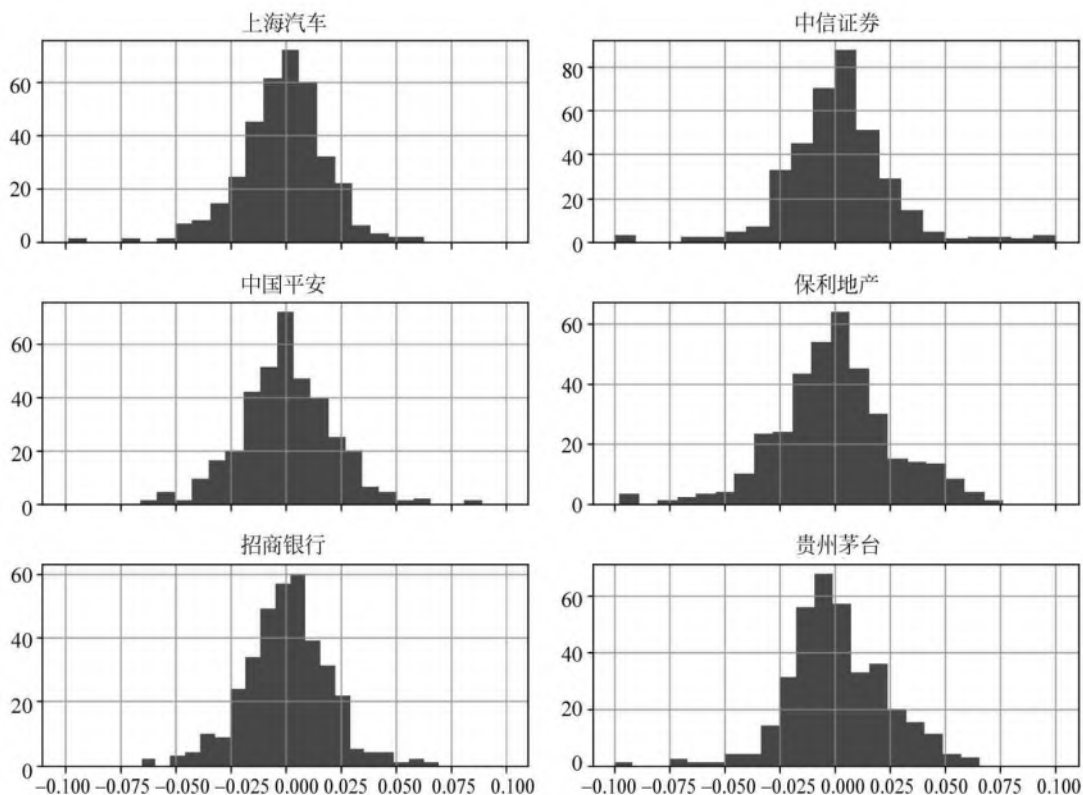


图20.5 每日收益直方图

20.1.4 股价相关性分析

从事投资的人都知道鸡蛋不要放在一个篮子里面，那么如何才能知道鸡蛋不在一个篮子里呢？很简单，我们可以通过相关性分析来获得答案，利用`corr()`函数可以计算相关性，代码如下。

```
>>> corrs = stocks_pct.corr()
```

之后利用Seaborn提供的`heatmap()`函数，可以将结果可视化，如图20.6所示。

从图20.6可以看出，由于中国平安与招商银行都属于金融行业，二者具有极强的相关性；而保利地产与上海汽车基本没有什么关系。如果个人从事股票投资，那么在选择好了股票后，最好进行一下相关

性分析，看看持仓个股是否都是强相关的，如果是，那么很可能你的投资就是在一个“篮子”里面。

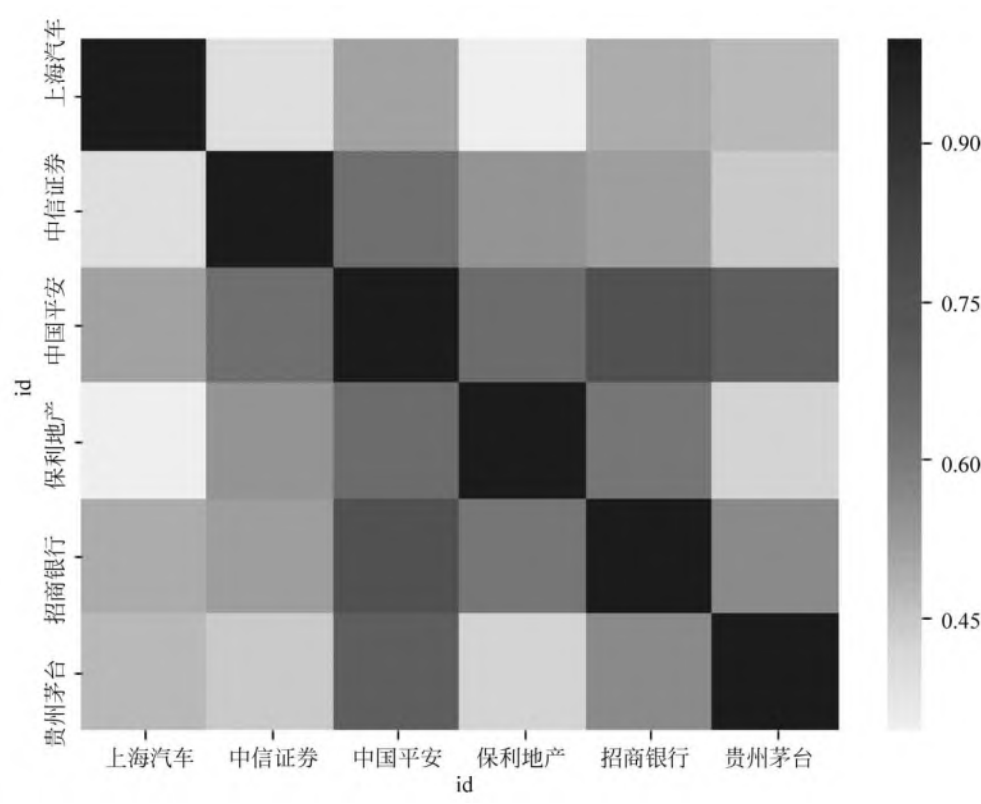


图20.6 收益相关性

20.2 CAPM资产定价模型选股

20.2.1 CAPM公式

资本资产定价模型（Capital Asset Pricing Model, CAPM）是由美国学者 William Sharpe、John Lintner、Jack Treynor 和 Jan Mossin 等人于1964年在资产组合理论和资本市场理论的基础上发展起来的，主要研究证券市场中资产的预期收益率与风险资产之间的关系，以及均衡价格是如何形成的，是现代金融市场价格理论的支柱，

广泛应用于投资决策和公司理财领域。下面让我们来看看CAPM在个人投资中的应用。CAPM的公式定义如下：

$$R_i - R_f = \alpha + \beta (R_m - R_f) + \varepsilon$$

其中， R_i 代表个股； R_f 代表无风险资产，通常使用银行存款或国债年化收益率代替； R_m 代表市场指数收益率，即大盘收益率。在CAPM模型中，假设 α 服从正态分布，并且所有资产的 α 应该为0或接近于0，如果有显著偏差，说明个股有异常收益。而 $\beta = 1$ 代表个股收益与大盘的收益波动一致， $\beta < 1$ 则说明个股波动程度小于大盘， $\beta > 1$ 则说明个股波动程度大于大盘。如果某只股票的 $\alpha > 1$ ，而 $\beta < 1$ ，显然这种股票就是投资者想要的股票。

20.2.2 在Python中实现CAPM

首先选定市场指数为上证50，获取其历史数据并计算收益率，代码如下。

```
>>> sz50 = ts.get_hist_data('sz50', start = '2018-01-01', end = '2019-06-30')
>>> sz50.sort_index(inplace = True)
>>> sz50['pct'] = sz50['close'].pct_change()
>>> sz50['pct'].fillna(0, inplace = True)
```

接下来以3年期国债年化收益率作为无风险利率，计算 R_f ，代码如下。

```
# 目前3年期国债年化收益率大约为4%
>>> Rf_year = 0.04
# 年利率转化为日利率
>>> Rf = (1 + Rf_year) ** (1/365) - 1
```

之后对比个股以及50指数的收益，代码如下。

```

>>> ERet = stocks_pct - Rf
>>> ERet_50 = sz50['pct'] - Rf
>>> import statsmodels.api as sm
>>> for stock in ['中信证券', '招商银行', '中国平安', '保利地产']:
    model = sm.OLS(ERet[stock], sm.add_constant(ERet_50))
    result = model.fit()
    print(stock + '\n')
    print(result.summary())
    print('\n')

```

实际上CAPM公式就是一个线性回归问题，因此上述代码直接使用了最小二乘法来求解，利用statsmodels中的OLS（）函数来完成。从输出结果来看，保利地产的截距为负，说明 $\alpha < 0$ ，也就是其收益弱于上证50，同时其 $\beta = 1.3468$ ，说明波动大于指数。很显然投资者应该避免投资保利地产。剩下的3只个股的 $\alpha > 0$ ，说明收益强于指数，同时招商银行的波动又最小，投资者可能会考虑买入招商银行股票。

20.3 留给读者的思考

前面在分析多只股票收益时，书中的代码存在大量重复，读者可以考虑写一个函数来完成这一功能。另外，对于股票数据分析，除了本章提到的方法，读者还可以尝试自己构建移动平均线来验证交易策略。读者也可以试试流行的深度学习方法，用长短期记忆网络（Long Short Term Memory, LSTM）来预测股价，或者对股票进行聚类分析，用数据分析来寻找价值低估股、财报造假股等。最后衷心希望读者能学以致用！

第21章 大规模数据处理

海纳百川，有容乃大；

壁立千仞，无欲则刚。

——林则徐

人类是数据的创造者和使用者，自结绳记事起，数据就已慢慢产生。随着计算机和互联网的广泛应用，人类产生、创造的数据量呈爆炸式增长。中国已成为全球数据总量最大，数据类型最丰富的国家之一。仿佛一夜之间，所有人都开始讨论大数据了，企业都在讨论Hadoop、Hive、MapReduce等。那么面对大规模数据时，我们使用Pandas能处理吗？本章将讨论利用Pandas处理大规模数据的一些方法。

21.1 不同规模数据处理工具的选择

不知道为什么，个人感觉我们做事总是一窝蜂而上。突然之间，所有人都在讨论大数据；之后大家又言必称机器学习、人工智能、深度学习；再后来，各个企业都说自己在做区块链。对于Pandas也一样，许多人还没有将其研究透，就觉得Pandas无法处理大规模数据，应该学习一个新的工具了。很多企业在没有相应技术储备，同时自己数据量并不大时，盲目追求大，反而是一种灾难。以下是作者在不同规模数据处理上的一些经验，供读者在选择数据处理工具时参考。

- 如果数据大小只有几十兆字节（MB），也许用Excel、Pandas就是不错的选择。

- 如果数据达到了几十兆字节（MB）到几吉字节（GB），此时Pandas或数据库都是不错的选择，Pandas相比数据库有时还更具灵活性与高效性。
- 如果数据规模从几吉字节（GB）到几十吉字节（GB），此时Pandas和数据库都能胜任。如果选择采用Pandas处理这些数据，可以提前考虑进行数据类型转换，或者使用下一节提到的方法。作者曾经用自己的一台16GB内存的笔记本电脑，对一个20GB的数据文件进行分析，处理起来毫无压力。
- 如果数据已经超过了几百吉字节（GB），进入TB级，此时开始考虑采用大数据系统无疑是不错的选择。

为了给大家一个更直观的判断，我们再举一个例子。假设有一数据集，一共有20列，每列需要40B来存储（极端假设所有UTF-8编码的中文字符都需要4B），那么存储一行数据需要800B，存储100万行数据需要760MB左右，存储1000万行不到8GB。读者据此可以判断，有多少情况下需要处理的数据是超过了这个规模的。

21.2 利用Pandas处理大规模数据

21.2.1 文件分块读入

由于Pandas是首先将数据文件读入到内存中再进行分析，于是很多人就认为内存大小决定了能处理的数据的大小。其实我们可以对此进行变通，如将文件分块读入进行处理。

```

>>> csv_file = '../data/gapminder - FiveYearData.csv'
>>> chunksize = 500
>>> for gm_chunk in pd.read_csv(csv_file, chunksize = chunksize):
    print(gm_chunk.shape)
(500, 6)
(500, 6)
(500, 6)
(204, 6)

```

上述代码就将文件分块读入到gm_chunk中，此时我们可以对每个块单独进行数据过滤，之后再合并进行处理。

21.2.2 使用数据库

对于数据文件超过内存，除了使用Pandas分块读入分析的方法，还可以考虑与数据库结合来应对文件大小超过系统内存的情况。如下代码先建立了一个数据库csv_database.db，这里为了简单，使用的是SQLite，对于MySQL、PostgreSQL，方法也是一样。之后利用pd.read_csv()函数提供的分块读取文件的功能，每次读取一部分数据。最后我们将其以append的方式添加到了数据库的service表中。

```

>>> import pandas as pd
>>> from sqlalchemy import create_engine
>>> csv_database = create_engine(r'sqlite:///d:/data - ana - book/data/csv_database.db')
>>> file = '../data/311 - service - requests.csv'
>>> chunksize = 100000
>>> i = 0
>>> j = 1
>>> for df in pd.read_csv(file, chunksize = chunksize, \
    iterator = True, low_memory = False):
    df = df.rename(columns = {c: c.replace(' ', '') for c in df.columns})
    df.index += j
    i += 1
    df.to_sql('service', csv_database, if_exists = 'append')
    j = df.index[-1] + 1

```

将数据放到数据库中后，就可以利用pd.read_sql_query()函数来执行各种SQL语句，选择特定数据来进行分析。例如，如下代码从数

据库中将Descriptor="Loud Talking"的数据筛选了出来。

```
>>> df = pd.read_sql_query('SELECT Descriptor,\n    ComplaintType FROM service where \n    Descriptor = "Loud Talking" ', csv_database)
```

	Descriptor	ComplaintType
0	Loud Talking	Noise - Street/Sidewalk
1	Loud Talking	Noise - Commercial
2	Loud Talking	Noise - Street/Sidewalk
3	Loud Talking	Noise - Street/Sidewalk
4	Loud Talking	Noise - Park

21.2.3 使用DASK

DASK是一款用于分析计算的灵活并行计算库，在进行大规模的数据分析时，本机内存往往不够，同时又不想使用Spark等大数据工具的话，DASK是一个不错的替代选择。通常DASK有两种应用场景，一种是利用dask.array、dask.dataframe等来分析大型数据集，这与数据库、Spark等类似；另一种应用场景就是自定义任务计划，这与Luigi、Airflow、Celery和Makefiles类似。

本书将介绍第一种应用场景，由于DASK的API使用和Pandas很相似，对于从Pandas数据分析过渡来的用户使用起来非常方便。下面同样以刚才的数据为例，演示一下如何使用DASK来处理大规模数据。首先导入DASK库，使用read_csv（）函数读入文件。

```
>>> import dask.dataframe as dd\n>>> df = dd.read_csv('../data/311-service-requests.csv', dtype='str')
```

与Pandas不同的是，此时数据文件并不会读入到内存，同样地，我们可以用head（）函数查看数据的前几行，如图21.1所示。

```
>>> df.head(2)
```

	Unique Key	Created Date	Closed Date	Agency	Agency Name	...	Ferry Direction	Ferry Terminal Name	Latitude	Longitude	Location
0	26589651	10/31/2013 02:08:41 AM	NaN	NYPD	New York City Police Department	...	NaN	NaN	40.70827532593202	-73.79160395779721	(40.70827532593202, -73.79160395779721)
1	26593698	10/31/2013 02:01:04 AM	NaN	NYPD	New York City Police Department	...	NaN	NaN	40.721040535628305	-73.90945306791765	(40.721040535628305, -73.90945306791765)

2 rows x 52 columns

图21.1 查看前两行数据

还可以使用Pandas中类似的方式过滤数据并统计，代码如下。

```
>>> radiator_df = df[df['Descriptor'] == 'RADIATOR']
>>> radiator_df['Descriptor'].count()
dd.Scalar < series - ..., dtype = int32 >
```

不过此时你会发现DASK采用了与MapReduce/Spark一样的方法，只有最后计算时才会输出结果，前面的count（）函数只会在系统标记，因此需要调用compute（）函数才会触发最后的计算。

```
>>> radiator_df.compute()
```

只有触发了计算才会得到最终的结果。读者如果对DASK感兴趣，可以访问网站<https://dask.org/>了解更多关于DASK的使用方法。

21.3 其他可选方法

除了上面提到的方法，Python还可以利用如下方法来处理大规模数据。

- 使用Pytables，详见<http://www.pytables.org>。
- 使用HDF（Hierarchical Data Format）格式文件。
- 对读入的数据预先指定数据类型来降低内存占用。
- 使用Koalas在Spark上扩展Pandas代码，数据科学家可以从单个机器迁移到分布式环境，而无须学习新的框架。

21.4 留给读者的思考

没试过，你永远不知道自己的极限在哪里！应用工具也是如此，与其频繁变换新工具，不如将一个工具研究透彻。对一个工具或一门开发语言，我们开始学习时是看山是山，但是随着学习的深入，会发现看山不是山。最后当我们跳出这一工具，开始学习其他工具时，会发现看山还是山。

邓立国 著



清华大学出版社

Python 大数据分析 算法与实例

邓立国 著



清华大学出版社
北京



本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13801310933

图书在版编目（CIP）数据

Python大数据分析算法与实例 / 邓立国著. —北京：清华大学出版社，2020.4

ISBN 978-7-302-55106-5

I. ①P… II. ①邓… III. ①软件工具—程序设计 IV. ①TP311.561

中国版本图书馆CIP数据核字（2020）第046755号

责任编辑：夏毓彦

封面设计：王翔

责任校对：闫秀华

责任印制：宋林

出版发行：清华大学出版社

网 址：<http://www.tup.com.cn>, <http://www.wqbook.com>

地 址：北京清华大学学研大厦A座

邮 编：100084

社总机：010-62770175

邮 购： 010-62786544

投稿与读者服务： 010-62776969, c-service@tup.tsinghua.edu.cn

质量反馈： 010-62772015, zhiliang@tup.tsinghua.edu.cn

印 装 者： 北京嘉实印刷有限公司

经 销： 全国新华书店

开 本： 190mm×260mm

印 张： 21.25

字 数： 544千字

版 次： 2020年5月第1版

印 次： 2020年5月第1次印刷

产品编号： 085897-01

内容简介

大数据时代，大数据分析是关键技术。Python是一款优秀的大数据分析软件，本书以Python 3结合第三方开源工具进行大数据分析，以最小的代价编程实现数据的提取、处理、分析和可视化。

本书分为8章，首先介绍大数据分析的背景和行业应用，给出了数据特征算法分析；然后基于Python 3介绍常用典型第三方大数据分析工具的场景应用；最后比较翔实地阐述大数据分析算法与经典实例应用。

本书适合从事大数据分析的研究人员、计算机或数学等相关专业的从业者参考学习，也可以作为计算机或数学等专业本科高年级或研究生的专业用书。

前言

大数据可以概括为5个V，数据量大（Volume）、速度快（Velocity）、类型多（Variety）、价值（Value）、真实性（Veracity）。随着大数据时代的来临，大数据分析也应运而生。大数据分析是指对规模巨大的数据进行分析。大数据分析在研究大量的数据的过程中寻找模式、相关性和其他有用的信息，可以帮助企业更好地适应经营环境的变化，并做出更明智的决策。

大数据分析6个基本流程：可视化分析、数据挖掘算法、预测判断、语义引擎、数据质量和数据管理、数据存储和数据仓库，本书从数据分析可视化入手实现数据展示。数据可视化借助于图形化的手段，清晰、快捷、有效地传达与沟通信息。从用户的角度，数据可视化可以让人快速抓住信息要点，使得关键的数据点从人类的眼睛快速通往大脑。数据可视化一般具备几个特点：准确性、创新性和简洁性。

本书的目的是展现基于Python 3大数据分析方法中的核心算法与实践，介绍的重点是数据特征算法分析及第三方开源库的场景应用，如NumPy、SciPy、Matplotlib、Pandas、StatsModels、Gensim等在大数据分析中的算法与实例应用。

本书可以作为计算机科学与工程、计算统计学和社会科学等专业的大学生或研究生的专业参考书，也可作为软件研究人员或从业人员的参考资料。由于大数据分析专业素材的多学科性，读者可以根据对应的知识背景参考对应的专业图书。

本书内容

本书分为8章，系统讲解了大数据的数据特征算法分析。第1、2章简要介绍了大数据分析的背景、应用及数据特征分析，第3~7章是典型开源工具介绍与场景应用，第8章是基于Python 3的典型大数据分析算法及应用实例。本书的例子都是在Python 3集成开发环境Anaconda 3中经过实际调试通过的典型案例，书中示例的大部分实验数据来源于GitHub，很多例子也给出了源代码的网址，读者可以参考实现。

代码下载

本书示例代码请扫描下边的二维码下载。

如果下载有问题，请联系booksaga@163.com，邮件主题为“Python大数据分析算法与实例”。



致谢

这里要感谢家人的支持与关爱。同时也要感谢我的同事，与他们的交流和探讨使得本书得以修正错误和完善知识结构。由于作者水平有限，书中有纰漏之处在所难免，恳请读者不吝赐教。

作者
2020年1月

目 录

前言

第1章 大数据分析概述

- 1.1 大数据分析背景
- 1.2 大数据分析的应用
- 1.3 大数据分析算法
- 1.4 大数据分析工具
- 1.5 本章小结

第2章 数据特征算法分析

- 2.1 数据分布性分析
 - 2.1.1 数据分布特征集中趋势的测定
 - 2.1.2 数据分布特征离散程度的测定
 - 2.1.3 数据分布特征偏态与峰度的测定
- 2.2 数据相关性分析
 - 2.2.1 数据相关关系
 - 2.2.2 数据相关分析的主要内容
 - 2.2.3 相关关系的测定
- 2.3 数据聚类性分析
 - 2.3.1 聚类分析定义
 - 2.3.2 聚类类型
 - 2.3.3 聚类应用
- 2.4 数据主成分分析
 - 2.4.1 主成分分析的原理及模型
 - 2.4.2 数据主成分分析的几何解释
 - 2.4.3 数据主成分的导出
 - 2.4.4 证明主成分的方差是依次递减的
 - 2.4.5 数据主成分分析的计算
- 2.5 数据动态性分析

[2.6 数据可视化](#)

[2.7 本章小结](#)

[第3章 大数据分析工具：NumPy](#)

[3.1 NumPy简介](#)

[3.2 NumPy环境安装配置](#)

[3.3 ndarray对象](#)

[3.4 数据类型](#)

[3.5 数组属性](#)

[3.6 数组创建例程](#)

[3.7 切片和索引](#)

[3.8 广播](#)

[3.9 数组与元素操作](#)

[3.9.1 数值迭代](#)

[3.9.2 数组变形](#)

[3.9.3 数组翻转](#)

[3.9.4 修改维度](#)

[3.9.5 数组连接](#)

[3.9.6 数组分割](#)

[3.9.7 添加 / 删除元素](#)

[3.10 位操作与字符串函数](#)

[3.11 数学运算函数](#)

[3.12 算术运算](#)

[3.13 统计函数](#)

[3.14 排序、搜索和计数函数](#)

[3.15 字节交换](#)

[3.16 副本和视图](#)

[3.17 矩阵库](#)

[3.18 线性代数模块](#)

[3.19 Matplotlib库](#)

[3.20 Matplotlib 绘制直方图](#)

[3.21 IO文件操作](#)

[3.22 NumPy实例：GPS定位](#)

[3.23 本章小结](#)

[第4章 大数据分析工具：SciPy](#)

[4.1 SciPy简介](#)

[4.2 文件输入和输出：SciPy.io](#)

[4.3 特殊函数：SciPy.special](#)

[4.4 线性代数操作：SciPy.linalg](#)

[4.5 快速傅里叶变换：sipy.fftpack](#)

[4.6 优化器：SciPy.optimize](#)

[4.7 统计工具：SciPy.stats](#)

[4.8 SciPy实例](#)

[4.8.1 最小二乘拟合](#)

[4.8.2 函数最小值](#)

[4.9 本章小结](#)

[第5章 大数据分析工具：Matplotlib](#)

[5.1 初级绘制](#)

[5.2 图像、子区、子图、刻度](#)

[5.3 其他种类的绘图](#)

[5.4 本章小结](#)

[第6章 大数据分析工具：Pandas](#)

[6.1 Pandas系列](#)

[6.2 Pandas数据帧](#)

[6.3 Pandas面板](#)

[6.4 Pandas快速入门](#)

[6.5 本章小结](#)

[第7章 大数据分析工具：Statsmodels与Gensim](#)

[7.1 Statsmodels](#)

[7.1.1 Statsmodels统计数据库](#)

[7.1.2 Statsmodels典型的拟合模型概述](#)

[7.1.3 Statsmodels举例](#)

[7.2 Gensim](#)

- [7.2.1 基本概念](#)
- [7.2.2 训练语料的预处理](#)
- [7.2.3 主题向量的变换](#)
- [7.2.4 文档相似度的计算](#)

[7.3 本章小结](#)

[第8章 大数据分析算法与实例](#)

- [8.1 描述统计](#)
- [8.2 假设检验](#)
- [8.3 信度分析](#)
- [8.4 列联表分析](#)
- [8.5 相关分析](#)
- [8.6 方差分析](#)
 - [8.6.1 单因素方差分析](#)
 - [8.6.2 多因素方差分析](#)
- [8.7 回归分析](#)
- [8.8 聚类分析](#)
- [8.9 判别分析](#)
- [8.10 主成分分析](#)
- [8.11 因子分析](#)
- [8.12 时间序列分析](#)
- [8.13 生存分析](#)
- [8.14 典型相关分析](#)
- [8.15 RoC分析](#)
- [8.16 距离分析](#)
- [8.17 对应分析](#)
- [8.18 决策树分析](#)
- [8.19 神经网络—深度学习](#)
 - [8.19.1 深度学习的基本模型](#)
 - [8.19.2 新闻分类实例](#)
- [8.20 蒙特·卡罗模拟](#)
 - [8.20.1 蒙特·卡罗模拟基本模型](#)

[8.20.2 蒙特·卡罗模拟计算看涨期权实例](#)

[8.21 关联规则](#)

[8.21.1 关联规则的概念](#)

[8.21.2 Apriori算法及实例](#)

[8.21.3 FP树频集算法](#)

[8.22 Uplift Modeling](#)

[8.23 集成方法](#)

[8.24 异常检测](#)

[8.25 文本挖掘](#)

[8.26 Boosting算法（提升法和Gradient Boosting）](#)

[8.27 本章小结](#)

[参考文献](#)

[返回总目录](#)

第1章 大数据分析概述

大数据作为时下火热的IT行业的词汇，随之而来的数据仓库、数据安全、数据分析、数据挖掘等围绕大数据商业价值的利用，逐渐成为行业人士争相追捧的利润焦点。随着大数据时代的来临，大数据分析应运而生。

1.1 大数据分析背景

1. 大数据的狭隘定义

大数据（Big Data）是指无法在特定时间范围内用规范化手段进行捕获、处理和筛选的数据集合，是需要新处理模式才能具有更强的决策力、洞察发现力和流程优化能力的海量、高增长率和多样化的信息资产。

2. 大数据的产生

“大数据”的名称来自于未来学家托夫勒所著的《第三次浪潮》，《自然》杂志在2008年9月推出了名为“大数据”的封面专栏。从2009年开始“大数据”才成为互联网技术行业中的热门词汇。2004年出现的社交媒体则把全世界每个人都转变成了潜在的数据生成器，向摩尔定律铸成的巨鼎中贡献数据，这是“大容量”形成的主要原因。

3. 大数据的特征

- 容量 (Volume)：数据的大小决定所考虑的数据的价值和潜在的信息。
- 种类 (Variety)：数据类型的多样性。
- 速度 (Velocity)：指获得数据的速度。
- 可变性 (Variability)：妨碍了处理和有效地管理数据的过程。
- 真实性 (Veracity)：数据的质量。
- 复杂性 (Complexity)：数据量巨大，来源多渠道。
- 价值 (Value)：合理运用大数据，以低成本创造高价值。

4. 大数据的结构

大数据包括结构化、半结构化和非结构化数据，非结构化数据越来越成为数据的主要部分。据IDC的调查报告显示：企业中80%的数据都是非结构化数据，这些数据每年都按指数增长60%。

5. 大数据分析

大数据分析的产生旨在IT管理，企业可以将实时数据流分析和历史相关数据相结合，然后进行大数据分析并发现它们所需的模型。反过来，帮助预测和预防未来运行中断和性能问题。

6. 大数据分析的意义

现在的社会是一个高速发展的社会，科技发达，信息流通，人们之间的交流越来越密切，生活也越来越方便，大数据就是这个高科技时代的产物。阿里巴巴创始人马云在演讲中就提到，“未来的时代将不是IT的时代，而是DT的时代”，DT就是Data Technology（数据科技），可以看出大数据对于阿里巴巴集团来说举足轻重。

有人把数据比喻为蕴藏能量的煤矿。煤炭按照性质有焦煤、无烟煤、肥煤、贫煤等分类，而露天煤矿、深山煤矿的挖掘成本又不一樣。与此类似，大数据并不在“大”，而在于“有用”。价值含量、挖掘成本比数量更重要。对于很多行业而言，如何利用这些大规模的数据是赢得竞争的关键。

1.2 大数据分析的应用

未来将是一个“大数据”引领的智慧科技的時代，随着社交网络的逐渐成熟，移动带宽迅速提升，云计算、物联网应用更加丰富，更多的传感设备、移动终端接入网络，由此产生的数据及增长速度将比历史上的任何时期都要多、要快。

虽然大数据在不同领域有不同的应用，但是总的来说，大数据的应用主要体现在三个方面，分别是分析预测、决策制定和技术创新。同时，大数据在很大程度上推动了人工智能的发展。

1. 分析预测

分析预测是比较早的落地应用之一，同时能够比较直观地获得价值，所以当前大数据的场景分析依然是比较重要的落地应用。分析预测涉及的行业非常多，比如舆情分析、流感预测、金融预测、销售分析等，随着传统行业信息化改造的推进，数据分析将是比较常见的大数据应用。

2. 决策制定

决策制定通常是大数据应用的重要目的，销售部门需要根据数据分析来制定产品的销售策略，设计部门需要根据数据分析来制定产品

的设计策略，生产部门需要根据数据分析来优化生产流程，人事部门需要根据数据来衡量员工的工作价值从而制定考核策略，财务部门需要根据数据分析来制定财务策略，等等。通常来说，数据分析一个重要的目的就是为了制定相应的策略。

3. 技术创新

大数据应用能够全面促进企业创新，不仅体现在技术创新上，还体现在管理创新上。通过数据能够挖掘出更多关于产品和市场的信息，这些信息会指导企业进行相应产品的设计，以满足市场的需求。同时在企业管理方面，以数据为驱动的管理方式能够极大地调动员工的能动性。

1.3 大数据分析算法

1. 大数据分析 with 数据分析的区别

大数据分析是指无法在可承受的时间范围内用常规软件工具进行捕捉、管理和处理的数据集合，是需要新处理模式才能具有更强的决策力、洞察发现力和流程优化能力的海量、高增长率和多样化的信息处理模式。

数据分析是指用适当的统计分析方法对收集来的大量数据进行分析，提取有用信息和形成结论，从而对数据加以详细研究和概括总结的过程。

大数据分析的优势是能清楚地阐述数据采集、大数据处理过程及最终结果的解读，同时提出模型的优化和改进之处，以利于提升大数据分析的商业价值。

大数据分析 with 数据分析的核心区别是处理的数据规模不同，由此导致两个方向从业者的技能也不同。大数据分析 with 数据分析的根本区别是分析的思维与分析所用的工具不同。

2. 机器学习和数据挖掘的联系与区别

从数据分析的角度来看，数据挖掘与机器学习（Machine Learning, ML）有很多相似之处，但不同之处也十分明显，例如，数据挖掘并没有机器学习探索人的学习机制这一科学发现任务，数据挖掘中的数据分析是针对海量数据进行的，从某种意义上来说，机器学习的科学成分更重一些，而数据挖掘的技术成分更重一些。

机器学习是一门多领域交叉学科，涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科。其专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或技能，重新组织已有的知识结构，使之不断改善自身的性能。

数据挖掘是从海量数据中获取有效的、新颖的、潜在有用的、最终可理解的模式的非平凡过程。数据挖掘中用到了大量的机器学习领域提供的数据分析技术，以及数据库领域提供的数据管理技术。

机器学习不仅涉及对人的认知学习过程的探索，还涉及对数据的分析处理。实际上，机器学习已经成为计算机数据分析技术的创新源头之一。由于几乎所有的学科都要面对数据分析任务，因此机器学习已经开始影响计算机科学的众多领域，甚至影响计算机科学之外的很多学科。机器学习是数据挖掘中的一种重要工具。然而数据挖掘不仅仅要研究、拓展、应用一些机器学习方法，还要通过许多非机器学习技术解决数据仓储、大规模数据、数据噪声等实践问题。机器学习的涉及面很宽，常用在数据挖掘上的方法是“从数据学习”。然而机器

学习不仅仅可以用在数据挖掘上，一些机器学习的子领域甚至与数据挖掘关系不大，如增强学习与自动控制等。

3. 统计学习与机器学习的联系与区别

统计学和机器学习之间的界定一直很模糊。无论是业界还是学界一直认为机器学习只是统计学披了一层光鲜的外衣。而事实是统计学与机器学习的不同，统计模型与机器学习的不同。机器学习和统计的主要区别在于它们的目的。机器学习模型旨在使最准确的预测成为可能。统计模型是为推断变量之间的关系而设计的。

首先，我们必须明白，统计和统计建模是不一样的。统计是对数据的数学研究，除非有数据，否则无法进行统计。统计模型是数据的模型，主要用于推断数据中不同内容的关系，或创建能够预测未来值的模型。通常情况下，这两者是相辅相成的。机器学习通常会牺牲可解释性以获得强大的预测能力。例如，从线性回归到神经网络，尽管解释性变差，但是预测能力却大幅提高。

统计模型与机器学习在线性回归的应用上是有差异的，或许是因为统计建模和机器学习中使用方法的相似性，使人们认为它们是同一类算法，但事实上不是这样的。统计模型和机器学习在回归分析建模方法的相似性，是造成这种误解的主要原因，其实它们的目的是不同的。线性回归是一种统计方法，通过这种方法既可以训练一个线性回归器，又可以通过最小二乘法拟合一个统计回归模型。机器学习（这里特指有监督学习）的目的是获得一个可反复预测的模型，通常不关心模型是否可以解释，只在乎结果。而统计建模更多的是为了寻找变量之间的关系和确定关系的显著性，恰巧迎合了预测。

4. 统计学与数据挖掘的联系与区别

统计学和数据挖掘有着共同的目标：发现数据中的结构。事实上，由于它们的目标相似，有人认为数据挖掘是统计学的分支。这种看法有偏差。因为数据挖掘还应用了其他领域的思维、工具和算法，尤其是计算机科学技术，例如数据库技术和机器学习，而且数据挖掘关注的某些领域和统计学家关注的有很大差别。

5. 大数据分析的10个统计方法

数据分析师不完全是软件工程师，应该是编程、统计和批判性思维三者的结合体。统计学习是培养现代数据分析师的一个基本素材。下面分享10个统计方法，任何数据分析师都应该学习，进而更高效地处理大数据集。

（1）线性回归

线性回归是一种通过拟合因变量和自变量之间的最佳线性关系来预测目标变量的方法。线性回归主要分为简单线性回归和多元线性回归。简单线性回归使用一个自变量，通过拟合一个最佳线性关系来预测因变量；而多元线性回归使用一个以上的自变量来预测因变量。

（2）分类

分类是一种数据挖掘技术，用来将一个整体数据集分成几个类别，以便更准确地预测和分析。

（3）重采样方法

重采样是从原始数据样本中反复抽样的方法，是一种非参数统计推断方法。重采样在实际数据的基础上生成唯一的抽样分布。

（4）子集选择

子集选择首先确定我们认为与反应有关的P预测因子的一个子集，然后使用子集特征的最小二乘拟合模型。

（5）特征缩减技术

通过对损失函数加入正则项，使得在训练求解参数的过程中将影响较小的特征的系数衰减到0，只保留重要的特征。

（6）降维

降维是将估计 $P+1$ 个系数减少为 $M+1$ 个系数，其中 M 可以将主成分回归描述为从一组大的变量中导出低纬度特征集的方法。

（7）非线性回归

非线性回归是回归分析的一种形式，在这种分析中，观测数据通过模型参数和因变量的非线性组合函数建模，数据用逐次逼近法进行拟合。

（8）树形方法

树形方法可以用于回归和分类问题，这些涉及将预测空间分层或分段为一些简单的区域。由于分割预测空间的分裂规则可以用树形总结，因此这类方法也被称为决策树方法。

（9）支持向量机

支持向量机（Support Vector Machine, SVM）是一种分类技术，简单地说，就是寻找一个超平面以最好地将两类点与最大边界区分开。

（10）无监督学习

无监督学习就是在无类别信息的情况下寻找到好的特征。

1.4 大数据分析工具

1. 大数据分析前端展现

用于展现分析的前端开源工具有 JasperSoft、Pentaho、Spagobi、Openi、Birt等。

用于展现分析的商用分析工具有 Style Intelligence、RapidMiner Radoop、Cognos、BO、Microsoft Power BI、Oracle、MicroStrategy、QlikView、Tableau等。

国内大数据分析工具有BDP、国云数据（大数据魔镜）、思迈特、FineBI等。

2. 大数据分析数据仓库

有Teradata AsterData、EMC GreenPlum、HP Vertica等。

3. 大数据分析数据集市

有QlikView、Tableau、Style Intelligence等。

4. 统计分析

统计分析法是指通过对研究对象的规模、速度、范围、程度等数量关系的分析研究，认识和揭示事物间的相互关系、变化规律和发展趋势，借以实现对事物的正确解释和预测的一种研究方法。

5. 可视化辅助工具

数据可视化技术的基本思想，是将数据库中每一个数据项作为单个图元元素表示，大量的数据集构成数据图像，同时将数据的各个属性值以多维数据的形式表示，可以从不同的维度观察数据，从而对数据进行更深入的观察和分析。一旦原始数据流被以图像形式表示时，

以此做决策就变得容易多了。为了满足并超越客户的期望，大数据可视化工具应该具备这些特征：

- 能够处理不同种类型的传入数据。
- 能够应用不同种类的过滤器来调整结果。
- 能够在分析过程中与数据集进行交互。
- 能够连接其他软件来接收输入数据，或为其他软件提供输入数据。
- 能够为用户提供协作选项。

下面介绍目前比较实用和流行的4种大数据可视化工具，它们提供了上述所有或者部分的特征。

（1）Jupyter：大数据可视化的一站式商店

Jupyter是一个开源项目，通过十多种编程语言实现大数据分析、可视化和软件开发的实时协作。Jupyter的界面包含代码输入窗口，通过运行输入的代码基于所选择的可视化技术提供视觉可读的图像。

（2）Tableau：AI、大数据和机器学习应用可视化的最佳解决方案

Tableau是大数据可视化的市场领导者之一，在为大数据操作、深度学习算法和多种类型的AI应用程序提供交互式数据可视化方面尤为高效。

（3）Google Chart：Google支持的免费而强大的整合功能

Google Chart（谷歌图表）是大数据可视化的最佳解决方案之一，它是完全免费的，并得到了Google的大力技术支持。

（4）D3.js：以任何用户需要的方式直观地显示大数据

D3.js代表Data Driven Document，是一个用于实时交互式大数据可视化的JS库。由于D3.js不是一个工具，因此用户在使用它处理数据

之前需要对JavaScript有一个很好的理解，并且要以一种能被其他人理解的形式呈现。除此以外，这个JS库将数据以SVG和HTML 5格式呈现，所以像IE 7和IE 8这样的旧式浏览器不能使用D3.js的功能。

6. 大数据处理框架

随着这些年全世界数据的几何式增长，数据的存储和运算都将成为世界级的难题。分布式文件系统解决的是大数据存储的问题。下面介绍一些分布式计算框架。

(1) Hadoop框架

Hadoop是目前世界上应用最广泛的大数据工具，它凭借极高的容错率和极低的硬件价格，在大数据市场上风生水起。Hadoop是第一个在开源社区上引发高度关注的批处理框架，它提出的Map和Reduce计算模式简洁而优雅。迄今为止，Hadoop已经成为一个广阔的生态圈，实现了大量算法和组件。由于Hadoop的计算任务需要在集群的多个节点上多次读写，因此在速度上会稍显劣势，但是其吞吐量是其他框架所不能匹敌的。

(2) Storm框架

与Hadoop的批处理模式不同，Storm采用的是流计算框架，由Twitter开源并且托管在GitHub上。与Hadoop类似的是，Storm也提出了两个计算角色，分别为Spout和Bolt。

(3) Samza框架

Smaza是一种流计算框架，但它目前只支持JVM语言，灵活度上略显不足，并且Samza必须和Kafka共同使用。相应地，其也继承了Kafka的低延时、分区、避免回压等优势。

(4) Spark框架

Spark属于Hadoop和Storm两种框架形式的集合体，是一种混合式的计算框架。它既有自带的实时流处理工具，又可以和Hadoop集成，代替其中的MapReduce，甚至Spark还可以单独拿出来部署集群，但是还得借助HDFS等分布式存储系统。Spark的强大之处在于其运算速度，与Storm类似，Spark也是基于内存的，并且在内存满负载的时候，硬盘也能运算。运算结果显示，Spark的速度大约为Hadoop的一百倍，并且其成本可能比Hadoop更低。但是Spark目前还没有像Hadoop那样拥有上万级别的集群，因此现阶段的Spark和Hadoop搭配起来使用更加合适。

7. 数据库

数据库可视为电子化的文件柜—存储电子文件的处所，用户可以对文件中的数据进行新增、查询、更新、删除等操作。

8. 数据仓库 / 商业智能

数据仓库（Data Warehouse，DW或DWH），是企业所有级别的决策制定过程提供所有类型数据支持的战略集合。它是为单个数据存储，出于分析性报告和决策支持的目的而创建的。数据仓库可以为需要业务智能的企业提供业务流程改进指导，监视时间、成本、质量以及控制。

商业智能（Business Intelligence，BI）又称商业智慧或商务智能，指用现代数据仓库技术、线上分析处理技术、数据挖掘和数据展现技术进行数据分析以实现商业价值。

伴随数据库技术的提高和数据处理技术的发展以及各行业业务自动化的实现，商业领域产生了大量的业务数据，想要从这些海量数据中提取出真正有价值的信息，将数据转化为知识，以支持商业决策，

需要用到能提取和存储有用信息，并能支持决策的数据仓库、联机分析处理（On-Line Analysis Processing, OLAP）以及数据挖掘（Data Mining, DM）等技术。因此，从技术层面讲，商业智能不是什么新技术，它是数据仓库、联机分析处理和数据挖掘等技术的综合运用。

9. 数据挖掘

数据挖掘，又译为资料探勘、数据采矿。它是数据库知识发现中的一个步骤。数据挖掘一般是指从大量的数据中通过统计、在线分析处理、情报检索、机器学习、专家系统（通过算法搜索隐藏于其中信息的过程。数据挖掘通常与计算机科学有关，靠过去的经验法则）和模式识别等诸多方法来实现数据中发现知识。

10. 编程语言

做好大数据分析不能缺少编程语言基础，如掌握Python、R、Ruby、Java等编程知识是必不可少的。

1.5 本章小结

大数据技术经过多年的发展已经趋于成熟，逐渐形成了一个较为清晰的产业链，包括数据的采集、整理、分析、呈现等，不同的环节往往都有众多的参与者，随着大数据逐渐落地到广大的传统行业，大数据的应用场景会得到进一步的拓展，大数据的价值也将逐渐提升。本章简要介绍了大数据分析的背景知识、场景应用、分析算法和大数据分析的必备技能和工具。

第2章 数据特征算法分析

大数据分析挖掘是一门多领域交叉学科，涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科。数据和特征决定了大数据分析的模型构建，模型和算法是逼近这个大数据分析的工具手段，特征工程的目的是最大限度地从原始数据中提取特征以供算法和模型使用。

2.1 数据分布性分析

统计数据的分布特征可以从两方面进行描述：一是数据分布的集中趋势；二是数据分布的离散程度。集中趋势和离散程度是数据分布特征对立统一的两方面。本节通过介绍平均指标和变异指标这两种统计指标的概念及计算来讨论反映数据集中趋势和离散程度这两方面的特征。

2.1.1 数据分布特征集中趋势的测定

集中趋势是指一组数据向某中心值靠拢的倾向，集中趋势的测度实际上就是对数据一般水平代表值或中心值的测度。不同类型的数据用不同的集中趋势测度值，低层次数据的集中趋势测度值适用于高层次的测量数据；反过来，高层次数据的集中趋势测度值并不适用于低层次的测量数据。选用哪一个测度值来反映数据的集中趋势，要根据所掌握的数据的类型来确定。

通常用平均指标作为集中趋势测度指标。本节重点介绍众数、中位数两个位置平均数和算术平均数、调和平均数及几何平均数3个数值型平均数。

1. 众数

众数是指一组数据中出现次数最多的变量值，用 M_0 表示。从变量分布的角度看，众数是具有明显集中趋势点的数值，一组数据分布的最高峰点所对应的变量值即为众数。当然，如果数据的分布没有明显的集中趋势或最高峰点，众数就可以不存在；如果有多个高峰点，就有多个众数。

(1) 定类数据和定序数据众数的测定

在使用定类数据与定序数据计算众数时，只需找出出现次数最多的组所对应的变量值即可。

(2) 未分组数据或单变量值分组数据众数的确定

在使用未分组数据或单变量值分组数据计算众数时，只需找出出现次数最多的变量值即可。

(3) 组距分组数据众数的确定

对于组距分组数据来说，众数的数值与其相邻两组的频数分布有一定的关系，这种关系可作如下理解：

设众数组的频数为 f_m ，众数前一组频数为 f_{-1} ，众数后一组的频数为 f_{+1} 。当众数相邻两组的频数相等时，即 $f_{-1} = f_{+1}$ ，众数组的组中值即为众数；当众数组的前一组的频数多于众数组后一组的频数时，即 $f_{-1} > f_{+1}$ ，众数会向其前一组靠，众数小于其组中值；当众数组后一组的频数多于众数组前一组的频数时，即 $f_{-1} < f_{+1}$ ，众数会向其后一组靠，众数大于其组中值。基于这种思路，借助几何图形而导出的分组数据众数的计算公式如下：

$$\begin{aligned}
M_0 &\doteq L + \frac{f_m - f_{-1}}{(f_m - f_{-1}) + (f_m - f_{+1})} \times i \\
M_0 &\doteq U - \frac{f_m - f_{+1}}{(f_m - f_{-1}) + (f_m - f_{+1})} \times i
\end{aligned}
\tag{2.1}$$

其中， L 表示众数所在组的下限， U 表示众数所在组的上限， i 表示众数所在组的组距， f_m 为众数组的频数， f_{-1} 为众数组前一组的频数， f_{+1} 为众数组后一组的频数。

上述下限和上限公式是假定数据分布具有明显的集中趋势，且众数组的频数在该组内是均匀分布的，若这些假定不成立，则众数的代表性会很差。从众数的计算公式可以看出，众数是根据众数组及相邻组的频率分布信息来确定数据中心点位置的，因此众数是一个位置代表值，它不受数据中极端值的影响。

2. 中位数

中位数是将总体各单位标志值按大小顺序排列后，处于中间位置的那个数值。各变量值与中位数的离差绝对值之和最小，即：

$$\sum_{i=1}^n |X_i - M_e| = \min
\tag{2.2}$$

(1) 定序数据中位数的确定

定序数据中位数确定的关键是确定中间位置，中间位置所对应的变量值即为中位数。

①未分组原始资料中间位置的确定

$$\begin{cases} \text{中位数位置} = \frac{N+1}{2} & N \text{ 为奇数} \\ \text{中位数位置} = \frac{N}{2} & N \text{ 为偶数} \end{cases}
\tag{2.3}$$

②分组数据中间位置的确定

$$\text{中位数位置} = \frac{\sum f}{2} \quad (2.4)$$

(2) 数值型数据中位数的确定

$$\text{数值型数据资料} = \begin{cases} \text{未分组资料} \\ \text{分组资料} \begin{cases} \text{单变量值分组资料} \\ \text{组距分组资料} \end{cases} \end{cases}$$

①未分组资料

首先必须将标志值按大小排序。设排序的结果为： $x_1 \leq x_2 \leq x_3 \leq \dots \leq x_n$ ，则：

$$M_e = \begin{cases} X_{\left(\frac{N+1}{2}\right)} & \text{当 } N \text{ 为奇数时} \\ \frac{1}{2} \left(X_{\frac{N}{2}} + X_{\frac{N}{2}+1} \right) & \text{当 } N \text{ 为偶数时} \end{cases} \quad (2.5)$$

②单变量分组资料

$$M_e = \begin{cases} X_{\left(\frac{\sum f+1}{2}\right)} & \sum f \text{ 为奇数时} \\ X_{\left(\frac{\sum f}{2}\right)} & \sum f \text{ 为偶数时} \end{cases} \quad (2.6)$$

③组距分组资料

根据位置公式确定中位数所在的组，假定在中位数组内的各单位是均匀分布的，则可利用下面的公式计算中位数的近似值：

$$M_e = L + \frac{\frac{\sum f}{2} - S'_{m-1}}{f_m} \cdot i$$

$$M_e = U - \frac{\frac{\sum f}{2} - S'_{m+1}}{f_m} \cdot i \quad (2.7)$$

其中， S_{m-1} 是到中位数组前面一组为止的向上累计频数， S'_{m+1} 则是到中位数组后面一组为止的向下累计频数， f_m 为中位数组的频数， i 为中位数组的组距。

3. 算术平均数

算术平均数 (Arithmetic Mean) 也称为均值 (Mean)，是全部数据算术平均的结果。算术平均法是计算平均指标最基本、最常用的方法。算术平均数在统计学中具有重要的地位，是集中趋势的主要测度值，通常用 $\bar{x}$ 表示。根据所掌握数据形式的不同，算术平均数有简单算术平均数和加权算术平均数。

(1) 简单算术平均数 (Simple Arithmetic Mean)

未经分组整理的原始数据，其算术平均数的计算就是直接将一组数据的各个数值相加除以数值个数。设总体数据为 $X_1, X_2, \dots, X_n$ ，样本数据为 $x_1, x_2, \dots, x_n$ ，则统计总体均值 $\bar{X}$ 和样本均值 $\bar{x}$ 的计算公式为：

$$\bar{X} = \frac{X_1 + X_2 + \dots + X_N}{N} = \frac{\sum_{i=1}^N X_i}{N}$$

$$\bar{x} = \frac{x_1 + x_2 + \dots + x_n}{n} = \frac{\sum_{i=1}^n x_i}{n} \quad (2.8)$$

(2) 加权算术平均数 (Weighted Arithmetic Mean)

根据分组整理的数据计算的算术平均数,就要以各组变量值出现的次数或频数为权数计算加权的算术平均数。设原始数据(总体或样本数据)被分成 K 或 k 组,各组的变量值为 $X_1, X_2, \dots, X_K$,或 $x_1, x_2, \dots, x_k$,各组变量值的次数或频数分别为 $F_1, F_2, \dots, F_K$,或 $f_1, f_2, \dots, f_k$,则总体或样本的加权算术平均数为:

$$\begin{aligned}\bar{X} &= \frac{X_1 F_1 + X_2 F_2 + \dots + X_K F_K}{F_1 + F_2 + \dots + F_K} = \frac{\sum_{i=1}^K X_i F_i}{\sum_{i=1}^K F_i} \\ \bar{x} &= \frac{x_1 f_1 + x_2 f_2 + \dots + x_k f_k}{f_1 + f_2 + \dots + f_k} = \frac{\sum_{i=1}^k x_i f_i}{\sum_{i=1}^k f_i}\end{aligned}\quad (2.9)$$

公式(2.9)中是用各组的组中值代表各组的实际数据,使用代表值时是假定各组数据在各组中是均匀分布的,但实际情况与这一假定会有一定的偏差,使得利用分组资料计算的平均数与实际平均值会产生误差,它是实际平均值的近似值。

加权算术平均数其数值的大小不仅受各组变量值 x_i 大小的影响,而且受各组变量值出现的频数(权数 f_i)大小的影响。如果某一组的权数大,说明该组的数据较多,那么该组数据的大小对算术平均数的影响就越大,反之,则越小。实际上,我们将上式变形为下面公式(2.10)的形式,就能更清楚地看出这一点。

$$\bar{x} = \frac{\sum_{i=1}^K x_i f_i}{\sum_{i=1}^K f_i} = \sum_{i=1}^K x_i \frac{f_i}{\sum_{i=1}^K f_i}\quad (2.10)$$

由上式可以清楚地看出，加权算术平均数受各组变量值（ x_i ）和各组权数（频率 $f_i / \sum f_i$ ）大小的影响。频率越大，相应的变量值计入平均数的份额也越大，对平均数的影响就越大；反之，频率越小，相应的变量值计入平均数的份额也越小，对平均数的影响就越小。这就是权数权衡轻重作用的实质。

算术平均数在统计学中具有重要的地位，它是进行统计分析和统计推断的基础。从统计思想上看，算术平均数是一组数据的重心所在，它是消除了一些随机因素影响后或者数据误差相互抵消后的必然性结果。

算术平均数具有下面一些重要的数学性质，这些数学性质在实际中有着广泛的应用，同时也体现了算术平均数的统计思想。

（1）各变量值与其算术平均数的离差之和等于零，即：

$$\sum_{i=1}^n (x_i - \bar{x}) = 0 \quad (2.11)$$

（2）各变量值与其算术平均数的离差平方和最小，即：

$$\sum_{i=1}^n (x_i - \bar{x})^2 = \min \text{ 或 } \sum_{i=1}^k (x_i - \bar{x})^2 f_i = \min \quad (2.12)$$

4. 调和平均数 (Harmonic Mean)

在实际工作中，经常会遇到只有各组变量值和各组标志总量而缺少总体单位数的情况，这时就要用调和平均数法计算平均指标。调和平均数是各个变量值倒数的算术平均数的倒数，习惯上用 H 表示。计算公式如下：

$$H = \frac{m_1 + m_2 + \cdots + m_k}{\frac{m_1}{x_1} + \frac{m_2}{x_2} + \cdots + \frac{m_k}{x_k}} = \frac{\sum_{i=1}^K m_i}{\sum_{i=1}^K \frac{m_i}{x_i}} \quad (2.13)$$

调和平均数和算术平均数在本质上是一致的，唯一的区别是计算时使用了不同的数据。在实际应用时可掌握这样的原则：当计算算术平均数其分子资料未知时，就采用加权算术平均数计算平均数；当分母资料未知时，就采用加权调和平均数计算平均数。

$$H = \frac{\sum_{i=1}^K m_i}{\sum_{i=1}^K \frac{m_i}{x_i}} = \frac{\sum_{i=1}^K x_i f_i}{\sum_{i=1}^K \frac{x_i f_i}{x_i}} = \frac{\sum_{i=1}^K x_i f_i}{\sum_{i=1}^K f_i} = \bar{x} \quad (2.14)$$

5. 几何平均数 (Geometric Mean)

几何平均数是适应于特殊数据的一种平均数，在实际生活中，通常用来计算平均比率和平均速度。当所掌握的变量值本身是比率的形式，而且各比率的乘积等于总的比率时，就应采用几何平均法计算平均比率。

$$G_M = \sqrt[N]{X_1 \times X_2 \times \cdots \times X_N} = \sqrt[N]{\prod_{i=1}^N X_i} \quad (2.15)$$

也可以看作算术平均数的一种变形：

$$\log G_M = \frac{1}{N} (\log X_1 + \log X_2 + \cdots + \log X_N) = \frac{\sum_{i=1}^N \log X_i}{N} \quad (2.16)$$

6. 众数、中位数与算术平均数的关系

算术平均数与众数、中位数的关系取决于频数分布的状况。它们的关系如下：

(1) 当数据具有单一众数且频数分布对称时，算术平均数与众数、中位数三者完全相等，即 $M_0 = M_e = \bar{X}$ 。

(2) 当频数分布呈现右偏态时，说明数据存在最大值，必然拉动算术平均数向极大值一方靠，则三者之间的关系为 $\bar{X} > M_e > M_0$ 。

(3) 当频数分布呈现左偏态时，说明数据存在最小值，必然拉动算术平均数向极小值一方靠，而众数和中位数由于是位置平均数，不受极值的影响，因此三者之间的关系为 $\bar{X} < M_e < M_0$ 。

当频数分布出现偏态时，极端值对算术平均数产生很大的影响，而对众数、中位数没有影响，此时用众数、中位数作为一组数据的中心值比算术平均数有较高的代表性。算术平均数与众数、中位数从数值上的关系看，当频数分布的偏斜程度不是很大时，无论是左偏还是右偏，众数与中位数的距离约为算术平均数与中位数的距离的两倍，即：

$$\begin{aligned} |M_e - M_0| &= 2|\bar{X} - M_e| \\ M_0 &= \bar{X} - 3(\bar{X} - M_e) = 3M_e - 2\bar{X} \end{aligned} \quad (2.17)$$

2.1.2 数据分布特征离散程度的测定

数据分布的离散程度是描述数据分布的另一个重要特征，反映各变量值远离其中心值的程度，因此也称为离中趋势，从另一个侧面说明了集中趋势测度值的代表程度，不同类型的数据有不同的离散程度测度值。描述数据离散程度的测度值主要有异众比率、极差、四分位差、平均差、方差和标准差、离散系数等，这些指标又称为变异指标。

1. 异众比率

异众比率的作用是衡量众数对一组数据的代表性程度的指标。异众比率越大，说明非众数组的频数占总频数的比重就越大，众数的代表性就越差；反之，异众比率越小，众数的代表性就越好。异众比率主要用于测度定类数据、定序数据的离散程度。

$$V_r = \frac{\sum F_i - F_m}{\sum F_i} = 1 - \frac{F_m}{\sum F_i} \quad (2.18)$$

其中， $\sum F_i$ 为变量值的总频数， F_m 为众数组的频数。

2. 极差

极差是一组数据的最大值与最小值之差，是离散程度的最简单测度值。极差的测度如下：

(1) 未分组数据

$$R = \max(X_i) - \min(X_i) \quad (2.19)$$

(2) 组距分组数据

R 最高组上限-最低组下限

3. 四分位差

中位数是从中间点将全部数据等分为两部分。与中位数类似的还有四分位数、八分位数、十分位数和百分位数等。它们分别是用3个点、7个点、9个点和99个点将数据四等分、八等分、十等分和100等分后各分位点上的值。这里只介绍四分位数的计算，其他分位数与之类似。

一组数据排序后处于25%和75%位置上的值称为四分位数，也称四分位点。四分位数是通过3个点将全部数据等分为4部分，其中每部分包含25%的数据。很显然，中间的分位数就是中位数，因此通常所说的四分位数是指处在25%位置上的数值（下四分位数）和处在75%位置上的数值（上四分位数）。与中位数的计算方法类似，根据未分组数据计算四分位数时，首先对数据进行排序，然后确定四分位数所在的位置。

（1）四分位数确定

设下四分位数为 Q_L ，上四分位数为 Q_U ：

①未分组数据

$$Q_L = X_{\frac{n+1}{4}} \quad Q_U = X_{\frac{3(n+1)}{4}} \quad (2.20)$$

当四分位数的位置不在某一个位置上时，可根据四分位数的位置按比例分摊四分位数两侧的差值。

②单变量值分组数据

$$Q_L = X_{\frac{\sum f}{4}} \quad Q_U = X_{\frac{3\sum f}{4}} \quad (2.21)$$

③组距分组数据

$$Q_L = L + \frac{\frac{\sum f}{4} - S_L}{f_L} \cdot i \quad Q_U = U + \frac{\frac{3\sum f}{4} - S_U}{f_U} \cdot i \quad (2.22)$$

（2）四分位差

四分位数是离散程度的测度值之一，是上四分位数与下四分位数之差，又称为四分位差，亦称为内距或四分间距（Inter-Quartile Range），用 Q_d 表示。四分位差的计算公式为：

$$Q_d = Q_U - Q_L \quad (2.23)$$

4. 平均差 (Mean Deviation)

平均差是离散程度的测度值之一，是各变量值与其算术平均数离差绝对值的平均数，用 M_d 表示。平均差能全面反映一组数据的离散程度，但该方法数学性质较差，实际中应用较少。

(1) 简单平均法

对于未分组资料采用简单平均法。其计算公式为：

$$M_D = \frac{\sum_{i=1}^N |X_i - \bar{X}|}{N} \quad (2.24)$$

(2) 加权平均法

在资料分组的情况下，应采用加权平均法。其计算公式为：

$$M_D = \frac{\sum_{i=1}^K |X_i - \bar{X}| F_i}{\sum_{i=1}^K F_i} \quad (2.25)$$

5. 方差 (Variance) 和标准差 (Standard Deviation)

方差和标准差同平均差一样，也是根据全部数据计算的，反映每个数据与其算术平均数相比平均相差的数值，因此能够准确地反映数据的差异程度。但与平均差的不同之处是在计算时的处理方法不同，平均差是取离差的绝对值消除正负号，而方差、标准差是取离差的平方消除正负号，这更便于数学上的处理。因此，方差、标准差是实际中应用广泛的离中程度度量值。

(1) 总体的方差和标准差

①设总体的方差为 σ^2 ，标准差为 σ ，对于未分组整理的原始资料，方差和标准差的计算公式分别为：

$$\sigma^2 = \frac{\sum_{i=1}^N (X_i - \bar{X})^2}{N} \quad \sigma = \sqrt{\frac{\sum_{i=1}^N (X_i - \bar{X})^2}{N}} \quad (2.26)$$

②对于分组数据，方差和标准差的计算公式分别为：

$$\sigma^2 = \frac{\sum_{i=1}^K (X_i - \bar{X})^2 F_i}{\sum_{i=1}^K F_i} \quad \sigma = \sqrt{\frac{\sum_{i=1}^K (X_i - \bar{X})^2 F_i}{\sum_{i=1}^K F_i}} \quad (2.27)$$

(2) 样本的方差和标准差

样本的方差、标准差与总体的方差、标准差在计算上有所差别。总体的方差和标准差在对各个离差平方平均时是除以数据个数或总频数，而样本的方差和标准差在对各个离差平方平均时是用样本数据个数或总频数减1（自由度）去除总离差平方和。

设样本的方差为 S^2 ，标准差为 S ，对于未分组整理的原始资料，方差和标准差的计算公式为：

$$S_{n-1}^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1} \quad S_{n-1} = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}} \quad (2.28)$$

对于分组数据，方差和标准差的计算公式为：

$$S_{n-1}^2 = \frac{\sum_{i=1}^k (x_i - \bar{x})^2 f_i}{\sum_{i=1}^k f_i - 1} \quad S_{n-1} = \sqrt{\frac{\sum_{i=1}^k (x_i - \bar{x})^2 f_i}{\sum_{i=1}^k f_i - 1}} \quad (2.29)$$

当 n 很大时，样本方差 S^2 与总体方差 σ^2 的计算结果相差很小，这时样本方差也可以用总体方差的公式来计算。

6. 相对离散程度：离散系数

前面介绍的全距、平均差、方差和标准差都是反映一组数值变异程度的绝对值，其数值的大小不仅取决于数值的变异程度，还与变量值水平的高低、计量单位的不同有关。所以，不宜直接利用上述变异指标对不同水平、不同计量单位的现象进行比较，应当先进行无量纲化处理，即将上述反映数据的绝对差异程度的变异指标转化为反映相对差异程度的指标，再进行对比。离散系数通常用 V 表示，常用的离散系数为标准差系数。测度了数据的相对离散程度，用于对不同组别数据离散程度进行比较的计算公式为：

$$V_{\sigma} = \frac{\sigma}{\bar{X}} \quad \text{或} \quad V_s = \frac{S}{\bar{x}} \quad (2.30)$$

2.1.3 数据分布特征偏态与峰度的测定

偏态和峰度就是对这些分布特征的描述。偏态是对数据分布的偏移方向和程度所做的进一步描述，峰度是对数据分布的扁平程度所做的描述。对于偏斜程度的描述用偏态系数，对于扁平程度的描述用峰度系数。

1. 动差法

动差又称矩，原是物理学上用以表示力与力臂对重心关系的术语，这个关系和统计学中变量与权数对平均数的关系在性质上很类似，所以统计学也用动差来说明频数分布的性质。

一般来说，取变量的 a 值为中点，所有变量值与 a 之差的 K 次方的平均数称为变量 X 关于 a 的 K 阶动差。用式子表示即为：

$$\frac{\sum(X-a)^K}{N} \quad (2.31)$$

当 $a=0$ 时，即变量以原点为中心，上式称为 K 阶原点动差，用大写英文字母 M 表示。

一阶原点动差：

$$M_1 = \frac{\sum X}{N} \quad (2.32)$$

二阶原点动差：

$$M_2 = \frac{\sum X^2}{N} \quad (2.33)$$

三阶原点动差：

$$M_3 = \frac{\sum X^3}{N} \quad (2.34)$$

当 $a=\bar{X}$ 时，即变量以算术平均数为中心，上式称为 K 阶中心动差，用小写英文字母 m 表示。

一阶中心动差：

$$m_1 = \frac{\sum(X-\bar{X})}{N} = 0 \quad (2.35)$$

二阶中心动差：

$$m_2 = \frac{\sum(X-\bar{X})^2}{N} = \sigma^2 \quad (2.36)$$

三阶中心动差：

$$m_3 = \frac{\sum (X - \bar{X})^3}{N} \quad (2.37)$$

2. 偏态及其测度

偏态是对分布偏斜方向及程度的度量。从前面的内容中我们已经知道，频数分布有对称的，有不对称的（偏态的）。在偏态的分布中，又有两种不同的形态，即左偏和右偏。我们可以利用众数、中位数和算术平均数之间的关系判断分布是左偏还是右偏的，但要度量分布偏斜的程度就需要计算偏态系数了。

采用动差法计算偏态系数是用变量的三阶中心动差 m_3 与 σ^3 进行对比，计算公式为：

$$\alpha = \frac{m_3}{\sigma^3} \quad (2.38)$$

当分布对称时，变量的三阶中心动差 m_3 由于离差三次方后正负相互抵消而取得0值，因此 $a=0$ ；当分布不对称时，正负离差不能抵消，就形成正的或负的三阶中心动差 m_3 。当 m_3 为正值时，表示正偏离差值比负偏离差值大，可以判断为正偏或右偏；反之，当 m_3 为负值时，表示负偏离差值比正偏离差值大，可以判断为负偏或左偏。 $|m_3|$ 越大，表示偏斜的程度就越大。由于三阶中心动差 m_3 含有计量单位，为消除计量单位的影响，就用 σ^3 去除 m_3 ，使其转化为相对数。同样地， a 的绝对值越大，表示偏斜的程度就越大。

3. 峰度及其测度

峰度是用来衡量分布的集中程度或分布曲线的尖峭程度的指标。计算公式如下：

$$\alpha_4 = \frac{m_4}{\sigma^4} = \frac{\sum (X - \bar{X})^4 F_i}{\sigma^4 \cdot \sum F_i} \quad (2.39)$$

分布曲线的尖峭程度与偶数阶中心动差的数值大小有直接的关系， m_2 是方差，于是就以四阶中心动差 m_4 来度量分布曲线的尖峭程度。 m_4 是一个绝对数，含有计量单位，为消除计量单位的影响，将 m_4 除以 σ^4 ，就得到无量纲的相对数。衡量分布的集中程度或分布曲线的尖峭程度往往是以正态分布的峰度作为比较标准的。在正态分布条件下， $m_4 / \sigma^4 = 3$ ，将各种不同分布的尖峭程度与正态分布比较。

当峰度 $\alpha_4 > 3$ 时，表示分布的形状比正态分布更瘦更高，这意味着分布比正态分布更集中在平均数周围，这样的分布称为尖峰分布，如图2.1（a）所示；当 $\alpha_4 = 3$ 时，分布为正态分布；当 $\alpha_4 < 3$ 时，表示分布比正态分布更扁平，意味着分布比正态分布更分散，这样的分布称为扁平分布，如图2.1（b）所示。

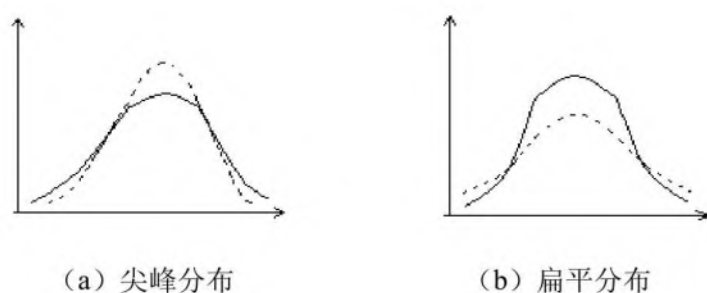


图2.1 尖峰与平峰分布示意图

2.2 数据相关性分析

数据相关性是指数据之间存在某种关系。大数据时代，数据相关性分析因其具有可以快捷、高效地发现事物间内在关联的优势而受到广泛关注，它能有效地应用于推荐系统、商业分析、公共管理、医疗诊断等领域。数据相关性可以用时序分析、空间分析等方法进行分析。数据相关性分析面临着高维数据、多变量数据、大规模数据、增长性数据及其可计算方面的挑战。

2.2.1 数据相关关系

数据相关关系是指两个或两个以上变量取值之间在某种意义下存在的规律，其目的在于探寻数据集里所隐藏的相关关系网。从统计学角度看，变量之间的关系大体可分两种类型：函数关系和相关关系。一般情况下，数据很难满足严格的函数关系，而相关关系要求宽松，所以被人们广泛接受。需要进一步说明的是，研究变量之间的相关关系主要从两个方向进行：一个是相关分析，即通过引入一定的统计指标量化变量之间的相关程度；另一个是回归分析，回归分析不仅刻画相关关系，更重要的是刻画因果关系。

1. 相关系数

对于不同测量尺度的变数，有不同的相关系数可用：

(1) Pearson相关系数 (Pearson Correlation Coefficient)：衡量两个等距尺度或等比尺度变数的相关性，是最常见的，也是学习统计学时第一个接触的相关系数。

(2) 净相关 (Partial Correlation)：在模型中有多个自变数（或解释变数）时，去除掉其他自变数的影响，只衡量特定一个自变数与因变数之间的相关性。自变数和因变数皆为连续变数。

(3) 相关比 (Correlation Ratio) : 衡量两个连续变数的相关性。

(4) Gamma相关系数: 衡量两个次序尺度变数的相关性。

(5) Spearman等级相关系数: 衡量两个次序尺度变数的相关性。

(6) Kendall等级相关系数 (Kendall Tau Rank Correlation Coefficient) : 衡量两个人为次序尺度变数 (原始资料为等距尺度) 的相关性。

(7) Kendall和谐系数: 衡量两个次序尺度变数的相关性。

(8) Phi相关系数 (Phi Coefficient) : 衡量两个真正名目尺度的二分变数的相关性。

(9) 列联相关系数 (Contingency Coefficient) : 衡量两个真正名目尺度变数的相关性。

(10) 四分相关 (Tetrachoric Correlation) : 衡量两个人为名义尺度 (原始资料为等距尺度) 的二分变数的相关性。

(11) Kappa一致性系数 (K Coefficient of Agreement) : 衡量两个名义尺度变数的相关性。

(12) 点二系列相关系数 (Point-Biserial Correlation Coefficient) : X变数是真正名义尺度二分变数。Y变数是连续变数。

(13) 二系列相关系数 (Biserial Correlation Coefficient) : X变数是人为名义尺度二分变数。Y变数是连续变数。

2. 数据种类

(1) 高维数据的相关分析

在探索随机向量间相关性度量的研究中，随机向量的高维特征导致巨大的矩阵计算量，这成为高维数据相关分析中的关键困难问题。面临高维特征空间的相关分析时，数据可能呈现块分布现象，如医疗数据仓库、电子商务推荐系统。探测高维特征空间中是否存在数据的块分布现象，并发现各数据块对应的特征子空间，从本质上来看，这是基于相关关系度量的特征子空间发现问题。结合子空间聚类技术发现相关特征子空间，并以此为基础探索新的分块矩阵计算方法，有望为高维数据相关分析与处理提供有效的求解途径。然而，面临的挑战在于：①如果数据维度很高、数据表示非常稀疏，如何保证相关关系度量的有效性？②分块矩阵的计算可以有效提升计算效率，但是，如何对分块矩阵的计算结果进行融合？

（2）多变量数据的相关分析

在现实的大数据相关分析中，往往面临多变量的情况。显然，发展多变量非线性相关关系的度量方法是我们面临的一个重要的挑战。

（3）大规模数据的相关分析

大数据时代，相关分析面向的是数据集的整体，因此高效地开展相关分析与处理仍然非常困难。为了快速计算大数据的相关性，需要探索数据集整体的拆分与融合策略。显然，在这种“分而治之”的策略中，如何有效保持整体的相关性是大规模数据相关分析中必须解决的关键问题。有关学者给出了一种可行的拆分与融合策略，指出随机拆分策略是可能的解决路径。当然，在设计拆分与融合策略时，如何确定样本子集规模、如何保持子集之间的信息传递、如何设计各子集结果的融合原理等都是具有挑战性的问题。

（4）增长性数据的相关分析

在大数据中，数据呈现快速增长的特征。更为重要的是，诸如电商精准推荐等典型增长性数据相关分析任务迫切需要高效的在线相关

分析技术。就增长性数据而言，可表现为样本规模的增长、维数规模的增长以及数据取值的动态更新。显然，对增长性数据相关分析而言，特别是对在线相关分析任务而言，每次对数据整体进行重新计算对于用户都是难以接受的，更难以满足用户的实时性需求。

我们认为，无论何种类型的数据增长，往往与原始数据集存在某种的关联模式，利用已有的关联模式设计具有递推关系的批增量算法是一种行之有效的计算策略。那么，面向大数据的相关分析任务，探测增长性数据与原始数据集的关联模式，进而发展具有递推关系的高效批增量算法，可为增长性数据相关分析尤其是在线相关分析提供有效的技术手段。

3. 相关关系的种类

现象之间的相互关系很复杂，它们涉及的变动因素多少不同，作用方向不同，表现出来的形态也不同。相关关系大体分为以下几种：

（1）正相关与负相关

按相关关系的方向可分为正相关和负相关。当两个因素（或变量）的变动方向相同时，即自变量 x 的值增大（或减小），因变量 y 的值也相应地增大（或减小），这样的关系就是正相关。例如家庭消费支出随收入的增加而增加就属于正相关。如果两个因素（或变量）变动的方向相反，即自变量 x 的值增大（或减小），因变量 y 的值随之减小（或增大），就称为负相关。例如商品流通费用率随商品经营的规模增大而逐渐减小就属于负相关。

（2）单相关与复相关

按自变量的多少可分为单相关和复相关。单相关是指两个变量之间的相关关系，即所研究的问题只涉及一个自变量和一个因变量，如职工的生活水平与工资之间的关系就是单相关。复相关是指3个或3个

以上变量之间的相关关系，即所研究的问题涉及若干个自变量与一个因变量，如同时研究成本、市场供求状况、消费倾向对利润的影响时，这几个因素之间的关系就是复相关。

（3）线性相关与非线性相关

按相关关系的表现形态可分为线性相关与非线性相关。线性相关是指在两个变量之间，当自变量 x 的值发生变动时，因变量 y 的值发生大致均等的变动，在相关图的分布上，近似地表现为直线形式。比如，商品销售额与销售量即为线性相关。非线性相关是指在两个变量之间，当自变量 x 的值发生变动时，因变量 y 的值发生不均等的变动，在相关图的分布上，表现为抛物线、双曲线、指数曲线等非直线形式。比如，从人的生命全过程来看，年龄与医疗费支出呈非线性相关。

（4）完全相关、不完全相关与不相关

按相关程度可分为完全相关、不完全相关和不相关。完全相关是指两个变量之间具有完全确定的关系，即因变量 y 的值完全随自变量 x 的值的变动而变动，它在相关图上表现为所有的观察点都落在同一条直线上，这时相关关系就转化为函数关系。不相关是指两个变量之间不存在相关关系，即两个变量的变动彼此互不影响。自变量 x 的值变动时，因变量 y 的值不随之做相应变动。比如，家庭收入多少与孩子多少之间不存在相关关系。不完全相关是介于完全相关和不相关之间的一种相关关系。比如，农作物产量与播种面积之间的关系。不完全相关关系是统计研究的主要对象。

2.2.2 数据相关分析的主要内容

相关分析是指对客观现象的相互依存关系进行分析、研究，这种分析方法叫相关分析法。相关分析的目的在于研究相互关系的密切程

度及其变化规律，以便做出判断，进行必要的预测和控制。下面介绍相关分析的主要内容。

（1）确定现象之间有无相关关系

这是相关与回归分析的起点，只有存在相互依存关系，才有必要进行进一步的分析。

（2）确定相关关系的密切程度和方向

确定相关关系的密切程度主要是通过绘制相关图表和计算相关系数。只有对达到一定密切程度的相关关系，才可配合具有一定意义的回归方程。

（3）确定相关关系的数学表达式

为确定现象之间变化上的一般关系，我们必须使用函数关系的数学公式作为相关关系的数学表达式。如果现象之间表现为直线相关，就可采用配合直线方程的方法；如果现象之间表现为曲线相关，就可采用配合曲线方程的方法。

（4）确定因变量估计值的误差程度

使用配合直线或曲线的方法可以找到现象之间一般的变化关系，也就是自变量 x 变化时，因变量 y 将会发生多大的变化。根据得出的直线方程或曲线方程可以给出自变量的若干数值，求得因变量的若干个估计值。估计值与实际值是有出入的，确定因变量估计值误差大小的指标是估计标准误差。估计标准误差大，表明估计不太精确；估计标准误差小，表明估计较精确。

2.2.3 相关关系的测定

相关分析的主要方法有相关表、相关图和相关系数3种。下面详细介绍这3种方法。

（1）相关表

在统计中，制作相关表或相关图可以直观地判断现象之间大致存在的相关关系的方向、形式和密切程度。

在对现象总体中两种相关变量进行相关分析，以研究其相互依存关系时，如果将实际调查取得的一系列成对变量值的资料顺序地排列在一张表格上，这张表格就是相关表。相关表是统计表的一种。根据资料是否分组，相关表可以分为简单相关表和分组相关表。

①简单相关表

简单相关表是资料未经分组的相关表，它是一种把自变量按从小到大的顺序并配合因变量一一对应、平行排列起来的统计表。

②分组相关表

在大量观察的情况下，原始资料很多，运用简单相关表就很难表示。这时就要将原始资料进行分组，然后编制相关表，这种相关表称为分组相关表。分组相关表包括单变量分组相关表和双变量分组相关表两种。

- 单变量分组相关表。在原始资料很多时，对自变量数值进行分组，而对应的因变量不分组，只计算其平均值。根据资料的具体情况，自变量可以是单项式，也可以是组距式。
- 双变量分组相关表。对两种有关变量都进行分组，交叉排列，并列两种变量各组间的共同次数，这种统计表称为双变量分组相关表。这种表格形似棋盘，故又称棋盘式相关表。

（2）相关图

相关图又称散点图。它是以直角坐标系的横轴代表自变量 x ，纵轴代表因变量 y ，将两个变量间相对应的变量值用坐标点的形式描绘出来，用来反映两个变量之间相关关系的图形。

相关图可以按未经分组的原始资料来编制，也可以按分组的资料，包括按单变量分组相关表和双变量分组相关表来编制。通过相关图将会发现，当 y 对 x 是函数关系时，所有的相关点都会分布在某一条线上；在相关关系的情况下，由于其他因素的影响，这些点并非处在一条线上，但所有相关点的分布会显示出某种趋势。所以相关图会很直观地显示现象之间相关的方向和密切程度。

(3) 相关系数

相关表和相关图大体说明变量之间有无关系，但它们的相关关系的紧密程度却无法表达，因此需运用数学解析方法构建一个恰当的数学模型来显示相关关系及其密切程度。如果要对现象之间的相关关系的紧密程度做出确切的数量说明，就需要计算相关系数。

接下来介绍相关系数的计算。相关系数是在直线相关条件下，说明两个现象之间关系密切程度的统计分析指标，记为 γ 。

相关系数的计算公式为：

$$\gamma = \frac{\sigma_{xy}}{\sigma_x \sigma_y} = \frac{\frac{1}{n} \sum (x - \bar{x}) \sum (y - \bar{y})}{\sqrt{\frac{1}{n} \sum (x - \bar{x})^2} \sqrt{\frac{1}{n} \sum (y - \bar{y})^2}} \quad (2.40)$$

式中， n 表示资料项数， $\bar{x}$ 表示 x 变量的算术平均数， $\bar{y}$ 表示 y 变量的算术平均数， σ_x 表示 x 变量的标准差， σ_y 表示 y 变量的标准差， σ_{xy} 表示 xy 变量的协方差。

在实际问题中，如果根据原始资料计算相关系数，可运用相关系数的简捷法计算，其计算公式为：

$$\gamma = \frac{n \sum xy - \sum x \sum y}{\sqrt{n \sum x^2 - (\sum x)^2} \sqrt{n \sum y^2 - (\sum y)^2}} \quad (2.41)$$

（4）相关系数的分析

明晰相关系数的性质是进行相关系数分析的前提。现将相关系数的性质总结如下：

①相关系数的数值范围在-1和+1之间，即 $-1 \leq \gamma \leq 1$ 。

②计算结果，当 $\gamma > 0$ 时， x 与 y 为正相关；当 $\gamma < 0$ 时， x 与 y 为负相关。

③相关系数 γ 的绝对值越接近1，表示相关关系越强；越接近于0，表示相关关系越弱。若 $|\gamma|=1$ ，则表示两个现象完全直线相关。若 $|\gamma|=0$ ，则表示两个现象完全不相关（不是直线相关）。

④相关系数 γ 的绝对值在0.3以下表示无直线相关，0.3以上表示有直线相关，0.3~0.5表示低度直线相关，0.5~0.8表示显著相关，0.8以上表示高度相关。

2.3 数据聚类性分析

所谓数据聚类，是指根据数据的内在性质将数据分成一些聚合类，每一聚合类中的元素尽可能具有相同的特性，不同聚合类之间的特性差别尽可能大。

聚类分析的目的是分析数据是否属于各个独立的分组，使一组中的成员彼此相似，而与其他组中的成员不同。聚类分析对一个数据对象的集合进行分析，但与分类分析不同的是，所划分的类是未知的，因此聚类分析也称为无指导或无监督的（Unsupervised）学习。聚类分析的一般方法是将数据对象分组为多个类或簇（Cluster），在同一簇中的对象具有较高的相似度，而不同簇中的对象差异较大。由于聚类分析的上述特征，在许多应用中，对数据集进行聚类分析后，可将一个簇中的各数据对象作为一个整体对待。

数据聚类（Cluster Analysis）是对静态数据进行分析的一门技术，在许多领域受到广泛应用，包括机器学习、数据挖掘、模式识别、图像分析以及生物信息。

2.3.1 聚类分析定义

1. 聚类应用

随着信息技术高速发展，数据库应用的规模、范围和深度不断扩大，导致积累了大量的数据，而这些激增的数据后面隐藏着许多重要的信息，因此人们希望能够对其进行更高层次的分析，以便更好地利用这些数据。目前的数据库系统可以高效、方便地实现数据的录入、查询、统计等功能，但是无法发现数据中存在的各种关系和规则，更无法根据现有的数据预测未来的发展趋势。而数据聚类分析正是解决这一问题的有效途径，它是数据挖掘的重要组成部分，用于发现在数据库中未知的对象类，为数据挖掘提供有力的支持，它是近年来广为研究的问题之一。

聚类分析是一个极富有挑战性的研究领域，采用基于聚类分析方法的数据挖掘在实践中已取得了较好的效果。聚类分析也可以作为其他算法的预处理步骤，聚类可以作为一个独立的工具来获知数据的分布情况，使数据形成簇，其他算法再在生成的簇上进行处理。聚类算法既可作为特征和分类算法的预处理步骤，也可将聚类结果用于进一步的关联分析。迄今为止，人们提出了许多聚类算法，这些算法都试图解决大规模数据的聚类问题。聚类分析还成功地应用在模式识别、图像处理、计算机视觉、模糊控制等领域，并在这些领域中取得了长足的发展。

2. 数据聚类

所谓聚类，就是将一个数据单位的集合分割成几个称为簇或类别的子集，每个类中的数据都有相似性，它的划分依据就是“物以类聚”。数据聚类分析是根据事物本身的特性，研究对被聚类的对象进行类别划分的方法。聚类分析依据的原则是使同一聚簇中的对象具有尽可能高的相似性，而不同聚簇中的对象具有尽可能高的相异性。聚类分析主要解决的问题是如何在没有先验知识的前提下，实现满足这种要求的聚簇的聚合。聚类分析称为无监督学习（Unsuper-Vised Study），主要体现在聚类学习的数据对象没有类别标记，需要由聚类学习算法自动计算。

2.3.2 聚类类型

经过持续了半个多世纪深入研究聚类算法，聚类技术已经成为常用的数据分析技术之一。其各种算法的提出、发展、演化使得聚类算法家族不断壮大。下面就针对目前数据分析和数据挖掘业界主流的认知对聚类算法进行介绍。

1. 划分方法

给定具有 n 个对象的数据集，采用划分方法对数据集进行 k 个划分，每个划分（每个组）代表一个簇。 $k \leq n$ ，并且每个簇至少包含一个对象，而且每个对象一般来说只能属于一个组。对于给定的 k 值，划分方法是：一般要做一个初始划分，然后采取迭代重新定位技术，通过让对象在不同组间移动来改进划分的准确度和精度。一个好的划分原则是：同一个簇中对象之间的相似性很高（或距离很近），而不同簇的对象之间相异度很高（或距离很远）。

（1）K-Means算法：又叫K均值算法，这是目前最著名、使用最广泛的聚类算法。在给定一个数据集和需要划分的数目 k 后，该算法可

以根据某个距离函数反复把数据划分到 k 个簇中，直到收敛为止。K-Means算法用簇中对象的平均值来表示划分的每个簇，大致的步骤是：首先把随机抽取的 k 个数据点作为初始的聚类中心（种子中心），然后计算每个数据点到每个种子中心的距离，并把每个数据点分配到距离它最近的种子中心；一旦所有的数据点都被分配完成，每个聚类的聚类中心（种子中心）按照本聚类（本簇）的现有数据点重新计算；这个过程不断重复，直到收敛，即满足某个终止条件为止，最常见的终止条件是误差平方和SSE（指令集）局部最小。

（2）K-Medoids算法：又叫K中心点算法，该算法用最接近簇中心的一个对象来表示划分的每个簇。K-Medoids算法与K-Means算法的划分过程相似，两者最大的区别是K-Medoids算法是用簇中最靠近中心点的一个真实的数据对象来代表该簇的，而K-Medoids算法是用计算出来的簇中对象的平均值来代表该簇的，这个平均值是虚拟的，并没有一个真实的数据对象具有这些平均值。

2. 层次方法

在给定 n 个对象的数据集后，可用层次方法（Hierarchical Methods）对数据集进行层次分解，直到满足某种收敛条件为止。按照层次分解的形式不同，层次方法又可以分为凝聚层次聚类和分裂层次聚类。

（1）凝聚层次聚类：又叫自底向上方法，一开始将每个对象作为单独的一类，然后相继合并与其相近的对象或类，直到所有小的类别合并成一个类，即层次的最上面，或者达到一个收敛，即终止条件为止。

（2）分裂层次聚类：又叫自顶向下方法，一开始将所有对象置于一个簇中，在迭代的每一步中，类会被分裂成更小的类，直到最终每

个对象在一个单独的类中，或者满足一个收敛，即终止条件为止。

3. 基于密度的方法

传统的聚类算法都是基于对象之间的距离，即距离作为相似性的描述指标进行聚类划分，但是这些基于距离的方法只能发现球状类型的数据，而对于非球状类型的数据来说，只根据距离来描述和判断是不够的。鉴于此，人们提出了基于密度的方法（Density-Based Methods），其原理是：只要邻近区域内的密度（对象的数量）超过了某个阈值，就继续聚类。换言之，给定某个簇中的每个数据点（数据对象），在一定范围内必须包含一定数量的其他对象。该算法从数据对象的分布密度出发，把密度足够大的区域连接在一起，因此可以发现任意形状的一类。该算法还可以过滤噪声数据（异常值）。基于密度的方法的典型算法包括DBSCAN（Density-Based Spatial Clustering of Application with Noise），以及其扩展算法OPTICS（Ordering Points to Identify the Clustering Structure）。其中，DBSCAN算法会根据一个密度阈值来控制簇的增长，将具有足够高密度的区域划分为类，并可在带有噪声的空间数据库里发现任意形状的聚类。尽管此算法优势明显，但是其最大的缺点就是，该算法需要用户确定输入参数，而且对参数十分敏感。

4. 基于网格的方法

基于网格的方法（Grid-Based Methods）将把对象空间量化为有限数目的单元，而这些单元则形成了网格结构，所有的聚类操作都是在这个网格结构中进行的。该算法的优点是处理速度快，其处理时间常常独立于数据对象的数目，只跟量化空间中每一维的单元数目有关。基于网格的方法的典型算法是STING（Statistical Information Grid，统计信息网格）算法。该算法是一种基于网格的多分辨率聚类

技术，将空间区域划分为不同分辨率级别的矩形单元，并形成一個层次结构，且高层的低分辨率单元会被划分为多个低一层次的较高分辨率单元。这种算法从最底层的网格开始逐渐向上计算网格内数据的统计信息并储存。网格建立完成后，用类似DBSCAN的方法对网格进行聚类。

2.3.3 聚类应用

1. 数据聚类需要解决的问题

在聚类分析的研究中，有许多急待进一步解决的问题，比如：

- 处理大数据量、具有复杂数据类型的数据集合时，聚类分析结果的精确性问题。
- 对高属性维数据的处理能力。
- 数据对象分布形状不规则时的处理能力。
- 处理噪声数据的能力，能够处理数据中包含的孤立点以及未知数据、空缺或者错误的數據。
- 对数据输入顺序的独立性，也就是对于任意的数据输入顺序产生相同的聚类结果。
- 减少对先决知识或参数的依赖性等问题。

这些问题的存在使得我们研究高正确率、低复杂度、I/O开销小、适合高维数据、具有高度的可伸缩性的聚类方法迫在眉睫，这也是今后聚类方法研究的方向。

2. 数据聚类的应用

聚类分析可以作为一个独立的工具来获得数据的分布情况，通过观察每个簇的特点，集中对特定的某些簇进行进一步的分析，以获得

需要的信息。聚类分析应用广泛，除了在数据挖掘、模式识别、图像处理、计算机视觉、模糊控制等领域的应用外，它还被应用在气象分析、食品检验、生物种群划分、市场细分、业绩评估等诸多方面。例如在商务上，聚类分析可以帮助市场分析人员从客户基本库中发现不同的客户群，并且用购买模式来刻画不同的客户群的特征；聚类分析还可以应用在欺诈探测中，聚类中的孤立点就可能预示着欺诈行为的存在。聚类分析的发展过程也是聚类分析的应用过程，目前聚类分析在相关领域已经取得了丰硕的成果。

2.4 数据主成分分析

在实际问题中，我们经常会遇到研究多个变量的问题，而且在多数情况下，多个变量之间常常存在一定的相关性。由于变量个数较多，再加上变量之间的相关性，势必增加了分析问题的复杂性。如何把多个变量综合为少数几个代表性变量，既能够代表原始变量的绝大多数信息，又互不相关，并且在新的综合变量的基础上可以进一步统计分析，就需要进行主成分分析。

2.4.1 主成分分析的原理及模型

1. 主成分分析的原理

主成分分析是采取一种数学降维的方法找出几个综合变量来代替原来众多的变量，使这些综合变量能尽可能地代表原来变量的信息量，而且彼此之间互不相关。这种把多个变量化为少数几个互相无关的综合变量的统计分析方法就叫作主成分分析或主分量分析。

主成分分析所要做的就是：设法将原来众多的具有一定相关性的变量重新组合为一组新的、相互无关的综合变量。通常，数学上的处

理方法是将原来的变量进行线性组合，作为新的综合变量，但若这种组合不加以限制，则会有很多，应该如何选择呢？如果将选取的第一个线性组合（第一个综合变量）记为 F_1 ，自然希望它尽可能多地反映原来变量的信息，这里“信息”用方差来测量，即希望 $Var(F_1)$ 越大，表示 F_1 包含的信息越多。因此，在所有的线性组合中所选取的 F_1 应该是方差最大的，故称 F_1 为第一主成分。如果第一主成分不足以代表原来 P 个变量的信息，再考虑选取 F_2 （第二个线性组合），为了有效地反映原来的信息， F_1 已有的信息就不需要再出现在 F_2 中，用数学语言表达就是要求 $Cov(F_1, F_2)=0$ ，称 F_2 为第二主成分，以此类推，可以构造出第三、第四……第 P 个主成分。

2. 主成分分析的数学模型

对于一个样本资料，观测 p 个变量 $x_1, x_2, \dots, x_p$ ， n 个样品的数据资料阵为：

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix} = (x_1, x_2, \dots, x_p) \quad (2.42)$$

$$\text{其中, } x_j = \begin{pmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{nj} \end{pmatrix}, \quad j = 1, 2, \dots, p。$$

主成分分析就是将 p 个观测变量综合成 p 个新的变量（综合变量），即：

$$\begin{cases} F_1 = a_{11}x_1 + a_{12}x_2 + \cdots + a_{1p}x_p \\ F_2 = a_{21}x_1 + a_{22}x_2 + \cdots + a_{2p}x_p \\ \cdots \\ F_p = a_{p1}x_1 + a_{p2}x_2 + \cdots + a_{pp}x_p \end{cases} \quad (2.43)$$

简写为：

$$F_j = \alpha_{j1}x_1 + \alpha_{j2}x_2 + \cdots + \alpha_{jp}x_p \quad (2.44)$$

$$j = 1, 2, \cdots, p$$

要求模型满足以下条件：

① F_i 、 F_j 互不相关 ($i \neq j$, $i, j = 1, 2, \cdots, p$)。

② F_1 的方差大于 F_2 的方差大于 F_3 的方差。

③ $a_{k1}^2 + a_{k2}^2 + \cdots + a_{kp}^2 = 1$, $k = 1, 2, \cdots, p$ 。

于是，称 F_1 为第一主成分， F_2 为第二主成分，以此类推，有第 p 个主成分。主成分又叫主分量。这里 a_{ij} 称为主成分系数。

上述模型可用矩阵表示为： $F = AX$ 。

其中：

$$F = \begin{pmatrix} F_1 \\ F_2 \\ \vdots \\ F_p \end{pmatrix} \quad X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix} \quad (2.45)$$

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1p} \\ a_{21} & a_{22} & \cdots & a_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ a_{p1} & a_{p2} & \cdots & a_{pp} \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{pmatrix} \quad (2.46)$$

A 称为主成分系数矩阵。

2.4.2 数据主成分分析的几何解释

假设有 n 个样品，每个样品有两个变量，即在二维空间中讨论主成分的几何意义。设 n 个样品在二维空间中的分布大致为一个椭圆，如图2.2所示。

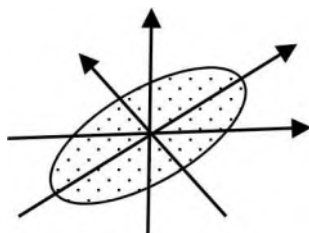


图2.2 主成分几何解释图

将坐标系正交旋转一个角度 θ ，使其在椭圆长轴方向取坐标 y_1 ，在椭圆短轴方向取坐标 y_2 ，旋转公式为：

$$\begin{cases} y_{1j} = x_{1j} \cos \theta + x_{2j} \sin \theta \\ y_{2j} = x_{1j} (-\sin \theta) + x_{2j} \cos \theta \end{cases} \quad j = 1, 2, \dots, n \quad (2.47)$$

写成矩阵形式为：

$$\begin{aligned} Y &= \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \end{bmatrix} \\ &= \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \cdot \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \end{bmatrix} = U \cdot X \end{aligned} \quad (2.48)$$

其中， U 为坐标旋转变换矩阵，它是正交矩阵，即有 $U' = U^{-1}$ ， $UU' = I$ ，即满足 $\sin^2 \theta + \cos^2 \theta = 1$ 。

经过旋转变换后，得到图2.3所示的新坐标。

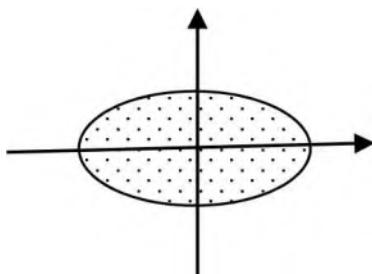


图2.3 新坐标

新坐标 $y_1 - y_2$ 有如下性质：

(1) n 个点的坐标 y_1 和 y_2 的相关几乎为零。

(2) 二维平面上的 n 个点的方差大部分都归结为 y_1 轴上，而 y_2 轴上的方差较小。

y_1 和 y_2 称为原始变量 x_1 和 x_2 的综合变量。由于 n 个点在 y_1 轴上的方差最大，因此将二维空间的点用在 y_1 轴上的一维综合变量来代替所损失的信息量最小，由此称 y_1 轴为第一主成分， y_2 轴与 y_1 轴正交，有较小的方差，称它为第二主成分。

2.4.3 数据主成分的导出

根据主成分分析的数学模型的定义，要进行主成分分析，就需要根据原始数据以及模型的3个条件的要求，求出主成分系数，以便得到主成分模型。这就是导出主成分所要解决的问题。

(1) 根据2.4.1节中主成分数学模型的条件①要求主成分之间互不相关，主成分之间的协差阵应该是一个对角阵。即，对于主成分：

$$F = AX \quad (2.49)$$

其协差阵应为：

$$\begin{aligned}
 \text{Var}(F) &= \text{Var}(AX) = (AX) \cdot (AX)' = AXX' A' \\
 &= \Lambda = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_p \end{pmatrix}
 \end{aligned} \tag{2.50}$$

(2) 设原始数据的协方差阵为 V ，若原始数据进行了标准化处理，则协方差阵等于相关矩阵，即有：

$$V = R = XX' \tag{2.51}$$

(3) 再由2.4.1节中主成分数学模型条件③和正交矩阵的性质，若能够满足条件③，则最好要求 A 为正交矩阵，即满足：

$$AA' = I \tag{2.52}$$

于是，将原始数据的协方差代入主成分的协方差阵公式得：

$$\begin{aligned}
 \text{Var}(F) &= AXX' A' = ARA' = \Lambda \\
 ARA' &= \Lambda \quad RA' = A'\Lambda
 \end{aligned} \tag{2.53}$$

展开上式得：

$$\begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1p} \\ r_{21} & r_{22} & \cdots & r_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ r_{p1} & r_{p2} & \cdots & r_{pp} \end{pmatrix} \cdot \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{p1} \\ a_{12} & a_{22} & \cdots & a_{p2} \\ \vdots & \vdots & \vdots & \vdots \\ a_{1p} & a_{2p} & \cdots & a_{pp} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{p1} \\ a_{12} & a_{22} & \cdots & a_{p2} \\ \vdots & \vdots & \vdots & \vdots \\ a_{1p} & a_{2p} & \cdots & a_{pp} \end{pmatrix} \cdot \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_p \end{pmatrix} \tag{2.54}$$

展开等式两边，根据矩阵相等的性质，这里只根据第一列得出的方程为：

$$\begin{cases} (r_{11} - \lambda_1)a_{11} + r_{12}a_{12} + \cdots + r_{1p}a_{1p} = 0 \\ r_{21}a_{11} + (r_{22} - \lambda_1)a_{12} + \cdots + r_{2p}a_{1p} = 0 \\ \cdots \\ r_{p1}a_{11} + r_{p2}a_{12} + \cdots + (r_{pp} - \lambda_1)a_{1p} = 0 \end{cases} \quad (2.55)$$

为了得到该齐次方程的解，要求其系数矩阵行列式为0，即：

$$\begin{vmatrix} r_{11} - \lambda_1 & r_{12} & \cdots & r_{1p} \\ r_{21} & r_{22} - \lambda_1 & \cdots & r_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ r_{p1} & r_{p2} & \cdots & r_{pp} - \lambda_1 \end{vmatrix} = 0 \quad (2.56)$$

$$|R - \lambda_1 I| = 0$$

显然， λ_1 是相关系数矩阵的特征值， $a_1 = (a_{11}, a_{12}, \dots, a_{1p})$ 是相应的特征向量。根据第二列、第三列等可以得到类似的方程，于是 λ_i 是特征方程 $|R - \lambda I| = 0$ 的特征根， a_j 是其特征向量的分量。

2.4.4 证明主成分的方差是依次递减的

设相关系数矩阵 R 的 p 个特征根为 $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_p$ ，相应的特征向量为 a_j 。

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1p} \\ a_{21} & a_{22} & \cdots & a_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ a_{p1} & a_{p2} & \cdots & a_{pp} \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{pmatrix} \quad (2.57)$$

相对于 F_1 的方差为:

$$Var(F_1) = a_1'XX'a_1' = a_1'Ra_1' = \lambda_1 \quad (2.58)$$

同样有: $Var(F_i) = \lambda_i$, 即主成分的方差依次递减。并且协方差为:

$$\begin{aligned} Cov(a_i'X', a_j'X) &= a_i'Ra_j \\ &= a_i'(\sum_{\alpha=1}^p \lambda_{\alpha} a_{\alpha} a_{\alpha}')a_j \\ &= \sum_{\alpha=1}^p \lambda_{\alpha} (a_i' a_{\alpha})(a_{\alpha}' a_j) = 0, \quad i \neq j \end{aligned} \quad (2.59)$$

综上所述, 根据证明得知, 主成分分析中的主成分协方差应该是对角矩阵, 其对角线上的元素恰好是原始数据相关矩阵的特征值, 而主成分系数矩阵 A 的元素则是原始数据相关矩阵特征值相应的特征向量。矩阵 A 是一个正交矩阵。

于是, 变量 $(x_1, x_2, \dots, x_p)$ 经过变换后得到新的综合变量:

$$\begin{cases} F_1 = a_{11}x_1 + a_{12}x_2 + \dots + a_{1p}x_p \\ F_2 = a_{21}x_1 + a_{22}x_2 + \dots + a_{2p}x_p \\ \dots \\ F_p = a_{p1}x_1 + a_{p2}x_2 + \dots + a_{pp}x_p \end{cases} \quad (2.60)$$

新的随机变量彼此不相关, 且方差依次递减。

2.4.5 数据主成分分析的计算

样本观测数据矩阵为:

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix} \quad (2.61)$$

(1) 对原始数据进行标准化处理:

$$x_{ij}^* = \frac{x_{ij} - \bar{x}_j}{\sqrt{\text{var}(x_j)}} \quad (i = 1, 2, \dots, n, j = 1, 2, \dots, p) \quad (2.62)$$

其中:

$$\bar{x}_j = \frac{1}{n} \sum_{i=1}^n x_{ij}, \text{var}(x_j) = \frac{1}{n-1} \sum_{i=1}^n (x_{ij} - \bar{x}_j)^2 \quad (2.63)$$

$$(j = 1, 2, \dots, p)$$

(2) 计算样本相关系数矩阵:

$$R = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1p} \\ r_{21} & r_{22} & \cdots & r_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ r_{p1} & r_{p2} & \cdots & r_{pp} \end{bmatrix} \quad (2.64)$$

为了方便, 假定原始数据标准化后仍用 X 表示, 则经标准化处理后的数据的相关系数为:

$$r_{ij} = \frac{1}{n-1} \sum_{t=1}^n x_{ti} x_{tj} \quad (2.65)$$

$$(i, j = 1, 2, \dots, p)$$

(3) 用雅克比方法求相关系数矩阵 R 的特征值 ($\lambda_1, \lambda_2, \dots, \lambda_p$) 和相应的特征向量 $a_i = (a_{i1}, a_{i2}, \dots, a_{ip})$, i

$=1, 2, \dots, p$ 。

(4) 选择重要的主成分，并写出主成分表达式。

主成分分析可以得到 p 个主成分，但是，由于各个主成分的方差是递减的，包含的信息量也是递减的，因此在实际分析时，一般不选取 p 个主成分，而是根据各个主成分累计贡献率的大小选取前 k 个主成分。这里贡献率是指某个主成分的方差占全部方差的比重，实际上就是某个特征值占全部特征值合计的比重，即：

$$\text{贡献率} = \frac{\lambda_i}{\sum_{i=1}^p \lambda_i} \quad (2.66)$$

贡献率越大，说明该主成分所包含的原始变量的信息越强。主成分个数 k 的选取主要根据主成分的累积贡献率来决定，即一般要求累计贡献率达到85%以上，这样才能保证综合变量包括原始变量的绝大多数信息。

另外，在实际应用中，选择了重要的主成分后，还要注意主成分的实际含义解释。主成分分析中一个很关键的问题是如何给主成分赋予新的意义，给出合理的解释。一般而言，这个解释是根据主成分表达式的系数结合定性分析来进行的。主成分是原来变量的线性组合，在这个线性组合中，变量的系数有大有小，有正有负，有的大小相当，因而不能简单地认为这个主成分是某个原变量的属性的作用结果，线性组合中各变量系数的绝对值大者表明该主成分主要综合了绝对值大的变量，有几个变量系数大小相当时，应认为这一主成分是这几个变量的总和，这几个变量综合在一起应赋予怎样的实际意义，要结合实际问题和专业给出恰当的解释，进而达到深刻分析的目的。

(5) 计算主成分得分。

根据标准化的原始数据，按照各个样品分别代入主成分表达式，就可以得到各主成分下的各个样品的数据，即为主成分得分。具体形式如下：

$$\begin{pmatrix} F_{11} & F_{12} & \cdots & F_{1k} \\ F_{21} & F_{22} & \cdots & F_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ F_{n1} & F_{n2} & \cdots & F_{nk} \end{pmatrix} \quad (2.67)$$

(6) 进一步的统计分析依据主成分得分的数据，可以进行进一步的统计分析。其中，常见的应用有主成分回归、变量子集合的选择、综合评价等。

2.5 数据动态性分析

动态数据是指观察或记录下来的、一组按时间先后顺序排列起来的数据序列。

1. 数据特征

(1) 构成

- 时间。
- 反映现象在一定时间条件下的数量特征的指标值。

(2) 表示

- $x(t)$ ：时间 t 为自变量。
- 整数：离散的、等间距的。
- 非整数：连续的，实际分析时必须进行采样处理。
- 时间单位：秒、分、小时、日、周、月、年。

2. 动态数据分类——按照指标值的表现形式

(1) 绝对数序列

- 时期序列：可加性。
- 时点序列：不可加性。

(2) 相对数 / 平均数序列

3. 时间数据分类——按照时间的表现形式

- 连续。
- 离散。
- 时间序列中，时间必须是等间隔的。

4. 动态数据的特点

- 数据取值随时间变化。
- 在每一时刻取什么值，不可能完全准确地用历史值预报。
- 前后时刻（不一定是相邻时刻）的数值或数据点有一定的相关性。
- 整体存在某种趋势或周期性。

5. 动态数据的构成与分解

时间序列=趋势+周期+平稳随机成分+白噪声

6. 动态数据分析模型分类

(1) 研究单变量或少数几个变量的变化

- 随机过程：周期分析和时间序列分析。
- 灰色系统：关联分析，GM模型。

(2) 研究多变量的变化

- 系统动力学建模。

7. 时间序列模型

- 研究一个或多个被解释变量随时间变化规律的模型。
- 模型主要用于预测分析。
- 目的—精确预测未来的变化。
- 数据要求—序列平稳。
- 研究角度：
 - 时间域。
 - 频率域。
- 模型内容：
 - 周期分析。
 - 时间序列预测。

时间序列模型的表示：

$$x_t = f(x_{t-1}, x_{t-2}, \dots) + \varepsilon_t \quad (2.68)$$

ε_t 表示白噪声。

8. 动态系统模型

- 研究具有时变特点的多个因素之间的相互作用，以及这些作用与系统整体发展之间的关系模型。

- 模型主要用于模拟和情景分析。
- 研究重点：各种因素是如何相互作用影响系统总体发展的。

9. 模型表示

- 因果反馈逻辑图。
- 未来系统要素变化趋势图。

10. 建模步骤

- (1) 分析数据的动态特征。
- (2) 进行数据序列分解。
- (3) 数据预处理。
- (4) 模型构建模型确认。

11. 建模方法

- (1) 时间序列模型
 - 统计学方法：随机过程理论。
 - 灰色系统方法。
- (2) 动态系统模型
 - 动态系统仿真方法。

12. 时间序列模型

- (1) 平稳随机过程

如果一个随机过程的均值和方差在时间过程上是常数，并且在任何两个时期之间的协方差值仅依赖于这两个时期间的距离和滞后，而不依赖于计算这个协方差的实际时间，那么这个随机过程称为平稳的随机过程。

- 严平稳：一种条件比较苛刻的平稳性定义。认为只有当序列所有的统计性质都不会随着时间的推移而发生变化时，该序列才能被认为是平稳的。
- 宽平稳：使用序列的特征统计量来定义的一种平稳性。认为序列的统计性质主要由它的低阶矩决定，所以只要保证序列低阶矩平稳（二阶），就能保证序列的主要性质近似稳定。

（2）平稳序列的统计性质

- 常数均值。
- 自协方差函数和自相关函数只依赖于时间的平移长度而与时间的起止点无关。

（3）自相关函数

$$\hat{\rho}_k = \frac{\sum_{t=1}^{n-k} (x_t - \bar{x})(x_{t+k} - \bar{x})}{\sum_{t=1}^n (x_t - \bar{x})^2} \quad (2.69)$$

其他的动态数据模型有线性模型法、非线性趋势等。

13. 时间序列建模

任何时间序列都可以看作是一个平稳的过程。所看到的数据集可以看作是平稳过程的一个实现。主要方法有自回归AR(p)、移动平均MA(q)与自回归移动平均ARMA(p, q)等。

(1) 自回归 (AR) 模型

时间序列可以表示成它的先前值和一个冲击值的函数：

$$x_t = \phi_1 x_{t-1} + \phi_{12} x_{t-2} + \cdots + \phi_p x_{t-p} + \varepsilon_t \quad (2.70)$$

(2) 滑动平均 (MA) 模型

序列值是现在和过去的误差或冲击值的线性组合：

$$x_t = \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \cdots - \theta_q \varepsilon_{t-q} \quad (2.71)$$

(3) 自回归滑动平均 (ARMA) 模型

序列值是现在和过去的误差或冲击值以及先前的序列值的线性组合：

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \cdots + \phi_p x_{t-p} + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \cdots - \theta_q \varepsilon_{t-q} \quad (2.72)$$

2.6 数据可视化

数据可视化是关于数据视觉表现形式的科学技术研究。其中，这种数据的视觉表现形式被定义为一种以某种概要形式抽提出来的信息，包括相应信息单位的各种属性和变量。这是一个处于不断演变之中的概念，其边界在不断地扩大。主要指的是技术上较为高级的技术方法，而这些技术方法允许利用图形、图像处理、计算机视觉以及用户界面，通过表达、建模以及对立体、表面、属性以及动画的显示，对数据加以可视化解释。与立体建模之类的特殊技术方法相比，数据可视化所涵盖的技术方法要广泛得多。为了有效地传达思想概念，美学形式与功能需要齐头并进，通过直观地传达关键的方面与特征，从而实现对于相当稀疏而又复杂的数据集的深入洞察。

数据可视化与信息图形、信息可视化、科学可视化以及统计图形密切相关。当前，在研究、教学和开发领域，数据可视化是一个极为

活跃而又关键的方面。“数据可视化”这条术语实现了成熟的科学可视化领域与较年轻的信息可视化领域的统一。

数据可视化技术包含以下几个基本概念。

(1) 数据空间：是由 n 维属性和 m 个元素组成的数据集所构成的多维信息空间。

(2) 数据开发：是指利用一定的算法和工具对数据进行定量的推演和计算。

(3) 数据分析：是指对多维数据进行切片、块、旋转等动作剖析数据，从而能够多角度多侧面观察数据。

(4) 数据可视化：是指将大型数据集中的数据以图形图像形式表示，并利用数据分析和开发工具发现其中未知信息的处理过程。

数据可视化已经提出了许多方法，这些方法根据其可视化的原理不同，可以划分为基于几何的技术、面向像素技术、基于图标的技术、基于层次的技术、基于图像的技术和分布式技术等。数据可视化的适用范围存在着不同的划分方法。一个常见的关注焦点就是信息的呈现。数据可视化的两个主要组成部分是：统计图形和主题图。

1. 数据的特性

要理解数据可视化，先要理解数据，再去掌握可视化的方法，这样才能实现高效的数据可视化。下面是常见的数据类型，在设计时，你可能会遇到以下几种数据类型：

(1) 量性：数据是可以计量的，所有的值都是数字。

(2) 离散性：数字类数据可能在有限的范围内取值。

(3) 持续性：数据可以测量，且在有限范围内。

(4) 范围性：数据可以根据编组和类别而分类。

可视化的意义是帮助人更好地分析数据，也就是说，这是一种高效的手段，并不是数据分析的必要条件。如果我们采用了可视化方案，就意味着机器并不能精确地分析。当然，要明确可视化不能直接带来结果，它需要人来介入分析结论。

2. 数据可视化方法及工具

下面介绍代表性的图形化数据的可视化方法。

- 柱形图。
- 散点图。
- 地图。
- 面积图。
- 漏斗图。
- 仪表盘。
- 饼图。
- 折线图。
- 矩形树图。

编程语言类数据可视化工具如下：

● R

R经常被称为“统计人员为统计人员开发的一种语言”。如果需要深奥的统计模型用于计算，那么可以在CRAN上找到它，CRAN叫综合R档案网络（Comprehensive R Archive Network）并非无缘无故。说到用于分析和标绘，没有什么比得过ggplot2。而如果想利用比机器提供的功能还强大的功能，那么可以使用SparkR绑定，在R上运行Spark。

● Scala

Scala是最轻松的语言，因为大家都欣赏其类型系统。Scala在JVM上运行，基本上成功地结合了函数范式和面向对象范式，目前它在金融界和需要处理海量数据的企业中取得了巨大进展，常常采用一种大规模分布式方式来处理（比如Twitter和LinkedIn）。它还是驱动Spark和Kafka的一种语言。

● Python

Python在学术界一直很流行，尤其是在自然语言处理（NLP）等领域。因而，如果你有一个需要NLP处理的项目，就会面临数量多得让人眼花缭乱的选择，包括经典的NLTK、使用GenSim的主题建模或者超快、准确的spaCy。同样，说到神经网络，Python同样游刃有余，有Theano和TensorFlow；还有面向机器学习的Scikit-Learn，以及面向数据分析的NumPy和Pandas。

● Java

Java很适合大数据项目。Hadoop MapReduce是用Java编写的。HDFS也是用Java编写的。连Storm、Kafka和Spark都可以在JVM上运行（使用Clojure和Scala），这意味着Java是这些项目中的“一等公民”。另外，还有像Google Cloud Dataflow（现在是Apache Beam）这些新技术，直到最近它们还只支持Java。

在大数据时代，可视化图表工具不可能“单独作战”，而我们都知大数据的价值在于数据挖掘，一般数据可视化都是和数据分析功能组合，数据分析又需要数据接入整合、数据处理、ETL等数据功能，发展成为一站式的大数据分析平台。

2.7 本章小结

数据和特征决定了机器学习的上限，而模型和算法只是逼近这个上限而已。机器学习数据分析的目的其实就是直观地展现数据，例如

让花费数小时甚至更久才能归纳的数据量转化成一眼就能读懂的指标；通过加减乘除、各类公式权衡计算得到两组数据的差异，在图中通过元素的颜色敏感、长短大小形成对比。

本章从机器学习的数据分布性、数据相关性、数据聚类性、数据成分、数据动态性及数据可视化等方面介绍了机器学习的数据特征。

第3章 大数据分析工具：NumPy

Python是数据处理常用的工具，可以处理数量级从几千字节至几兆字节不等的数据，具有较高的开发效率和可维护性，还具有较强的通用性和跨平台性。Python可用于数据分析，但其单纯依赖Python本身自带的库进行数据分析还是具有一定的局限性的，需要安装第三方扩展库来增强分析和挖掘能力。

Python数据分析需要安装的第三方扩展库有：NumPy、Pandas、SciPy、Matplotlib、Scikit-Learn、Keras、Gensim等。

3.1 NumPy简介

NumPy是Python的一个科学计算的库，提供了矩阵运算的功能，其一般与SciPy、Matplotlib一起使用。其实，List已经提供了类似于矩阵的表示形式，不过NumPy为我们提供了更多的函数。如果接触过Matlab、Scilab，那么NumPy很好入手。

NumPy是一个Python包。它代表Numeric Python。它是一个由多维数组对象和用于处理数组的例程集合组成的库。

Numeric是NumPy的前身，是由Jim Hugunin开发的。他还开发了另一个包Numarray，拥有一些额外的功能。2005年，Travis Oliphant通过将Numarray的功能集成到Numeric包中来创建NumPy包。这个开源项目有很多贡献者。

使用NumPy，开发人员可以执行以下操作：

- 数组的算术和逻辑运算。
- 傅里叶变换和用于图形操作的例程。

- 与线性代数有关的操作，NumPy拥有线性代数和随机数生成的内置函数。

NumPy通常与SciPy（Scientific Python）和Matplotlib（绘图库）一起使用。这种组合广泛用于替代Matlab，是一个流行的技术计算平台。Python作为Matlab的替代方案，现在被视为一种更加现代和完整的编程语言。

3.2 NumPy环境安装配置

标准的Python发行版不会与NumPy模块捆绑在一起。一个轻量级的替代方法是使用流行的Python包安装程序pip来安装NumPy（<http://www.lfd.uci.edu/~gohlke/pythonlibs/>）。

```
pip install numpy
```

启用NumPy的方法是使用操作系统的可安装的二进制包。这些二进制包包含完整的SciPy技术栈（包括NumPy、SciPy、Matplotlib、IPython、SymPy以及Python核心自带的其他包）。

1. Windows

Anaconda（www.continuum.io）是一个带有SciPy技术栈的免费Python发行版。它可用于Linux和Mac系统。

Canopy（www.enthought.com/products/canopy/）是可用的免费和商业发行版，带有完整的SciPy技术栈，可用于Windows，Linux和Mac系统。

Python (x, y)（www.python-xy.github.io/）是一个免费的Python发行版，带有SciPy技术栈和Spyder IDE，可用于Windows。

2. Linux

Linux发行版的相应软件包管理器可用于安装一个或多个SciPy技术栈中的软件包。

3. Ubuntu

```
sudo apt-get install python-numpy
python-scipy python-matplotlibpythonipynotebook python-
pandas
python-sympy python-nose
```

4. Fedora

```
sudo yum install numpy scipy python-matplotlibpython
python-pandas sympy python-nose atlas-devel
```

5. 从源码构建

核心Python（2.6.x、2.7.x和3.2.x起）必须安装distutils，zlib模块应该启用。GNU（4.2及以上）C编译器（GCC）必须可用。要安装NumPy，请运行以下命令。

```
Python setup.py install
```

要测试NumPy模块是否正确安装，可尝试从Python提示符导入它。如果未安装，那么将显示以下错误消息：

```
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    import numpy
ImportError: No module named 'numpy'
```

3.3 ndarray对象

NumPy中定义的最重要的对象是称为ndarray的N维数组类型。它描述相同类型的元素集合，可以使用基于零的索引访问集合中的项目。

ndarray中的每个元素在内存中使用相同大小的块。ndarray中的每个元素是数据类型对象的对象（称为dtype）。

从ndarray对象提取的任何元素（通过切片）由一个数组标量类型的Python对象表示。图3.1显示了ndarray数据类型对象（dtype）和数组标量类型之间的关系。

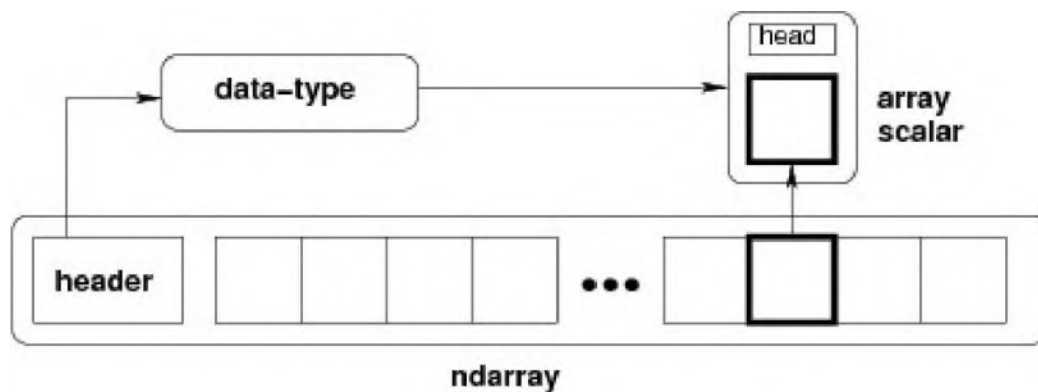


图3.1 ndarray数据类型对象（dtype）和数组标量类型之间的关系

ndarray类的实例可以通过不同的数组创建例程来构造。基本的ndarray是使用NumPy中的数组函数创建的，如下所示：

```
numpy.array
```

它从任何暴露数组接口的对象或从返回数组的任何方法创建一个ndarray。

```
numpy.array(object, dtype = None, copy = True, order = None,
subok = False, ndmin
= 0)
```

上面的构造器接收以下参数：

- object: 任何暴露数组接口方法的对象都会返回一个数组或任何（嵌套）序列。
- dtype: 数组所需的数据类型，可选。
- copy: 可选，默认为true，对象是否被复制。
- order: C（按行）、F（按列）或A（任意，默认）。
- subok: 默认情况下，返回的数组强制为基类数组。如果为true，就返回子类。
- ndmin: 指定返回数组的最小维数。

【例3.1】

```
import numpy as np
a = np.array([1,2,3])
print a
```

输出如下:

```
[1, 2, 3]
```

【例3.2】

```
# 多于一个维度
import numpy as np
a = np.array([[1, 2], [3, 4]])
print a
```

输出如下:

```
[[1, 2]
 [3, 4]]
```

【例3.3】

```
# 最小维度
import numpy as np
a = np.array([1, 2, 3,4,5], ndmin = 2)
print a
```

输出如下：

```
[[1, 2, 3, 4, 5]]
```

【例3.4】

```
# dtype 参数
import numpy as np
a = np.array([1, 2, 3], dtype = complex)
print a
```

输出如下：

```
[ 1.+0.j,  2.+0.j,  3.+0.j]
```

ndarray对象由计算机内存中的一维连续区域组成，带有将每个元素映射到内存块中某个位置的索引方案。内存块以按行（C风格）或按列（Fortran或MATLAB风格）的方式保存元素。

3.4 数据类型

1. NumPy数据类型

NumPy支持比Python更多种类的数据类型。表3.1定义了不同标量的数据类型。

表3.1 不同标量的数据类型

数据类型	描述
bool	存储为一字节的布尔值（真或假）
int	默认为整数，相当于 C 的 long，通常为 int32 或 int64
intc	相当于 C 的 int，通常为 int32 或 int64
intp	用于索引的整数，相当于 C 的 size_t，通常为 int32 或 int64
int8	字节（-128 ~ 127）
int16	16 位整数（-32768 ~ 32767）
int32	32 位整数（-2147483648 ~ 2147483647）
int64	64 位整数（-9223372036854775808 ~ 9223372036854775807）
uint8	8 位无符号整数（0 ~ 255）
uint16	16 位无符号整数（0 ~ 65535）
uint32	32 位无符号整数（0 ~ 4294967295）
uint64	64 位无符号整数（0 ~ 18446744073709551615）
float_	float64 的简写
float16	半精度浮点数，包括 1 个符号位，5 个指数位，10 个尾数位
float32	单精度浮点数，包括 1 个符号位，8 个指数位，23 个尾数位
float64	双精度浮点数，包括 1 个符号位，11 个指数位，52 个尾数位
complex_	complex128 的简写
complex64	复数，由两个 32 位浮点数表示（实部和虚部）
complex128	复数，由两个 64 位浮点数表示（实部和虚部）

NumPy 数值类型是 dtype（数据类型）对象的实例，每个对象具有唯一的特征。这些类型可以是 np.bool_、np.float32 等。

2. 数据类型对象（dtype）

数据类型对象描述了对应于数组的固定内存块的解释，取决于以下5个方面：

- 数据类型（整数、浮点数或者Python对象）。
- 数据大小。

- 字节序（小端或大端）。
- 在结构化类型的情况下，字段的名称、每个字段的数据类型和每个字段占用的内存块部分。
- 数据类型是子序列的形状和数据类型。

字节顺序取决于数据类型的前缀“<”或“>”。<意味着编码是小端（最小有效字节存储在最小地址中），>意味着编码是大端（最大有效字节存储在最小地址中）。

dtype可由以下语法构造：

```
numpy.dtype(object, align, copy)
```

参数说明如下：

- object：被转换为数据类型的对象。
- align：如果为true，就向字段添加间隔，使其类似C的结构体。
- copy：生成dtype对象的新副本，如果为false，结果就是内建数据类型对象的引用。

【例3.5】

```
# 使用数组标量类型
import numpy as np
dt = np.dtype(np.int32)
print dt
```

输出如下：

```
int32
```

【例3.6】

```
# 首先创建结构化数据类型
import numpy as np
```

```
dt = np.dtype([('age', np.int8)])  
print dt
```

输出如下：

```
[('age', 'i1')]
```

【例3.7】

```
# 现在将其应用于 ndarray 对象  
import numpy as np  
  
dt = np.dtype([('age', np.int8)])  
a = np.array([(10,), (20,), (30,)], dtype = dt)  
print a
```

输出如下：

```
[(10,) (20,) (30,)]
```

每个内建类型都有一个唯一定义它的字符代码：

- 'b'：布尔值。
- 'i'：符号整数。
- 'u'：无符号整数。
- 'f'：浮点数。
- 'c'：复数浮点数。
- 'm'：时间间隔。
- 'M'：日期时间。
- 'O'：Python对象。
- 'S', 'a'：字节串。
- 'U'：Unicode。
- 'V'：原始数据（void）。

3.5 数组属性

这一节讨论NumPy的多种数组属性。

(1) shape

语法: ndarray.shape

这一数组属性返回一个包含数组维度的元组，它也可以用于调整数组大小。

【例3.8】

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
print a.shape
```

输出如下:

```
(2, 3)
```

【例3.9】

```
# 这会调整数组大小
import numpy as np
a = np.array([[1,2,3],[4,5,6]]) a.shape = (3,2)
print a
```

输出如下:

```
[[1, 2]
 [3, 4]
 [5, 6]]
```

【例3.10】

NumPy还提供了reshape函数来调整数组大小。

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
b = a.reshape(3,2)
print b
```

输出如下：

```
[[1, 2]
 [3, 4]
 [5, 6]]
```

(2) ndim

语法：ndarray. ndim

这一数组属性返回数组的维数。

【例3.11】

```
# 等间隔数字的数组
import numpy as np
a = np.arange(24)
print a.ndim
```

输出如下：

```
[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
18 19 20 21 22 23]
```

【例3.12】

```
# 一维数组
import numpy as np
a = np.arange(24) a.ndim
# 现在调整其大小
b = a.reshape(2,4,3)
print b
# b 现在拥有3个维度
```

输出如下：

```
[[[ 0, 1, 2]
 [ 3, 4, 5]
 [ 6, 7, 8]
 [ 9, 10, 11]]
 [[12, 13, 14]
 [15, 16, 17]]]
```

```
[18, 19, 20]
[21, 22, 23]]]
```

(3) itemsize

语法: `numpy. itemsize`

这一数组属性返回数组中每个元素的字节单位长度。

【例3.13】

```
# 数组的 dtype 为 int8(一字节)
import numpy as np
x = np.array([1,2,3,4,5], dtype = np.int8)
print x.itemsize
```

输出如下:

```
1
```

(4) flags

语法: `numpy. flags`

ndarray对象拥有以下属性。这个函数返回了它们的当前值。

【例3.14】

展示当前的标志。

```
import numpy as np
x = np.array([1,2,3,4,5])
print x.flags
```

输出如下:

```
C_CONTIGUOUS : True
F_CONTIGUOUS : True
OWNDATA : True
WRITEABLE : True
ALIGNED : True
UPDATEIFCOPY : False
```


表3.2列出了ndarray对象的属性及描述。

表3.2 ndarray对象属性

属性	描述
C_CONTIGUOUS (C)	数组位于单一的、C 风格的连续区段内
F_CONTIGUOUS (F)	数组位于单一的、Fortran 风格的连续区段内
OWNDATA (O)	数组的内存从其他对象处借用
WRITEABLE (W)	数据区域可写入。将它设置为 false 会锁定数据，使其只读
ALIGNED (A)	数据和所有元素都适当地对齐到硬件上
UPDATEIFCOPY (U)	这个数组是另一个数组的副本。当这个数组释放时，源数组会由这个数组中的元素更新

3.6 数组创建例程

1. NumPy数组创建例程

新的ndarray对象可以通过下列数组创建例程或使用低级ndarray构造函数构造。

(1) 语法: `numpy.empty`

创建指定形状和dtype的未初始化数组。该函数的用法如下:

```
numpy.empty(shape, dtype = float, order = 'C')
```

构造器接收下列参数:

- `shape`: 空数组的形状, 整数或整数元组。
- `dtype`: 所需的输出数组类型, 可选。
- `order`: 'C' 为按行的C风格数组, 'F' 为按列的Fortran风格数组。

【例3.15】

下面是一个展示空数组的例子：

```
import numpy as np
x = np.empty([3,2], dtype = int)
print x
```

输出如下：

```
[[22649312  1701344351]
 [1818321759  1885959276]
 [16779776   156368896]]
```

注意：数组元素为随机值，因为它们未初始化。

(2) 语法：numpy.zeros

返回特定大小，以0填充的新数组。该函数的用法如下：

```
numpy.zeros(shape, dtype = float, order = 'C')
```

构造器接收下列参数：

- shape：空数组的形状，整数或整数元组。
- dtype：所需的输出数组类型，可选。
- order：'C' 为按行的C风格数组，'F' 为按列的Fortran风格数组。

【例3.16】

```
# 含有 5 个 0 的数组，默认类型为 float
import numpy as np
x = np.zeros(5)
print x
```

输出如下：

```
[ 0.  0.  0.  0.  0.]
```

(3) 语法：numpy.ones

返回特定大小，以1填充的新数组。该函数的用法如下：

```
numpy.ones(shape, dtype = None, order = 'C')
```

构造器接收下列参数：

- shape: 空数组的形状，整数或整数元组。
- dtype: 所需的输出数组类型，可选。
- order: 'C' 为按行的C风格数组，'F' 为按列的Fortran风格数组。

【例3.17】

```
# 含有 5 个 1 的数组，默认类型为 float
import numpy as np
x = np.ones(5)  print x
```

输出如下：

```
[ 1.  1.  1.  1.  1.]
```

2. NumPy现有数据数组

(1) 语法: `numpy.asarray`

该函数类似于`numpy.array`，除了它有较少的参数外。这个例程对于将Python序列转换为`ndarray`非常有用。该函数的用法如下：

```
numpy.asarray(a, dtype = None, order = None)
```

构造器接收下列参数：

- a: 任意形式的输入参数，比如列表、列表的元组、元组、元组的元组、元组的列表。
- dtype: 通常输入数据的类型会应用到返回的`ndarray`。

- order: 'C' 为按行的C风格数组, 'F' 为按列的Fortran风格数组。

【例3.18】

```
# 将列表转换为 ndarray
import numpy as np

x = [1,2,3]
a = np.asarray(x)
print a
```

输出如下:

```
[1  2  3]
```

【例3.19】

```
# 设置了 dtype
import numpy as np

x = [1,2,3]
a = np.asarray(x, dtype = float)
print a
```

输出如下:

```
[ 1.  2.  3.]
```

(2) 语法: numpy.frombuffer

此函数将缓冲区解释为一维数组。暴露缓冲区接口的任何对象都用作参数来返回ndarray。该函数的用法如下:

numpy.frombuffer(buffer, dtype=float, count=-1, offset=0)

构造器接收下列参数:

- buffer: 任何暴露缓冲区接口的对象。
- dtype: 返回数组的数据类型, 默认为float。

- count: 需要读取的数据数量, 默认为-1, 读取所有数据。
- offset: 需要读取的起始位置, 默认为0。

【例3.20】

frombuffer函数的用法:

```
import numpy as np
s = 'Hello World'
a = np.frombuffer(s, dtype = 'S1')
print a
```

输出如下:

```
['H' 'e' 'l' 'l' 'o' ' ' 'W' 'o' 'r' 'l' 'd']
```

(3) 语法: numpy.fromiter

此函数从任何可迭代对象构建一个ndarray对象, 返回一个新的一维数组。该函数的用法如下:

numpy.fromiter(iterable, dtype, count=-1)

构造器接收下列参数:

- iterable: 任何可迭代对象。
- dtype: 返回数组的数据类型。
- count: 需要读取的数据数量, 默认为-1, 读取所有数据。

【例3.21】

使用内置的range()函数返回列表对象, 此列表的迭代器用于形成ndarray对象。

```
# 使用 range 函数创建列表对象
import numpy as np
list = range(5)
print list
```

输出如下：

```
[0, 1, 2, 3, 4]
```

3. NumPy数值范围数组

(1) 语法：numpy.arange

该函数从数值范围创建数组，返回ndarray对象，包含给定范围内的等间隔值。该函数的用法如下：

```
numpy.arange(start, stop, step, dtype)
```

构造器接收下列参数：

- start: 范围的起始值，默认为0。
- stop: 范围的终止值（不包含）。
- step: 两个值的间隔，默认为1。
- dtype: 返回ndarray的数据类型，如果没有提供，就会使用输入数据的类型。

【例3.22】

```
import numpy as np
x = np.arange(5)
print x
```

输出如下：

```
[0 1 2 3 4]
```

(2) 语法：numpy.linspace

该函数类似于arange()函数。在此函数中，指定了范围之间的均匀间隔数量，而不是步长。该函数的用法如下：

```
numpy.linspace(start, stop, num, endpoint, retstep,
dtype)
```

构造器接收下列参数：

- start: 序列的起始值。
- stop: 序列的终止值，如果endpoint为true，该值就包含于序列中。
- num: 要生成的等间隔样例数量，默认为50。
- endpoint: 序列中是否包含stop值，默认为ture。
- retstep: 如果为true，就返回样例以及连续数字之间的步长。
- dtype: 输出ndarray的数据类型。

【例3.23】

```
import numpy as np
x = np.linspace(10,20,5)
print x
```

输出如下：

```
[10.  12.5  15.  17.5  20.]
```

(3) 语法：numpy.logspace

该函数返回一个ndarray对象，其中包含在对数刻度上均匀分布的数字。刻度的开始和结束端点是某个底数的幂，通常为10。该函数的用法如下：

```
numpy.logscale(start, stop, num, endpoint, base, dtype)
```

logspace函数的输出由以下参数决定：

- start: 起始值是base ** start。
- stop: 终止值是base ** stop。

- num: 范围内的数值数量, 默认为50。
- endpoint: 如果为true, 终止值就包含在输出数组中。
- base: 对数空间的底数, 默认为10。
- dtype: 输出数组的数据类型, 如果没有提供, 就取决于其他参数。

【例3.24】

```
import numpy as np
# 默认底数是 10
a = np.logspace(1.0, 2.0, num = 10)
print a
```

输出如下:

```
[ 10.          12.91549665   16.68100537   21.5443469   27.82559402
 35.93813664   46.41588834   59.94842503   77.42636827  100.        ]
```

3.7 切片和索引

ndarray对象的内容可以通过索引或切片来访问和修改, 就像Python的内置容器对象一样。如前所述, ndarray对象中的元素遵循基于零的索引。有3种可用的索引方法类型: 字段访问、基本切片和高级索引。

1. 切片

基本切片是Python中基本切片概念到n维的扩展。通过将start、stop和step参数提供给内置的slice函数来构造一个Python slice对象。此slice对象被传递给数组来提取数组的一部分。

【例3.25】


```
import numpy as np
a = np.arange(10)
s = slice(2,7,2)
print a[s]
```

输出如下：

```
[2  4  6]
```

在【例3.25】中，ndarray对象由arange()函数创建。然后，分别用起始值、终止值和步长值2、7和2定义切片对象。当这个切片对象传递给ndarray时，会对它的一部分进行切片，从索引2到7，步长为2。通过将冒号分隔的切片参数（start：stop：step）直接提供给ndarray对象也可以获得相同的结果。

【例3.26】

```
import numpy as np
a = np.arange(10)
b = a[2:7:2]
print b
```

输出如下：

```
[2  4  6]
```

如果只输入一个参数，那么将返回与索引对应的单个项目。如果使用a:，那么从该索引向后的所有项目将被提取。如果使用两个参数（以:分隔），那么对两个索引（不包括停止索引）之间的元素以默认步骤进行切片。

【例3.27】

```
import numpy as np
a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print a
# 对始于索引的元素进行切片
print '现在我们从索引 a[1:] 开始对数组切片'
print a[1:]
```

输出如下：

```
[[1 2 3]
 [3 4 5]
 [4 5 6]]
现在我们从索引 a[1:] 开始对数组切片
[[3 4 5]
 [4 5 6]]
```

切片还可以包括省略号（...），来使选择元组的长度与数组的维度相同。如果在行位置使用省略号，就会返回包含行中元素的 ndarray。

2. NumPy高级索引

如果一个 ndarray 是非元组序列，数据类型为整数或布尔值的 ndarray，或者至少一个元素为序列对象的元组，我们就能够用它来索引 ndarray。高级索引始终返回数据的副本。与此相反，切片只提供了一个视图。有两种类型的高级索引：整数和布尔值。

（1）整数索引

这种机制有助于基于N维索引来获取数组中的任意元素。每个整数数组表示该维度的下标值。当索引的元素个数就是目标 ndarray 的维度时，会变得相当直接。

以下示例获取了 ndarray 对象中每一行指定列的一个元素。因此，行索引包含所有行号，列索引指定要选择的元素。

【例3.28】

```
import numpy as np

x = np.array([[1, 2], [3, 4], [5, 6]])
y = x[[0,1,2], [0,1,0]]
print y
```

输出如下：

```
[1  4  5]
```

该结果包括数组中(0, 0)、(1, 1)和(2, 0)位置处的元素。

下面的示例获取了4×3数组中的每个角处的元素。行索引是[0, 0]和[3, 3]，而列索引是[0, 2]和[0, 2]。

【例3.29】

```
import numpy as np
x = np.array([[ 0,  1,  2],[ 3,  4,  5],[ 6,  7,  8],[
9, 10, 11]])
print '我们的数组是：'
print x
print '\n'
rows = np.array([[0,0],[3,3]])
cols = np.array([[0,2],[0,2]])
y = x[rows,cols]
print '这个数组的每个角处的元素是：'
print y
```

输出如下：

```
我们的数组是：
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

这个数组的每个角处的元素是：
[[ 0  2]
 [ 9 11]]
```

返回的结果是包含每个角元素的ndarray对象。

高级和基本索引可以通过使用切片（:）或省略号（...）与索引数组组合。以下示例使用slice作为列索引和高级索引。当切片用于两

者时，结果是相同的。但高级索引会导致复制，并且可能有不同的内存布局。

（2）布尔值索引

当结果对象是布尔运算（例如比较运算符）的结果时，将使用此类型的高级索引。

【例3.30】

这个例子中，大于5的元素会作为布尔索引的结果返回。

```
import numpy as np
x = np.array([[ 0,  1,  2],[ 3,  4,  5],[ 6,  7,  8],[
9, 10, 11]])
print '我们的数组是：'
print x
print '\n'
# 现在我们会打印出大于 5 的元素
print '大于 5 的元素是：'
print x[x > 5]
```

输出如下：

```
我们的数组是：
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

大于 5 的元素是：
[ 6  7  8  9 10 11]
```

3.8 广播

广播是指NumPy在算术运算期间处理不同形状的数组的能力。对数组的算术运算通常在相应的元素上进行。如果两个阵列具有完全相同的形状，这些操作就会被无缝执行。

【例3.31】

```
import numpy as np
a = np.array([1,2,3,4])
b = np.array([10,20,30,40])
c = a * b
print c
```

输出如下：

```
[10  40  90 160]
```

如果两个数组的维数不相同，元素到元素的操作就是不可能的。然而，在NumPy中仍然可以对形状不相似的数组进行操作，因为它拥有广播功能。较小的数组会广播到较大数组的大小，以便使它们的形状可兼容。

如果满足以下规则，就可以进行广播：

- ndim较小的数组会在前面追加一个长度为1的维度。
- 输出数组的每个维度的大小是输入数组该维度大小的最大值。
- 若输入数组的某个维度的长度为1时，则沿着此维度运算时都用此维度上的第一组值。
- 若输入的某个维度大小为1，则该维度中的第一个数据元素将用于该维度的所有计算。

如果上述规则产生有效结果，并且满足以下条件之一，那么数组被称为可广播的。

- 数组拥有相同形状。
- 数组拥有相同的维数，每个维度拥有相同长度，或者长度为1。
- 数组拥有极少的维度，可以在其前面追加长度为1的维度，使上述条件成立。

【例3.32】

```
import numpy as np
a = np.array([[0.0,0.0,0.0],[10.0,10.0,10.0],
[20.0,20.0,20.0],[30.0,30.0,
30.0]])
b = np.array([1.0,2.0,3.0])
print '第一个数组: '
print a
print '\n'
print '第二个数组: '
print b
print '\n'
print '第一个数组加第二个数组: '
print a + b
```

输出如下:

```
第一个数组:
[[ 0.  0.  0.]
 [ 10. 10. 10.]
 [ 20. 20. 20.]
 [ 30. 30. 30.]]

第二个数组:
[ 1.  2.  3.]

第一个数组加第二个数组:
[[ 1.  2.  3.]
 [ 11. 12. 13.]
 [ 21. 22. 23.]
 [ 31. 32. 33.]]
```

3.9 数组与元素操作

3.9.1 数值迭代

NumPy包含一个迭代器对象`numpy.nditer`。它是一个有效的多维迭代器对象，可以用于在数组上进行迭代。数组的每个元素可使用Python的标准Iterator接口来访问。

【例3.33】使用arange()函数创建一个 3×4 数组，并使用nditer对它进行迭代。

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
print '原始数组是: '
print a
print '\n'
print '修改后的数组是: '
for x in np.nditer(a):
    print x,
Python
```

输出如下:

```
原始数组是:
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]

修改后的数组是:
0 5 10 15 20 25 30 35 40 45 50 55
```

【例3.34】迭代的顺序匹配数组的内容布局，而不考虑特定的排序。这可以通过迭代上述数组的转置来看到。

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
print '原始数组是: '
print a
print '\n'
print '原始数组的转置是: '
b = a.T
print b
print '\n'
print '修改后的数组是: '
for x in np.nditer(b):
    print x,
Python
```

输出如下:

原始数组是：

```
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]
```

原始数组的转置是：

```
[[ 0 20 40]
 [ 5 25 45]
 [10 30 50]
 [15 35 55]]
```

修改后的数组是：

```
0 5 10 15 20 25 30 35 40 45 50 55
```

【例3.35】 可以通过显式提醒来强制nditer对象使用某种顺序。

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
print '原始数组是：'
print a
print '\n'
print '以 C 风格顺序排序：'
for x in np.nditer(a, order = 'C'):
    print x,
print '\n'
print '以 F 风格顺序排序：'
for x in np.nditer(a, order = 'F'):
    print x,
Python
```

输出如下：

原始数组是：

```
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]
```

以 C 风格顺序排序：

```
0 5 10 15 20 25 30 35 40 45 50 55
```

以 F 风格顺序排序：

```
0 20 40 5 25 45 10 30 50 15 35 55
```


【例3.36】nditer对象有另一个可选参数op_flags，其默认值为只读，也可以设置为读写或只写模式。这将允许使用此迭代器修改数组元素。

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
print '原始数组是: '
print a
print '\n'
for x in np.nditer(a, op_flags=['readwrite']):
    x[...] = 2*x
print '修改后的数组是: '
print a
```

输出如下：

```
原始数组是:
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]

修改后的数组是:
[[ 0 10 20 30]
 [ 40 50 60 70]
 [ 80 90 100 110]]
```

nditer类的构造器拥有flags参数，它可以接收下列值：

- c_index：可以跟踪C顺序的索引。
- f_index：可以跟踪Fortran顺序的索引。
- multi-index：每次迭代可以跟踪一种索引类型。
- external_loop：给出的值是具有多个值的一维数组，而不是零维数组。

【例3.37】迭代器遍历对应于每列的一维数组。

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
```

```

print '原始数组是: '
print a
print '\n'
print '修改后的数组是: '
for x in np.nditer(a, flags = ['external_loop'], order =
'F'):
    print x,

```

输出如下:

```

原始数组是:
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]

修改后的数组是:
[ 0 20 40] [ 5 25 45] [10 30 50] [15 35 55]

```

【例3.38】如果两个数组是可广播的，nditer组合对象就能够同时迭代它们。假设数组a具有维度 3×4 ，并且存在维度为 1×4 的另一个数组b，则使用以下类型的迭代器（数组b被广播到a的大小）。

```

import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
print '第一个数组: '
print a
print '\n'
print '第二个数组: '
b = np.array([1, 2, 3, 4], dtype = int)
print b
print '\n'
print '修改后的数组是: '
for x,y in np.nditer([a,b]):
    print "%d:%d" % (x,y),

```

输出如下:

```

第一个数组:
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]

```

第二个数组:

```
[1 2 3 4]
```

修改后的数组是:

```
0:1 5:2 10:3 15:4 20:1 25:2 30:3 35:4 40:1 45:2 50:3 55:4
```

3.9.2 数组变形

NumPy包中有几个例程用于处理ndarray对象中的元素。它们可以分为以下类型:

- reshape: 在不改变数据的条件下修改形状。
- flat: 数组上的一维迭代器。
- flatten: 返回折叠为一维的数组副本。
- ravel: 返回连续的展开数组。

(1) 语法: `numpy.reshape`

这个函数在不改变数据的条件下修改形状, 它接收如下参数:

```
numpy.reshape(arr, newshape, order')
```

其中:

- arr: 要修改形状的数组。
- newshape: 整数或者整数数组, 新的形状应当兼容原有形状。
- order: 'C' 为C风格顺序, 'F' 为F风格顺序, 'A' 为保留原顺序。

【例3.39】

```
import numpy as np
a = np.arange(8)
print '原始数组: '
print a
print '\n'
```

```
b = a.reshape(4,2)
print '修改后的数组: '
print b
Python
```

输出如下:

```
原始数组:
[0 1 2 3 4 5 6 7]

修改后的数组:
[[0 1]
 [2 3]
 [4 5]
 [6 7]]
```

(2) 语法: `numpy.ndarray.flat`

该函数返回数组上的一维迭代器, 行为类似Python内建的迭代器。

【例3.40】

```
import numpy as np
a = np.arange(8).reshape(2,4)
print '原始数组: '
print a
print '\n'

print '调用 flat 函数之后: '
# 返回展开数组中的下标的对应元素
print a.flat[5]
Python
```

输出如下:

```
原始数组:
[[0 1 2 3]
 [4 5 6 7]]

调用 flat 函数之后:
5
```

(3) 语法: `numpy.ndarray.flatten`

该函数返回折叠为一维的数组副本, 函数接收下列参数:

`ndarray.flatten(order)`

其中:

- `order`: 'C' 为按行, 'F' 为按列, 'A' 为原顺序, 'k' 为元素在内存中出现的顺序。

【例3.41】

```
import numpy as np
a = np.arange(8).reshape(2,4)

print '原数组: '
print a
print '\n'
# default is column-major

print '展开的数组: '
print a.flatten()
print '\n'

print '以 F 风格顺序展开的数组: '
print a.flatten(order = 'F')
Python
```

输出如下:

```
原数组:
[[0 1 2 3]
 [4 5 6 7]]

展开的数组:
[0 1 2 3 4 5 6 7]

以 F 风格顺序展开的数组:
[0 4 1 5 2 6 3 7]
```

(4) 语法: `numpy.ravel`

这个函数返回展开的一维数组，并且按需生成副本。返回的数组和输入数组拥有相同数据类型。这个函数接收两个参数。

```
numpy.ravel(a, order)
```

其中：

- a：要修改形状的数组。
- order：'C' 为按行，'F' 为按列，'A' 为原顺序，'k' 为元素在内存中出现的顺序。

【例3.42】

```
import numpy as np
a = np.arange(8).reshape(2,4)

print '原数组: '
print a
print '\n'

print '调用 ravel 函数之后: '
print a.ravel()
print '\n'

print '以 F 风格顺序调用 ravel 函数之后: '
print a.ravel(order = 'F')
Python
```

输出如下：

```
原数组:
[[0 1 2 3]
 [4 5 6 7]]

调用 ravel 函数之后:
[0 1 2 3 4 5 6 7]

以 F 风格顺序调用 ravel 函数之后:
[0 4 1 5 2 6 3 7]
```

3.9.3 数组翻转

- `transpose`: 翻转数组的维度。
- `ndarray.T`: 与`self.transpose()`作用相同。
- `rollaxis`: 向后滚动指定的轴。
- `swapaxes`: 互换数组的两个轴。

(1) 语法: `numpy.transpose`

这个函数翻转给定数组的维度。如果可能的话，它会返回一个视图。函数接收下列参数：

`numpy.transpose(arr, axes)`

其中：

- `arr`: 要转置的数组。
- `axes`: 整数的列表，对应维度，通常所有维度都会翻转。

【例3.43】

```
import numpy as np
a = np.arange(12).reshape(3,4)

print '原数组: '
print a
print '\n'

print '转置数组: '
print np.transpose(a)
Python
```

输出如下：

```
原数组:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

转置数组:
[[ 0  4  8]
 [ 1  5  9]]
```

```
[ 2  6 10]
[ 3  7 11]]
```

(2) 语法: `numpy.ndarray.T`

该函数属于`ndarray`类, 行为类似于`numpy.transpose`。

【例3.44】

```
import numpy as np
a = np.arange(12).reshape(3,4)

print '原数组: '
print a
print '\n'

print '转置数组: '
print a.T
Python
```

输出如下:

```
原数组:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

转置数组:
[[ 0  4  8]
 [ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]]
```

(3) 语法: `numpy.rollaxis`

该函数向后滚动特定的轴, 直到一个特定位置。这个函数接收3个参数:

`numpy.rollaxis(arr, axis, start)`

其中:

- `arr`: 输入数组。

- axis: 要向后滚动的轴，其他轴的相对位置不会改变。
- start: 默认为零，表示完整的滚动，会滚动到特定位置。

【例3.45】

```
# 创建了三维的 ndarray
import numpy as np
a = np.arange(8).reshape(2,2,2)

print '原数组: '
print a
print '\n'
# 将轴 2 滚动到轴 0 (宽度到深度)

print '调用 rollaxis 函数: '
print np.rollaxis(a,2)
# 将轴 0 滚动到轴 1: (宽度到高度)
print '\n'

print '调用 rollaxis 函数: '
print np.rollaxis(a,2,1)
Python
```

输出如下:

```
原数组:
[[[0 1]
  [2 3]
  [4 5]
  [6 7]]]

调用 rollaxis 函数:
[[[0 2]
  [4 6]
  [1 3]
  [5 7]]]

调用 rollaxis 函数:
[[[0 2]
  [1 3]
  [4 6]
  [5 7]]]
```

(4) 语法: `numpy.swapaxes`

该函数交换数组的两个轴。对于1.10之前的NumPy版本，会返回交换后数组的视图。这个函数接收下列参数：

`numpy.swapaxes(arr, axis1, axis2)`

- `arr`: 要交换其轴的输入数组。
- `axis1`: 对应第一个轴的整数。
- `axis2`: 对应第二个轴的整数。

【例3.46】

```
# 创建了三维的 ndarray
import numpy as np
a = np.arange(8).reshape(2,2,2)

print '原数组: '
print a
print '\n'
# 现在交换轴 0 (深度方向) 到轴 2 (宽度方向)

print '调用 swapaxes 函数后的数组: '
print np.swapaxes(a, 2, 0)
Python
```

输出如下：

```
原数组:
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]

调用 swapaxes 函数后的数组:
[[[0 4]
  [2 6]]

 [[1 5]
  [3 7]]]
```

3.9.4 修改维度

- `broadcast`: 产生模仿广播的对象。
- `broadcast_to`: 将数组广播到新形状。
- `expand_dims`: 扩展数组的形状。
- `squeeze`: 从数组的形状中删除单维条目。

(1) 语法: `broadcast`

如前所述, NumPy已经内置了对广播的支持。此功能模仿广播机制。它返回一个对象, 该对象封装了将一个数组广播到另一个数组的结果。该函数使用两个数组作为输入参数:

```
numpy.broadcast(x, y)
```

下面的例子说明了它的用法。

【例3.47】

```
import numpy as np
x = np.array([[1], [2], [3]])
y = np.array([4, 5, 6])

# 对 y 广播 x
b = np.broadcast(x, y)
# 它拥有 iterator 属性, 基于自身组件的迭代器元组

print '对 y 广播 x: '
r, c = b.iters
print r.next(), c.next()
print r.next(), c.next()
print '\n'
# shape 属性返回广播对象的形状

print '广播对象的形状: '
print b.shape
print '\n'
# 手动使用 broadcast 将 x 与 y 相加
b = np.broadcast(x, y)
c = np.empty(b.shape)
```

```

print '手动使用 broadcast 将 x 与 y 相加: '
print c.shape
print '\n'
c.flat = [u + v for (u,v) in b]

print '调用 flat 函数: '
print c
print '\n'
# 获得了和 NumPy 内建的广播支持相同的结果

print 'x 与 y 的和: '
print x + y
Python

```

输出如下:

```

对 y 广播 x:
1 4
1 5

广播对象的形状:
(3, 3)

手动使用 broadcast 将 x 与 y 相加:
(3, 3)

调用 flat 函数:
[[ 5.  6.  7.]
 [ 6.  7.  8.]
 [ 7.  8.  9.]]

x 与 y 的和:
[[5 6 7]
 [6 7 8]
 [7 8 9]]

```

(2) 语法: `numpy.broadcast_to`

此函数将数组广播到新形状。它在原始数组上返回只读视图，通常不连续。如果新形状不符合NumPy的广播规则，该函数就可能会抛出 `ValueError`。该函数接收以下参数:

`numpy.broadcast_to(array, shape, subok)`

【例3.48】

```
import numpy as np
a = np.arange(4).reshape(1,4)

print '原数组: '
print a
print '\n'

print '调用 broadcast_to 函数之后: '
print np.broadcast_to(a, (4,4))
Python
```

输出如下:

```
[[0  1  2  3]
 [0  1  2  3]
 [0  1  2  3]
 [0  1  2  3]]
```

(3) 语法: `numpy.expand_dims`

函数通过在指定位置插入新的轴来扩展数组形状。该函数需要两个参数:

`numpy.expand_dims(arr, axis)`

其中:

- `arr`: 输入数组。
- `axis`: 新轴插入的位置。

【例3.49】

```
import numpy as np
x = np.array([1,2],[3,4])

print '数组 x: '
print x
print '\n'
```

```

y = np.expand_dims(x, axis = 0)

print '数组 y: '
print y
print '\n'

print '数组 x 和 y 的形状: '
print x.shape, y.shape
print '\n'
# 在位置 1 插入轴
y = np.expand_dims(x, axis = 1)

print '在位置 1 插入轴之后的数组 y: '
print y
print '\n'

print 'x.ndim 和 y.ndim: '
print x.ndim, y.ndim
print '\n'

print 'x.shape 和 y.shape: '
print x.shape, y.shape
Python

```

输出如下:

```

数组 x:
[[1 2]
 [3 4]]

数组 y:
[[[1 2]
  [3 4]]]

数组 x 和 y 的形状:
(2, 2) (1, 2, 2)

在位置 1 插入轴之后的数组 y:
[[[1 2]]
 [[3 4]]]

x.ndim 和 y.ndim:
2 3

```

```
x.shape 和 y.shape:  
(2, 2) (2, 1, 2)
```

(4) 语法: `numpy.squeeze`

函数从给定数组的形状中删除一维条目。此函数需要两个参数:

`numpy.squeeze(arr, axis)`

其中:

- `arr`: 输入数组。
- `axis`: 整数或整数元组, 用于选择形状中单一维度条目的子集。

【例3.50】

```
import numpy as np  
x = np.arange(9).reshape(1,3,3)  
  
print '数组 x: '  
print x  
print '\n'  
y = np.squeeze(x)  
  
print '数组 y: '  
print y  
print '\n'  
  
print '数组 x 和 y 的形状: '  
print x.shape, y.shape  
Python
```

输出如下:

```
数组 x:  
[[[0 1 2]  
  [3 4 5]  
  [6 7 8]]]  
  
数组 y:  
[[0 1 2]  
 [3 4 5]]
```

```
[6 7 8]]
```

数组 x 和 y 的形状:

```
(1, 3, 3) (3, 3)
```

3.9.5 数组连接

- `concatenate`: 沿着现存的轴连接数据序列。
- `stack`: 沿着新轴连接数组序列。
- `hstack`: 水平堆叠序列中的数组（列方向）。
- `vstack`: 竖直堆叠序列中的数组（行方向）。

(1) 语法: `numpy.concatenate`

连接沿现有轴的数组序列。此函数用于沿指定轴连接相同形状的两个或多个数组。该函数接收以下参数:

```
numpy.concatenate((a1, a2, ...), axis)
```

其中:

- `a1, a2, ...`: 相同类型的数组序列。
- `axis`: 沿着它连接数组的轴, 默认为0。

【例3.51】

```
import numpy as np
a = np.array([[1,2],[3,4]])

print '第一个数组: '
print a
print '\n'
b = np.array([[5,6],[7,8]])

print '第二个数组: '
print b
print '\n'
# 两个数组的维度相同

print '沿轴 0 连接两个数组: '
```



```
print np.concatenate((a,b))
print '\n'

print '沿轴 1 连接两个数组：'
print np.concatenate((a,b),axis = 1)
Python
```

输出如下：

```
第一个数组：
[[1 2]
 [3 4]]

第二个数组：
[[5 6]
 [7 8]]

沿轴 0 连接两个数组：
[[1 2]
 [3 4]
 [5 6]
 [7 8]]

沿轴 1 连接两个数组：
[[1 2 5 6]
 [3 4 7 8]]
```

(2) 语法：numpy.stack

此函数沿新轴连接数组序列。此功能添加自NumPy版本1.10.0。需要提供以下参数：

`numpy.stack(arrays, axis)`

其中：

- `arrays`：相同形状的数组序列。
- `axis`：返回数组中的轴，输入数组沿着它来堆叠。

【例3.52】

```
import numpy as np
a = np.array([[1,2],[3,4]])
```

```

print '第一个数组: '
print a
print '\n'
b = np.array([[5,6],[7,8]])

print '第二个数组: '
print b
print '\n'

print '沿轴 0 堆叠两个数组: '
print np.stack((a,b),0)
print '\n'

print '沿轴 1 堆叠两个数组: '
print np.stack((a,b),1)
Python

```

输出如下:

```

第一个数组:
[[1 2]
 [3 4]]

第二个数组:
[[5 6]
 [7 8]]

沿轴 0 堆叠两个数组:
[[[1 2]
  [3 4]]
 [[5 6]
  [7 8]]]

沿轴 1 堆叠两个数组:
[[[1 2]
  [5 6]]
 [[3 4]
  [7 8]]]

```

(3) 语法: `numpy.hstack`

`numpy.stack`函数的变体, 通过堆叠来生成水平的单个数组。

【例3.53】

```

import numpy as np
a = np.array([[1,2],[3,4]])

print '第一个数组: '
print a
print '\n'
b = np.array([[5,6],[7,8]])

print '第二个数组: '
print b
print '\n'

print '水平堆叠: '
c = np.hstack((a,b))
print c
print '\n'
Python
输出如下:
第一个数组:
[[1 2]
 [3 4]]

第二个数组:
[[5 6]
 [7 8]]

水平堆叠:
[[1 2 5 6]
 [3 4 7 8]]

```

(4) 语法: `numpy.vstack`

`numpy.stack`函数的变体，通过堆叠来生成竖直的单个数组。

【例3.54】

```

import numpy as np
a = np.array([[1,2],[3,4]])

print '第一个数组: '
print a
print '\n'
b = np.array([[5,6],[7,8]])

print '第二个数组: '

```

```
print b
print '\n'

print '竖直堆叠: '
c = np.vstack((a,b))
print c
Python
```

输出如下:

第一个数组:

```
[[1 2]
 [3 4]]
```

第二个数组:

```
[[5 6]
 [7 8]]
```

竖直堆叠:

```
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
```

3.9.6 数组分割

- `split`: 将一个数组分割为多个子数组。
- `hsplit`: 将一个数组水平分割为多个子数组（按列）。
- `vsplit`: 将一个数组竖直分割为多个子数组（按行）。

(1) 语法: `numpy.split`

该函数沿特定的轴将数组分割为子数组。该函数接收3个参数:

`numpy.split(ary, indices_or_sections, axis)`

其中:

- `ary`: 被分割的输入数组。

- `indices_or_sections`: 可以是整数，表明要从输入数组创建的等大小的子数组的数量。若此参数是一维数组，则其元素表明要创建的新子数组的点。
- `axis`: 默认为0。

【例3.55】

```
import numpy as np
a = np.arange(9)

print '第一个数组: '
print a
print '\n'

print '将数组分为三个大小相等的子数组: '
b = np.split(a,3)
print b
print '\n'

print '将数组在一维数组中表明的位置分割: '
b = np.split(a,[4,7])
print b
Python
```

输出如下:

```
第一个数组:
[0 1 2 3 4 5 6 7 8]

将数组分为三个大小相等的子数组:
[array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8])]

将数组在一维数组中表明的位置分割:
[array([0, 1, 2, 3]), array([4, 5, 6]), array([7, 8])]
```

(2) 语法: `numpy.hsplit`

`numpy.hsplit`是`split()`函数的特例，其中轴为1表示水平分割，无论输入数组的维度是什么。

【例3.56】

```

import numpy as np
a = np.arange(16).reshape(4,4)

print '第一个数组: '
print a
print '\n'

print '水平分割: '
b = np.hsplit(a,2)
print b
print '\n'
Python

```

输出:

```

第一个数组:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]

水平分割:
[array([[ 0,  1],
        [ 4,  5],
        [ 8,  9],
        [12, 13]]), array([[ 2,  3],
        [ 6,  7],
        [10, 11],
        [14, 15]])]

```

(3) 语法: `numpy.vsplit`

`numpy.vsplit`是`split()`函数的特例，其中轴为0表示竖直分割，无论输入数组的维度是什么。

【例3.57】

```

import numpy as np
a = np.arange(16).reshape(4,4)

print '第一个数组: '
print a
print '\n'

```

```
print '竖直分割: '  
b = np.vsplit(a,2)  
print b  
Python
```

输出如下:

```
第一个数组:  
[[ 0 1 2 3]  
 [ 4 5 6 7]  
 [ 8 9 10 11]  
 [12 13 14 15]]  
  
竖直分割:  
[array([[0, 1, 2, 3],  
        [4, 5, 6, 7]]), array([[ 8,  9, 10, 11],  
        [12, 13, 14, 15]])]
```

3.9.7 添加 / 删除元素

- `resize`: 返回指定形状的新数组。
- `append`: 将值添加到数组末尾。
- `insert`: 沿指定轴将值插入指定下标之前。
- `delete`: 返回删掉某个轴的子数组的新数组。
- `unique`: 寻找数组内的唯一元素。

(1) 语法: `numpy.resize`

该函数返回指定大小的新数组。如果新大小大于原始大小,就包含原始数组中的元素的重复副本。该函数接收以下参数:

```
numpy.resize(arr, shape)
```

其中:

- `arr`: 要修改大小的输入数组。
- `shape`: 返回数组的新形状。

【例3.58】

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])

print '第一个数组: '
print a
print '\n'

print '第一个数组的形状: '
print a.shape
print '\n'
b = np.resize(a, (3,2))

print '第二个数组: '
print b
print '\n'

print '第二个数组的形状: '
print b.shape
print '\n'
# 要注意 a 的第一行在 b 中重复出现, 因为尺寸变大了

print '修改第二个数组的大小: '
b = np.resize(a, (3,3))
print b
Python
```

输出如下:

```
第一个数组:
[[1 2 3]
 [4 5 6]]

第一个数组的形状:
(2, 3)

第二个数组:
[[1 2]
 [3 4]
 [5 6]]

第二个数组的形状:
(3, 2)
```


修改第二个数组的大小：

```
[[1 2 3]
 [4 5 6]
 [1 2 3]]
```

(2) 语法：numpy.append

该函数在输入数组的末尾添加值。附加操作不是原地的，而是分配新的数组。此外，输入数组的维度必须匹配否则将生成ValueError。函数接收下列函数：

```
numpy.append(arr, values, axis)
```

其中：

- arr：输入数组。
- values：要向arr添加的值，比如和arr形状相同（除了要添加的轴外）。
- axis：沿着它完成操作的轴。如果没有提供，那么两个参数都会被展开。

【例3.59】

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])

print '第一个数组： '
print a
print '\n'

print '向数组添加元素： '
print np.append(a, [7,8,9])
print '\n'

print '沿轴 0 添加元素： '
print np.append(a, [[7,8,9]],axis = 0)
print '\n'

print '沿轴 1 添加元素： '
print np.append(a, [[5,5,5],[7,8,9]],axis = 1)
Python
```

输出如下：

第一个数组：

```
[[1 2 3]
 [4 5 6]]
```

向数组添加元素：

```
[1 2 3 4 5 6 7 8 9]
```

沿轴 0 添加元素：

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

沿轴 1 添加元素：

```
[[1 2 3 5 5 5]
 [4 5 6 7 8 9]]
```

(3) 语法：numpy.insert

该函数在给定索引之前，沿给定轴在输入数组中插入值。若值的类型转换为要插入类型，则它与输入数组不同。插入没有原地的，函数会返回一个新数组。此外，若未提供轴，则输入数组会被展开。该函数接收以下参数：

```
numpy.insert(arr, obj, values, axis)
```

其中：

- arr：输入数组。
- obj：在其之前插入值的索引。
- values：要插入的值。
- axis：沿着它插入的轴，如果未提供，输入数组就会被展开。

(4) 语法：numpy.delete

该函数返回从输入数组中删除指定子数组的新数组。与insert()函数的情况一样，若未提供轴参数，则输入数组将展开。该函数接收

以下参数：

`NumPy.delete(arr, obj, axis)`

其中：

- `arr`：输入数组。
- `obj`：可以被切片，整数或者整数数组，表明要从输入数组删除的子数组。
- `axis`：沿着它删除给定子数组的轴，若未提供，则输入数组会被展开。

【例3.60】

```
import numpy as np
a = np.arange(12).reshape(3,4)

print '第一个数组： '
print a
print '\n'

print '未传递 Axis 参数。 在插入之前输入数组会被展开。 '
print np.delete(a,5)
print '\n'

print '删除第二列： '
print np.delete(a,1,axis = 1)
print '\n'

print '包含从数组中删除的替代值的切片： '
a = np.array([1,2,3,4,5,6,7,8,9,10])
print np.delete(a, np.s_[::2])
Python
```

输出如下：

```
第一个数组：
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

未传递 `Axis` 参数。 在插入之前输入数组会被展开。

```
[ 0  1  2  3  4  6  7  8  9 10 11]
```

删除第二列:

```
[[ 0  2  3]
 [ 4  6  7]
 [ 8 10 11]]
```

包含从数组中删除的替代值的切片:

```
[ 2  4  6  8 10]
```

(5) 语法: `numpy.unique`

该函数返回输入数组中的去重元素数组。该函数能够返回一个元组，包含去重数组和相关索引的数组。索引的性质取决于函数调用中返回参数的类型。该函数接收以下参数:

```
numpy.unique(arr,          return_index,          return_inverse,
return_counts)
```

其中:

- `arr`: 输入数组。如果不是一维数组，就会展开。
- `return_index`: 如果为`true`，就返回输入数组中的元素下标。
- `return_inverse`: 如果为`true`，就返回去重数组的下标，可以用于重构输入数组。
- `return_counts`: 如果为`true`，就返回去重数组中的元素在原数组中出现的次数。

【例3.61】

```
import numpy as np
a = np.array([5,2,6,2,7,5,6,8,2,9])

print '第一个数组: '
print a
print '\n'

print '第一个数组的去重值: '
```

```

u = np.unique(a)
print u
print '\n'

print '去重数组的索引数组: '
u,indices = np.unique(a, return_index = True)
print indices
print '\n'

print '我们可以看到每个和原数组下标对应的数值: '
print a
print '\n'

print '去重数组的下标: '
u,indices = np.unique(a,return_inverse = True)
print u
print '\n'

print '下标为: '
print indices
print '\n'

print '使用下标重构原数组: '
print u[indices]
print '\n'

print '返回去重元素的重复数量: '
u,indices = np.unique(a,return_counts = True)
print u
print indices
Python

```

输出如下:

```

第一个数组:
[5 2 6 2 7 5 6 8 2 9]

第一个数组的去重值:
[2 5 6 7 8 9]

去重数组的索引数组:
[1 0 2 4 7 9]

我们可以看到每个和原数组下标对应的数值:
[5 2 6 2 7 5 6 8 2 9]

```

去重数组的下标:

```
[2 5 6 7 8 9]
```

下标为:

```
[1 0 2 0 3 1 2 4 0 5]
```

使用下标重构原数组:

```
[5 2 6 2 7 5 6 8 2 9]
```

返回唯一元素的重复数量:

```
[2 5 6 7 8 9]  
[3 2 2 1 1 1]
```

3.10 位操作与字符串函数

1. 位操作

下面是NumPy包中可用的位操作函数。

- `bitwise_and`: 对数组元素执行位与操作。
- `bitwise_or`: 对数组元素执行位或操作。
- `invert`: 计算位非。
- `left_shift`: 向左移动二进制表示的位。
- `right_shift`: 向右移动二进制表示的位。

(1) `bitwise_and`

通过`np.bitwise_and()`函数，对输入数组中的整数的二进制表示的相应位执行位与运算。

【例3.62】

```
import numpy as np  
print '13 和 17 的二进制形式: '  
a,b = 13,17  
print bin(a), bin(b)  
print '\n'
```

```
print '13 和 17 的位与: '  
print np.bitwise_and(13, 17)
```

输出如下:

```
13 和 17 的二进制形式:  
0b1101 0b10001
```

```
13 和 17 的位与:  
1
```

(2) bitwise_or

通过`np.bitwise_or()`函数,对输入数组中的整数的二进制表示的相应位执行位或运算。

【例3.63】

```
import numpy as np  
a,b = 13,17  
print '13 和 17 的二进制形式: '  
print bin(a), bin(b)  
  
print '13 和 17 的位或: '  
print np.bitwise_or(13, 17)
```

输出如下:

```
13 和 17 的二进制形式:  
0b1101 0b10001
```

```
13 和 17 的位或:  
29
```

(3) invert

此函数计算输入数组中整数的位非结果。对于有符号整数返回补码。

【例3.64】

```
import numpy as np

print '13 的位反转, 其中 ndarray 的 dtype 是 uint8: '
print np.invert(np.array([13], dtype = np.uint8))
print '\n'
# 比较 13 和 242 的二进制表示, 我们发现了位的反转

print '13 的二进制表示: '
print np.binary_repr(13, width = 8)
print '\n'

print '242 的二进制表示: '
print np.binary_repr(242, width = 8)
```

输出如下:

```
13 的位反转, 其中 ndarray 的 dtype 是 uint8:
[242]

13 的二进制表示:
00001101

242 的二进制表示:
11110010
```

注: `np.binary_repr()` 函数返回给定宽度中十进制数的二进制表示。

2. 字符串函数

以下函数用于对 `dtype` 为 `numpy.string_` 或 `numpy.unicode_` 的数组执行向量化字符串操作。它们基于 Python 内置库中的标准字符串函数。

- `add()`: 返回两个 `Str` 或 `Unicode` 数组的逐个字符串连接。
- `multiply()`: 返回按元素多重连接后的字符串。
- `center()`: 返回给定字符串的副本, 其中元素位于特定字符串的中央。

- `capitalize()`: 返回给定字符串的副本，其中只有第一个字符串为大写。
- `title()`: 返回字符串或Unicode的按元素标题转换版本。
- `lower()`: 返回一个数组，其元素转换为小写。
- `upper()`: 返回一个数组，其元素转换为大写。
- `split()`: 返回字符串中的单词列表，并使用分隔符来分割。
- `splitlines()`: 返回元素中的行列表，以换行符分割。
- `strip()`: 返回数组副本，其中元素移除了开头或者结尾处的特定字符。
- `join()`: 返回一个字符串，它是序列中字符串的连接。
- `replace()`: 返回字符串的副本，其中所有子字符串出现的位置都被新字符串取代。
- `decode()`: 按元素调用`str.decode`。
- `encode()`按元素调用`str.encode`。

这些函数在字符数组类（`numpy.char`）中定义。较旧的Numarray包包含`chararray`类。`numpy.char`类中的上述函数，在执行向量化字符串操作时非常有用。

(1) `numpy.char.add()`，执行按元素的字符串连接。

【例3.65】

```
import numpy as np
print '连接两个字符串: '
print np.char.add(['hello'], [' xyz'])
print '\n'

print '连接示例: '
print np.char.add(['hello', 'hi'], [' abc', ' xyz'])
Python
```

输出如下：

连接两个字符串：

```
['hello xyz']
```

连接示例：

```
['hello abc' 'hi xyz']
```

(2) `numpy.char.multiply()`，执行多重连接。

【例3.66】

```
import numpy as np
print np.char.multiply('Hello ',3)
```

输出如下：

```
Hello Hello Hello
```

(3) `numpy.char.center()`，返回所需宽度的数组，以便输入字符串位于中心，并使用`fillchar`在左侧和右侧进行填充。

【例3.67】

```
import numpy as np
# np.char.center(arr, width,fillchar)
print np.char.center('hello', 20,fillchar = '*')
```

输出如下：

```
*****hello*****
```

(4) `numpy.char.capitalize()`，返回字符串的副本，其中第一个字母大写。

【例3.68】

```
import numpy as np
print np.char.capitalize('hello world')
```

输出如下：

```
Hello world
```

其他函数就不一一列举了。

3.11 数学运算函数

NumPy包含大量的数学运算函数。NumPy提供标准的三角函数、算术运算函数、复数处理函数等。

1. 三角函数

NumPy拥有标准的三角函数，它为弧度制单位的给定角度返回三角函数比值。

【例3.69】

```
import numpy as np
a = np.array([0,30,45,60,90])
print '不同角度的正弦值: '
# 通过乘 pi/180 转化为弧度
print np.sin(a*np.pi/180)
print '\n'
print '数组中角度的余弦值: '
print np.cos(a*np.pi/180)
print '\n'
print '数组中角度的正切值: '
print np.tan(a*np.pi/180)
Python
```

输出如下：

不同角度的正弦值:

```
[ 0.          0.5          0.70710678  0.8660254   1.          ]
```

数组中角度的余弦值:

```
[ 1.00000000e+00  8.66025404e-01  7.07106781e-01  5.00000000e-01  
 6.12323400e-17]
```

数组中角度的正切值:

```
[ 0.00000000e+00  5.77350269e-01  1.00000000e+00  1.73205081e+00  
 1.63312394e+16]
```

`arcsin`、`arccos`和`arctan`函数返回给定角度的`sin`、`cos`和`tan`的反三角函数。这些函数的结果可以通过`numpy.degrees()`函数将弧度制转换为角度制来验证。

2. 舍入函数

`numpy.around()`函数返回四舍五入到所需精度的值。该函数接收以下参数:

- `a`: 输入数组。
- `Decimals`: 要舍入的小数位数, 默认值为0。如果为负, 整数将四舍五入到小数点左侧的位置。

【例3.70】

```
import numpy as np  
a = np.array([1.0, 5.55, 123, 0.567, 25.532])  
print '原数组: '  
print a  
print '\n'  
print '舍入后: '  
print np.around(a)  
print np.around(a, decimals = 1)  
print np.around(a, decimals = -1)
```

输出如下:

```
原数组:  
[ 1.      5.55 123.      0.567 25.532]
```

```
舍入后:  
[ 1.    6.  123.    1.  26. ]  
[ 1.    5.6 123.    0.6 25.5]  
[ 0.    10. 120.    0.  30. ]
```

3. numpy.floor()

该函数返回不大于输入参数的最大整数，即标量 x 的下限是最大的整数 i ，使得 $i \leq x$ 。注意：在Python中，向下取整总是从0舍入。

【例3.71】

```
import numpy as np  
a = np.array([-1.7, 1.5, -0.2, 0.6, 10])  
print '提供的数组: '  
print a  
print '\n'  
print '修改后的数组: '  
print np.floor(a)
```

输出如下：

```
提供的数组:  
[ -1.7    1.5   -0.2    0.6   10. ]  
  
修改后的数组:  
[ -2.    1.   -1.    0.   10.]
```

4. numpy.ceil()

`ceil()`函数返回输入值的上限，即标量 x 的上限是最小的整数 i ，使得 $i \geq x$ 。

【例3.72】

```
import numpy as np
a = np.array([-1.7, 1.5, -0.2, 0.6, 10])
print '提供的数组: '
print a
print '\n'
print '修改后的数组: '
print np.ceil(a)
```

输出如下:

```
提供的数组:
[ -1.7   1.5  -0.2   0.6  10. ]

修改后的数组:
[ -1.    2.   -0.    1.   10.]
```

3.12 算术运算

用于执行算术运算的输入数组，如 `add()`、`subtract()`、`multiply()` 和 `divide()`，必须具有相同的形状或符合数组广播规则。

【例3.73】

```
import numpy as np
a = np.arange(9, dtype = np.float_).reshape(3,3)
print '第一个数组: '
print a
print '\n'
print '第二个数组: '
b = np.array([10,10,10])
print b
print '\n'
print '两个数组相加: '
print np.add(a,b)
print '\n'
print '两个数组相减: '
print np.subtract(a,b)
print '\n'
print '两个数组相乘: '
print np.multiply(a,b)
print '\n'
```

```
print '两个数组相除：'  
print np.divide(a,b)
```

输出如下：

```
第一个数组：  
[[ 0.  1.  2.]  
 [ 3.  4.  5.]  
 [ 6.  7.  8.]]  
  
第二个数组：  
[10 10 10]  
  
两个数组相加：  
[[ 10. 11. 12.]  
 [ 13. 14. 15.]  
 [ 16. 17. 18.]]  
  
两个数组相减：  
[[-10. -9. -8.]  
 [ -7. -6. -5.]  
 [ -4. -3. -2.]]  
  
两个数组相乘：  
[[ 0. 10. 20.]  
 [ 30. 40. 50.]  
 [ 60. 70. 80.]]  
  
两个数组相除：  
[[ 0. 0.1 0.2]  
 [ 0.3 0.4 0.5]  
 [ 0.6 0.7 0.8]]
```

1. numpy.reciprocal()

该函数返回参数逐元素的倒数。由于Python处理整数除法的方式，对于绝对值大于1的整数元素结果始终为0，因此对于整数0发出溢出警告。

【例3.74】

```

import numpy as np
a = np.array([0.25, 1.33, 1, 0, 100])
print '我们的数组是: '
print a
print '\n'
print '调用 reciprocal 函数: '
print np.reciprocal(a)
print '\n'
b = np.array([100], dtype = int)
print '第二个数组: '
print b
print '\n'
print '调用 reciprocal 函数: '
print np.reciprocal(b)

```

输出如下:

```

的数组是:
[ 0.25  1.33  1.    0.   100. ]

调用 reciprocal 函数:
main.py:9: RuntimeWarning: divide by zero encountered in reciprocal
  print np.reciprocal(a)
[ 4.          0.7518797  1.          inf  0.01       ]

第二个数组:
[100]

调用 reciprocal 函数:
[0]

```

2. numpy.power()

该函数将第一个输入数组中的元素作为底数，计算它与第二个输入数组中相应元素的幂。

【例3.75】

```

import numpy as np
a = np.array([10,100,1000])
print '我们的数组是: '
print a

```



```

print '\n'
print '调用 power 函数: '
print np.power(a,2)
print '\n'
print '第二个数组: '
b = np.array([1,2,3])
print b
print '\n'
print '再次调用 power 函数: '
print np.power(a,b)

```

输出如下:

```

我们的数组是:
[ 10 100 1000]

调用 power 函数:
[ 100 10000 1000000]

第二个数组:
[1 2 3]

再次调用 power 函数:
[ 10 10000 1000000000]

```

3. numpy.mod()

该函数返回输入数组中相应元素的除法余数。函数 `numpy.remainder()` 也产生相同的结果。

【例3.76】

```

import numpy as np
a = np.array([10,20,30])
b = np.array([3,5,7])
print '第一个数组: '
print a
print '\n'
print '第二个数组: '
print b
print '\n'
print '调用 mod() 函数: '

```

```
print np.mod(a,b)
print '\n'
print '调用 remainder() 函数: '
print np.remainder(a,b)
```

输出如下:

```
第一个数组:
[10 20 30]

第二个数组:
[3 5 7]

调用 mod() 函数:
[1 0 2]

调用 remainder() 函数:
[1 0 2]
```

以下函数用于对含有复数的数组执行操作。

- `numpy.real()`: 返回复数类型参数的实部。
- `numpy.imag()`: 返回复数类型参数的虚部。
- `numpy.conj()`: 返回通过改变虚部的符号而获得的共轭复数。
- `numpy.angle()`: 返回复数参数的角度。函数的参数是 `degree`。如果为 `true`, 返回的角度就以角度制来表示, 否则以弧度制来表示。

3.13 统计函数

NumPy有很多有用的统计函数, 用于从数组中给定的元素中查找最小元素、最大元素、百分位标准差和方差等。

1. `numpy.amin()` 和 `numpy.amax()`

这两个函数从给定数组的元素中沿指定轴返回最小值和最大值。

【例3.77】

```
import numpy as np
a = np.array([[3,7,5],[8,4,3],[2,4,9]])
print '我们的数组是: '
print a
print '\n'
print '调用 amin() 函数: '
print np.amin(a,1)
print '\n'
print '再次调用 amin() 函数: '
print np.amin(a,0)
print '\n'
print '调用 amax() 函数: '
print np.amax(a)
print '\n'
print '再次调用 amax() 函数: '
print np.amax(a, axis = 0)
```

输出如下:

```
我们的数组是:
[[3 7 5]
 [8 4 3]
 [2 4 9]]

调用 amin() 函数:
[3 3 2]

再次调用 amin() 函数:
[2 4 3]

调用 amax() 函数:
9

再次调用 amax() 函数:
[8 7 9]
```

2. numpy.ptp()

该函数返回沿轴的值的范围（最大值-最小值）。

【例3.78】

```
import numpy as np
a = np.array([[3,7,5],[8,4,3],[2,4,9]])
print '我们的数组是: '
print a
print '\n'
print '调用 ptp() 函数: '
print np.ptp(a)
print '\n'
print '沿轴 1 调用 ptp() 函数: '
print np.ptp(a, axis = 1)
print '\n'
print '沿轴 0 调用 ptp() 函数: '
print np.ptp(a, axis = 0)
```

输出如下：

```
我们的数组是:
[[3 7 5]
 [8 4 3]
 [2 4 9]]

调用 ptp() 函数:
7

沿轴 1 调用 ptp() 函数:
[4 5 7]

沿轴 0 调用 ptp() 函数:
[6 3 6]
```

3. numpy.percentile()

百分位数是统计中使用的度量，表示小于这个值的观察值的百分比。该函数接收以下参数：

`numpy.percentile(a, q, axis)`

- a: 输入数组。
- q: 要计算的百分位数，在0~100之间。
- axis: 沿着它计算百分位数的轴。

【例3.79】

```
import numpy as np
a = np.array([[30,40,70],[80,20,10],[50,90,60]])
print '我们的数组是: '
print a
print '\n'
print '调用 percentile() 函数: '
print np.percentile(a,50)
print '\n'
print '沿轴 1 调用 percentile() 函数: '
print np.percentile(a,50, axis = 1)
print '\n'
print '沿轴 0 调用 percentile() 函数: '
print np.percentile(a,50, axis = 0)
```

输出如下:

```
我们的数组是:
[[30 40 70]
 [80 20 10]
 [50 90 60]]

调用 percentile() 函数:
50.0

沿轴 1 调用 percentile() 函数:
[ 40. 20. 60.]

沿轴 0 调用 percentile() 函数:
[ 50. 40. 60.]
```

4. numpy.median()

中值，定义为将数据样本的上半部分与下半部分分开的值。

【例3.80】

```
import numpy as np
a = np.array([[30,65,70],[80,95,10],[50,90,60]])
print '我们的数组是: '
print a
print '\n'
print '调用 median() 函数: '
print np.median(a)
print '\n'
print '沿轴 0 调用 median() 函数: '
print np.median(a, axis = 0)
print '\n'
print '沿轴 1 调用 median() 函数: '
print np.median(a, axis = 1)
```

输出如下:

```
我们的数组是:
[[30 65 70]
 [80 95 10]
 [50 90 60]]

调用 median() 函数:
65.0

沿轴 0 调用 median() 函数:
[ 50. 90. 60.]

沿轴 1 调用 median() 函数:
[ 65. 80. 60.]
```

5. numpy.mean()

算术平均值是沿轴的元素总和除以元素的数量。numpy.mean() 函数返回数组中元素的算术平均值。如果提供了轴，就沿其轴进行计算。

6. numpy.average()

加权平均值是由每个分量乘以反映其重要性的因子得到的平均值。numpy.average()函数根据在另一个数组中给出的各自的权重，计算数组中元素的加权平均值。该函数可以接收一个轴参数。如果没有指定轴，数组就会被展开。

考虑数组[1, 2, 3, 4]和相应的权重[4, 3, 2, 1]，通过将相应元素的乘积相加，并将和除以权重的和来计算加权平均值。

$$\text{加权平均值} = (1*4 + 2*3 + 3*2 + 4*1) / (4 + 3 + 2 + 1)$$

7. 标准差

标准差是与均值的偏差的平方的平均值的平方根。标准差公式如下：

$$\text{std} = \sqrt{\text{mean}((x - x.\text{mean}())**2)} \quad (3.1)$$

若数组是[1, 2, 3, 4]，则其平均值为2.5。因此，差的平方是[2.25, 0.25, 0.25, 2.25]，其平均值的平方根除以4，即sqrt(5/4)是1.1180339887498949。

8. 方差

方差是偏差的平方的平均值，即mean((x - x.mean())** 2)。换句话说，标准差是方差的平方根。

3.14 排序、搜索和计数函数

NumPy中提供了各种排序相关功能。这些排序函数实现不同的排序算法，每个排序算法的特征在于执行速度、最坏情况的性能、所需的工作空间和算法的稳定性。表3.3显示了3种排序算法的比较。

表3.3 三种排序算法的比较

种类	速度	最坏情况	工作空间	稳定性
'quicksort'(快速排序)	1	$O(n^2)$	0	否
'mergesort'(归并排序)	2	$O(n \log(n))$	$\sim n/2$	是
'heapsort'(堆排序)	3	$O(n \log(n))$		否

1. numpy.sort()

sort() 函数返回输入数组的排序副本。它有以下参数：

numpy.sort(a, axis, kind, order)

- a: 要排序的数组。
- axis: 要排序的数组的轴，如果没有数组就会被展开，沿着最后的轴排序。
- kind: 默认为'quicksort'（快速排序）。
- order: 如果数组包含字段，就是要排序的字段。

【例3.81】

```
import numpy as np
a = np.array([[3,7],[9,1]])
print '我们的数组是：'
print a
print '\n'
print '调用 sort() 函数：'
print np.sort(a)
print '\n'
print '沿轴 0 排序：'
print np.sort(a, axis = 0)
print '\n'
# 在 sort 函数中排序字段
dt = np.dtype([('name', 'S10'),('age', int)])
a = np.array [("raju",21),("anil",25),("ravi", 17),
("amar",27)], dtype =
dt)
print '我们的数组是：'
print a
```



```
print '\n'
print '按 name 排序: '
print np.sort(a, order = 'name')
```

输出如下:

我们的数组是:

```
[[3 7]
 [9 1]]
```

调用 `sort()` 函数:

```
[[3 7]
 [1 9]]
```

沿轴 0 排序:

```
[[3 1]
 [9 7]]
```

我们的数组是:

```
[('raju', 21) ('anil', 25) ('ravi', 17) ('amar', 27)]
```

按 name 排序:

```
[('amar', 27) ('anil', 25) ('raju', 21) ('ravi', 17)]
```

2. `numpy.argsort()`

该函数对输入数组沿给定轴执行间接排序, 并使用指定排序类型返回数据的索引数组。这个索引数组用于构造排序后的数组。

【例3.82】

```
import numpy as np
x = np.array([3, 1, 2])
print '我们的数组是: '
print x
print '\n'
print '对 x 调用 argsort() 函数: '
y = np.argsort(x)
print y
print '\n'
print '以排序后的顺序重构原数组: '
print x[y]
```

```
print '\n'
print '使用循环重构原数组: '
for i in y:
    print x[i],
```

输出如下:

我们的数组是:

```
[3 1 2]
```

对 x 调用 argsort() 函数:

```
[1 2 0]
```

以排序后的顺序重构原数组:

```
[1 2 3]
```

使用循环重构原数组:

```
1 2 3
```

3. numpy.lexsort()

该函数使用键序列执行间接排序。键可以看作电子表格中的一列。该函数返回一个索引数组，使用它可以获得排序数据。注意，最后一个键恰好是sort的主键。

【例3.83】

```
import numpy as np

nm = ('raju','anil','ravi','amar')
dv = ('f.y.', 's.y.', 's.y.', 'f.y.')
ind = np.lexsort((dv,nm))
print '调用 lexsort() 函数: '
print ind
print '\n'
print '使用这个索引来获取排序后的数据: '
print [nm[i] + ", " + dv[i] for i in ind]
```

输出如下:

调用 `lexsort()` 函数：

```
[3 1 0 2]
```

使用这个索引来获取排序后的数据：

```
['amar, f.y.', 'anil, s.y.', 'raju, f.y.', 'ravi, s.y.']
```

4. NumPy

NumPy模块有一些用于在数组内搜索的函数，它提供了用于找到最大值、最小值以及满足给定条件的元素的函数，如`numpy.argmax()`和`numpy.argmin()`函数。这两个函数分别沿给定轴返回最大元素和最小元素的索引。

5. 其他函数

- `numpy.nonzero()`：该函数返回输入数组中非零元素的索引。
- `numpy.where()`：该函数返回输入数组中满足给定条件的元素的索引。
- `numpy.extract()`：该函数返回满足任何条件的元素。

3.15 字节交换

存储在计算机内存中的数据取决于CPU使用的架构。它可以是小端（最小有效位存储在最小地址中）或大端（最小有效字节存储在最大地址中）。

`numpy.ndarray.byteswap()` 函数在两个表示：大端和小端之间切换。

【例3.84】

```
import numpy as np
a = np.array([1, 256, 8755], dtype = np.int16)
```

```
print '我们的数组是：'
print a
print '以十六进制表示内存中的数据：'
print map(hex,a)
# bytearray() 函数通过传入 true 来原地交换
print '调用 bytearray() 函数：'
print a.bytearray(True)
print '十六进制形式：'
print map(hex,a)
# 我们可以看到字节已经交换了
```

输出如下：

```
我们的数组是：
[1 256 8755]

以十六进制表示内存中的数据：
['0x1', '0x100', '0x2233']

调用 bytearray() 函数：
[256 1 13090]

十六进制形式：
['0x100', '0x1', '0x3322']
```

3.16 副本和视图

在执行函数时，其中一些返回输入数组的副本，而另一些返回视图。当内容物理存储在另一个位置时，称为副本。如果提供了相同内存内容的不同视图，就将其称为视图。

1. 无复制

简单的赋值不会创建数组对象的副本。相反，使用原始数组的相同id()来访问它。id()返回Python对象的通用标识符，类似于C中的指针。

此外，一个数组的任何变化都反映在另一个数组上。例如，一个数组的形状改变会改变另一个数组的形状。

【例3.85】

```
import numpy as np
a = np.arange(6)
print '我们的数组是: '
print a
print '调用 id() 函数: '
print id(a)
print 'a 赋值给 b: '
b = a
print b
print 'b 拥有相同 id(): '
print id(b)
print '修改 b 的形状: '
b.shape = 3,2
print b
print 'a 的形状也修改了: '
print a
```

输出如下：

```
我们的数组是:
[0 1 2 3 4 5]

调用 id() 函数:
139747815479536

a 赋值给 b:
[0 1 2 3 4 5]
b 拥有相同 id():
139747815479536

修改 b 的形状:
[[0 1]
 [2 3]
 [4 5]]

a 的形状也修改了:
[[0 1]
 [2 3]
 [4 5]]
```

2. 视图或浅复制

NumPy拥有`ndarray.view()`方法，它是一个新的数组对象，并可查看原始数组的相同数据。与前一种情况不同，新数组的维数更改不会更改原始数据的维数。

【例3.86】

```
import numpy as np
# 最开始 a 是个 3x2 的数组
a = np.arange(6).reshape(3,2)
print '数组 a: '
print a
print '创建 a 的视图: '
b = a.view()
print b
print '两个数组的 id() 不同: '
print 'a 的 id(): '
print id(a)
print 'b 的 id(): '
print id(b)
# 修改 b 的形状，并不会修改 a
b.shape = 2,3
print 'b 的形状: '
print b
print 'a 的形状: '
print a
```

输出如下：

```
数组 a:
[[0 1]
 [2 3]
 [4 5]]
创建 a 的视图:
[[0 1]
 [2 3]
 [4 5]]
两个数组的 id() 不同:
a 的 id():
140424307227264
```

```
b 的 id():  
140424151696288  
  
b 的形状:  
[[0 1 2]  
 [3 4 5]]  
a 的形状:  
[[0 1]  
 [2 3]  
 [4 5]]
```

3.17 矩阵库

NumPy包包含一个Matrix库numpy.matlib。此模块的函数返回矩阵而不是返回ndarray对象。

1. matlib.empty()

该函数返回一个新的矩阵，而不初始化元素。该函数接收以下参数：

numpy.matlib.empty(shape, dtype, order)

- shape: 定义新矩阵形状的整数或整数元组。
- dtype: 可选，输出的数据类型。
- order: C或者F。

【例3.87】

```
import numpy.matlib  
import numpy as np  
print np.matlib.empty((2,2))  
# 填充为随机数据
```

输出如下：

```
[[ 2.12199579e-314,  4.24399158e-314]
 [ 4.24399158e-314,  2.12199579e-314]]
```

2. numpy.matlib.zeros()

该函数返回以0填充的矩阵。

【例3.88】

```
import numpy.matlib
import numpy as np
print np.matlib.zeros((2,2))
```

输出如下：

```
[[ 0.  0.]
 [ 0.  0.]]
```

3. numpy.matlib.ones()

该函数返回以1填充的矩阵。

【例3.89】

```
import numpy.matlib
import numpy as np
print np.matlib.ones((2,2))
```

输出如下：

```
[[ 1.  1.]
 [ 1.  1.]]
```

4. numpy.matlib.eye()

该函数返回一个矩阵，对角线元素为1，其他位置为0。该函数接收以下参数：

`numpy.matlib.eye(n, M, k, dtype)`

- `n`: 返回矩阵的行数。
- `M`: 返回矩阵的列数，默认为`n`。
- `k`: 对角线的索引。
- `dtype`: 输出的数据类型。

【例3.90】

```
import numpy.matlib
import numpy as np
print np.matlib.eye(n = 3, M = 4, k = 0, dtype = float)
```

输出如下：

```
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]])
```

5. `numpy.matlib.identity()`

该函数返回给定大小的单位矩阵。单位矩阵是主对角线元素都为1的方阵。

【例3.91】

```
import numpy.matlib
import numpy as np
print np.matlib.identity(5, dtype = float)
```

输出如下：

```
[[ 1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.]
 [ 0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  1.]])
```

6. numpy.matlib.rand()

该函数返回给定大小的填充随机值的矩阵。

【例3.92】

```
import numpy.matlib
import numpy as np
print np.matlib.rand(3,3)
```

输出如下：

```
[[ 0.82674464  0.57206837  0.15497519]
 [ 0.33857374  0.35742401  0.90895076]
 [ 0.03968467  0.13962089  0.39665201]]
```

注意，矩阵总是二维的，而ndarray是一个n维数组。两个对象是可互换的。

3.18 线性代数模块

NumPy包包含numpy.linalg模块，提供线性代数所需的所有功能。此模块中的一些重要功能如下。

- dot：两个数组的点积。
- vdot：两个向量的点积。
- inner：两个数组的内积。
- matmul：两个数组的矩阵积。
- determinant：数组的行列式。
- solve：求解线性矩阵方程。
- inv：寻找矩阵的乘法逆矩阵。

1. numpy.dot()

该函数返回两个数组的点积。对于二维向量，其等效于矩阵乘法。对于一维数组，它是向量的内积。对于N维数组，它是a的最后一个轴上的和与b的倒数第二个轴的乘积。

【例3.93】

```
import numpy.matlib
import numpy as np

a = np.array([[1,2],[3,4]])
b = np.array([[11,12],[13,14]])
np.dot(a,b)
```

输出如下：

```
[[37 40]
 [85 92]]
```

要注意点积计算为：

```
[[1*11+2*13, 1*12+2*14], [3*11+4*13, 3*12+4*14]]
```

2. numpy.vdot()

该函数返回两个向量的点积。如果第一个参数是复数，那么它的共轭复数会用于计算。如果参数id是多维数组，它就会被展开。

【例3.94】

```
import numpy as np
a = np.array([[1,2],[3,4]])
b = np.array([[11,12],[13,14]])
print np.vdot(a,b)
```

输出如下：

注意： $1*11 + 2*12 + 3*13 + 4*14 = 130$ 。

3. `numpy.inner()`

该函数返回一维数组的向量内积。对于更高的维度，它返回最后一个轴上的和的乘积。

【例3.95】

```
import numpy as np
print np.inner(np.array([1,2,3]),np.array([0,1,0]))
# 等价于 1*0+2*1+3*0
```

输出如下：

2

4. `numpy.matmul()`

该函数返回两个数组的矩阵乘积。虽然它返回二维数组的正常乘积，但若任一参数的维数大于2，则将其视为存在于最后两个索引的矩阵的栈，并进行相应广播。若任一参数是一维数组，则通过在其维度上附加1来将其提升为矩阵，并在乘法之后被去除。

【例3.96】

```
# 对于二维数组，它就是矩阵乘法
import numpy.matlib
import numpy as np

a = [[1,0],[0,1]]
b = [[4,1],[2,2]]
print np.matmul(a,b)
```

输出如下：

```
[[4 1]
 [2 2]]
```

5. numpy.linalg.det()

行列式在线性代数中是非常有用的值。它从方阵的对角元素计算。对于 2×2 矩阵，它是左上和右下元素的乘积与其他两个的乘积的差。换句话说，对于矩阵 $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ ，行列式计算为 $ad-bc$ 。较大的方阵被认为是 2×2 矩阵的组合。

【例3.97】

```
b = np.array([[6,1,1], [4, -2, 5], [2,8,7]])
print b
print np.linalg.det(b)
print 6*(-2*7 - 5*8) - 1*(4*7 - 5*2) + 1*(4*8 - -2*2)
```

输出如下：

```
[[ 6 1 1]
 [ 4 -2 5]
 [ 2 8 7]]

-306.0

-306
```

6. numpy.linalg.solve()

该函数给出了矩阵形式的线性方程的解。

【例3.98】

考虑以下线性方程：

```
x + y + z = 6
2y + 5z = -4
2x + 5y - z = 27
```

可以使用矩阵表示为：

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 5 \\ 2 & 5 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 6 \\ -4 \\ 27 \end{bmatrix}$$

如果矩阵成为A、X和B，方程可变为：

$$AX = B$$

或

$$X = A^{-1}B$$

7. numpy.linalg.inv()

该函数用来计算矩阵的逆。矩阵的逆是这样的，若它乘以原始矩阵，则得到单位矩阵。

【例3.99】

```
import numpy as np

x = np.array([[1,2],[3,4]])
y = np.linalg.inv(x)
print x
print y
print np.dot(x,y)
```

输出如下：

```
[[1 2]
 [3 4]]
[[-2.   1. ]
 [ 1.5 -0.5]]
[[ 1.00000000e+00  1.11022302e-16]
 [ 0.00000000e+00  1.00000000e+00]]
```

3.19 Matplotlib库

Matplotlib是Python的绘图库。它可与NumPy一起使用，提供了一种有效的MatLab开源替代方案。它也可以和图形工具包一起使用，如PyQt和wxPython。

Matplotlib模块最初是由John D. Hunter编写的。自2012年以来，Michael Droettboom是主要开发者。目前，Matplotlib 1.5.1是可用的稳定版本。该软件包可以进行二进制分发，其源代码形式在www.matplotlib.org上提供。

通常，通过添加以下语句将包导入Python脚本中：

```
from matplotlib import pyplot as plt
```

这里`pyplot()`是Matplotlib库中最重要的函数，用于绘制2D数据。

【例3.100】

使用以下脚本绘制方程 $y = 2x + 5$ 。

```
import numpy as np
from matplotlib import pyplot as plt

x = np.arange(1,11)
y = 2 * x + 5
plt.title("Matplotlib demo")
plt.xlabel("x axis caption")
plt.ylabel("y axis caption")
plt.plot(x,y) plt.show()
```

ndarray对象`x`由`np.arange()`函数创建为`x`轴上的值，`y`轴上的对应值存储在另一个数组对象`y`中。这些值使用Matplotlib软件包的`pyplot`子模块的`plot()`函数绘制，如图3.2所示。图形由`show()`函数展示。

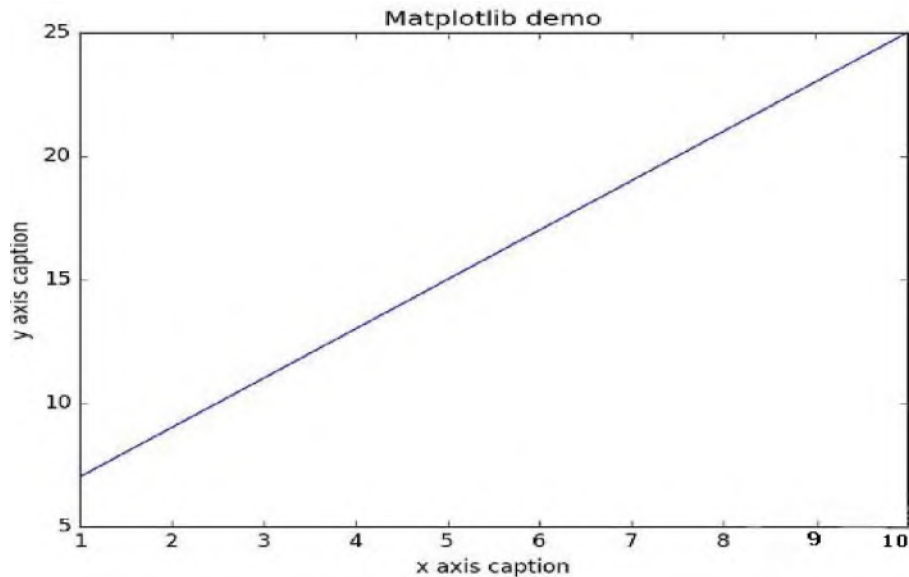


图3.2 使用`plot()`函数绘制方程 $y = 2x + 5$

作为线性图的替代，可以通过向`plot()`函数添加格式字符串来显示离散值。可以使用以下格式化字符。

- `'-'`：实线样式。
- `'--'`：短横线样式。
- `'-.'`：点划线样式。
- `':'`：虚线样式。
- `'.'`：点标记。
- `','`：像素标记。
- `'o'`：圆标记。
- `'v'`：倒三角标记。
- `'^'`：正三角标记。
- `'<'`：左三角标记。
- `'>'`：右三角标记。
- `'1'`：下箭头标记。
- `'2'`：上箭头标记。
- `'3'`：左箭头标记。
- `'4'`：右箭头标记。

- 's': 正方形标记。
- 'p': 五边形标记。
- '*': 星形标记。
- 'h': 六边形标记1。
- 'H': 六边形标记2。
- '+': 加号标记。
- 'x': X标记。
- 'D': 菱形标记。
- 'd': 窄菱形标记。
- '|': 竖直线标记。
- '_': 水平线标记。

还定义了以下颜色缩写。

- 'b': 蓝色。
- 'g': 绿色。
- 'r': 红色。
- 'c': 青色。
- 'm': 品红色。
- 'y': 黄色。
- 'k': 黑色。
- 'w': 白色。

3.20 Matplotlib 绘制直方图

NumPy有一个`numpy.histogram()`函数，它用于实现数据的频率分布的图形表示。水平尺寸相等的矩形对应于类间隔，称为bin，变量`height`对应频率。

1. `numpy.histogram()`

该函数将输入数组和bin作为两个参数。bin数组中的连续元素用作每个bin的边界。

【例3.101】

```
import numpy as np

a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
np.histogram(a,bins = [0,20,40,60,80,100])
hist,bins = np.histogram(a,bins = [0,20,40,60,80,100])
print hist
print bins
```

输出如下：

```
[3 4 5 2 1]
[0 20 40 60 80 100]
```

2. plt()

Matplotlib可以将直方图的数字表示转换为图形。pyplot子模块的plt()函数将包含数据和bin数组的数组作为参数，并转换为直方图。

【例3.102】

```
from matplotlib import pyplot as plt
import numpy as np

a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
plt.hist(a, bins = [0,20,40,60,80,100])
plt.title("histogram")
plt.show()
```

效果如图3.3所示。

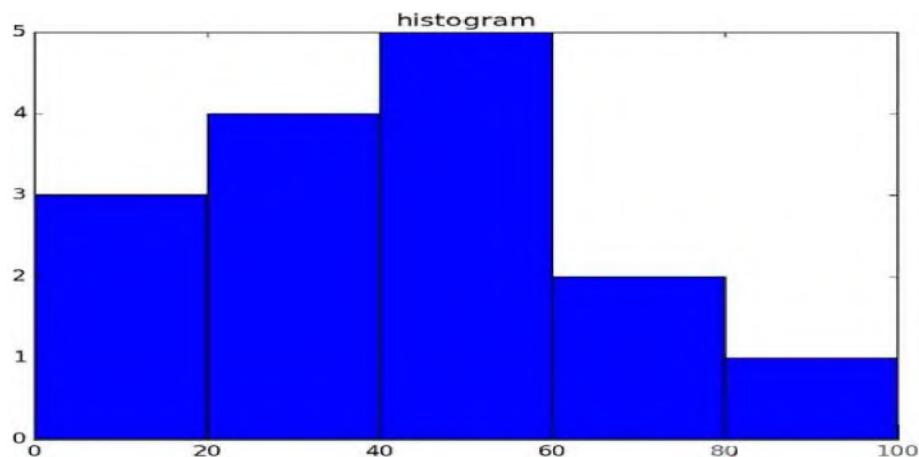


图3.3 pyplot子模块的plt()函数将包含数据转换为直方图

3.21 IO文件操作

ndarray对象可以保存到磁盘文件并从磁盘文件加载。可用的IO功能有：

- load() 和 save() 函数处理NumPy二进制文件（带.npy扩展名）。
- loadtxt() 和 savetxt() 函数处理正常的文本文件。

NumPy为ndarray对象引入了一个简单的文件格式。这个.npy文件在磁盘文件中，存储重建ndarray所需的数据、图形、dtype和其他信息，以便正确获取数组，即使该文件在具有不同架构的另一台机器上。

1. numpy.save()

该函数将输入数组存储在具有.npy扩展名的磁盘文件中。

【例3.103】

```
import numpy as np
a = np.array([1,2,3,4,5])
```

```
np.save('outfile',a)
```

为了从outfile.npy重建数组，请使用load()函数。

```
import numpy as np
b = np.load('outfile.npy')
print b
```

输出如下：

```
array([1, 2, 3, 4, 5])
```

save()和load()函数接收一个附加的布尔参数allow_pickle。Python中的pickle用于在保存到磁盘文件或从磁盘文件读取之前对对象进行序列化和反序列化。

2. savetxt()

以简单文本文件格式存储和获取数组数据，是通过savetxt()和loadtxt()函数完成的。

【例3.104】

```
import numpy as np

a = np.array([1,2,3,4,5])
np.savetxt('out.txt',a)
b = np.loadtxt('out.txt')
print b
```

输出如下：

```
[ 1.  2.  3.  4.  5.]
```

savetxt()和loadtxt()函数接收附加的可选参数，例如页首、页尾和分隔符。

3.22 NumPy实例：GPS定位

以GPS系统为例介绍卫星定位的计算方法。GPS定位的基本原理：根据高速运动卫星的瞬间位置作为已知的起算数据，采用空间距离后方交会的方法确定待测点的位置。

假设 t 时刻在地面待测点上安置GPS接收机，可以测定GPS信号到达接收机的时间 Δt ，再加上接收机所接收到的卫星星历等其他数据，就可以确定一个方程组来对位置信息进行求解。假设地球上一个点 R ，同时收到6颗卫星（ $S_1, S_2, \dots, S_6$ ）发射的信号，假设接收信息如图3.4所示（图来源于网络）。其中 x 、 y 表示卫星的经纬度， z 表示卫星的高度。

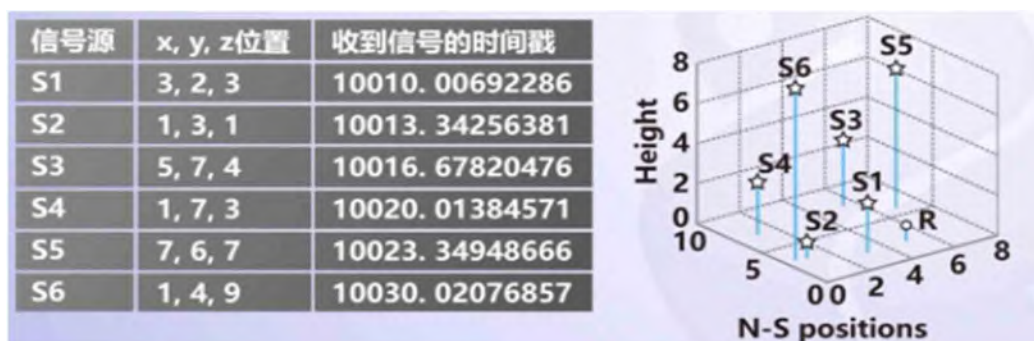


图3.4 6颗卫星的空间位置和信号接收时间戳

【例3.105】

由于上述6个卫星和地球在高速运动，从卫星发出的位置信息以光速传输到GPS接收端需要一定的时间。假设 (x, y, z, t) 表示 R 当前的位置， t 是 R 的相对时间，卫星 S_1 发出信号时刻到当前接收时刻满足以下关系（其中 c 是光速）：

$$(x-3)^2 + (y-2)^2 + (z-3)^2 = [(10010.00692286 - t) * c]^2$$

该公式表示以 (x, y, z, t) 为参数的（欧式空间距离）与信号传输距离相等。

对于卫星 $S_1, S_2, \dots, S_6$ ，满足方程组：

$$\begin{cases} (x-3)^2 + (y-2)^2 + (z-3)^2 - [(10010.00692286-t)*c]^2 = 0 \\ (x-1)^2 + (y-3)^2 + (z-1)^2 - [(10013.34256381-t)*c]^2 = 0 \\ (x-5)^2 + (y-7)^2 + (z-4)^2 - [(10016.67820476-t)*c]^2 = 0 \\ (x-1)^2 + (y-7)^2 + (z-3)^2 - [(10020.01384571-t)*c]^2 = 0 \\ (x-7)^2 + (y-6)^2 + (z-7)^2 - [(10023.34948666-t)*c]^2 = 0 \\ (x-1)^2 + (y-4)^2 + (z-9)^2 - [(10030.02076857-t)*c]^2 = 0 \end{cases} \quad (3.2)$$

其中，光速为常数 $c=0.299792458\text{km/us}$ 。上述方程组是非线性的，但很容易将所有二次项都消去（每个公式减去第一个公式），从而得到：

$$\begin{pmatrix} 4 & -4 & -12 & 3.59751 \\ 0 & -2 & -16 & 2.99792 \\ 8 & 6 & -10 & 2.39834 \\ 0 & 6 & -12 & 1.79875 \\ 12 & 4 & -4 & 1.19917 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix} = \begin{pmatrix} 35971.1 \\ 29957.2 \\ 24031.4 \\ 17993.5 \\ 12059.7 \end{pmatrix} \quad (3.3)$$

此时，上述等式变成了 $A*X=B$ 形式，根据线性代数方法， $X=A^{-1}*B$ ，即只需对系数矩阵求逆，再乘以常数矩阵便可以得到方程组的解。

GPS定位的问题建模，上面给出了GPS的定位原理，如何利用计算机辅助GPS的定位计算呢？以6颗卫星为例，GPS定位计算问题的IPO模式描述如下：

- 输入：6颗卫星的欧式坐标和信号时间戳。
- 处理：GPS定位算法。
- 输出：GPS接收设备的地理坐标和当前时间。

假设第 i 颗卫星的坐标和时间戳表示为 (x_i, y_i, z_i, t_i) ，结合上述例子，GPS定位算法可以描述为如下公式：

$$\begin{pmatrix} x_1 - x_2 & y_1 - y_2 & z_1 - z_2 & c^2(t_1 - t_2) \\ x_1 - x_3 & y_1 - y_3 & z_1 - z_3 & c^2(t_1 - t_3) \\ x_1 - x_4 & y_1 - y_4 & z_1 - z_4 & c^2(t_1 - t_4) \\ x_1 - x_5 & y_1 - y_5 & z_1 - z_5 & c^2(t_1 - t_5) \\ x_1 - x_6 & y_1 - y_6 & z_1 - z_6 & c^2(t_1 - t_6) \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix} = \begin{pmatrix} [(t_2^2 - t_1^2)c^2 - (x_2^2 - x_1^2 + y_2^2 - y_1^2 + z_2^2 - z_1^2)]/2 \\ [(t_3^2 - t_1^2)c^2 - (x_3^2 - x_1^2 + y_3^2 - y_1^2 + z_3^2 - z_1^2)]/2 \\ [(t_4^2 - t_1^2)c^2 - (x_4^2 - x_1^2 + y_4^2 - y_1^2 + z_4^2 - z_1^2)]/2 \\ [(t_5^2 - t_1^2)c^2 - (x_5^2 - x_1^2 + y_5^2 - y_1^2 + z_5^2 - z_1^2)]/2 \\ [(t_6^2 - t_1^2)c^2 - (x_6^2 - x_1^2 + y_6^2 - y_1^2 + z_6^2 - z_1^2)]/2 \end{pmatrix} \quad (3.4)$$

下面将使用NumPy函数库实现上述矩阵操作。程序中用到的函数如下：

- `numpy.dot(a, b)`：计算矩阵a与矩阵b的点积。
- `numpy.linalg.inv(a)`：求矩阵a的逆矩阵。

GPS定位实现的Python代码如下：

```

from numpy import *
def main_GPSLocation():
    i = 1
    c = 0.299792458 # 光速 0.299792458km/us
    x = zeros((6, 4)) #存储6个卫星的 (x,y,z,t) 参数
    while i<=6:
        print(" %s %d" % ("please input (x,y,z,t) of group",i) )
        temp=input()
        x[i-1]=temp.split()
        j=0
        while j<4:
            x[i-1][j]=float(x[i-1][j])
            j=j+1
        i=i+1
    a=zeros((4,4)) #系数矩阵
    b=zeros((4,1)) #常数项
    j=0
    while j<4:
        a[j][0]=2*(x[5][0]-x[j][0])
        a[j][1]=2*(x[5][1]-x[j][1])
        a[j][2]=2*(x[5][2]-x[j][2])
        a[j][3]=2*c*c*(x[j][3]-x[5][3])
        b[j][0]=x[5][0] * x[5][0] - x[j][0] * x[j][0] + \
            x[5][1] * x[5][1] - x[j][1] * x[j][1] + \
            x[5][2] * x[5][2] - x[j][2] * x[j][2] + \
            c*c*(x[j][3] * x[j][3] - x[5][3] * x[5][3])
        j=j+1
    a_ni=linalg.inv(a) #系数矩阵求逆
    print(dot(a_ni,b))

main_GPSLocation()

```

其中，zeros是NumPy提供的函数，用来建立指定维度的数组。zeros用来生成数组x用于存储接收来自外部输入的6颗卫星的坐标，数组a、b用来存放前面算法中的系数矩阵。例程中还展示了两种数组的索引方法，最后调用求矩阵逆的函数及点乘操作完成坐标计算。

运行程序后，依次输入6颗卫星的坐标，运算结果如图3.5所示。


```
please input (x,y,z,t) of group 1
3 2 3 10010.00692286
please input (x,y,z,t) of group 2
1 3 1 10013.34256381
please input (x,y,z,t) of group 3
5 7 4 10016.67820476
please input (x,y,z,t) of group 4
1 7 3 10020.01384571
please input (x,y,z,t) of group 5
7 6 7 10023.34948666
please input (x,y,z,t) of group 6
1 4 9 10030.02076857
[[ 5.00000000e+00]
 [ 3.00000000e+00]
 [ 1.00000000e+00]
 [ 1.00000000e+04]]
Process finished with exit code 0
```

图3.5 GPS定位的程序实现运算结果

3.23 本章小结

Python没有提供数组功能，NumPy可以提供数组支持以及相应的高效处理函数，是Python数据分析的基础，也是SciPy、Pandas等数据分析和科学计算库基本的函数功能库，且其数据类型对Python数据分析十分有用。本章详细介绍了NumPy的语法及操作等。

第4章 大数据分析工具：SciPy

SciPy库构建于NumPy之上，提供了一个用于在Python中进行科学计算的工具集，如数值计算的算法和一些功能函数，可以方便地处理数据。SciPy库主要包含以下内容：

- 统计
- 优化
- 集成
- 线性代数
- 傅里叶变换
- 信号和图像处理

4.1 SciPy简介

SciPy是一个高级的科学计算库，它和NumPy的联系很密切，SciPy一般都是操控NumPy数组来进行科学计算，所以可以说SciPy是基于NumPy的。

SciPy由一些特定功能的子模块组成：

- `SciPy.cluster`：向量量化。
- `SciPy.constants`：数学常量。
- `SciPy.fftpack`：快速傅里叶变换。
- `SciPy.integrate`：积分。
- `SciPy.interpolate`：插值。
- `SciPy.io`：数据输入输出。
- `SciPy.linalg`：线性代数。
- `SciPy.ndimage`：N维图像。

- SciPy.odr: 正交距离回归。
- SciPy.optimize: 优化算法。
- SciPy.signal: 信号处理。
- SciPy.sparse: 稀疏矩阵。
- SciPy.spatial: 空间数据结构和算法。
- SciPy.special: 特殊数学函数。
- SciPy.stats: 统计函数。

它们全依赖NumPy，但是每个模块之间基本独立。导入NumPy和这些SciPy模块的标准方式是：

```
import numpy as np
from scipy import stats # 其他子模块相同
```

主SciPy命名空间大多包含真正的NumPy函数（尝试SciPy.cos就是np.cos）。这些仅仅是由于历史原因，通常没有理由在代码中使用import SciPy。

4.2 文件输入和输出：SciPy.io

（1）这个模块可以加载和保存Matlab文件

【例4.1】

```
>>> from scipy import io as spio
>>> a = np.ones((3, 3))
>>> spio.savemat('file.mat', {'a': a}) # 保存字典到 file.mat
>>> data = spio.loadmat('file.mat', struct_as_record=True)
>>> data['a']
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

（2）读取图片

【例4.2】

```
In [16]: from scipy import misc
In [17]: misc.imread('scikit.png')
Out[17]:
array([[[255, 255, 255, 255],
        [255, 255, 255, 255],
        [255, 255, 255, 255],
        ...,
        ...,
        [255, 255, 255, 255],
        [255, 255, 255, 255],
        [255, 255, 255, 255]]], dtype=uint8)

In [18]: import matplotlib.pyplot as plt
In [19]: plt.imread('scikit.png')
Out[19]:
array([[[ 1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.],
        ...,
        ...,
        [ 1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.],
        ...,
        ...,
        [ 1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.] ]], dtype=float32)
```

(3) 载入TXT文件

```
numpy.loadtxt()
numpy.savetxt()
```

(4) 智能导入文本/CSV文件

```
numpy.genfromtxt()
numpy.recfromcsv()
```

(5) 高速、有效率，但NumPy特有的二进制格式

```
numpy.save()  
numpy.load()
```

4.3 特殊函数: SciPy.special

特殊函数是先验函数，常用的有：

- 贝塞尔函数，如SciPy.special.jn()（整数n阶贝塞尔函数）。
- 椭圆函数，如SciPy.special.ellipj()（雅可比椭圆函数）。
- 伽马函数，如SciPy.special.gamma()。还要注意SciPy.special.gammaln()，这个函数给出对数坐标的伽马函数，因此有更高的数值精度。

4.4 线性代数操作: SciPy.linalg

SciPy.linalg模块提供标准线性代数运算，依赖于底层有效率的实现（BLAS，LAPACK）。

假如要计算一个方阵的行列式，需要调用det()函数：

【例4.3】

```
>>> from scipy import linalg  
>>> arr = np.array([[1, 2],  
...                 [3, 4]])  
>>> linalg.det(arr)  
-2.0  
>>> arr = np.array([[3, 2],  
...                 [6, 4]])  
>>> linalg.det(arr)  
0.0
```

比如求一个矩阵的转置：

【例4.4】

```
>>> arr = np.array([[1, 2],  
...                 [3, 4]])  
>>> iarr = linalg.inv(arr)  
>>> iarr  
array([[ -2. ,  1. ],  
       [ 1.5, -0.5]])
```

4.5 快速傅里叶变换: `scipy.fftpack`

首先用NumPy初始化正弦信号，示例代码如下：

【例4.5】

```
>>> import numpy as np  
>>> time_step = 0.02  
>>> period = 5.  
>>> time_vec = np.arange(0, 20, time_step)  
>>> sig = np.sin(2 * np.pi / period * time_vec) + \  
...       0.5 * np.random.randn(time_vec.size)
```

如果我们要计算该信号的采样频率，那么可以用`SciPy.fftpack.fftfreq()`函数。计算它的快速傅里叶变换使用`SciPy.fftpack.fft()`函数：

【例4.6】

```
>>> from scipy import fftpack  
>>> sample_freq = fftpack.fftfreq(sig.size, d=time_step)  
>>> sig_fft = fftpack.fft(sig)
```

NumPy中也有用于计算快速傅里叶变换的模块：`numpy.fft`。但是`SciPy.fftpack`是我们的首选，因为它应用了更多底层的工具，工作效率要高一些。

4.6 优化器: `SciPy.optimize`

优化是找到最小值或等式的数值解的问题。SciPy.optimize子模块提供了函数最小值（标量或多维）、曲线拟合和寻找等式的根的有效算法。SciPy.optimize通常用来最小化一个函数值，构建一个函数并绘制函数图。

【例4.7】

```
>>> def f(x):  
...     return x**2 + 10*np.sin(x)  
>>> x = np.arange(-10, 10, 0.1)  
>>> plt.plot(x, f(x))  
>>> plt.show()
```

效果如图4.1所示。

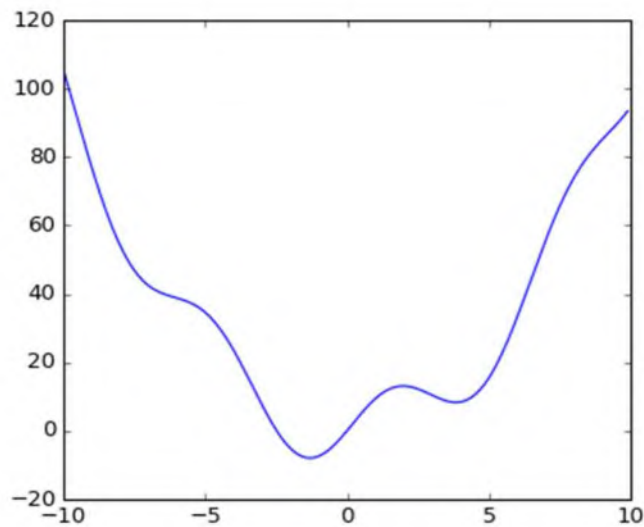


图4.1 SciPy.optimize构建一个函数并绘制函数图

如果要找出这个函数的最小值，也就是曲线的最低点，就可以使用 BFGS 优化算法（Broyden Fletcher Goldfarb Shanno Algorithm）：

```
>>> optimize.fmin_bfgs(f, 0)
Optimization terminated successfully.
      Current function value: -7.945823
      Iterations: 5
      Function evaluations: 24
      Gradient evaluations: 8
array([-1.30644003])
```

可以得到最低点的值为-1.30644003, `optimize.fmin_bfgs(f, 0)` 第二个参数0表示从0点的位置最小化, 找到最低点(该点刚好为全局最低点)。假如从3点的位置开始梯度下降, 那么得到的将会是局部最低点3.83746663:

```
>>> optimize.fmin_bfgs(f, 3, disp=0)
array([ 3.83746663])
```

4.7 统计工具: SciPy.stats

Python有一个很好的统计推断包, 就是SciPy里面的stats。

SciPy的stats模块包含了多种概率分布的随机变量, 随机变量分为连续的和离散的两种。所有的连续随机变量都是`rv_continuous`的派生类的对象, 而所有的离散随机变量都是`rv_discrete`的派生类的对象。各个随机过程的随机数生成器可以从`numpy.random`中找到。

首先随机生成1000个服从正态分布的数, 如图4.2所示。

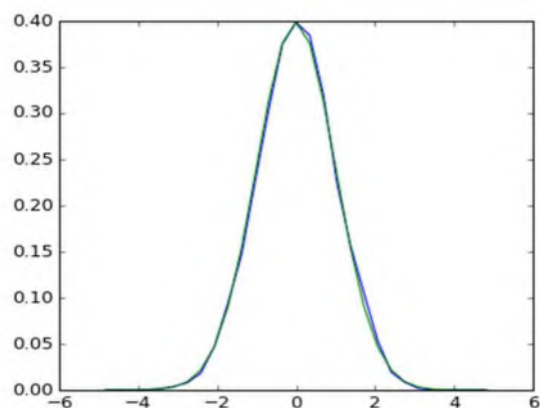


图4.2 随机生成服从正态分布的数

【例4.8】

```
>>> a = np.random.normal(size=1000)
#用stats模块计算该分布的均值和标准差
>>> loc, std = stats.norm.fit(a)
>>> loc
0.0314345570...
>>> std
0.9778613090...
#中位数
>>> np.median(a)
0.04041769593...
```

SciPy的其他模块（计算积分、信号处理、图像处理的模块）就不一一介绍了。其实机器学习最基础的部分还是统计算法和优化算法。

4.8 SciPy实例

4.8.1 最小二乘拟合

假设有一组实验数据 $(x[i], y[i])$ ，知道它们之间的函数关系： $y = f(x)$ ，通过这些已知信息，需要确定函数中的一些参数项。例如，如果 f 是一个线型函数 $f(x) = k * x + b$ ，那么参数 k 和 b 就是

需要确定的值。如果将这些参数用 P 表示的话，就要找到一组 P 值使得如下公式中的 S 函数最小：

$$S(P) = \sum_{i=1}^m [y_i - f(x_i, P)]^2 \quad (4.1)$$

这种算法被称为最小二乘拟合（Least-Square Fitting）。

用已知数据集 (x, y) 去确定一组参数 (k, b) ，使得实际的 y 值减去预测的 y 值的平方和最小。

SciPy中的子函数库optimize已经提供了实现最小二乘拟合算法的函数leastsq。下面是用leastsq进行数据拟合的一个例子。

【例4.9】

```

# -*- coding: utf-8 -*-
# 注意，如果代码里有中文，那么文件开头必须添加上面这句话，或者添加“#encoding:utf-8”
import numpy as np
from scipy.optimize import leastsq
import pylab as pl

def func(x, p):
    """
    数据拟合所用的函数：A*sin(2*pi*k*x + theta)
    """
    A, k, theta = p
    return A*np.sin(2*np.pi*k*x+theta)

def residuals(p, y, x):
    """
    实验数据 x, y 和拟合函数之间的差，p 为拟合需要找到的系数
    """
    return y - func(x, p)

x = np.linspace(0, -2*np.pi, 100)
A, k, theta = 10, 0.34, np.pi/6          # 真实数据的函数参数
y0 = func(x, [A, k, theta])              # 真实数据
y1 = y0 + 2 * np.random.randn(len(x))    # 加入噪声之后的实验数据

p0 = [7, 0.2, 0]                          # 第一次猜测的函数拟合参数

# 调用 leastsq 进行数据拟合
# residuals 为计算误差的函数
# p0为拟合参数的初始值
# args 为需要拟合的实验数据
plsq = leastsq(residuals, p0, args=(y1, x))

print u"真实参数:", [A, k, theta]
print u"拟合参数", plsq[0]                # 实验数据拟合后的参数

pl.plot(x, y0, label=u"真实数据")
pl.plot(x, y1, label=u"带噪声的实验数据")
pl.plot(x, func(x, plsq[0]), label=u"拟合数据")
pl.legend()
pl.show()

```

这个例子中要拟合的函数是一个正弦波函数，它有3个参数：A、k和theta，分别对应振幅、频率和相角。假设实验数据是一组包含噪声的数据x, y1，其中y1是在真实数据y0的基础上加入噪声得到的。

通过leastsq函数对带噪声的实验数据x, y1进行数据拟合, 可以找到x和真实数据y0之间的正弦关系的3个参数: A、k和theta。

最后可以发现拟合参数虽然和真实参数完全不同, 但是由于正弦函数具有周期性, 实际上拟合参数得到的函数和真实参数对应的函数是一致的。

4.8.2 函数最小值

optimize 库提供了几个求函数最小值的算法: fmin、fmin_powell、fmin_cg和fmin_bfgs。

对于一个离散的线性时不变系统传递函数h, 如果它的输入是x, 那么其输出y可以用x和h的卷积表示: $y = x (*) h$ 。

现在的问题是如果已知系统的输入x和输出y, 如何计算系统的传递函数h; 或者已知系统的传递函数h和系统的输出y, 如何计算系统的输入x。这种运算被称为反卷积运算, 是十分困难的, 特别是在实际的运用中, 测量系统的输出总是存在误差的。

下面用fmin计算反卷积, 这种方法只能用在很小规模的数列上, 因此没有很大的实用价值, 不过用来评价fmin函数的性能还是不错的。

【例4.10】

```

# -*- coding: utf-8 -*-
# 本程序用各种 fmin 函数求卷积的逆运算

import scipy.optimize as opt
import numpy as np

def test_fmin_convolve(fminfunc, x, h, y, yn, x0):
    """
    x (*) h = y, (*)表示卷积
    yn 为在 y 的基础上添加一些干扰噪声的结果
    x0为求解 x 的初始值
    """
    def convolve_func(h):
        """
        计算 yn - x (*) h 的 power
        fmin 将通过计算使得此 power 最小
        """
        return np.sum((yn - np.convolve(x, h))**2)

    # 调用 fmin 函数, 以 x0为初始值
    h0 = fminfunc(convolve_func, x0)

    print fminfunc.__name__
    print "-----"
    # 输出 x (*) h0 和 y 之间的相对误差
    print "error of y:", np.sum((np.convolve(x, h0)-y)**2)/np.sum(y**2)
    # 输出 h0 和 h 之间的相对误差
    print "error of h:", np.sum((h0-h)**2)/np.sum(h**2)
    print

def test_n(m, n, nscale):
    """
    随机产生 x, h, y, yn, x0等数列, 调用各种 fmin 函数求解 b
    m 为 x 的长度, n 为 h 的长度, nscale 为干扰的强度
    """
    x = np.random.rand(m)
    h = np.random.rand(n)
    y = np.convolve(x, h)
    yn = y + np.random.rand(len(y)) * nscale
    x0 = np.random.rand(n)

    test_fmin_convolve(opt.fmin, x, h, y, yn, x0)
    test_fmin_convolve(opt.fmin_powell, x, h, y, yn, x0)
    test_fmin_convolve(opt.fmin_cg, x, h, y, yn, x0)
    test_fmin_convolve(opt.fmin_bfgs, x, h, y, yn, x0)

if __name__ == "__main__":
    test_n(200, 20, 0.1)

if __name__ == "__main__"

```

上面这段代码的功能是，在当前文件下编译运行的时候执行if下面的语句。如果在其他文件内通过import调用本文件，if内语句就不执行。

4.9 本章小结

SciPy依赖于NumPy，提供了真正的矩阵，包含的功能有最优化、线性代数、积分、插值、拟合、特殊函数、快速傅里叶变换、信号处理、图像处理、常微分方程求解器等。SciPy是高端科学计算工具包，由一些特定功能的子模块组成。

第5章 大数据分析工具：Matplotlib

Matplotlib是Python 2D绘图领域使用广泛的套件。它能让使用者很轻松地将数据图形化，并且提供多样化的输出格式。本章将会探索Matplotlib的常见用法。在Matplotlib中使用最多的模块是Pyplot。Pyplot是Matplotlib面向对象绘图库的一个接口。它的语法和Matlab十分相近。也就是说，它主要的绘图命令和Matlab对应的命令有相似的参数。

5.1 初级绘制

使用Matplotlib库绘图，原理很简单，就是下面这5步：

- 创建一个图纸（figure）。
- 在图纸上创建一个或多个绘图（plotting）区域（也叫子图、坐标系 / 轴）。
- 在plotting区域上描绘点、线等各种marker。
- 为plotting添加修饰标签（绘图线上的或坐标轴上的）。
- 其他各种DIY。

在上面的过程中，主要涉及下面3个元素：

- 变量。
- 函数。
- 图纸（figure）和子图（axes，也可以理解成坐标轴）。

其中，变量和函数通过改变figure和axes中的元素（例如标题、标签、点和线等）一起描述figure和axes，也就是在画布上绘图。图片结构如图5.1所示。

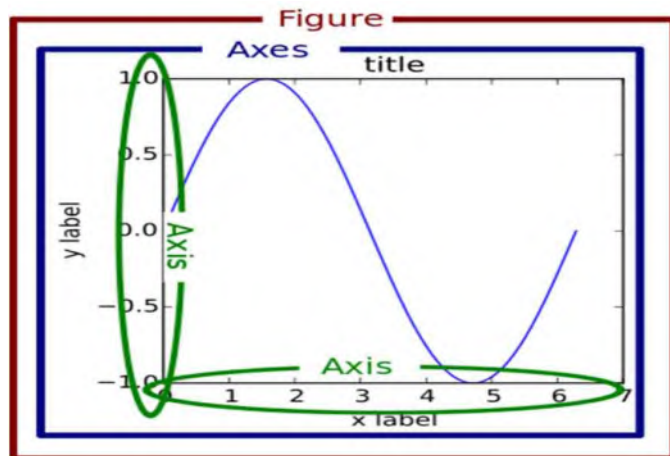


图5.1 Matplotlib库绘图图片结构

这一节将从简到繁：先尝试用默认配置在同一张图上绘制正弦和余弦函数图像，然后逐步美化它。

下面取得正弦函数和余弦函数的值：

```
from pylab import *  
  
X = np.linspace(-np.pi, np.pi, 256, endpoint=True)  
C,S = np.cos(X), np.sin(X)
```

X现在是一个NumPy数组，包含从 $-\pi$ 到 $+\pi$ （包含 π ）等差分布的256个值。C是正弦值（256个值），S是余弦值（256个值）。

可以在IPython的交互模式下测试代码，也可以执行Python文件。

```
python exercise_1.py
```

1. 使用默认配置

Matplotlib的默认配置都允许用户自定义。可以调整大多数的默认配置：图片大小和分辨率（dpi）、线宽、颜色、风格、坐标轴以及网格的属性、文字与字体属性等。不过，Matplotlib的默认配置在大

多数情况下已经做得足够好，只在很少的情况下才会更改这些默认配置。

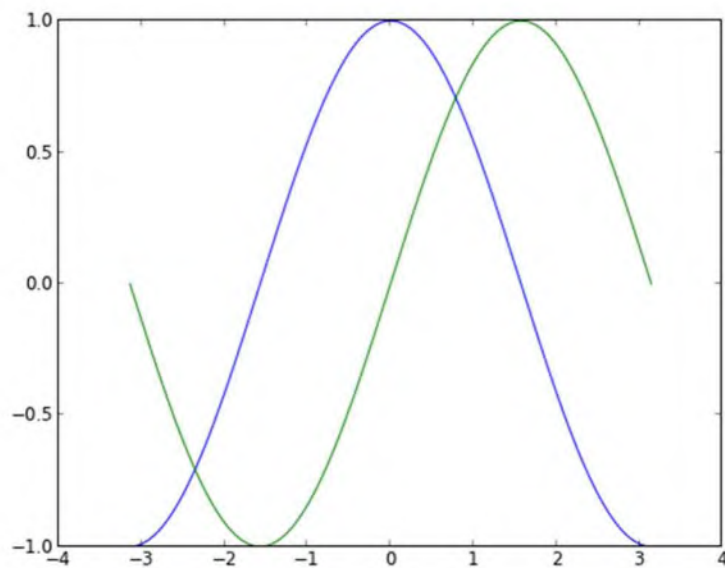
```
from pylab import *

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C,S = np.cos(X), np.sin(X)

plot(X,C)
plot(X,S)

show()
```

Matplotlib使用默认配置的正余弦图如图5.2所示。



下面的代码中展现了Matplotlib的默认配置并辅以注释说明，这部分配置包含有关绘图样式的所有配置。代码中的配置与默认配置完全相同，可以在交互模式中修改其中的值来观察效果。

```
# 导入Matplotlib 的所有内容 (NymPy 可以用 np 这个名字)
from pylab import *

# 创建一个 8 * 6 点 (point) 的图，并设置分辨率为 80
```

```

figure(figsize=(8,6), dpi=80)

# 创建一个新的 1 * 1 的子图，接下来的图样绘制在其中的第 1 块（也是唯一的一块）
subplot(1,1,1)

X = np.linspace(-np.pi, np.pi, 256,endpoint=True)
C,S = np.cos(X), np.sin(X)

# 绘制余弦曲线，使用蓝色的、连续的、宽度为 1 （像素）的线条
plot(X, C, color="blue", linewidth=1.0, linestyle="-")

# 绘制正弦曲线，使用绿色的、连续的、宽度为 1 （像素）的线条
plot(X, S, color="green", linewidth=1.0, linestyle="-")

# 设置横轴的上下限
xlim(-4.0,4.0)

# 设置横轴记号
xticks(np.linspace(-4,4,9,endpoint=True))

# 设置纵轴的上下限
ylim(-1.0,1.0)

# 设置纵轴记号
yticks(np.linspace(-1,1,5,endpoint=True))

# 以分辨率 72 来保存图片
# savefig("exercice_2.png",dpi=72)

# 在屏幕上显示
show()

```

2. 改变线条的颜色和粗细

以蓝色和红色分别表示余弦和正弦函数，而后将线条变粗一点。在水平方向拉伸一下整个图，如图5.3所示。

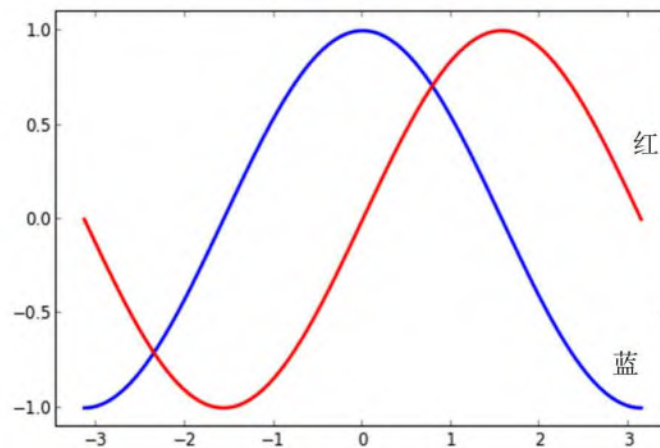


图5.3 改变线条的颜色和粗细

```
figure(figsize=(10,6), dpi=80)
plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plot(X, S, color="red", linewidth=2.5, linestyle="-")
```

3. 设置边界

当前的图像边界有点太紧了，而且想要预留一点空间使数据点更清晰。

```
xlim(X.min()*1.1, X.max()*1.1)
ylim(C.min()*1.1, C.max()*1.1)
```

4. 设置刻度

当前的刻度并不理想，因为不显示正余弦中我们感兴趣的值（ $\pm\pi$, $\pm\pi/2$ ）。我们将进行更改，让其只显示这些值。

```
xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
yticks([-1, 0, +1])
```

效果如图5.4所示。

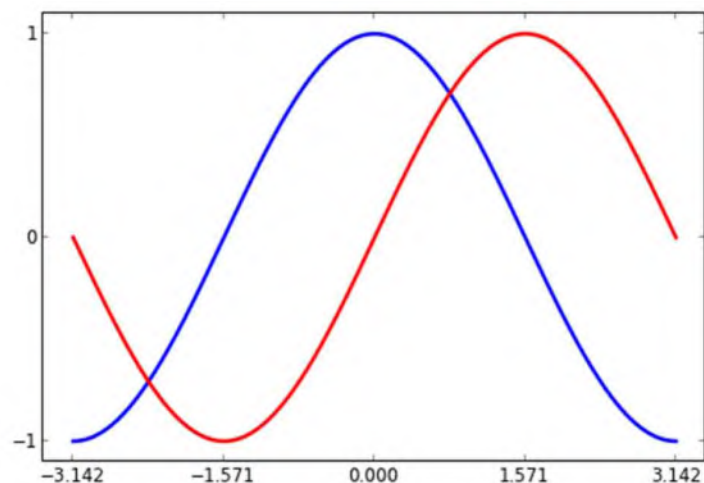


图5.4 设置刻度

5. 设置刻度标签

刻度已经设置得很合适了，但是其标签并不是很清楚，可以猜出 3.142 是 π ，但是最好让它更直接。当设置刻度值时，也可以在第二个参数列表中提供相应的标签。注意，用 latex 可以获得更好渲染的标签。

```
xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
        [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])

yticks([-1, 0, +1],
        [r'$-1$', r'$0$', r'$+1$'])
```

效果如图5.5所示。

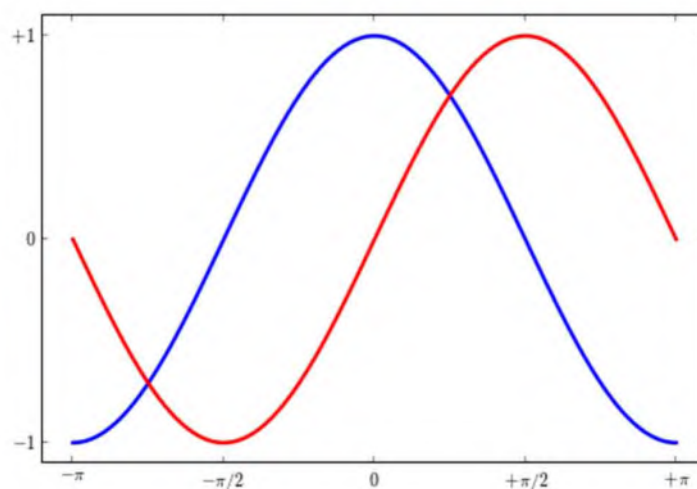


图5.5 设置刻度标签

6. 移动轴线

轴线（spines）是连接刻度标志和标示数据区域边界的线。它们可以被放置在任意地方，现在是轴的边界。改变这点，让它们位于中间。一共有4个轴线（上 / 下 / 左 / 右）。通过将它们的颜色设置成None舍弃位于顶部和右部的轴线。然后把底部和左部的轴线移动到数据空间坐标中的零点。

```
ax = gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
```

7. 添加图例

下面在图片左上角添加一个图例。这仅仅需要向plot命令添加关键字参数label（之后将被图例框使用）。

```
plot(X, C, color="blue", linewidth=2.5, linestyle="-",  
label="cosine")  
plot(X, S, color="red", linewidth=2.5, linestyle="-",  
label="sine")  
  
legend(loc='upper left')
```

效果如图5.6所示。

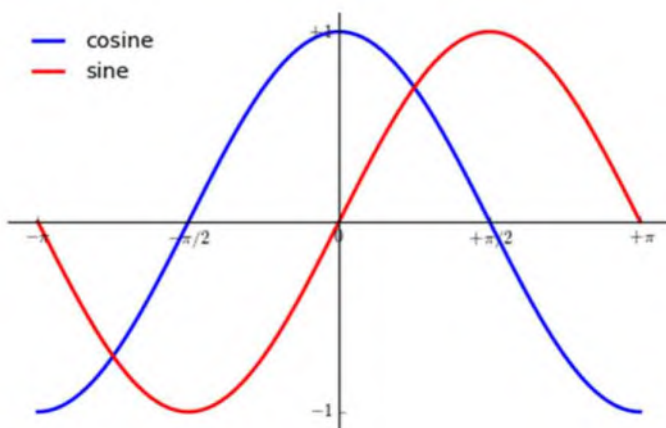


图5.6 添加图例

8. 注解某些点

现在使用`annotate`命令注解一些我们感兴趣的点。选择 $2\pi/3$ 作为想要注解的正弦值和余弦值。我们将在曲线上做一个标记和一个垂直的虚线。然后，使用`annotate`命令来显示一个箭头和一些文本。

```

t = 2*np.pi/3
plot([t,t],[0,np.cos(t)], color='blue', linewidth=2.5, linestyle="--")
scatter([t],[np.cos(t)], 50, color='blue')

annotate(r'$\sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
        xy=(t, np.sin(t)), xycoords='data',
        xytext=(+10, +30), textcoords='offset points', fontsize=16,
        arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

plot([t,t],[0,np.sin(t)], color='red', linewidth=2.5, linestyle="--")
scatter([t],[np.sin(t)], 50, color='red')

annotate(r'$\cos(\frac{2\pi}{3})=-\frac{1}{2}$',
        xy=(t, np.cos(t)), xycoords='data',
        xytext=(-90, -50), textcoords='offset points', fontsize=16,
        arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

```

效果如图5.7所示。

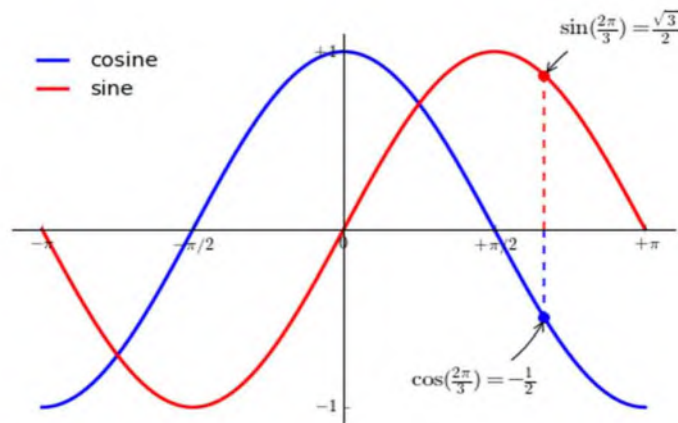


图5.7 注解某些点

5.2 图像、子区、子图、刻度

到目前为止，我们都用隐式的方法来绘制图像和坐标轴。在快速绘图中，这种方法很方便。我们也可以显式地控制图像（Figures）、子图（Axes）和坐标轴。Matplotlib中的“图像”指的是用户在界面上看到的整个窗口内容。在图像里面有所谓的“子图”。子图的位置

是由坐标网格确定的，而“坐标轴”却不受此限制，可以放在图像的任意位置。我们已经隐式地使用过图像和子图：在调用plot函数的时候，Matplotlib调用gca()函数和gcf()函数来获取当前的坐标轴和图像。如果无法获取图像，就会调用figure()函数来创建一个图像，严格地说，是用subplot(1, 1, 1)创建一个只有一个子图的图像。

1. 图像

所谓图像，就是GUI里以“Figure #”为标题的那些窗口。图像编号从1开始，与Matlab的风格一致，而与Python从0开始编号的风格不同。表5.1所示的参数是图像的属性。

表5.1 图像的属性参数

参数	默认值	描述
num	1	图像的数量
figsize	figure(figsize	图像的长和宽（英寸）
dpi	figure.dpi	分辨率（点/英寸）
facecolor	figure.facecolor	绘图区域的背景颜色
edgecolor	figure.edgecolor	绘图区域边缘的颜色
frameon	True	是否绘制图像边缘

除了图像数量这个参数外，其余的参数都很少修改。

在图形界面中可以单击右上角的“×”来关闭窗口，Matplotlib还提供了名为close的函数来关闭这个窗口。和其他对象一样，你可以使用setp或者set_something来设置图像的属性。

2. 子图

用户可以用子图来将图样（Plot）放在均匀的坐标网格中，如图5.8所示。用subplot函数的时候需要指明网格的行列数量，以及希望将图样放在哪个网格区域中。此外，gridspec函数的功能更强大，也可以选择它来实现这个功能。



图5.8 子图样式

3. 坐标轴

坐标轴和子图的功能类似，不过它可以放在图像的任意位置，样例图如图5.9所示。因此，如果希望在一幅图中绘制一个小图，就可以用这个功能。

4. 刻度

良好的刻度是图像的重要组成部分。Matplotlib的刻度系统里的各个细节都可以由用户个性化配置。可以用Tick Locators来指定在哪些位置放置刻度，用Tick Formatters来调整刻度的样式。主要和次要的刻度可以以不同的方式呈现。默认情况下，每一个次要的刻度都是隐藏的，也就是说，次要刻度列表是空的（NullLocator）。坐标轴样例图如图5.9所示。

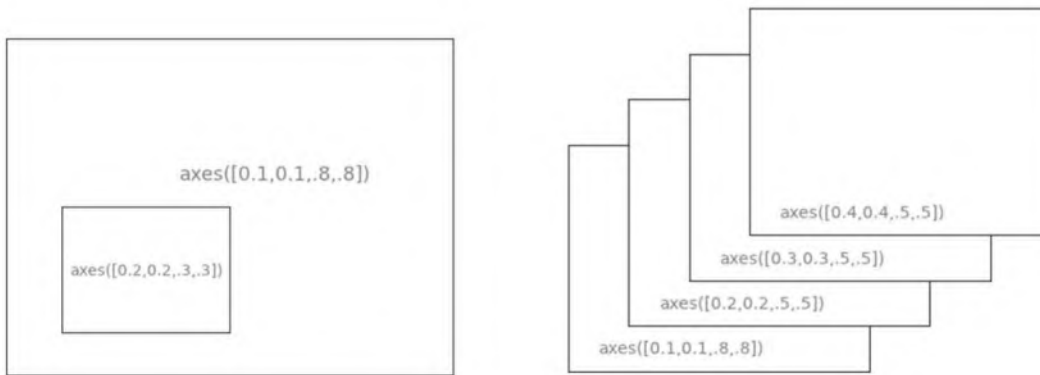
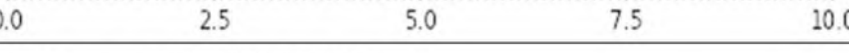
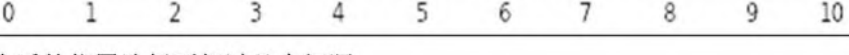


图5.9 坐标轴样例图

表5.2所示为不同需求设计的一些Locators。这些Locators都是`matplotlib.ticker.Locator`的子类，用户可以据此定义自己的Locator。以日期为刻度特别复杂，因此Matplotlib提供了`matplotlib.dates`来实现这一功能。

表5.2 刻度Locators

类型	说明
NullLocator	无标记 
IndexLocator	在每一个点的倍数上做一个标记 
FixedLocator	标记固定点 
LinearLocator	决定标记的位置 
MultipleLocator	在每一个基数的倍数上设置一个标记 
AutoLocator	在合适的位置选择不超过几个间隔 
LogLocator	预测确定日志轴的刻度位置 

这些定位器源于Matplotlib的基类`matplotlib.ticker.Locator`。用户可以源于它创建自己的定位器。处理时间刻度可能非常棘手，因此，Matplotlib在`matplotlib.dates`中提供了特殊的定位器。

5.3 其他种类的绘图

1. 常规绘图

下面尝试生成图5.10所示的图形。

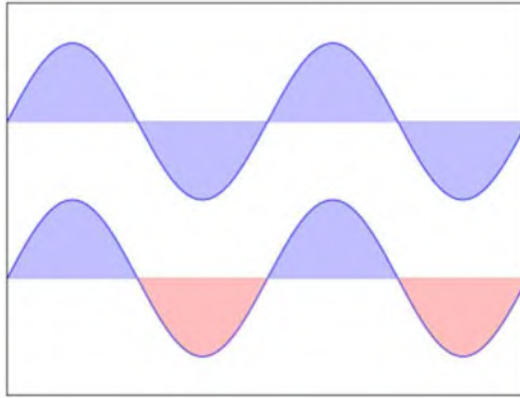


图5.10 普通图

```
#
-----
# Copyright (c) 2015, Nicolas P. Rougier. All Rights Reserved.
# Distributed under the (new) BSD License. See LICENSE.txt for more info.
#
-----

import numpy as np
import matplotlib.pyplot as plt

n = 256
X = np.linspace(-np.pi, np.pi, n, endpoint=True)
Y = np.sin(2*X)

plt.axes([0.025, 0.025, 0.95, 0.95])

plt.plot(X, Y+1, color='blue', alpha=1.00)
plt.fill_between(X, 1, Y+1, color='blue', alpha=.25)

plt.plot(X, Y-1, color='blue', alpha=1.00)
plt.fill_between(X, -1, Y-1, (Y-1) > -1, color='blue', alpha=.25)
plt.fill_between(X, -1, Y-1, (Y-1) < -1, color='red', alpha=.25)

plt.xlim(-np.pi, np.pi), plt.xticks([])
plt.ylim(-2.5, 2.5), plt.yticks([])
# savefig('../figures/plot_ex.png', dpi=48)
plt.show()
```

提示：需要使用fill_between命令。

2. 散点图

下面尝试生成图5.11所示的图形，注意标记大小、颜色和透明度。

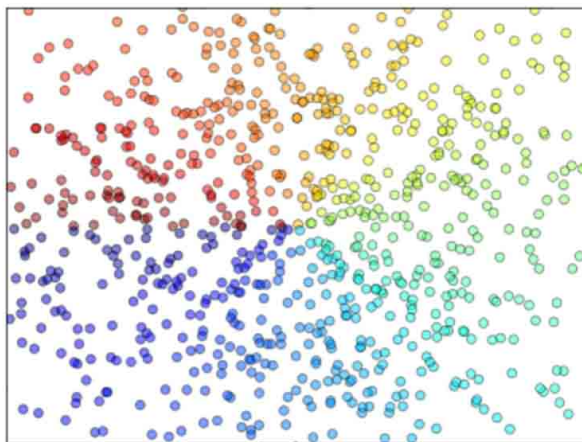


图5.11 散点图

```
#-----  
# Copyright (c) 2015, Nicolas P. Rougier. All Rights Reserved.  
# Distributed under the (new) BSD License. See LICENSE.txt for more info.  
#-----  
  
import numpy as np  
import matplotlib.pyplot as plt  
  
n = 1024  
X = np.random.normal(0,1,n)  
Y = np.random.normal(0,1,n)  
T = np.arctan2(Y,X)  
  
plt.axes([0.025,0.025,0.95,0.95])  
plt.scatter(X,Y, s=75, c=T, alpha=.5)  
  
plt.xlim(-1.5,1.5), plt.xticks([])  
plt.ylim(-1.5,1.5), plt.yticks([])  
# savefig('../figures/scatter_ex.png',dpi=48)  
plt.show()
```

提示：色彩由(X,Y)角度给出。

3. 条形图

下面尝试生成图5.12所示的图形。

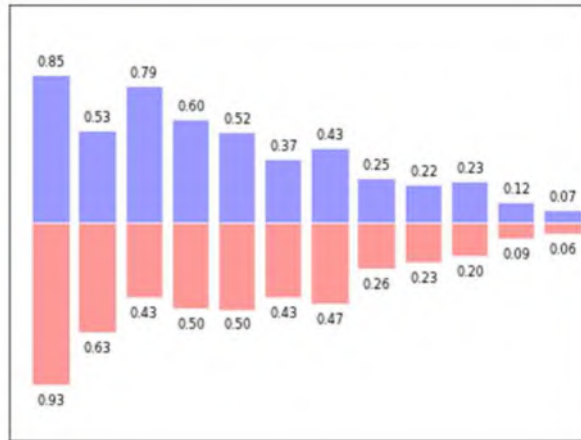


图5.12 条形图

```

#
-----
# Copyright (c) 2015, Nicolas P. Rougier. All Rights Reserved.
# Distributed under the (new) BSD License. See LICENSE.txt for more info.
#
-----

import numpy as np
import matplotlib.pyplot as plt

n = 12
X = np.arange(n)
Y1 = (1-X/float(n)) * np.random.uniform(0.5,1.0,n)
Y2 = (1-X/float(n)) * np.random.uniform(0.5,1.0,n)

plt.axes([0.025,0.025,0.95,0.95])
plt.bar(X, +Y1, facecolor='#9999ff', edgecolor='white')
plt.bar(X, -Y2, facecolor='#ff9999', edgecolor='white')

for x,y in zip(X,Y1):
    plt.text(x+0.4, y+0.05, '%.2f' % y, ha='center', va= 'bottom')

for x,y in zip(X,Y2):
    plt.text(x+0.4, -y-0.05, '%.2f' % y, ha='center', va= 'top')

plt.xlim(-.5,n), plt.xticks([])
plt.ylim(-1.25,+1.25), plt.yticks([])

# savefig('../figures/bar_ex.png', dpi=48)
plt.show()

```

提示：要注意文本对齐。

4. 等高线图

下面尝试生成图5.13所示的图形。

下面尝试生成图5.14所示的图形。

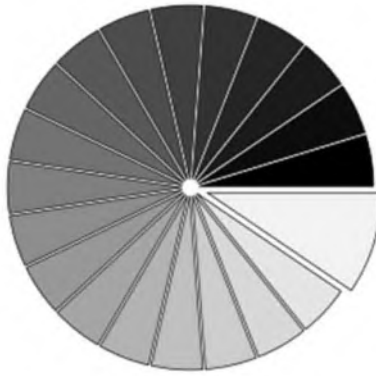


图5.14 饼图

```
#  
-----  
# Copyright (c) 2015, Nicolas P. Rougier. All Rights Reserved.  
# Distributed under the (new) BSD License. See LICENSE.txt for more info.  
#  
-----  
  
import numpy as np  
import matplotlib.pyplot as plt  
  
n = 20  
Z = np.ones(n)  
Z[-1] *= 2  
  
plt.axes([0.025,0.025,0.95,0.95])  
  
plt.pie(Z, explode=Z*.05, colors = ['%f' % (i/float(n)) for i in range(n)])  
plt.gca().set_aspect('equal')  
plt.xticks([], plt.yticks([]))  
  
# savefig('../figures/pie_ex.png',dpi=48)  
plt.show()
```

提示：需要改变Z。

6. 矢量图

下面尝试生成图5.15所示的图形。

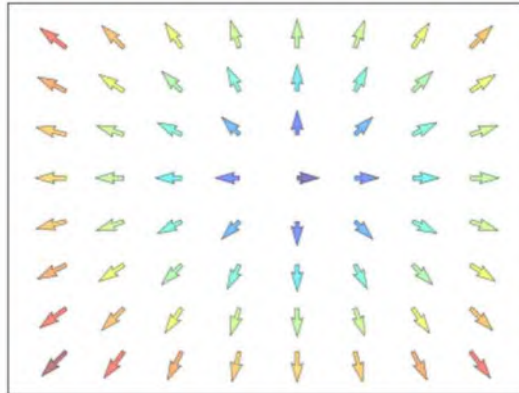


图5.15 矢量图

```
#  
  
# Copyright (c) 2015, Nicolas P. Rougier. All Rights Reserved.  
# Distributed under the (new) BSD License. See LICENSE.txt for more info.  
#  
  
import numpy as np  
import matplotlib.pyplot as plt  
  
n = 8  
X,Y = np.mgrid[0:n,0:n]  
T = np.arctan2(Y-n/2.0, X-n/2.0)  
R = 10+np.sqrt((Y-n/2.0)**2+(X-n/2.0)**2)  
U,V = R*np.cos(T), R*np.sin(T)  
  
plt.axes([0.025,0.025,0.95,0.95])  
plt.quiver(X,Y,U,V,R, alpha=.5)  
plt.quiver(X,Y,U,V, edgecolor='k', facecolor='None', linewidth=.5)  
  
plt.xlim(-1,n), plt.xticks([])  
plt.ylim(-1,n), plt.yticks([])  
  
# savefig('../figures/quiver_ex.png',dpi=48)  
plt.show()
```

提示：注意色彩和方向，你需要画两次箭头。

7. 极轴图

下面尝试生成图5.16所示的图形。

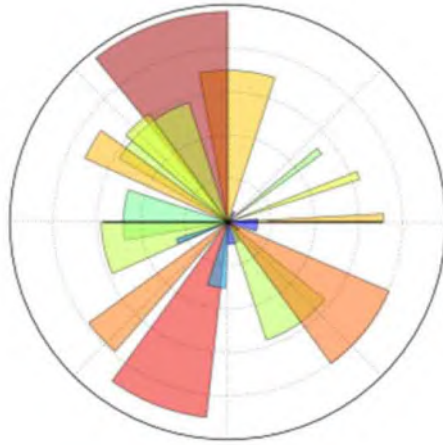


图5.16 极轴图

```
#  
-----  
# Copyright (c) 2015, Nicolas P. Rougier. All Rights Reserved.  
# Distributed under the (new) BSD License. See LICENSE.txt for more info.  
#  
-----  
  
import numpy as np  
import matplotlib.pyplot as plt  
  
ax = plt.axes([0.025,0.025,0.95,0.95], polar=True)  
  
N = 20  
theta = np.arange(0.0, 2*np.pi, 2*np.pi/N)  
radii = 10*np.random.rand(N)  
width = np.pi/4*np.random.rand(N)  
bars = plt.bar(theta, radii, width=width, bottom=0.0)  
  
for r,bar in zip(radii, bars):  
    bar.set_facecolor( plt.cm.jet(r/10.))  
    bar.set_alpha(0.5)  
  
ax.set_xticklabels([])  
ax.set_yticklabels([])  
# savefig('../figures/polar_ex.png',dpi=48)  
plt.show()
```

提示：仅仅需要修改axes这行。

8. 三维绘图

下面尝试生成图5.17所示的图形。

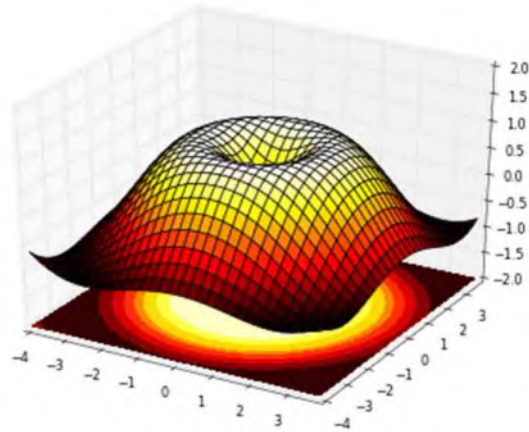


图5.17 三维图

```
# -----  
# Copyright (c) 2015, Nicolas P. Rougier. All Rights Reserved.  
# Distributed under the (new) BSD License. See LICENSE.txt for more info.  
# -----  
  
import numpy as np  
import matplotlib.pyplot as plt  
from mpl_toolkits.mplot3d import Axes3D  
  
fig = plt.figure()  
ax = Axes3D(fig)  
X = np.arange(-4, 4, 0.25)  
Y = np.arange(-4, 4, 0.25)  
X, Y = np.meshgrid(X, Y)  
R = np.sqrt(X**2 + Y**2)  
Z = np.sin(R)  
  
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=plt.cm.hot)  
ax.contourf(X, Y, Z, zdir='z', offset=-2, cmap=plt.cm.hot)  
ax.set_zlim(-2,2)  
  
# savefig('../figures/plot3d_ex.png',dpi=48)  
plt.show()
```

提示：需要使用contourf命令。

5.4 本章小结

Matplotlib是Python的绘图库。它可以与NumPy一起使用，提供了一种有效的Matlab开源替代方案。它也可以和图形工具包一起使用，如PyQt和wxPython。

本章主要介绍了Matplotlib初级绘图的使用法及其绘图种类。

第6章 大数据分析工具：Pandas

Pandas 是 Python 的一个数据分析包，最初由 AQR Capital Management 于 2008 年 4 月开发，并于 2009 年底开源出来，目前由专注于 Python 数据包开发的 PyData 开发团队继续开发和维护，属于 PyData 项目的一部分。Pandas 最初被作为金融数据分析工具而开发出来，因此 Pandas 为时间序列分析提供了很好的支持。Pandas 的名称来自于面板数据（Panel Data）和 Python 数据分析（Data Analysis）。面板数据是经济学中关于多维数据集的一个术语，在 Pandas 中提供了 Panel 的数据类型。

标准的 Python 发行版并没有将 Pandas 模块捆绑在一起发布。安装 Pandas 模块的一个轻量级的替代方法是使用流行的 Python 包安装程序用 pip 来安装 Pandas。Pandas 在 Python 上的安装同样使用 pip 进行：`pip install pandas`。

Pandas 处理以下 3 种数据结构：

- 系列（Series）。
- 数据帧（DataFrame）。
- 面板（Panel）。

6.1 Pandas 系列

系列（Series）是具有均匀数据的一维数组结构。Series 像 Python 中的数据类型 List 一样，每个数据都有自己的索引。

系列可以使用以下构造函数创建：

```
pandas.Series( data, index, dtype, copy)
```

- data: 数据采取各种形式, 如ndarray、list、constants。
- index: 索引值必须是唯一的和散列的, 与数据的长度相同。
默认为np.arange(n), 如果没有索引被传递。
- dtype: 用于数据类型。如果没有, 那么将推断数据类型。
- copy, 复制数据, 默认为false。

1. 从List创建 Series

【例6.1】

```
>>> import pandas as pd
>>> s1 = pd.Series([100,23,'bugingcode'])
>>> s1
0          100
1           23
2    bugingcode
dtype: object
>>>
```

2. 在Series 中添加相应的索引

【例6.2】

```
>>> import numpy as np
>>> ts = pd.Series(np.random.randn(365),
index=np.arange(1,366))
>>> ts
```

在index中设置索引值, 是一个从1到366的值。

3. 创建一个基本系列, 是一个空系列

【例6.3】

```
#import the pandas library and aliasing as pd
import pandas as pd
s = pd.Series()
print s
```

输出结果如下：

```
Series([], dtype: float64)
```

4. 从ndarray创建一个系列

【例6.4】

如果数据是ndarray，传递的索引就必须具有相同的长度。如果没有传递索引值，那么默认的索引将是范围（n），其中n是数组长度，即[0, 1, 2, 3... range(len(array))-1]-1]。

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = np.array(['a', 'b', 'c', 'd'])
s = pd.Series(data)
print s
```

输出结果如下：

```
0    a
1    b
2    c
3    d
dtype: object
```

这里没有传递任何索引，因此它分配了从0到len(data)-1的索引，即0到3。

5. 从字典创建一个系列

【例6.5】

字典（Dict）可以作为输入传递，如果没有指定索引，就按排序顺序取得字典键以构造索引。如果传递了索引，索引中与标签对应的数据中的值就会被拉出。

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data)
print s
```

输出结果如下：

```
a 0.0
b 1.0
c 2.0
dtype: float64
```

注意：字典键用于构建索引。

6. 从标量创建一个系列

【例6.6】

如果数据是标量值，就必须提供索引。将重复该值以匹配索引的长度。

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
s = pd.Series(5, index=[0, 1, 2, 3])
print s
```

输出结果如下：

```
0    5
1    5
2    5
```

```
3 5
dtype: int64
```

7. 从具有位置的系列中访问数据

【例6.7】

系列中的数据可以类似于访问ndarray中的数据来访问。

检索第一个元素。比如已经知道数组从0开始计数，第一个元素存储在0位置等。

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve the first element
print s[0]
```

得到以下结果：

```
1
```

8. 使用标签检索数据（索引）

【例6.8】

一个系列就像一个固定大小的字典，可以通过索引标签获取和设置值。使用索引标签值检索单个元素。

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve a single element
print s['a']
```

得到以下结果：

```
1
```

6.2 Pandas数据帧

数据帧（DataFrame）是二维数据结构，即数据以行和列的表格方式排列。

数据帧的功能特点如下：

- 潜在的列是不同的类型。
- 大小可变。
- 标记轴（行和列）。
- 可以对行和列执行算术运算。

Pandas中的DataFrame可以使用以下构造函数创建：

```
pandas.DataFrame( data, index, columns, dtype, copy)
```

- data：数据采取各种形式，如ndarray、series、map、lists、dict、constant和另一个DataFrame。
- index：对于行标签，要用于结果帧的索引是可选默认值np.arange(n)，如果没有传递索引值。
- columns：对于列标签，可选的默认语法是np.arange(n)。只有在没有索引传递的情况下才是这样。
- dtype：每列的数据类型。
- copy：如果默认值为False，此命令就用于复制数据。

Pandas数据帧可以使用各种输入创建：

- 列表。
- 字典。
- 系列。
- NumPy ndarrays。
- 另一个数据帧。

1. 创建一个空的数据帧

创建基本数据帧，是空数据帧。

【例6.9】

```
#import the pandas library and aliasing as pd
import pandas as pd
df = pd.DataFrame()
print df
```

得到以下结果：

```
Empty DataFrame
Columns: []
Index: []
```

2. 从列表创建数据帧

可以使用单个列表或嵌套多个创建数据帧。

【例6.10】

```
import pandas as pd
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print df
```

得到以下结果：

```
   0
0  1
1  2
2  3
3  4
4  5
```

3. 从ndarrays/Lists的字典创建数据帧

所有的ndarrays必须具有相同的长度。如果传递了索引（Index），索引的长度就应等于数组的长度。

如果没有传递索引，那么默认情况下，索引将为range(n)，其中n为数组长度。

【例6.11】

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':
[28,34,29,42]}
df = pd.DataFrame(data)
print df
```

得到以下结果：

	Age	Name
0	28	Tom
1	34	Jack
2	29	Steve
3	42	Ricky

注意：观察值0、1、2、3，它们是分配给每个使用函数range(n)的默认索引。

4. 从字典列表创建数据帧

字典列表可作为输入数据传递以用来创建数据帧，字典键默认为列名。

【例6.12】

以下代码显示如何通过传递字典列表来创建数据帧。

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
print df
```

得到以下结果：

	a	b	c
0	1	2	NaN
1	5	10	20.0

注意：观察到NaN（不是数字）被附加在缺失的区域。

5. 从系列的字典创建数据帧

字典的系列可以传递以形成一个数据帧。所得到的索引是通过索引的所有系列的并集。

【例6.13】

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print df
```

得到以下结果：

	one	two
a	1.0	1
b	2.0	2
c	3.0	3
d	NaN	4

注意：对于第一个系列，观察到没有传递标签'd'，但在结果中，对于d标签附加了NaN。

6. 列选择

下面将从数据帧中选择一列。

【例6.14】

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
print df ['one']
```

得到以下结果：

```
a    1.0
b    2.0
c    3.0
d     NaN
Name: one, dtype: float64
```

7. 行选择

可以通过将行标签传递给loc()函数来选择行。

【例6.15】

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print df.loc['b']
```

得到以下结果：

```
one 2.0
two 2.0
Name: b, dtype: float64
```

结果是一系列标签作为数据帧的列名称。而且，系列的名称是检索的标签。

6.3 Pandas面板

面板（Panel）是3D容器的数据。面板数据一词来源于计量经济学，部分源于名称：Pandas-pan(el)-da(ta)-s。

3个轴（Axis）这个名称旨在给出描述涉及面板数据的操作的一些语义。它们是：

- items-axis 0：每个项目对应于内部包含的数据帧。
- major_axis-axis 1：每个数据帧的索引（行）。
- minor_axis-axis 2：每个数据帧的列。

1. pandas.Panel()

使用以下构造函数创建面板：

```
pandas.Panel(data, items, major_axis, minor_axis, dtype,
copy)
```

构造函数的参数说明如下。

- data：数据采取各种形式，如ndarray、series、map、lists、dict、constant和另一个数据帧。
- items：axis=0。
- major_axis：axis=1。
- minor_axis：axis=2。
- dtype：每列的数据类型。
- copy：复制数据，默认为false。

2. 从3D ndarray创建面板

【例6.16】


```
# creating an empty panel
import pandas as pd
import numpy as np

data = np.random.rand(2,4,5)
p = pd.Panel(data)
print p
```

得到以下结果：

```
<class 'pandas.core.panel.Panel'>
Dimensions: 2 (items) x 4 (major_axis) x 5 (minor_axis)
Items axis: 0 to 1
Major_axis axis: 0 to 3
Minor_axis axis: 0 to 4
```

注意：观察空面板和上面板的尺寸大小，所有对象都不同。

3. 从数据帧对象的dict创建面板

【例6.17】

```
#creating an empty panel
import pandas as pd
import numpy as np
data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
        'Item2' : pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print p
```

得到以下结果：

```
<class 'pandas.core.panel.Panel'>
Dimensions: 2 (items) x 4 (major_axis) x 5 (minor_axis)
Items axis: 0 to 1
Major_axis axis: 0 to 3
Minor_axis axis: 0 to 4
```

4. 创建一个空面板

可以使用Panel的构造函数创建一个空面板：

【例6.18】

```
#creating an empty panel
import pandas as pd
p = pd.Panel()
print p
```

得到以下结果：

```
<class 'pandas.core.panel.Panel'>
Dimensions: 0 (items) x 0 (major_axis) x 0 (minor_axis)
Items axis: None
Major_axis axis: None
Minor_axis axis: None
```

5. 从面板中选择数据

要从面板中选择数据，可以使用以下方式：

- Items。
- Major_axis。
- Minor_axis。

(1) 使用Items

【例6.19】

```
# creating an empty panel
import pandas as pd
import numpy as np
data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
        'Item2' : pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print p['Item1']
```

得到以下结果：

	0	1	2
0	0.488224	-0.128637	0.930817
1	0.417497	0.896681	0.576657
2	-2.775266	0.571668	0.290082
3	-0.400538	-0.144234	1.110535

上面的示例有两个数据项，这里只检索item1。结果是具有4行和3列的数据帧，它们是Major_axis和Minor_axis维。

(2) 使用major_axis

可以使用panel.major_axis(index)方法访问数据。

【例6.20】

```
# creating an empty panel
import pandas as pd
import numpy as np
data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
        'Item2' : pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print p.major_xs(1)
```

得到以下结果：

	Item1	Item2
0	0.417497	0.748412
1	0.896681	-0.557322
2	0.576657	NaN

(3) 使用minor_axis

使用panel.minor_axis(index)方法访问数据。

【例6.21】

```
# creating an empty panel
import pandas as pd
import numpy as np
data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
        'Item2' : pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print p.minor_xs(1)
```

得到以下结果：

	Item1	Item2
0	-0.128637	-1.047032
1	0.896681	-0.557322
2	0.571668	0.431953
3	-0.144234	1.302466

注意：观察尺寸大小的变化。

6.4 Pandas快速入门

假设用户已安装Anaconda，现在启动Anaconda开始示例。测试工作环境是否已安装Pandas，导入相关包如下：

【例6.22】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
print("Hello, Pandas")
```

1. 对象创建

(1) 通过传递值列表来创建一个系列，让Pandas创建一个默认的整数索引。

【例6.23】

```
import pandas as pd
import numpy as np

s = pd.Series([1,3,5,np.nan,6,8])

print(s)
```

执行后输出结果如下：

```

runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64

```

(2) 通过传递NumPy数组，使用datetime索引和标记列来创建数据帧。

【例6.24】

```

import pandas as pd
import numpy as np

dates = pd.date_range('20180101', periods=7)
print(dates)

print("--"*16)
df = pd.DataFrame(np.random.randn(7,4), index=dates,
columns=list('ABCD'))
print(df)

```

执行后输出结果如下：

```

runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')
DatetimeIndex(['2018-01-01', '2018-01-02', '2018-01-03', '2018-01-04',
              '2018-01-05', '2018-01-06', '2018-01-07'],
              dtype='datetime64[ns]', freq='D')
-----
              A          B          C          D
2018-01-01 -2.577667  1.572909  0.261591 -0.622097
2018-01-02  0.339740  0.350866  1.156210 -1.031649
2018-01-03  0.214324  0.755249 -1.918018 -1.818146
2018-01-04 -0.272122 -0.556245  0.901669  1.986347
2018-01-05  0.349880 -0.361535 -0.639621  1.073277
2018-01-06  0.379895 -1.881535 -0.281546  1.845020
2018-01-07 -0.826106  0.039449 -0.198268 -1.153146

```

(3) 通过传递可以转换为类似系列的对象的字典来创建数据帧，参考以下示例代码：

【例6.25】

```
import pandas as pd
import numpy as np

df2 = pd.DataFrame({ 'A' : 1.,
                      'B' : pd.Timestamp('20181112'),
                      'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
                      'D' : np.array([3] * 4,dtype='int32'),
                      'E' : pd.Categorical(["test","train","test","train"]),
                      'F' : 'foo' })

print(df2)
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')
   A      B      C  D      E      F
0  1.0 2018-11-12  1.0  3   test   foo
1  1.0 2018-11-12  1.0  3  train   foo
2  1.0 2018-11-12  1.0  3   test   foo
3  1.0 2018-11-12  1.0  3  train   foo
```

2. 查看数据

(1) 查看框架的顶部和底部的数据行，参考以下示例代码：

【例6.26】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=7)
df = pd.DataFrame(np.random.randn(7,4), index=dates,
columns=list('ABCD'))
print(df.head())
print("-----" * 10)
print(df.tail(3))
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')
      A      B      C      D
2018-11-16 -0.322896  0.238304  1.323155 -0.783943
2018-11-17  0.139943  0.439021  1.246504 -0.638173
2018-11-18  0.501084  1.362034 -0.753448  0.488802
2018-11-19 -1.290280 -0.433372  1.771574 -0.322251
2018-11-20 -0.504033 -0.141004 -0.553480 -1.658107
-----
      A      B      C      D
2018-11-20 -0.504033 -0.141004 -0.553480 -1.658107
2018-11-21  1.331755  1.011874 -0.158176  0.740578
2018-11-22  0.746351 -0.435700 -0.819356  0.192756
```

(2) 显示索引、列和底层NumPy数据，参考以下代码：

【例6.27】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=7)
df = pd.DataFrame(np.random.randn(7,4), index=dates,
columns=list('ABCD'))
print("index is :")
print(df.index)
print("columns is :")
print(df.columns)
print("values is :")
print(df.values)
```

执行上面的示例代码后，输出结果如下：

```

runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')
index is :
DatetimeIndex(['2018-11-16', '2018-11-17', '2018-11-18', '2018-11-19',
               '2018-11-20', '2018-11-21', '2018-11-22'],
              dtype='datetime64[ns]', freq='D')
columns is :
Index(['A', 'B', 'C', 'D'], dtype='object')
values is :
[[ 1.14743672  1.27744643 -1.29618648  0.39069711]
 [-0.48348226 -0.85702047 -0.28095306 -2.28861502]
 [ 0.82365145 -0.84981074 -0.29244328 -2.45052732]
 [ 1.39668655  1.59837496  1.50166845 -1.23930783]
 [-0.61215929 -2.00881509 -0.08310447 -0.46637797]
 [ 0.25758817 -1.00247889 -0.86132333 -0.31025041]
 [ 1.67716193 -2.08318153 -1.28382321  0.29126785]]

```

(3) 描述显示数据的快速统计摘要，参考以下示例代码：

【例6.28】

```

import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=7)
df = pd.DataFrame(np.random.randn(7,4), index=dates,
columns=list('ABCD'))
print(df.describe())

```

执行上面的示例代码后，输出结果如下：

```

runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')

```

	A	B	C	D
count	7.000000	7.000000	7.000000	7.000000
mean	0.278102	0.442469	-0.471948	-0.349542
std	0.903016	0.763967	0.594038	0.925314
min	-1.163529	-0.694474	-1.461945	-1.578866
25%	-0.079049	-0.029647	-0.745407	-1.011644
50%	0.252560	0.652621	-0.445792	-0.387773
75%	0.627838	0.958775	-0.124039	0.300048
max	1.760107	1.280878	0.342991	0.943038

(4) 调换数据，参考以下示例代码：

【例6.29】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))
print(df.T)
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')
      2018-11-16  2018-11-17      ...      2018-11-20  2018-11-21
A      0.642181    0.075754      ...      1.497843   -0.997976
B      1.388360    0.341654      ...      0.659674   -0.781704
C     -1.498715    1.657816      ...      0.187711    0.866206
D      0.801267    0.794530      ...     -1.481244   -1.778276
```

(5) 通过轴排序，参考以下示例代码：

【例6.30】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))
print(df.sort_index(axis=1, ascending=False))
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')
      D      C      B      A
2018-11-16  0.392651  2.533227  0.144140  1.660218
2018-11-17  0.538997  1.290620  0.153564  0.109048
2018-11-18  0.823326  0.672298  0.040840 -0.309626
2018-11-19 -0.682314 -0.851813 -0.491239  1.323530
2018-11-20 -0.537231  0.420444 -1.626126 -0.014370
2018-11-21 -1.849932 -1.849171  1.357071  0.571947
```

(6) 按值排序，参考以下示例代码：

【例6.31】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))
print(df.sort_values(by='B'))
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')

      A      B      C      D
2018-11-19  0.433790 -1.523115  0.403432 -0.896684
2018-11-21  0.789746 -0.518838 -0.951376 -0.221729
2018-11-20 -1.039951 -0.467941 -0.494231  0.170267
2018-11-18  0.998579  0.078212 -1.825454 -0.363920
2018-11-17  0.313478  0.677431  1.040781  0.984559
2018-11-16 -0.612155  1.687324  0.724159 -0.535083
```

3. 选择区块

注意，虽然用于选择和设置的标准Python/NumPy表达式是直观的，可用于交互式工作，但对于生产代码，建议使用优化的Pandas数据访问方法`.at`、`.iat`、`.loc`、`.iloc`和`.ix`。

(1) 获取

①选择一列，产生一个系列，相当于`df.A`，参考以下示例代码：

【例6.32】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
```

```
columns=list('ABCD'))  
  
print(df['A'])
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',  
wdir='C:/Users/DLG/.spyder-py3')  
2018-11-16    -1.074002  
2018-11-17     1.440750  
2018-11-18     1.812373  
2018-11-19     0.877045  
2018-11-20     0.313987  
2018-11-21     0.425046  
Freq: D, Name: A, dtype: float64
```

②通过[]操作符选择切片行，参考以下示例代码：

【例6.33】

```
import pandas as pd  
import numpy as np  
  
dates = pd.date_range('20181116', periods=6)  
df = pd.DataFrame(np.random.randn(6,4), index=dates,  
columns=list('ABCD'))  
  
print(df[0:3])  
  
print("====指定选择日期====")  
  
print(df['20181116':'20181117'])
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')
      A      B      C      D
2018-11-16  0.038735 -1.136200  0.421581  0.425085
2018-11-17 -1.564266  0.724515  1.776787  0.747415
2018-11-18 -0.136437  2.079560 -1.015210  0.055421
===== 指定选择日期 =====
      A      B      C      D
2018-11-16  0.038735 -1.136200  0.421581  0.425085
2018-11-17 -1.564266  0.724515  1.776787  0.747415
```

（2）按标签选择

①使用标签获取横截面，参考以下示例代码：

【例6.34】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))

print(df.loc[dates[0]])
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')
A   -0.310334
B    0.033213
C    0.143139
D   -0.979762
Name: 2018-11-16 00:00:00, dtype: float64
```

②通过标签选择多轴，参考以下示例代码：

【例6.35】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=6)
```

```
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))

print(df.loc[:,['A','B']])
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')

      A      B
2018-11-16  0.915099  1.851329
2018-11-17 -1.183457  0.025724
2018-11-18  0.500575  0.313305
2018-11-19 -0.404541  0.121724
2018-11-20  0.032216 -0.531506
2018-11-21  0.945294 -1.074906
```

③显示标签切片，包括两个端点，参考以下示例代码：

【例6.36】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))

print(df.loc['20181116':'20181117',['A','B']])
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')

      A      B
2018-11-16  0.774270  0.028909
2018-11-17 -0.267173  0.066412
```

④减少返回对象的尺寸（大小），参考以下示例代码：

【例6.37】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20170101', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))

print(df.loc['20181116',['A','B']])
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')
A   -1.429766
B    1.101248
Name: 2018-11-16 00:00:00, dtype: float64
```

⑤获得标量值，参考以下示例代码：

【例6.38】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))

print(df.loc[dates[0],'A'])
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')
-1.709403357791696
```

⑥快速访问标量（等同于先前的方法），参考以下示例代码：

【例6.39】

```
import pandas as pd
import numpy as np
```

```
dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))

print(df.at[dates[0], 'A'])
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')
-0.38763045860339174
```

(3) 通过位置选择

①通过传递的整数的位置选择，参考以下示例代码：

【例6.40】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))

print(df.iloc[3])
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')
A    1.105575
B   -0.441827
C    0.230463
D    1.308419
Name: 2018-11-19 00:00:00, dtype: float64
```

②通过整数切片，类似于NumPy/Python，参考以下示例代码：

【例6.41】

```
import pandas as pd
import numpy as np
```

```

dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))

print(df.iloc[3:5,0:2])

```

执行上面的示例代码后，输出结果如下：

```

runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')

```

	A	B
2018-11-19	-1.097961	-0.063939
2018-11-20	-0.530755	1.535225

③整数位置的列表，类似于NumPy/Python样式，参考以下示例代码：

【例6.42】

```

import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))

print(df.iloc[[1,2,4],[0,2]])

```

执行上面的示例代码后，输出结果如下：

```

runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')

```

	A	C
2018-11-17	0.645021	0.711071
2018-11-18	-1.366864	-0.623005
2018-11-20	0.248483	-0.942709

④明确切片行，参考以下示例代码：

【例6.43】


```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))

print(df.iloc[1:3,:])
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')

      A      B      C      D
2018-11-17 -0.447935 -0.021304 -0.643159 -0.952132
2018-11-18  0.832708 -0.864145 -0.943967  0.593821
```

⑤明确切片列，参考以下示例代码：

【例6.44】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))

print(df.iloc[:,1:3])
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')

      B      C
2018-11-16  1.037712 -2.346177
2018-11-17  0.363997 -0.102346
2018-11-18  0.530748  0.180533
2018-11-19 -1.133693  1.336118
2018-11-20 -2.550474 -0.569274
2018-11-21  0.255712  1.190824
```

⑥明确获取值，参考以下示例代码：

【例6.45】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))

print(df.iloc[1,1])
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')
-1.4410621738508806
```

⑦快速访问标量（等同于先前的方法），参考以下示例代码：

【例6.46】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))

print(df.iat[1,1])
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')
-1.1139807251032545
```

（4）布尔索引

①使用单列的值来选择数据，参考以下示例代码：

【例6.47】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))

print(df[df.A > 0])
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')

      A      B      C      D
2018-11-16  1.849350 -0.216369 -0.722925  1.283104
2018-11-17  0.480921  0.500803  1.330278  0.281972
2018-11-19  1.251439 -1.361345  1.024343  1.759472
2018-11-20  1.465461 -0.084952  0.261507  0.804989
```

②从满足布尔条件的数据帧中选择值，参考以下示例代码：

【例6.48】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))

print(df[df > 0])
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')

      A      B      C      D
2018-11-16  NaN  0.778964  1.722535  NaN
2018-11-17  NaN   NaN     NaN     0.108527
2018-11-18  NaN  0.056008  0.009676  NaN
2018-11-19  0.569408  0.566211   NaN     1.048052
2018-11-20  NaN  0.230177  1.733346  NaN
2018-11-21  NaN  0.208917  0.092021  NaN
```

③使用isin()方法进行过滤，参考以下示例代码：

【例6.49】

```
import pandas as pd
import numpy as np

dates = pd.date_range('20181116', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))

df2 = df.copy()
df2['E'] = ['one', 'one', 'two', 'three', 'four', 'three']

print(df2)

print("===== start to filter ===== ")

print(df2[df2['E'].isin(['two', 'four'])])
```

执行上面的示例代码后，输出结果如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')

      A      B      C      D      E
2018-11-16  2.447493  0.318740 -0.012785  1.538562  one
2018-11-17  0.648087  1.269157  1.280157 -0.953285  one
2018-11-18 -0.375997 -0.808582  3.055245 -1.128035  two
2018-11-19 -0.268321  0.285642  0.723543 -0.783271  three
2018-11-20 -0.020216  1.292080 -2.111235  0.689906  four
2018-11-21  1.472228  0.260298 -0.625512  0.280217  three
===== start to filter =====
      A      B      C      D      E
2018-11-18 -0.375997 -0.808582  3.055245 -1.128035  two
2018-11-20 -0.020216  1.292080 -2.111235  0.689906  four
```

6.5 本章小结

Pandas是基于NumPy的一种工具，该工具是为了解决数据分析任务而创建的。Pandas纳入了大量库和一些标准的数据模型，提供了高效地操作大型数据集所需的工具。Pandas提供了大量能快速便捷地处理

数据的函数和方法。本章主要介绍Pandas大数据分析的基础操作，同时有针对性地介绍了Pandas的系列、数据帧和面板等功能。

第7章 大数据分析工具：Statsmodels与Gensim

Statsmodels是一个Python模块，它提供对许多不同统计模型估计的类和函数，并且可以进行统计测试和统计数据的探索。

Gensim是一个用于从文档中自动提取语义主题的Python库，足够智能。Gensim可以处理原生、非结构化的数值化文本（纯文本）。

7.1 Statsmodels

Statsmodels是一个有很多统计模型的Python库，能完成很多统计测试、数据探索以及可视化。它还包含一些经典的统计方法，比如贝叶斯方法和一个机器学习的模型。

Statsmodels中的模型和方法包括：

- 线性模型（linear models）、广义线性模型（generalized linear models）和鲁棒线性模型（robust linear models）。
- 线性混合效应模型（Linear mixed effects models）。
- 方差分析（Analysis of Variance, ANOVA）方法。
- 时间序列处理（Time Series Processes）和状态空间（State Space）模型。
- 广义矩估计方法（Generalized Method of Moments）。

7.1.1 Statsmodels统计数据库

Statsmodels包含统计模型和统计数据的库。这个库里有样本数据可以提供项目训练。

以下代码列出了这个库包含的所有数据和每个数据的简短介绍。

【例7.1】

```
#coding:utf-8

import statsmodels.api as sm
from pandas import DataFrame

dataDict = {'name':[], 'describe_short':[]}

for modstr in dir(sm.datasets):
    try:
        mod = eval('sm.datasets.%s' % modstr)
        dataDict['describe_short'].append(mod.DESCRSHORT)
        dataDict['name'].append(modstr)
    except Exception as e:
        print("该模块无 DESCRSHORT 属性:\n", e)
        continue

dataDf = DataFrame({'describe_short':dataDict['describe_short']},
index=dataDict['name'])
print(dataDf)
```

输出结果如下（可以看到这个库里的数据还是比较多的，如 sunspots、scotland、china_smoking等，可供项目训练使用）：

anes96	This data is a subset of the American National...
cancer	Breast Cancer and county population
ccard	William Greene's credit scoring data
china_smoking	Co-occurrence of lung cancer and smoking in 8 ...
co2	Atmospheric CO2 from Continuous Air Samples at...
committee	Number of bill assignments in the 104th House ...
copper	World Copper Market 1951-1975
cpunish	Number of state executions in 1997
elnino	Averaged monthly sea surface temperature - Pac...
engel	Engel food expenditure data.
fair	Extramarital affair data.
fertility	Total fertility rate represents the number of ...
grunfeld	Grunfeld (1950) Investment Data for 11 U.S. Fi...
heart	Survival times after receiving a heart transplant
interest	inflation (West) German interest and inflation rate
1972...	
longley	
macrodata	US Macroeconomic Data for 1959Q1 - 2009Q3
modechoice	Data used to study travel mode choice between ...
nile	This dataset contains measurements on the annu...
randhie	The RAND Co. Health Insurance Experiment Data
scotland	Taxation Powers' Yes Vote for Scottish Parliam...
spector	Experimental data on the effectiveness of the ...
stackloss	Stack loss plant data of Brownlee (1965)
star98	Math scores for 303 student with 10 explanator...
statecrime	State crime data 2009
strikes	Contains data on the length of strikes in US m...
sunspots	Yearly (1700-2008) data on sunspots from the N...

那么如何调用一组数据呢？以scotland为例，想要查看scotland的数据，就用下面的几行代码。

【例7.2】

```
#coding:utf-8

import statsmodels.api as sm
from pandas import DataFrame

china_smoking_data = sm.datasets.china_smoking.load_pandas()
# print(type(scotland_data))
# print(scotland_data)
df = china_smoking_data.data
print(type(df)) # DataFrame 类型的数据
print(df)
```


代码输出如下：

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')
<class 'pandas.core.frame.DataFrame'>
```

	smoking_yes_cancer_yes	...	smoking_no_cancer_no
Location		...	
Beijing	126	...	61
Shanghai	908	...	807
Shenyang	913	...	598
Nanjng	235	...	121
Harbin	402	...	215
Zhengzhou	182	...	98
Taiyuan	60	...	43
Nanchang	104	...	36

```
[8 rows x 4 columns]
```

7.1.2 Statsmodels典型的拟合模型概述

1. 模型拟合和描述

Statsmodels典型的拟合模型涉及3个简单的步骤：

```
# step 1 Describe model
mod = sm.OLS(y, X)
# step 2 Fit model
res = mod.fit()
# step 3 Summarize model
print(res.summary())

>>> res.params    # 获取模型参数
>>> dir(res)      # 查看完整的属性列表
```

2. 诊断和规格测试

```
sm.stats.linear_rainbow(res)
```

3. 使用R型公式来拟合模型

```
import statsmodels.formula.api as smf
```

4. endog、exog参数

Statsmodels使用endog和exog为模型数据参数名称，作为估计器的观测变量。

endog	exog
y	x
y variable	x variable
left hand side (LHS)	right hand side (RHS)
dependent variable (因变量)	independent variable (自变量)
regressand	regressors
outcome	design
response variable (响应变量)	explanatory variable (解释变量)

7.1.3 Statsmodels举例

1. 线性模型

【例7.3】

```
# Load modules and data
import numpy as np
import statsmodels.api as sm
spector_data = sm.datasets.spector.load()
spector_data.exog = sm.add_constant(spector_data.exog,
prepend=False)
```

```
# Fit and summarize OLS model
mod = sm.OLS(spector_data.endog, spector_data.exog)
res = mod.fit()
print(res.summary())
```

输出结果:

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')

OLS Regression Results

=====
Dep. Variable:          y          R-squared:          0.416
Model:                  OLS Adj.   R-squared:          0.353
Method:                 Least Squares   F-statistic:        6.646
Date:                   Wed, 12 Dec 2018   Prob (F-statistic):  0.00157
Time:                   14:05:09          Log-Likelihood:     -12.978
No. Observations:      32                AIC:               33.96
Df Residuals:          28                BIC:               39.82
Df Model:               3
Covariance Type:       nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
x1	0.4639	0.162	2.864	0.008	0.132	0.796
x2	0.0105	0.019	0.539	0.594	-0.029	0.050
x3	0.3786	0.139	2.720	0.011	0.093	0.664
const	-1.4980	0.524	-2.859	0.008	-2.571	-0.425

```
=====
Omnibus:                0.176   Durbin-Watson:          2.346
Prob(Omnibus):          0.916   Jarque-Bera (JB):        0.167
Skew:                   0.141   Prob(JB):                0.920
Kurtosis:               2.786   Cond. No.                176.
=====
```

2. 广义线性模型

【例7.4】

```
import statsmodels.api as sm
data = sm.datasets.scotland.load()
data.exog = sm.add_constant(data.exog)
# Instantiate a gamma family model with the default link
```

```
function.
gamma_model = sm.GLM(data.endog, data.exog,
family=sm.families.Gamma())
gamma_results = gamma_model.fit()
print(gamma_results.summary())
```

输出结果:

```
runfile('C:/Users/DLG/.spyder-py3/temp.py',
wdir='C:/Users/DLG/.spyder-py3')
d:\Anaconda3\lib\site-packages\statsmodels\genmod\generalized_linear_model.
py:302: DomainWarning: The inverse_power link function does not respect the domain
of the Gamma family.
DomainWarning)

Generalized Linear Model Regression Results

=====
Dep. Variable:          y          No. Observations:          32
Model:                  GLM          Df Residuals:           24
Model Family:           Gamma          Df Model:             7
Link Function:          inverse_power  Scale:                 0.0035843
Method:                 IRLS          Log-Likelihood:        - 83.017
Date:                   Wed, 12 Dec 2018  Deviance:             0.087389
Time:                   14:30:39          Pearson chi2:          0.0860
No. Iterations:         6              Covariance Type:      nonrobust
=====
coef      std err        z      P>|z|    [0.025    0.975]
-----
const    -0.0178     0.011    -1.548    0.122    -0.040     0.005
x1       4.962e-05   1.62e-05   3.060    0.002     1.78e-05   8.14e-05
x2       0.0020       0.001     3.824    0.000     0.001     0.003
x3      -7.181e-05   2.71e-05   -2.648    0.008    -0.000    -1.87e-05
x4       0.0001       4.06e-05   2.757    0.006     3.23e-05   0.000
x5      -1.468e-07   1.24e-07   -1.187    0.235    -3.89e-07   9.56e-08
x6      -0.0005       0.000     -2.159    0.031    -0.001    -4.78e-05
x7      -2.427e-06   7.46e-07   -3.253    0.001    -3.89e-06   -9.65e-07
=====
```

7.2 Gensim

Gensim是一款开源的第三方Python工具包，用于从原始的非结构化文本中无监督地学习文本隐层的主题向量表达。它支持包括TF-

IDF、LSA、LDA和Word2vec在内的多种主题模型算法，支持流式训练，并提供了诸如相似度计算、信息检索等一些常用任务的API接口。

从宏观来看，Gensim提供了一个发现文档语义结构的工具，用于检查词出现的频率。Gensim读取一段语料（Corpus），输出一个向量（Vector），表示文档中的一个词。词向量可以用来训练各种分类器模型。这3个模型是Gensim的核心概念。

7.2.1 基本概念

- 语料：一组原始文本的集合，用于无监督地训练文本主题的隐层结构。语料中不需要人工标注附加信息。在Gensim中，语料通常是一个可迭代的对象（比如列表）。每一次迭代返回一个可用于表达文本对象的稀疏向量。
- 向量：由一组文本特征构成的列表，是一段文本在Gensim中的内部表达。
- 稀疏向量（Sparse Vector）：通常可以略去向量中多余的0元素。此时，向量中的每一个元素是一个(key, value)的元组。
- 模型：一个抽象的术语，定义了两个向量空间的变换（从文本的一种向量表达变换为另一种向量表达）。

7.2.2 训练语料的预处理

语料是指一组电子文档的集合。这个集合是Gensim的输入，Gensim会从这个语料中推断出它的结构、主题等。从语料中推断出的隐含结构可以用来对一个新的文档指定一个主题。我们也把这个集合叫作训练语料。这个训练过程不需要人工参与，所以主题分类是无监督的。

【例7.5】

```
raw_corpus = ["This software depends on NumPy and Scipy",
              "two Python packages for scientific computing",
              "You must have them installed prior to installing gensim",
              "It is also recommended you install a fast BLAS library
before installing NumPy",
              "This is optional, but using an optimized BLAS such as ATLAS
or OpenBLAS
              is known to improve performance by as much as an order
of magnitude",
              "On OS X, NumPy picks up the BLAS that comes with it
automatically",
              "so you don't need to do anything special"]
```

收集语料之后，需要做一些预处理。这里的预处理比较简单，移除了一些英文虚词以及在预料中仅出现一次的词。这个处理过程需要对数据进行单词切分。通常，我们需要先对原始的文本进行分词、去除停用词等操作，得到每一篇文档的特征列表。把文档分割成一个个单词字典（这里以空格作为分隔符）。

```
stoplist = set('for a of the and to in'.split(' '))
texts = [[word for word in document.lower().split() if word not in stoplist]
          for document in raw_corpus]

from collections import defaultdict
frequency = defaultdict(int)
for text in texts:
    for token in text:
        frequency[token] += 1

precessed_corpus = [[token for token in text if frequency[token] > 1] for text
in texts]
precessed_corpus
```

分词后的特征列表：

```
texts = ['this', 'software', 'depends', 'on', 'numpy',
'scipy']
['two', 'python', 'packages', 'scientific', 'computing']
['you', 'must', 'have', 'them', 'installed', 'prior',
'installing', 'gensim']
```

```

['it', 'is', 'also', 'recommended', 'you', 'install',
'fast', 'blas', 'library',
'before', 'installing', 'numpy']
['this', 'is', 'optional,', 'but', 'using', 'an',
'optimized', 'blas', 'such',
'as', 'atlas', 'or', 'openblasis', 'known', 'improve',
'performance', 'by', 'as',
'much', 'as', 'an', 'order', 'magnitude']
['on', 'os', 'x,', 'numpy', 'picks', 'up', 'blas', 'that',
'comes', 'with', 'it',
'automatically']
['so', 'you', 'don't', 'need', 'do', 'anything', 'special']

```

其中，corpus的每一个元素对应一篇文档。

接下来，我们可以调用Gensim提供的API建立语料特征（此处就是word）的索引字典，并将文本特征的原始表达转化成词袋模型对应的稀疏向量的表达。依然以词袋模型为例：

```

from gensim import corpora
dictionary = corpora.Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]
print (corpus[0] )

```

输出：

```
[(0, 1), (1, 1), (2, 1)]
```

到这里，训练语料的预处理工作就完成了。我们得到了语料中每一篇文档对应的稀疏向量（这里是bow向量）。向量的每一个元素代表一个word在这篇文档中出现的次数。值得注意的是，虽然词袋模型是很多主题模型的基本假设，但是这里介绍的doc2bow函数并不是将文本转化成稀疏向量的唯一途径。技术手册中有更多的向量变换函数。

最后，出于内存优化的考虑，Gensim支持文档的流式处理。我们需要做的只是将前面的列表封装成一个Python迭代器，每一次迭代都返回一个稀疏向量。

```
class MyCorpus(object):
    def __iter__(self):
        for line in open('mycorpus.txt'):
            # assume there's one document per line, tokens separated by whitespace
            yield dictionary.doc2bow(line.lower().split())
```

输出：

```
[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1)]
```

7.2.3 主题向量的变换

对文本向量的变换是Gensim的核心。通过挖掘语料中隐藏的语义结构特征，我们最终可以变换出一个简洁高效的文本向量。

在Gensim中，每一个向量变换的操作都对应着一个主题模型，例如上述提到的对应词袋模型的doc2bow变换。每一个模型又都是一个标准的Python对象。下面以TF-IDF模型为例介绍Gensim模型的一般使用方法。

首先是模型对象的初始化。通常，Gensim模型都接收一段训练语料（注意在Gensim中，语料对应一个稀疏向量的迭代器）作为初始化的参数。显然，越复杂的模型需要配置的参数越多。

【例7.6】

```
from gensim import models
tfidf = models.TfidfModel(corpus)
```

其中，corpus是一个返回bow向量的迭代器。这两行代码将完成对corpus中出现的每一个特征>IDF值的统计工作。

接下来，我们可以调用这个模型将任意一段语料（依然是bow向量的迭代器）转化成TFIDF向量（的迭代器）。需要注意的是，这里的bow向量必须与训练语料的bow向量共享同一个特征字典（共享同一个向量空间）。


```
doc_bow = [(0, 1), (1, 1)]
print (tfidf[doc_bow])
```

输出：

```
[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1)]
[(0, 0.9168545678312987), (1, 0.39922136897576327)]
```

注意，同样是出于内存的考虑，`model[corpus]`方法返回的是一个迭代器。如果要多次访问`model[corpus]`的返回结果，那么可以先将结果向量序列化到磁盘上。

我们也可以将训练好的模型持久化到磁盘上，以便下一次使用：

```
tfidf.save("./model.tfidf")
tfidf = models.TfidfModel.load("./model.tfidf")
```

Gensim内置了多种主题模型的向量变换，包括LDA、LSI、RP、HDP等。这些模型通常以bow向量或tfidf向量的语料为输入，生成相应的主题向量。所有的模型都支持流式计算。

7.2.4 文档相似度的计算

在得到每一篇文档对应的主题向量后，我们就可以计算文档之间的相似度，进而完成如文本聚类、信息检索之类的任务。在Gensim中提供了这一类任务的API接口。

以信息检索为例，对于一篇待检索的query，我们的目标是从文本集合中检索出主题相似度最高的文档。

首先，我们需要将待检索的query和文本放在同一个向量空间里进行表达（以LSI向量空间为例）：

```
# 构造LSI模型并将待检索的query和文本转化为LSI主题向量
# 转换之前的corpus和query均是BOW向量
lsi_model = models.LsiModel(corpus, id2word=dictionary,
```

```
num_topics=2)
documents = lsi_model[corpus]
query_vec = lsi_model[query]
```

接下来，我们用待检索的文档向量初始化一个相似度计算的对象：

```
index = similarities.MatrixSimilarity(documents)
```

我们也可以通过save()和load()方法持久化这个相似度矩阵：

```
index.save('/tmp/deerwester.index')
index =
similarities.MatrixSimilarity.load('/tmp/deerwester.index')
```

注意，如果待检索的目标文档过多，使用similarities.MatrixSimilarity类往往会带来内存不够用的问题。此时，可以改用similarities.Similarity类。二者的接口基本保持一致。

最后，我们借助index对象计算任意一段query和所有文档的（余弦）相似度：

```
sims = index[query_vec] # return: an iterator of tuple (idx,
sim)
```

7.3 本章小结

Pandas着眼于数据的读取、处理和探索，而Statsmodels则更加注重数据的统计建模分析，它使得Python有了R语言的味道。Statsmodels支持与Pandas进行数据交互，因此它与Pandas结合成为Python下强大的数据挖掘组合。

Gensim里面的算法，比如Latent Semantic Analysis（潜在语义分析，LSA）、Latent Dirichlet Allocation（隐狄利克雷分布，

LAD)、Random Projections (随机投影), 通过在语料库的训练下检验词的统计共生模式 (Statistical Co-Occurrence Patterns) 来发现文档的语义结构。

第8章 大数据分析算法与实例

Python是现在最受欢迎的动态编程语言之一（还有Perl、Ruby等）。Python不但拥有强大的数据处理功能，而且完全可以用它构建生产系统。在众多解释型语言中，由于各种历史和文化的因素，Python发展出了一个巨大而活跃的科学计算（Scientific Computing）社区。在过去的10年，Python从一个边缘或“自担风险”的科学计算语言成为数据科学、机器学习、学界和工业界软件开发最重要的语言之一。

数据分析与可视化是指对数据进行控制、处理、整理、分析的过程。数据分析和建模大部分时间都用在数据准备上，数据的准备过程包括：加载、清理、转换与重塑。

本章主要讲述基于Python的数据分析方法，尽量翔实地阐述Python数据分析方法的原理与实践。

8.1 描述统计

统计学分为描述统计学和推断统计学。描述统计学是使用特定的数字或图表来体现数据的集中程度或离散程度，如平均数、中位数、众数、方差、标准差；推断统计学是根据样本数据来推断总体特征，如产品检查，一般采用抽检，根据所抽样本的质量合格率作为总体质量合格率的一个估计。

在数值分析的过程中，我们往往要计算出数据的统计特征，用来做科学计算的NumPy和SciPy工具可以满足我们的需求。Matplotlib工具可用来绘制图形，满足图分析的需求。

1. 基本概念

与Python中原生的List类型不同，NumPy中用ndarray类型来描述一组数据：

```
from numpy import array
from numpy.random import normal, randint
#使用List来创造一组数据
data = [1, 2, 3]
#使用ndarray来创造一组数据
data = array([1, 2, 3])
#创造一组服从正态分布的定量数据
data = normal(0, 10, size=10)
#创造一组服从均匀分布的定性数据
data = randint(0, 10, size=10)
```

2. 中心位置（均值、中位数、众数）

数据的中心位置是我们最容易想到的数据特征。借由中心位置，我们可以知道数据的一个平均情况，如果要对新数据进行预测，那么平均情况是非常直观的选择。数据的中心位置可分为均值（Mean）、中位数（Median）和众数（Mode）。其中，均值和中位数用于定量的数据，众数用于定性的数据。

对于定量数据来说，均值是总和除以总量（N），中位数是数值大小位于中间（奇偶总量处理不同）的值：

$$\text{Mean} = \frac{\sum_i^N \text{Data}[i]}{N} \quad (8.1)$$

$$\text{Median} = \begin{cases} \text{Select}(\text{Data}, \frac{N+1}{2}), N\%2 == 1 \\ \frac{\text{Select}(\text{Data}, \frac{N}{2}) + \text{Select}(\text{Data}, \frac{N}{2} + 1)}{2}, N\%2 == 0 \end{cases} \quad (8.2)$$

均值相对中位数来说，包含的信息量更大，但是容易受异常的影响。使用NumPy计算均值与中位数：

```
from numpy import mean, median
#计算均值
mean(data)
#计算中位数
median(data)
```

对于定性数据来说，众数是出现次数最多的值，使用SciPy计算众数：

```
from scipy.stats import mode
#计算众数
mode(data)
```

3. 发散程度（极差、方差、标准差、变异系数）

对数据的中心位置有所了解以后，一般我们会想知道数据以中心位置为标准有多发散。如果以中心位置来预测新数据，那么发散程度决定了预测的准确性。数据的发散程度可用极差（PTP）、方差（Variance）、标准差（STD）、变异系数（CV）来衡量，它们的计算方法如下：

$$PTP = \text{Max}(\text{Data}) - \text{Min}(\text{Data}) \quad (8.3)$$

$$\text{Variance} = \frac{\sum_i^N (\text{Data}[i] - \text{Mean})^2}{N} \quad (8.4)$$

$$\text{STD} = \sqrt{\text{Variance}} \quad (8.5)$$

$$\text{CV} = \frac{\text{STD}}{\text{Mean}} \quad (8.6)$$

极差是只考虑了最大值和最小值的发散程度指标；相对来说，方差包含更多的信息；标准差基于方差，但是与原始数据同量级；变异系数基于标准差，但是进行了无量纲处理。使用NumPy计算极差、方差、标准差和变异系数：

```
from numpy import mean, ptp, var, std
#极差
ptp(data)
#方差
var(data)
#标准差
std(data)
#变异系数
mean(data) / std(data)
```

4. 偏差程度（z-分数）

之前提到均值容易受异常值影响，那么如何衡量偏差、偏差到多少算异常是两个必须要解决的问题。定义z-分数（Z-Score）为测量值距均值相差的标准差数目：

$$Z\text{-Score} = \frac{X - \text{Mean}}{STD} \quad (8.7)$$

当标准差不为0且不为接近于0的数时，z-分数是有意义的，使用NumPy计算z-分数：

```
from numpy import mean, std
#计算第一个值的z-分数
(data[0]-mean(data)) / std(data)
```

5. 相关程度

有两组数据时，我们关心这两组数据是否相关，相关程度是多少。用协方差（COV）和相关系数（CORRCOEF）来衡量相关程度：

$$COV = \frac{\sum_i^N (Data_1[i] - Mean_1) * (Data_2[i] - Mean_2)}{N} \quad (8.8)$$

$$CORRCOEF = \frac{COV}{STD_1 * STD_2} \quad (8.9)$$

协方差的绝对值越大表示相关程度越高，协方差为正值表示正相关，负值为负相关，0为不相关。相关系数基于协方差，但进行了无量纲处理。使用NumPy计算协方差和相关系数：

```
from numpy import array, cov, corrcoef

data = array([data1, data2])

#计算两组数的协方差
#参数bias=1表示结果需要除以N，否则只计算了分子部分
#返回结果为矩阵，第i行第j列的数据表示第i组数与第j组数的协方差。对角线为方差
cov(data, bias=1)
#计算两组数的相关系数
#返回结果为矩阵，第i行第j列的数据表示第i组数与第j组数的相关系数。对角线为1
corrcoef(data)
```

6. 利用Matplotlib画图举例

利用Python画图需要使用Matplotlib库。

【例8.1】

```
import matplotlib.pyplot as plt
```

创建一组数据，该数据为30个中国CBA球员的体重，其中“\”表示换行接着写。

```
weight = [225,232,232,245,235,245,270,225,240,240,\
          217,195,225,185,200,220,200,210,271,240,\
          220,230,215,252,225,220,206,185,227,236]
```


(1) 直方图画法

- 找出最大值与最小值，确定数据的范围。
- 整理数据，将数据分为几组（尽量使每组都有数据），计算频数分布表。
- 根据频数分布表画出频数直方图。频数为纵坐标，分组类别为横坐标。通过直方图可以对数据分布有一个直观的了解。
- 除了频数直方图，还有频率直方图外。频率直方图的纵坐标为频率 / 组距。频率=频数 / 总数，组距是分组的极差。

```
from pylab import mpl #显示中文设置
mpl.rcParams['font.sans-serif'] = ['SimHei']#显示中文设置
#创建频数分布直方图
#weight为待绘制的定量数据，bins=5表示将数据划分为5个区间
#normed=False时为频数分布直方图
plt.hist(weight,bins=5,normed=False)
#x轴区间范围
plt.xlabel('weight')
plt.ylabel('frequency')
plt.title('CHINA CBA Histogram of weight frequency
distribution of players')
plt.show()
```

图8.1所示为频数分布直方图。

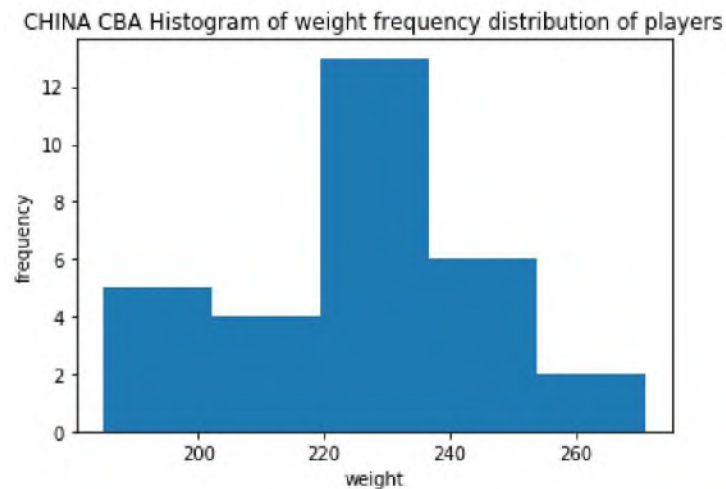


图8.1 频数分布直方图

(2) 箱线图画法

- 下四分位数 $Q1$ ：将数据按升序排序，位于25%处的数据。
- 上四分位数 $Q3$ ：将数据按升序排序，位于75%处的数据。
- 四分位距 $IQR=Q3-Q1$ ，是描述数据离散程度的一个统计量。
- 异常点是小于 $Q1-1.5IQR$ 或大于 $Q3+1.5IQR$ 的值。
- 下边缘是除异常点以外的数据中的最小值。
- 上边缘是除异常点以外的数据中的最大值。

```
#箱线图
plt.boxplot(weight, labels=['体重'])
plt.title('中国CBA球员体重箱线图')
plt.show()
```

图8.2所示为箱线图。



图8.2 箱线图

8.2 假设检验

假设检验是推论统计中用于检验统计假设的一种方法。统计假设是可通过观察一组随机变量的模型进行检验的科学假说。一旦能估计未知参数，就会希望根据结果对未知的真正参数值做出适当的推论。

统计上对参数的假设就是对一个或多个参数的论述。而其中欲检验其正确性的是零假设 (Null Hypothesis)，零假设通常由研究者决

定，反映研究者对未知参数的看法。相对于零假设，其他有关参数的论述是备择假设（Alternative Hypothesis），通常反映了执行检验的研究者对参数可能数值的另一种（对立的）看法（换句话说，备择假设通常才是研究者最想知道的）。

日常进行数据分析难免会用到一些假设检验方法进行数据探索和相关性、差异性分析，并且这是进行后续统计模型（机器学习类预测模型可以略过）预测的第一步。

1. 必要条件

- 标准Python科学计算环境（NumPy、SciPy、Matplotlib）。
- Pandas。
- Statsmodels。
- Seaborn。

要安装Python及这些依赖，推荐下载Anaconda Python或Enthought Canopy。如果使用Ubuntu或其他Linux系统，更应该使用包管理器。适用于贝叶斯模型的是PyMC，在Python中实现了概率编程语言。

2. 统计性检验的4部分

- 正态性检验。
- 相关性检验。
- 参数统计假设检验。
- 非参数统计假设检验。

（1）正态性检验

正态性检验是很多统计建模的第一步，例如，普通线性回归就对残差有正态性要求。正态性检验可用于检查数据是否具有高斯分布的统计检验。

w检验（Shapiro-wilk test）用于检验数据样本是否具有高斯分布。

【例8.2】

```
from scipy.stats import shapiro
data = [21,12,12,23,19,13,20,17,14,19]
stat,p = shapiro(data)
print("stat为: %f" %stat,"p值为: %f" %p)
```

输出：

```
stat为: 0.913007 p值为: 0.302289
```

（2）相关性检验

相关性检验是检查两个样本是否相关的统计检验。皮尔逊相关系数用于检验两个样本是否具有单调关系。

假设：

- 每个样本中的观察是独立同分布的。
- 每个样本的观察都是正态分布的。
- 每个样本的观察具有相同的方差。

【例8.3】

```
from scipy.stats import pearsonr
data1 = [21,12,12,23,19,13,20,17,14,19]
data2 = [12,11,8,9,10,15,16,17,10,16]
corr,p = pearsonr(data1,data2)
print("corr为: %f" %corr,"p值为: %f" %p)
```

输出：

corr为: 0.142814 p值为: 0.693889

(3) 卡方检验

检验两个分类变量是相关的还是独立的。

假设:

- 用于计算列联表的观察是独立的。
- 列联表的每个单元格中有25个或更多实例。

【例8.4】

```
from scipy.stats import chi2_contingency
data1 = [21,12,12,23,19,13,20,17,14,19]
data2 = [12,11,8,9,10,15,16,17,10,16]
stat,p,dof,expected = chi2_contingency(data1,data2)
print("stat为: %f" %stat,"p值为: %f" %p)
```

输出:

stat为: 0.000000 p值为: 1.000000

(4) 参数统计假设检验

①T检验

检验两个独立样本的均值是否存在显著差异。

假设:

- 每个样本中的观察是独立同分布的。
- 每个样本的观察都是正态分布的。
- 每个样本中的观察具有相同的方差。

【例8.5】

```
from scipy.stats import ttest_ind
data1 = [21,12,12,23,19,13,20,17,14,19]
data2 = [12,11,8,9,10,15,16,17,10,16]
```

```
stat,p = ttest_ind(data1,data2)
print("stat为: %f" %stat,"p值为: %f" %p)
```

输出:

```
stat为: 2.802933    p值为: 0.011763
```

②配对T检验

检验两个配对样本的均值是否存在显著差异。

假设:

- 每个样本中的观察是独立同分布的。
- 每个样本的观察都是正态分布的。
- 每个样本中的观察具有相同的方差。
- 每个样本的观察结果是成对的。

【例8.6】

```
from scipy.stats import ttest_rel
data1 = [21,12,12,23,19,13,20,17,14,19]
data2 = [12,11,8,9,10,15,16,17,10,16]
stat,p = ttest_rel(data1,data2)
print("stat为: %f" %stat,"p值为: %f" %p)
```

输出:

```
stat为: 3.022945    p值为: 0.014410
```

③方差分析

测试两个或两个以上独立样本的均值是否存在显著差异。

假设:

- 每个样本中的观察是独立同分布的。
- 每个样本的观察都是正态分布的。
- 每个样本中的观察具有相同的方差。

【例8.7】

```
from scipy.stats import f_oneway
data1 = [21,12,12,23,19,13,20,17,14,19]
data2 = [12,11,8,9,10,15,16,17,10,16]
stat,p = f_oneway(data1,data2)
print("stat为: %f" %stat,"p值为: %f" %p)
```

输出:

```
stat为: 7.856436      p值为: 0.011763
```

(5) 非参数统计假设检验

①曼-惠特尼U检验

检验两个独立样本的分布是否相等。

假设:

- 每个样本中的观察是独立同分布的。
- 可以对每个样本中的观察进行排序。

【例8.8】

```
from scipy.stats import mannwhitneyu
data1 = [21,12,12,23,19,13,20,17,14,19]
data2 = [12,11,8,9,10,15,16,17,10,16]
stat,p = mannwhitneyu(data1,data2)
print("stat为: %f" %stat,"p值为: %f" %p)
```

输出:

```
stat为: 17.500000      p值为: 0.007627
```

②威尔科克森符号秩检验

检验两个配对样本的分布是否均等。

假设:

- 每个样本中的观察是独立同分布。
- 可以对每个样本中的观察进行排序。

【例8.9】

```
from scipy.stats import wilcoxon
data1 = [21,12,12,23,19,13,20,17,14,19]
data2 = [12,11,8,9,10,15,16,17,10,16]
stat,p = wilcoxon(data1,data2)
print("stat为: %f" %stat,"p值为: %f" %p)
```

输出:

```
stat为: 2.000000    p值为: 0.014714
```

由于样本太小，因此无法精确计算相似性。

③Kruskal-Wallis H检验

检验两个或多个独立样本的分布是否相等。

假设:

- 每个样本中的观察是独立同分布的。
- 可以对每个样本中的观察进行排序。

```
from scipy.stats import kruskal
data1 = [21,12,12,23,19,13,20,17,14,19]
data2 = [12,11,8,9,10,15,16,17,10,16]
stat,p = kruskal(data1,data2)
print("stat为: %f" %stat,"p值为: %f" %p)
```

输出:

```
stat为: 6.072239    p值为: 0.013732
```

8.3 信度分析

很多人知道信度和效度是因为问卷数据分析需要用到它们，其实信度和效度的应用范围远远不止问卷的数据分析。信度是指一个测试或者使用其他任何测量工具对事物的测量可以保持一致性。观察值=真实值+误差值，误差越小越可信。信度的不同类型如下：

- 再测信度（一个测试在不同时间是否可信，两次测试值之间的相关系数）。
- 复本信度（一个测试的几个复本是否可信或者是否等价，两个复本之间的相关系数）。
- 内在一致性信度（每个项目的得分与总得分之间的相关系数）。
- 评分者信度（对一个观察结果的评价是否一致，检验不同评分者一致结论的百分比）。

1. 再测信度

两次测试的得分如下，计算相关系数。

【例8.10】

测试者编号	时期1得分	时期2得分
1	54	56
2	67	77
3	67	87
4	83	89
5	87	89
6	89	90
7	84	87
8	90	92
9	98	99
10	65	76

```
import scipy.stats.stats as stats
s1=[54,67,67,83,87,89,84,90,98,65]
s2=[56,77,87,89,89,90,87,92,99,76]
comparisonR=stats.pearsonr(s1,s2)[0]
print(comparisonR)
```

输出：

```
0.9005791967752049
```

2. 复本信度

自主记忆测试复本I和复本II的得分（时间间隔）。

【例8.11】

编号	复本 I 得分	复本 II 得分
1	4	5
2	5	6
3	3	5
4	6	6
5	7	7
6	5	6
7	6	7
8	4	8
9	4	7
10	3	7

```
import scipy.stats.stats as stats
i=[4,5,3,6,7,5,6,4,3,3]
ii=[5,6,5,6,7,6,7,8,7,7]
duplicateR=stats.pearsonr(i,ii)[0]
print (duplicateR)
```

输出：

```
0.12869789041755741
```

3. 一致性信度

计算克隆巴赫系数（Cronbach's），实际上是计算单项得分与总分直接的相关系数，并与每个单项得分的变异性比较，计算公式为：

$$\partial = \left(\frac{k}{k-1} \right) \left(\frac{s_y^2 - \sum s_i^2}{s_y^2} \right) \tag{8.10}$$

其中：

- k表示项目的个数。
- s_y^2 为所有被试项目所得总分的方差。
- s_i^2 表示所有被试在第i项目上的差异。

【例8.12】

10个被测者、5个项目的得分表如下。

编号	项目1	项目2	项目3	项目4	项目5
1	3	5	1	4	1
2	4	4	3	5	3
3	3	4	4	4	4
4	3	3	5	2	1
5	3	4	5	4	3
6	4	5	5	3	2
7	2	5	5	3	4
8	3	4	4	2	4
9	3	5	4	4	3
10	3	3	2	3	2

Python代码引入了Pandas包，应用Pandas的数据帧计算就变得简单了，其中sum()是求和，var()是求方差。

```
import numpy as np
import pandas as pd
import scipy.stats.stats as stats
import math
score=[
    [3,5,1,4,1],
    [4,4,3,5,3],
    [3,4,4,4,4],
    [3,3,5,2,1],
    [3,4,5,4,3],
    [4,5,5,3,2],
    [2,5,5,3,4],
    [3,4,4,2,4],
    [3,5,4,4,3],
    [3,3,2,3,2]]
df = pd.DataFrame(score)
total_row = df.sum(axis=1)
#print total_row
sy = total_row.var()
print( sy )
var_column = df.var()
si = var_column.sum()
print (si )
r = (5.0/4.0)*((sy-si)/sy)
print (round(r,2))
```

输出：

```
6.3999999999999995  
5.177777777777777  
0.24
```

8.4 列联表分析

列联表分析（Contingency Table Analysis）是基于列联表进行的相关统计分析与推断。列联表分析的基本问题是，判明所考察的各属性之间有无关联，即是否独立。列联表又称交互分类表，所谓交互分类，是指同时依据两个变量的值将所研究的个案分类。交互分类的目的是将两个变量分组，然后比较各组的分布状况，以寻找变量间的关系。

比较和对照是进行科学研究的基本手段。对于间距测度和比例测度的资料，进行分组比较时可以用均数检验、方差分析等方法。对于有较多可取值的序次测度资料，进行分组比较时可以用各种秩和检验方法。

而对于名义测度的资料和有序分类所得的资料（也属序次测度），分组比较时需用交叉分类进行统计描述，交叉分类所得的表格称为“列联表”，统计推断（检验）则要使用列联表分析的方法——卡方检验。卡方检验是用来研究两个定类变量间是否独立（是否存在某种关联性）的常用方法。

卡方检验的方法：假设两个变量是相互独立、互不关联的，这在统计上称为原假设。对于调查中得到的两个变量的数据，用一个表格的形式来表示它们的分布（频数和百分数），这里的频数叫观测频数，这种表格叫列联表。如果原假设成立，在这个前提下，可以计算出上面列联表中每个格子里的频数应该是多少，这叫期望频数。比较观测频数与期望频数的差，两者的差越大，表明实际情况与原假设相

差越远；差越小，表明实际情况与原假设越相近。这种差值用一个卡方统计量来表示。对卡方值进行检验，如果卡方检验的结果不显著，就不能拒绝原假设，即两个变量是相互独立、互不关联的，如果卡方检验的结果显著，就拒绝原假设，即两个变量间存在某种关联，至于是如何关联的，这要看列联表中数据的分布形态。

卡方检验的公式如下：

$$\chi^2 = \sum \left(\frac{(A-T)^2}{T} \right) \quad (8.11)$$

其中， A 是实际值， T 是理论值。

【例8.13】

列联表如下：

杀虫效果	甲	乙	丙	统计
死亡数	37	49	23	109
未死亡数	150	100	57	307
统计	187	149	80	416

```
import numpy as np
from scipy.stats import chi2_contingency

d = np.array([[37, 49, 23], [150, 100, 57]])
print (chi2_contingency(d))
```

输出如下：

```
(7.6919413561281065, 0.021365652322337315, 2,
array([[ 48.99759615,
39.04086538, 20.96153846], [138.00240385, 109.95913462,
59.03846154]]))
```

代码运行结果解析如下：

$$\begin{aligned} \text{chi-square-statistics} = & \frac{\left(37 - \frac{187 \cdot 109}{416}\right)^2}{\frac{187 \cdot 109}{416}} + \frac{\left(49 - \frac{149 \cdot 109}{416}\right)^2}{\frac{149 \cdot 109}{416}} + \frac{\left(23 - \frac{80 \cdot 109}{416}\right)^2}{\frac{80 \cdot 109}{416}} + \\ & \frac{\left(150 - \frac{187 \cdot 307}{416}\right)^2}{\frac{187 \cdot 307}{416}} + \frac{\left(100 - \frac{149 \cdot 307}{416}\right)^2}{\frac{149 \cdot 307}{416}} + \frac{\left(57 - \frac{80 \cdot 307}{416}\right)^2}{\frac{80 \cdot 307}{416}} = 7.6919413561281065 \end{aligned}$$

$$P\{ \chi^2(2) > 7.6919413561281065 \} = 0.021365652322337315$$

对于 `array([[ 48.99759615, 39.04086538, 20.96153846], [138.00240385, 109.95913462, 59.03846154]])` 而言：

$$48.99759615 = \frac{N_i \cdot N_j}{N} = \frac{109 \cdot 150}{416}$$

以此类推，得到其他结果。自由度为1时，代码底层存在Yate校正，此时手算结果会与程序运算结果不一致，属于正常现象。

8.5 相关分析

相关系数是变量间关联程度的基本测度之一，如果我们想知道两个变量之间的相关性，就可以计算相关系数进行判定。相关系数的基本特征说明如下。

方向：

- 正相关：两个变量变化方向相同。
- 负相关：两个变量变化方向相反。

量级（Magnitude）：

- 低度相关： $0 \leq |r| < 0.3$ 。
- 中度相关： $0.3 \leq |r| < 0.8$ 。
- 高度相关： $0.8 \leq |r| < 1$ 。

散点图：

在进行相关分析之前，通常会绘制散点图来观察变量之间的相关性，如果这些数据在二维坐标轴中构成的数据点分布在一条直线的周围，就说明变量间存在线性相关关系。

1. Python计算相关系数

$$r = \frac{\sum (Z_X Z_Y)}{N} \quad (8.12)$$

其中：

- r ：相关系数。
- Z_X ：变量 X 的 z 分数。
- Z_Y ：变量 Y 的 z 分数。
- N ： X 和 Y 取值的配对个数。

2. 场景案例

我们知道影响金融产品销量的因素很多。作为用户来讲，最直接的参考指标一定是产品的利率，金融机构为了吸引更多的用户能够持有或购买某项金融产品，往往会推出加息活动，那么加息活动这个变量与实际销量之间是否存在相关关系？

【例8.14】

A	B	C	D	E
X: 加息活动投入	Y: 销售额	Z_X	Z_Y	$Z_X + Z_Y$
52	162			
19	61			
7	22			
33	100			
2	6			

计算相关系数的Python代码如下：

```
import numpy
X = [52,19,7,33,2]
Y = [162,61,22,100,6]

#均值
XMean = numpy.mean(X)
YMean = numpy.mean(Y)

#标准差
XSD = numpy.std(X)
YSD = numpy.std(Y)

#z分数
ZX = (X-XMean)/XSD
ZY = (Y-YMean)/YSD
#相关系数
r = numpy.sum(ZX*ZY)/(len(X))
print(ZX,ZY)
print(r)
```

输出：

```
[ 1.61684964 -0.19798159 -0.85792022  0.57194681
-1.13289465] [ 1.63073842
-0.16342912 -0.85622649  0.52936824 -1.14045105]
0.999674032661831
```

则得到结果：

A	B	C	D	E
X:加息活动投入	Y:销售额	Z_x	Z_y	Z_x+Z_y
52	162	1.61684964	1.63073842	2.63665883
19	61	-0.19798159	-0.16342912	0.63665883
7	22	-0.85792022	-0.85622649	0.03235596
33	100	0.57194681	0.52936824	0.30277048
2	6	-1.13289465	-1.14045105	1.29201089

8.6 方差分析

方差分析（ANOVA）又称“变异数分析”或“F检验”，是由 R. A. Fister 发明的，用于两个及两个以上的样本均数差别的显著性检验。与通常的统计推断问题一样，方差分析的任务是先根据实际情况提出原假设 H_0 与备择假设 H_1 ，然后寻找适当的检验统计量进行假设检验。

试验中要考察的指标称为试验指标，影响试验指标的条件称为因素，因素所处的状态称为水平，若试验中只有一个因素改变，则称为单因素试验；若有两个因素改变，则称为双因素试验；若有多个因素改变，则称为多因素试验。方差分析就是对试验数据进行分析，检验方差相等的多个正态总体均值是否相等，进而判断各因素对试验指标的影响是否显著。根据影响试验指标条件的个数可以区分为单因素方差分析、双因素方差分析和多因素方差分析。

8.6.1 单因素方差分析

单因素方差分析是指对单因素试验结果进行分析，检验因素对试验结果有无显著性影响的方法。单因素方差分析是两个样本平均数比较的引申，用来检验多个平均数之间的差异，从而确定因素对试验结果有无显著性影响。

- 因素：影响研究对象的某一指标、变量。
- 水平：因素变化的各种状态或因素变化所分的等级或组别。
- 单因素试验：考虑的因素只有一个的试验叫单因素试验。

1. 假设前提

设单因素 A 具有 r 个水平，分别记为 $A_1, A_2, \dots, A_r$ ，在每个水平 A_i ($i=1, 2, \dots, r$) 下，要考察的指标可以看成一个总体，故有 r 个总体，并假设：

(1) 每个总体均服从正态分布，即 $X_{i1}, X_{i2}, \dots, X_{ini} \sim N(\mu_i, \sigma^2)$ ， $i=1, 2, \dots, r$ 。

(2) 每个总体的方差 σ^2 相同。

(3) 从每个总体中抽取的样本 $X_{i1}, X_{i2}, \dots, X_{ini}$ 相互独立， $i=1, 2, \dots, r$ 。

此处的 μ_i 、 σ^2 均未知。

那么，要比较各个总体的均值是否一致，就要检验各个总体的均值是否相等，设第 i 个总体的均值为 μ_i ，则：

假设检验为 $H_0: \mu_1 = \mu_2 = \dots = \mu_i$ 。

备择假设为 $H_1: \mu_1, \mu_2, \dots, \mu_i$ 不全相等。

2. 方差分析的任务

(1) 检验该模型中 r 个总体 $N(\mu_i, \sigma^2)$ ($i=1, 2, \dots, r$) 的均值是否相等。

(2) 作为未知参数 $\mu_1, \mu_2, \dots, \mu_i, \sigma^2$ 的估计。

Python实现单因素方差分析用到了SciPy.stats.f_oneway()方法，用法很简单，只不过在用之前需要先检验方差齐性，用到了levene test。

【例8.15】

```

#-*- coding: utf-8 -*-
from scipy import stats
import pandas as pd
import numpy as np
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm

#读取数据
fPath='altman_910.txt'
df=pd.read_csv(fPath,header=None,names=['value','group'])
print (df)

#数据分组
print('One-way ANOVA: -----')
inFile = 'altman_910.txt'
data = np.genfromtxt(inFile, delimiter=',')

# Sort them into groups, according to column 1
group1 = data[data[:,1]==1,0]
group2 = data[data[:,1]==2,0]
group3 = data[data[:,1]==3,0]

# First, check if the variances are equal, with the "Levene"-test
(W,p) = stats.levene(group1, group2, group3)
if p<0.05:
    print(('Warning: the p-value of the Levene test is <0.05: p={0}'.format(p)))

# Do the one-way ANOVA
F_statistic, pVal = stats.f_oneway(group1, group2, group3)
# --- >>> STOP stats <<< ---

# Print the results
print('Data form Altman 910:')
print((F_statistic, pVal))
if pVal < 0.05:
    print('One of the groups is significantly different.')

# Elegant alternative implementation, with pandas & statsmodels
df = pd.DataFrame(data, columns=['value', 'treatment'])
#如果没有大写 C(), 就会出错, 因为表示分类变量, category
model = ols('value ~ C(treatment)', df).fit()
anovaResults = anova_lm(model)
print(anovaResults)

# Check if the two results are equal. If they are, there is no output
#decimal=3表示精确到的小数位。如果两个数相等, 结果就为空, 否则出现异常提示
np.testing.assert_almost_equal(F_statistic, anovaResults['F'][0],decimal=3)

```

输出：

```
runfile('D:/Anaconda3/workspace/test.py', wdir='D:/Anaconda3/workspace')
      value  group
0      243     1
1      251     1
2      275     1
3      291     1
4      347     1
5      354     1
6      380     1
7      392     1
8      206     2
9      210     2
10     226     2
11     249     2
12     255     2
13     273     2
14     285     2
15     295     2
16     309     2
17     241     3
18     258     3
19     270     3
20     293     3
21     328     3

One-way ANOVA: -----
Warning: the p-value of the Levene test is <0.05: p=0.045846812634186246
Data form Altman 910:
(3.7113359882669763, 0.043589334959178244)
One of the groups is significantly different.

      df      sum_sq      mean_sq      F      PR(>F)
C(treatment)  2.0  15515.766414  7757.883207  3.711336  0.043589
Residual      19.0  39716.097222  2090.320906    NaN    NaN
```

8.6.2 多因素方差分析

多因素方差分析用来研究两个及两个以上的控制变量是否对观测变量产生显著影响。这里，由于研究多个因素对观测变量的影响，因此称为多因素方差分析。多因素方差分析不仅能够分析多个因素对观测变量的独立影响，还能够分析多个控制因素的交互作用能否对观测

变量的分布产生显著影响，进而最终找到有利于观测变量的最优组合。

【例8.16】

```
#-*- coding: utf-8 -*-
from scipy import stats
import pandas as pd
import numpy as np
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm
from statsmodels.stats.multicomp import pairwise_tukeyhsd
import matplotlib.pyplot as plt

#读取数据
fPath='altman_910_2.txt'
df=pd.read_csv(fPath,header=None,names=
['hs','fetus','observer'])
df=df.dropna()#删除空值
print (df)

formula='hs~C(fetus)+C(observer)+C(fetus):C(observer) '
anova_results=anova_lm(ols(formula,df).fit())
print(anova_results)

hsd=pairwise_tukeyhsd(df['hs'],df['fetus'])
print (hsd.summary())
```

输出：


```
runfile('D:/Anaconda3/workspace/test.py', wdir='D:/Anaconda3/workspace')
```

	hs	fetus	observer
--	----	-------	----------

0	14.3	1	1
1	14.0	1	1
2	14.8	1	1
3	13.6	1	2
4	13.6	1	2
5	13.8	1	2
6	13.9	1	3
7	13.7	1	3
8	13.8	1	3
9	13.8	1	4
10	13.7	1	4
11	13.9	1	4
12	19.7	2	1
13	19.9	2	1
14	19.8	2	1
15	19.8	2	2
16	19.3	2	2
17	19.8	2	2
18	19.5	2	3
19	19.8	2	3
20	19.5	2	3

	df	sum_sq	...	F	PR(>F)
C(fetus)	1.0	171.187659	...	3744.730035	2.086626e-18
C(observer)	3.0	0.721389	...	5.260127	1.222175e-02
C(fetus):C(observer)	3.0	0.235849	...	1.719734	2.087118e-01
Residual	14.0	0.640000	...	NaN	NaN

[4 rows x 5 columns]

Multiple Comparison of Means - Tukey HSD,FWER=0.05

=====

group1	group2	meandiff	lower	upper	reject
--------	--------	----------	-------	-------	--------

```
runfile('D:/Anaconda3/workspace/test.py', wdir='D:/Anaconda3/workspace')
```

	hs	fetus	observer
--	----	-------	----------

0	14.3	1	1
1	14.0	1	1
2	14.8	1	1
3	13.6	1	2
4	13.6	1	2
5	13.8	1	2
6	13.9	1	3
7	13.7	1	3
8	13.8	1	3
9	13.8	1	4
10	13.7	1	4
11	13.9	1	4


```

12  19.7    2    1
13  19.9    2    1
14  19.8    2    1
15  19.8    2    2
16  19.3    2    2
17  19.8    2    2
18  19.5    2    3
19  19.8    2    3
20  19.5    2    3

              df      sum_sq      ...      F      PR(>F)
C(fetus)          1.0    171.187659  ...    3744.730035  2.086626e-18
C(observer)        3.0     0.721389  ...     5.260127  1.222175e-02
C(fetus):C(observer) 3.0     0.235849  ...     1.719734  2.087118e-01
Residual          14.0     0.640000  ...      NaN      NaN

[4 rows x 5 columns]
Multiple Comparison of Means - Tukey HSD,FWER=0.05
=====
group1 group2 meandiff  lower  upper  reject
-----
1      2      5.7694   5.5029  6.036   True
-----

```

这是随即设计的两个因素方差分析的结果：显示fetus的主效应显著。输出的结果显示，3个水平均值均呈现显著差异（reject==True）。

8.7 回归分析

线性回归是一种有监督的学习算法，它介绍自变量和因变量之间的线性相关关系，分为一元线性回归和多元线性回归。一元线性回归是一个自变量和一个因变量间的回归，可以看成是多元线性回归的特例。线性回归可以用来预测和分类，从回归方程可以看出自变量和因变量的相互影响关系。

1. 线性回归模型

$$y = \beta_0 + \beta_1 + \cdots + \beta_k X_k + \varepsilon_i \quad (8.13)$$

对于线性回归的模型假定如下：

- (1) 误差项的均值为0，且误差项与解释变量之间线性无关。

$$E(\varepsilon_i) = 0 \quad E(X^T \varepsilon) = 0 \quad (8.14)$$

(2) 误差项是独立同分布的，即每个误差项之间相互独立且每个误差项的方差是相等的。

- (3) 解释变量之间线性无关。

- (4) 正态性假设，即误差项是服从正态分布的。

以上的假设是建立回归模型的基本条件，所以对于回归结果要一一进行验证，如果不满足假定，就要进行相关的修正。

2. 模型的参数求解

- (1) 矩估计

一般是通过样本矩来估计总体的参数，常见的是用样本的一阶原点矩来估计总体的均值，用二阶中心矩来估计总体的方差。

- (2) 最小二乘估计

最小二乘估计法，又称最小平方法，是一种数学优化技术。它通过最小化误差的平方和寻找数据的最佳函数匹配。利用最小二乘估计法可以简便地求得未知的数据，并使得这些数据与实际数据之间误差的平方和为最小。

- (3) 极大似然估计

极大似然估计是基于概率的思想，要求样本的概率分布是已知的，参数估计的值使得大量样本发生的概率最大，用似然函数来度

量，似然函数是各个样本的密度函数的乘积，为方便求解其导数，加负号求解极小值，得到参数的估计结果。

3. 用Python实现线性回归

目前使用Python较多，而Python中远近闻名的机器学习库要数Scikit-Learn了。这个库优点很多，简单易用，接口抽象得非常好。这里封装其中的很多机器学习算法，然后进行一次性测试，从而便于分析取优。当然，针对具体算法，超参调优也非常重要。

【例8.17】

下面利用线性回归算法预测波士顿的房价。波士顿房价数据集包含波士顿郊区住房价值的信息。波士顿数据集是Scikit-Learn的内置数据集，可以直接拿来使用。

```
#Python库导入
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_boston
import matplotlib.font_manager as fm
#数据获取和理解
boston = load_boston()
print(boston.keys())
```

输出：

```
dict_keys(['data', 'target', 'feature_names', 'DESCR'])
#波士顿数据集506个样本，14个特征
print(boston.data.shape)
```

输出：

```
(506, 13)
#数据集列名
```

```
print(boston.feature_names)
```

输出:

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD'  
'TAX' 'PTRATIO' 'B'  
'LSTAT']  
#波士顿数据集描述  
bos = pd.DataFrame(boston.data)  
print(bos.head())
```

输出:

Boston House Prices dataset

=====

Notes

Data Set Characteristics:

:Number of Instances: 506

:Number of Attributes: 13 numeric/categorical predictive

:Median Value (attribute 14) is usually the target

:Attribute Information (in order):

- CRIM per capita crime rate by town
- ZN proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS proportion of non-retail business acres per town
- CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX nitric oxides concentration (parts per 10 million)
- RM average number of rooms per dwelling
- AGE proportion of owner-occupied units built prior to 1940
- DIS weighted distances to five Boston employment centres
- RAD index of accessibility to radial highways
- TAX full-value property-tax rate per \$10,000
- PTRATIO pupil-teacher ratio by town
- B $1000(B_k - 0.63)^2$ where B_k is the proportion of blacks by town
- LSTAT % lower status of the population
- MEDV Median value of owner-occupied homes in \$1000's

:Missing Attribute Values: None

:Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.

<http://archive.ics.uci.edu/ml/datasets/Housing>

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978. Used in Belsley, Kuh & Welsch, 'Regression diagnostics ...', Wiley, 1980. N.B. Various transformations are used in the table on pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression problems.

References

- Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.

- Quinlan, R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.

- many more! (see <http://archive.ics.uci.edu/ml/datasets/Housing>)

```

0      1      2      3      4      ...      8      9      10      11      12
0 0.00632 18.0  2.31  0.0  0.538  ...   1.0 296.0 15.3 396.90  4.98
1 0.02731  0.0  7.07  0.0  0.469  ...   2.0 242.0 17.8 396.90  9.14
2 0.02729  0.0  7.07  0.0  0.469  ...   2.0 242.0 17.8 392.83  4.03
3 0.03237  0.0  2.18  0.0  0.458  ...   3.0 222.0 18.7 394.63  2.94
4 0.06905  0.0  2.18  0.0  0.458  ...   3.0 222.0 18.7 396.90  5.33

```

```

[5 rows x 13 columns]
#数据模型构建——线性回归
X = bos.drop('PRICE', axis=1)
lm = LinearRegression()
lm
lm.fit(X, bos.PRICE)
print('线性回归算法w值:', lm.coef_)
print('线性回归算法b值:', lm.intercept_)

```

输出:

```

      CRIM      ZN      INDUS      CHAS      NOX      ...      RAD      TAX      PTRATIO      B      LSTAT
0 0.00632 18.0  2.31  0.0  0.538  ...   1.0 296.0 15.3 396.90  4.98
1 0.02731  0.0  7.07  0.0  0.469  ...   2.0 242.0 17.8 396.90  9.14
2 0.02729  0.0  7.07  0.0  0.469  ...   2.0 242.0 17.8 392.83  4.03
3 0.03237  0.0  2.18  0.0  0.458  ...   3.0 222.0 18.7 394.63  2.94
4 0.06905  0.0  2.18  0.0  0.458  ...   3.0 222.0 18.7 396.90  5.33

[5 rows x 13 columns]
[24.  21.6 34.7 33.4 36.2]
线性回归算法w值: [-1.07170557e-01  4.63952195e-02  2.08602395e-02
2.68856140e+00
-1.77957587e+01  3.80475246e+00  7.51061703e-04 -1.47575880e+00
 3.05655038e-01 -1.23293463e-02 -9.53463555e-01  9.39251272e-03
-5.25466633e-01]
线性回归算法b值: 36.49110328036135
#散点图
myfont = fm.FontProperties(fname='C:/Windows/Fonts/msyh.ttc')
plt.scatter(bos.RM, bos.PRICE)
plt.xlabel(u'住宅平均房间数', fontproperties=myfont)
plt.ylabel(u'房屋价格', fontproperties=myfont)
plt.title(u'RM与PRICE的关系', fontproperties=myfont)
plt.show()

```

RM和PRICE的关系散点图如图8.3所示。

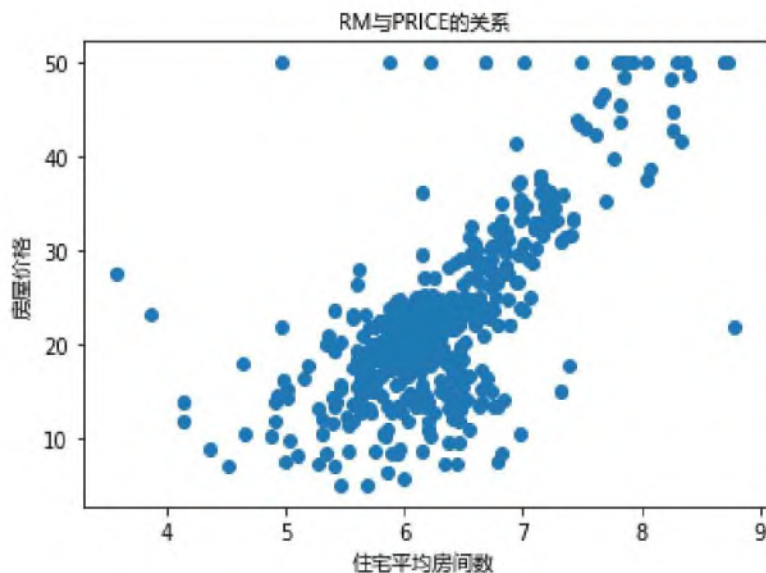


图8.3 RM和PRICE的关系散点图

```
#数据模型应用—预测房价
lm.predict(X)[0:5]
#预测
mse = np.mean((bos.PRICE - lm.predict(X)) ** 2)
print(mse)
```

输出：

```
21.897779217687496
```

使用.DESCR探索波士顿数据集，业务目标是预测波士顿郊区住房的房价，使用Scikit-Learn针对整个数据集拟合线性回归模型，并计算均方误差。

8.8 聚类分析

聚类分析是没有给定划分类别的情况下，根据样本相似度进行样本分组的一种方法，是一种非监督的学习算法。聚类的输入是一组未被标记的样本，聚类根据数据自身的距离或相似度划分为若干组，划分的原则是组内距离最小化而组间距离最大化。

常见的聚类分析算法如下：

- 分散性聚类：K-Means聚类也称为快速聚类法，在最小化误差函数的基础上将数据划分为预定的类数K。该算法的原理简单并便于处理大量数据。
- 密度算法：基于密度的方法（Density-Based Methods），与其他方法的根本区别是：它不是基于各种各样的距离的，而是基于密度的。这样就能克服基于距离的算法只能发现“类圆形”的聚类的缺点。这个方法的指导思想是，只要一个区域中的点的密度大过某个阈值，就把它加到与之相近的聚类中去。
- 系统聚类：也称为层次聚类，分类的单位由高到低呈树形结构，且所处的位置越低，其所包含的对象就越少，但这些对象间的共同特征越多。该聚类方法只适合在小数据量的时候使用，数据量大的时候速度会非常慢。

【例8.18】

为了看鸢尾花的3种聚类算法的直观区别，不用具体的算法实现，只需要调用相应的函数即可。

1. 分散性聚类

K-Means算法是典型的基于距离的非层次聚类算法，在最小化误差函数的基础上将数据划分为预定的类数K，采用距离作为相似性的评价指标，即认为两个对象的距离越近，其相似度就越大。

算法流程：

- （1）选择聚类的个数k。
- （2）任意产生k个聚类，然后确定聚类中心，或者直接生成k个中心。

(3) 对每个点确定其聚类中心点。

(4) 再计算其聚类新中心。

(5) 重复以上步骤直到满足收敛要求（通常是确定的中心点不再改变）。

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data[:, :4] #表示取特征空间中的4个维度
print(X.shape)

# 绘制数据分布图
plt.scatter(X[:, 0], X[:, 1], c="red", marker='o',
label='see')
plt.xlabel('petal length')
plt.ylabel('petal width')
plt.legend(loc=2)
plt.show()

estimator = KMeans(n_clusters=3) # 构造聚类器
estimator.fit(X) # 聚类
label_pred = estimator.labels_ # 获取聚类标签
# 绘制K-Means结果
x0 = X[label_pred == 0]
x1 = X[label_pred == 1]
x2 = X[label_pred == 2]
plt.scatter(x0[:, 0], x0[:, 1], c="red", marker='o',
label='label0')
plt.scatter(x1[:, 0], x1[:, 1], c="green", marker='*',
label='label1')
plt.scatter(x2[:, 0], x2[:, 1], c="blue", marker='+',
label='label2')
plt.xlabel('petal length')
plt.ylabel('petal width')
plt.legend(loc=2)
plt.show()
```

鸢尾花K-Means聚类如图8.4所示。

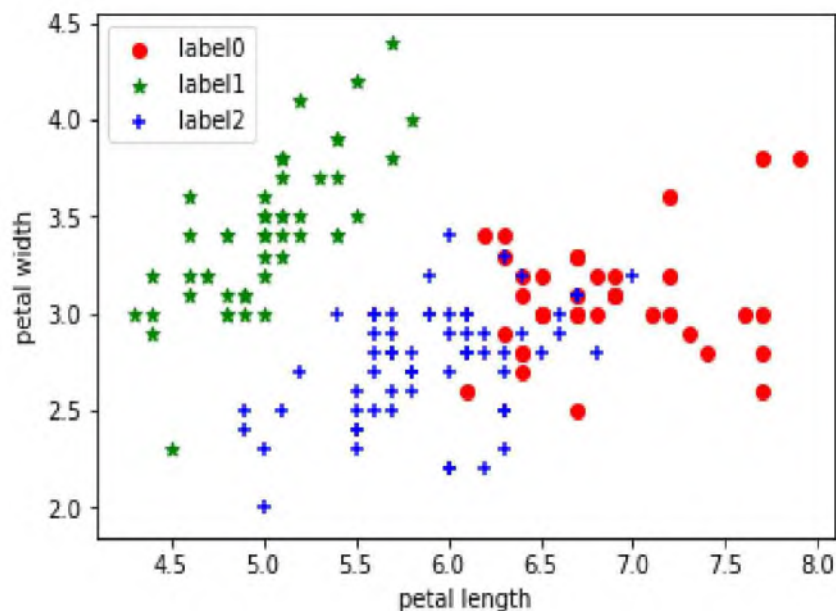


图8.4 鸢尾花K-Means聚类

K-Means算法试图找到使平方误差准则函数最小的簇。当潜在的簇形状是凸面的，簇与簇之间的区别较明显，且簇大小相近时，其聚类结果比较理想。对于处理大数据集合，该算法非常高效，且伸缩性较好。但该算法除了要事先确定簇数K和对初始聚类中心敏感外，经常以局部最优结束，同时对“噪声”和孤立点敏感，并且该方法不适于发现非凸面形状的簇或大小差别很大的簇。

2. 密度聚类之DBSCAN算法

DBSCAN算法需要两个参数，即 ϵ (eps) 和 minPts 形成高密度区域所需要的最少点数。由一个任意未被访问的点开始，然后探索这个点的 ϵ -邻域，如果 ϵ -邻域里有足够的点，就建立一个新的聚类，否则这个点被标记为杂音。注意，这个点之后可能被发现在其他点的 ϵ -邻域里，而该 ϵ -邻域可能有足够的点，届时这个点会被加入该聚类中。

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
```

```

from sklearn import datasets
from sklearn.cluster import DBSCAN
iris = datasets.load_iris()
X = iris.data[:, :4] #表示只取特征空间中的4个维度
print(X.shape)
# 绘制数据分布图
plt.scatter(X[:, 0], X[:, 1], c="red", marker='o',
label='see')
plt.xlabel('petal length')
plt.ylabel('petal width')
plt.legend(loc=2)
plt.show()

dbscan = DBSCAN(eps=0.4, min_samples=9)
dbscan.fit(X)
label_pred = dbscan.labels_

# 绘制K-Means结果
x0 = X[label_pred == 0]
x1 = X[label_pred == 1]
x2 = X[label_pred == 2]
plt.scatter(x0[:, 0], x0[:, 1], c="red", marker='o',
label='label0')
plt.scatter(x1[:, 0], x1[:, 1], c="green", marker='*',
label='label1')
plt.scatter(x2[:, 0], x2[:, 1], c="blue", marker='+',
label='label2')
plt.xlabel('petal length')
plt.ylabel('petal width')
plt.legend(loc=2)
plt.show()

```

鸢尾花DBSCAN聚类如图8.5所示。

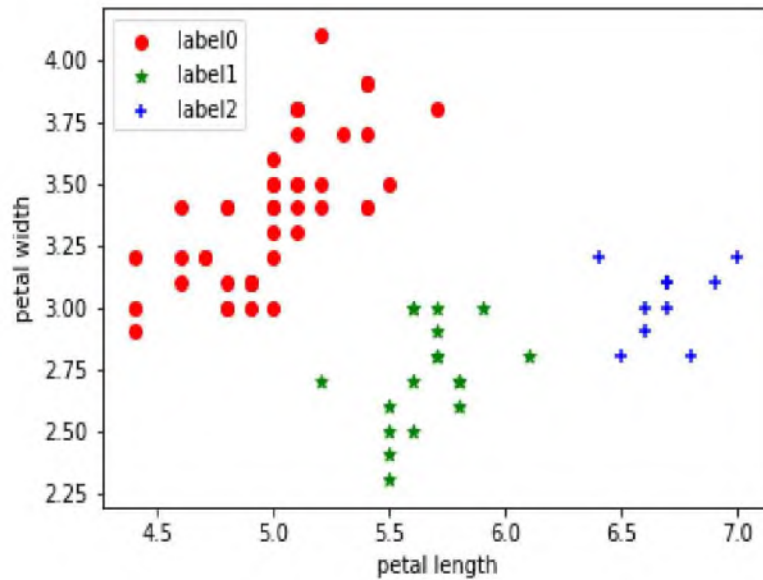


图8.5 鸢尾花DBSCAN聚类

3. 结构性聚类（层次聚类）

- 凝聚层次聚类（AGNES算法，自底向上）：首先将每个对象作为一个簇，然后合并这些原子簇为越来越大的簇，直到某个终结条件被满足。
- 分裂层次聚类（DIANA算法，自顶向下）：首先将所有对象置于一个簇中，然后逐渐细分为越来越小的簇，直到达到某个终结条件。

下面使用AGNES算法。

```
from sklearn import datasets
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import pandas as pd

iris = datasets.load_iris()
irisdata = iris.data

clustering = AgglomerativeClustering(linkage='ward',
n_clusters=3)
```

```

res = clustering.fit(irisdata)

print ("各个簇的样本数目: ")
print (pd.Series(clustering.labels_).value_counts())
print ("聚类结果: ")
print (confusion_matrix(iris.target, clustering.labels_))

plt.figure()
d0 = irisdata[clustering.labels_ == 0]
plt.plot(d0[:, 0], d0[:, 1], 'r.')
d1 = irisdata[clustering.labels_ == 1]
plt.plot(d1[:, 0], d1[:, 1], 'go')
d2 = irisdata[clustering.labels_ == 2]
plt.plot(d2[:, 0], d2[:, 1], 'b*')
plt.xlabel("Sepal.Length")
plt.ylabel("Sepal.Width")
plt.title("AGNES Clustering")
plt.show()

```

鸢尾花AGNES聚类如图8.6所示。

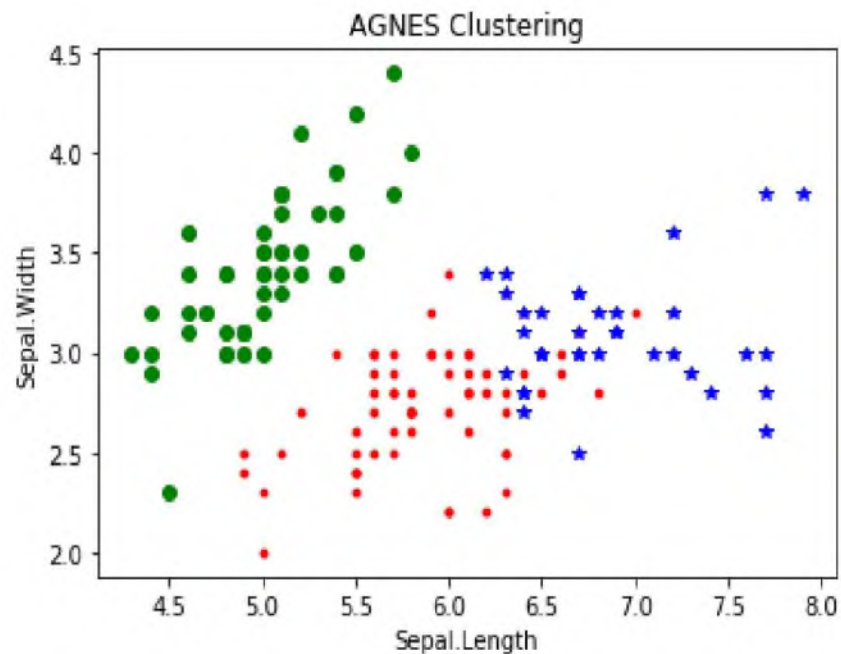


图8.6 鸢尾花AGNES聚类

从上面3种实验截图可以看出，K-Means聚类 and AGNES 层次聚类分析结果显示为三类，与DBSCAN的结果不一样。这主要取决于算法本身的优缺点。K-Means对于大型数据集简单高效、时间复杂度低、空间复杂度低。最重要的是数据集大时结果容易局部最优。需要预先设定K值，对最先的K个点的选取很敏感，对噪声和离群值非常敏感，只用于 numerical 类型的数据，不能解决非凸数据。DBSCAN对噪声不敏感，能发现任意形状的聚类，但是聚类的结果与参数有很大的关系。DBSCAN用固定参数识别聚类，但当聚类的稀疏程度不同时，相同的判定标准可能会破坏聚类的自然结构，即较稀的聚类会被划分为多个类，或密度较大且离得较近的类会被合并成一个聚类。

8.9 判别分析

判别分析是一种经典的分析方法，其利用已知类别的样本建立判别模型，对未知类别的样本进行分类。本节主要讨论费歇（Fisher）判别分析的方法。

1. Fisher线性判别原理

Fisher判别的思想是投影，使多维问题简化为一维问题来处理。选择一个适当的投影轴，使所有的样品点都投影到这个轴上得到一个投影值。对这个投影轴的方向的要求是：使每一类内的投影值所形成的类内离差尽可能小，而不同类间的投影值所形成的类间离差尽可能大。

Fisher判别原理示意图如图8.7所示。

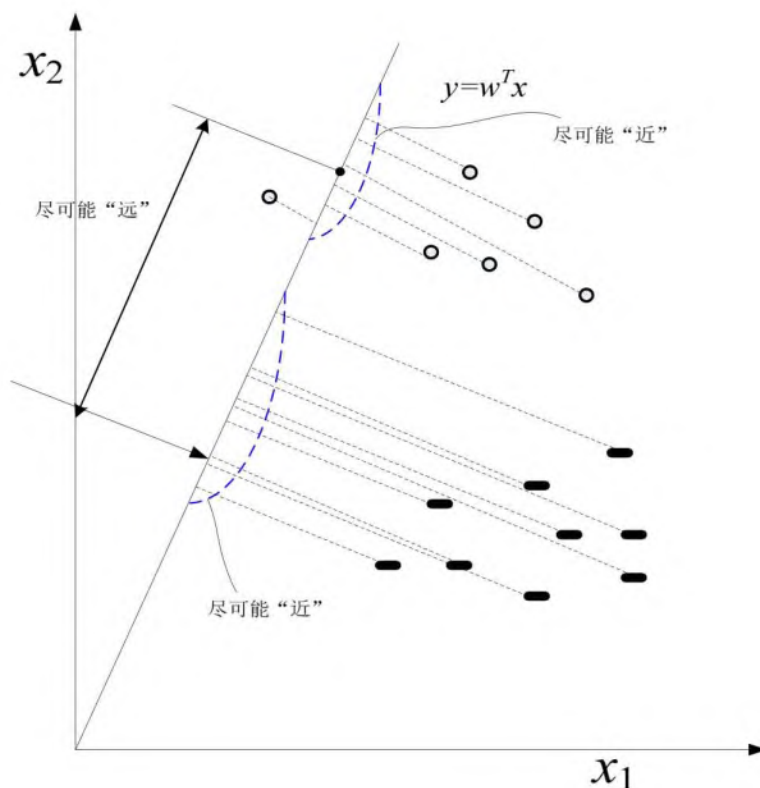


图8.7 Fisher判别原理示意图

2. 公式推导

图8.7给出了一个二维的示意图，接下来讨论将以二维的情况进行分类来逐步分析原理和实现。

对于给定的数据集 D （已经设置好分类标签）， X_i 、 U_i 和 Σ_i 分别表示给定类别 i 的集合、均值向量和协方差矩阵。现将数据投影到直线 $y = w^T x$ 上，则样本中心的投影为 $y = w_1 * u_1 + w_2 * u_2 + \dots + w_n * u_n$ （ n 为样本维度，接下来的讨论中将统一设置为2），写成向量形式则为 $w^T u = y$ 。若将所有的样本都投影到直线上，则两类样本的协方差分别为 $w^T \Sigma_0 w$ 和 $w^T \Sigma_1 w$ 。

要想达到较好的分类效果，应该使得同类样本的投影点尽可能接近，也就是让同类样本投影点的协方差尽可能小，即 $(w^T \Sigma_0 w + w^T$

$\Sigma_0 w$) 尽可能小。同时, 应该保证不同类样本的投影点尽可能互相远离, 即 $\|w^T u_0 - w^T u_1\|$ 尽可能大。如果同时考虑两者的关系, 可以得到下面的需要最大化的目标:

$$J = \frac{\|w^T u_0 - w^T u_1\|}{w^T \sum_0 w + w^T \sum_1 w} \quad (8.15)$$

这里定义类内散度矩阵 (Within-Class Scatter Matrix) :

$$S_w = \sum_0 + \sum_1 = \sum_{x \in X_0} (x - u_0)(x - u_0)^T + \sum_{x \in X_1} (x - u_1)(x - u_1)^T \quad (8.16)$$

以及类间离散度矩阵 (Between-Class Scatter Matrix) :

$$S_b = (u_0 - u_1)(u_0 - u_1)^T \quad (8.17)$$

则 J 可重写为:

$$J = \frac{w^T S_b w}{w^T S_w w} \quad (8.18)$$

3. 编程实现

【例8.19】

下面直接用Scikit-Learn的接口来生成数据。


```

#数据生成
from sklearn.datasets import make_multilabel_classification
import numpy as np

x, y = make_multilabel_classification(n_samples=20, n_features=2,
                                     n_labels=1, n_classes=1,
                                     random_state=2) # 设置随机数种子, 保证每次产生相
同的数据

# 根据类别分类
index1 = np.array([index for (index, value) in enumerate(y) if value == 0])
# 获取类别1的 indexes
index2 = np.array([index for (index, value) in enumerate(y) if value == 1])
# 获取类别2的 indexes

c_1 = x[index1] # 类别1的所有数据(x1, x2) in X_1
c_2 = x[index2] # 类别2的所有数据(x1, x2) in X_2

#fisher 算法实现
def cal_cov_and_avg(samples):
    """
    给定一个类别的数据, 计算协方差矩阵和平均向量
    :param samples:
    :return:
    """
    u1 = np.mean(samples, axis=0)
    cov_m = np.zeros((samples.shape[1], samples.shape[1]))
    for s in samples:
        t = s - u1
        cov_m += t * t.reshape(2, 1)
    return cov_m, u1

def fisher(c_1, c_2):
    """
    fisher 算法实现(请参考上面推导出来的公式, 那才是精华部分)
    :param c_1:
    :param c_2:

```

```

        :return:
        """
        cov_1, u1 = cal_cov_and_avg(c_1)
        cov_2, u2 = cal_cov_and_avg(c_2)
        s_w = cov_1 + cov_2
        u, s, v = np.linalg.svd(s_w) # 奇异值分解
        s_w_inv = np.dot(np.dot(v.T, np.linalg.inv(np.diag(s))), u.T)
        return np.dot(s_w_inv, u1 - u2)
#判定类别
def judge(sample, w, c_1, c_2):
    """
    true 属于1
    false 属于2
    :param sample:
    :param w:
    :param center_1:
    :param center_2:
    :return:
    """
    u1 = np.mean(c_1, axis=0)
    u2 = np.mean(c_2, axis=0)
    center_1 = np.dot(w.T, u1)
    center_2 = np.dot(w.T, u2)
    pos = np.dot(w.T, sample)
    return abs(pos - center_1) < abs(pos - center_2)

w = fisher(c_1, c_2) # 调用函数, 得到参数 w
out = judge(c_1[1], w, c_1, c_2) # 判断所属的类别
print(out)
# 绘图
import matplotlib.pyplot as plt

plt.scatter(c_1[:, 0], c_1[:, 1], c='#99CC99')
plt.scatter(c_2[:, 0], c_2[:, 1], c='#FFCC00')
line_x = np.arange(min(np.min(c_1[:, 0]), np.min(c_2[:, 0])),
                    max(np.max(c_1[:, 0]), np.max(c_2[:, 0])),
                    step=1)
line_y = - (w[0] * line_x) / w[1]
plt.plot(line_x, line_y)
plt.show()

```

Fisher线性判别如图8.8所示。

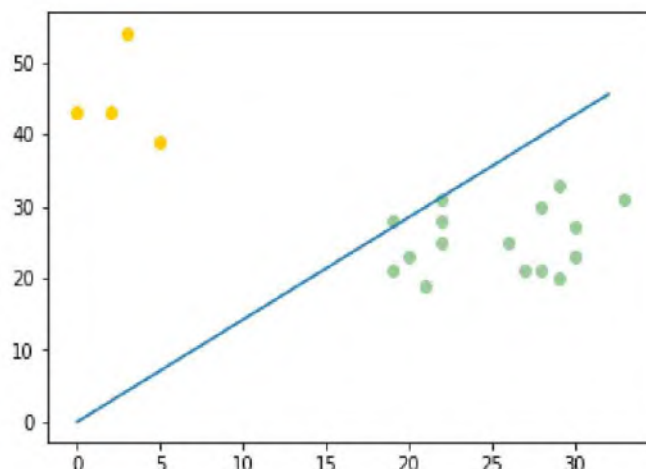


图8.8 Fisher线性判别

8.10 主成分分析

主成分分析 (Principal Components Analysis) 是一种对特征进行降维的方法。由于观测指标间存在相关性，将导致信息的重叠与低效，我们倾向于用少量的、尽可能多地反映原特征的新特征来替代它们，主成分分析因此产生。主成分分析可以看成是高维空间通过旋转坐标系找到最佳投影（几何上），生成新维度，其中新坐标轴每一个维度都是原维度的线性组合 $\theta'X$ （数学上），满足：

- 新维度特征之间的相关性尽可能小。
- 参数空间 θ 有界。
- 方差尽可能大，且每个主成分的方差递减。

1. 数学表示

对于样本 $i \in N$ 有 p 维特征 $X_{p \times 1}$ ，于是有 p 维主成分 $Z_{p \times 1}$ ，以及参数 $\theta_{p \times p}$ 满足：

$$Z_{p \times 1} = \theta_{p \times p} X_{p \times 1} \quad (8.19)$$

其中， $Z_1 = Z[0]$ 是样本 i 的第一主成分值，以此类推。 θ 可以看作是对原坐标系的正交变换。针对上述主成分分析的3点性质，分别对应有：

$$\text{Cov}(Z_i, Z_j) = 0 \quad i, j = 1, 2, \dots, p \text{ 且 } i \neq j \quad (8.20)$$

$$\sum_{i=1}^p \theta_{ij}^2 = 1 \quad j=1, 2, \dots, p \quad (8.21)$$

$$\text{Var}(Z_i) = \max(\text{Var}(\theta_i X)) \text{ 且 } \text{Var}(Z_i) > \text{Var}(Z_j) \text{ 如果 } i < j \quad (8.22)$$

问题转换为，如何在满足以上性质的情况下找到合适的 θ 。

2. 主成分分析的基本步骤

输入：样本集 $D = \{x_1, x_2, \dots, x_m\}$ ，低维空间维数 d' 。

过程：

(1) 对所有样本进行中心化： $x_i \leftarrow x_i - \frac{1}{m} \sum_{i=1}^m x_i$ 。

(2) 计算样本的协方差矩阵 XX^T 。

(3) 对协方差矩阵 XX^T 进行特征值分解。

(4) 取最大的 d' 个特征值所对应的特征向量 $w_1, w_2, \dots, w_{d'}$ 。

输出：投影矩阵 $W = (w_1, w_2, \dots, w_{d'})$ 。

3. Python源代码实现

我们通过Python的sklearn库来实现鸢尾花数据进行降维，数据本身是4维的，降维后变成2维的，可以在平面中画出样本点的分布。

【例8.20】

```

import matplotlib.pyplot as plt                    #加载 Matplotlib 用于数据的可视化
from sklearn.decomposition import PCA              #加载 PCA 算法包
from sklearn.datasets import load_iris

data=load_iris()
y=data.target
x=data.data
pca=PCA(n_components=2)                          #加载 PCA 算法，设置降维后主成分数目为2
reduced_x=pca.fit_transform(x)                   #对样本进行降维
red_x,red_y=[],[]
blue_x,blue_y=[],[]
green_x,green_y=[],[]

for i in range(len(reduced_x)):
    if y[i]==0:
        red_x.append(reduced_x[i][0])
        red_y.append(reduced_x[i][1])
    elif y[i]==1:
        blue_x.append(reduced_x[i][0])
        blue_y.append(reduced_x[i][1])
    else:
        green_x.append(reduced_x[i][0])
        green_y.append(reduced_x[i][1])

#可视化
plt.scatter(red_x,red_y,c='r',marker='x')
plt.scatter(blue_x,blue_y,c='b',marker='D')
plt.scatter(green_x,green_y,c='g',marker='.')
plt.show()

```

样本点的分布如图8.9所示。

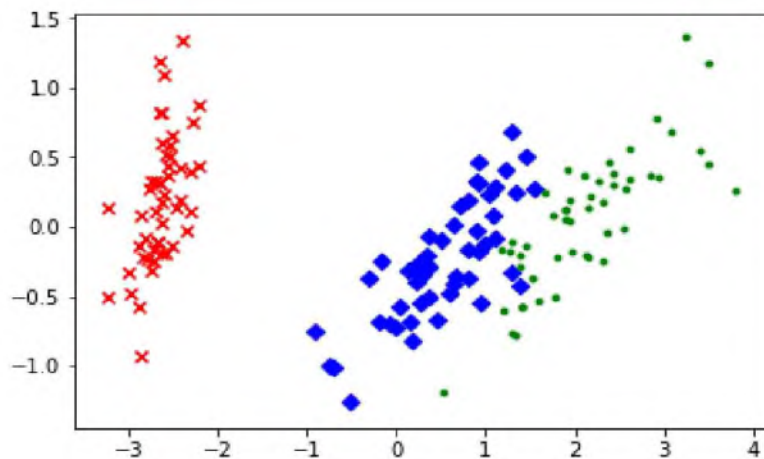


图8.9 主成分分析 (PCA)

可以看出，降维后的数据仍能够清晰地分成3类。这样不仅能削减数据的维度、降低分类任务的工作量，还能保证分类的质量。

8.11 因子分析

因子分析（Factor Analysis）是一种数据简化的技术。它通过研究众多变量之间的内部依赖关系探求观测数据中的基本结构，并用少数几个假想变量来表示其基本的数据结构。这几个假想变量能够反映原来众多变量的主要信息。原始的变量是可观测的显在变量，而假想变量是不可观测的潜在变量，称为因子。

例如，在企业形象或品牌形象的研究中，消费者可以通过一个由24个指标构成的评价体系评价百货商场的24个方面的优劣。但消费者主要关心三个方面，即商店的环境、商店的服务和商品的价格。因子分析方法可以通过24个变量找出反映商店环境、商店服务水平和商品价格的3个潜在的因子，对商店进行综合评价。

这3个公共因子可以表示为：

$$x_i = \mu_i + \alpha_{i1}F_1 + \alpha_{i2}F_2 + \alpha_{i3}F_3 + \varepsilon_i \quad i=1,2,\dots,24 \quad (8.23)$$

称 F_1 、 F_2 、 F_3 是不可观测的潜在因子。24个变量共享这3个因子，但是每个变量又有自己的个性，即不被包含的部分 ε_i ，称为特殊因子。

1. 因子分析与主成分分析的区别

- 主成分分析仅仅是变量变换，而因子分析需要构造因子模型。
- 因子分析：潜在的假想变量和随机影响变量的线性组合表示原始变量。

- 主成分分析：原始变量的线性组合表示新的综合变量，即主成分。

(1) 因子分析数学模型

假设有 p 个变量 X ，有 m 个因子（ $m \leq p$ ），则因子分析的数学模型可以表示如下：

$$X_i = \mu_i + a_{i1}F_1 + a_{i2}F_2 + \cdots + a_{im}F_m + \varepsilon_i \quad i=1, \dots, m \ (m \leq p) \quad (8.24)$$

称 $F_1, F_2, \dots, F_m$ 为公共因子，是不可观测的变量，它们的系数称为因子负载。 ε_i 是特殊因子，是不能被前 m 个公共因子包含的部分。其中，因子 $F_1, F_2, \dots, F_m$ 之间互不相关，方差等于1。因子载荷 a_{ij} 是第 i 个变量与第 j 个公共因子的相关系数，反映了第 j 个公共因子对第 i 个变量的影响程度。

2. 因子旋转

若因子分析中得出的各个因子有明确的含义，则因子分析的模型会更加易于解释和有实际意义。在因子分析中可以对因子载荷矩阵进行旋转，使每个变量仅在一个公共因子上有较大的载荷，而在其余的公共因子上的载荷比较小。通过旋转，因子可以有更加明确的含义。常用的一种方法是方差最大旋转。

3. 因子得分及其计算

前面我们主要解决了用公共因子的线性组合来表示一组观测变量的有关问题。如果要使用这些因子做其他的研究，比如把得到的因子作为自变量来进行回归分析，对样本进行分类或评价，就需要计算每个个体在每个因子上的得分。

要计算因子得分，需要估计以下表达式：

$$F_j = b_{j0} + b_{j1}X_1 + \cdots + b_{jp}X_p \quad (8.25)$$

因子得分有多种计算方法，常用的一种是回归法。若对变量都进行标准化，则模型中没有常数项。

4. 因子分析的步骤

因子分析解决的3个基本问题：

- 因子载荷阵A的估计。
- 当因子难以得到合理的解释时，对因子载荷阵进行正交变换，即因子旋转。对因子的实际意义做出合理的解释。
- 给出每个变量（或样品）关于m个公共因子的得分，通常表示为原始变量的线性组合，即因子得分函数。对公共因子做出估计。

因子分析的步骤：

- （1）根据问题选取原始变量。
- （2）求其相关阵R，探讨其相关性。
- （3）从R求解初始公共因子F及因子载荷矩阵A（主成分法）。
- （4）因子旋转，分析因子的含义。
- （5）计算因子得分函数。
- （6）根据因子得分值进行进一步分析（例如综合评价）。

5. 因子分析与主成分分析的区别与联系

(1) 因子分析、主成分分析都是重要的降维方法（即数据简化技术），因子分析可以看作主成分分析的推广和发展。

(2) 主成分分析不能作为一个模型来描述，它只能作为一般的变量变换，主成分是可观测的原始变量的线性组合。因子分析需要构造因子模型，公共因子是潜在的不可观测的变量，一般不能表示为原始变量的线性组合。

(3) 因子分析是用潜在的、不可观测的变量和随机影响变量的线性组合来表示原始变量，即通过这样的分解来分析原始变量的协方差结构（相依关系）。

6. 基于sklearn的因子分析

【例8.21】

用iris（鸢尾花）数据集进行因子分析。

```
#Getting ready
from sklearn import datasets
iris = datasets.load_iris()
from sklearn.decomposition import FactorAnalysis

fa = FactorAnalysis(n_components=2)
iris_two_dim = fa.fit_transform(iris.data)
iris_two_dim[:5]

#matplotlib inline
from matplotlib import pyplot as plt
f = plt.figure(figsize=(5, 5))
ax = f.add_subplot(111)
ax.scatter(iris_two_dim[:,0], iris_two_dim[:, 1],
c=iris.target)
ax.set_title("Factor Analysis 2 Components")
```

对iris（鸢尾花）数据集基于sklearn的因子分析如图8.10所示。

由于因子分析是一种概率性的转换方法，因此可以通过不同的角度来观察，例如模型观测值的对数似然估计值，通过模型比较对数似然估计值会更好。

因子分析也有不足之处。由于用户不是通过拟合模型直接预测结果，拟合模型只是一个中间步骤。这本身并非坏事，但是训练实际模型时误差就会产生。

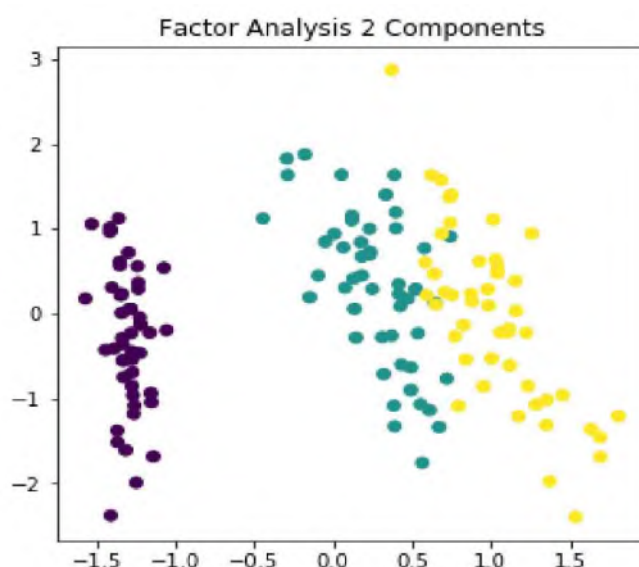


图8.10 对iris（鸢尾花）数据集基于sklearn的因子分析

因子分析的基本假设是，有个重要特征和它们的线性组合（加噪声）能够构成原始的 N 维数据集。也就是说，不需要指定结果变量（就是最终生成 N 维），而是要指定数据模型的因子数量。

8.12 时间序列分析

在生产和科学研究中，对某一个或者一组变量进行观察测量，将在一系列时刻所得到的离散数字组成的序列集合称为时间序列。时间序列分析是根据系统观察得到的时间序列数据，通过曲线拟合和参数

估计来建立数学模型的理论和方法。时间序列分析常用于国民宏观经济控制、市场潜力预测、气象预测、农作物害虫灾害预报等方面。

1. 基本模型

自回归移动平均模型（ARMA(p , q)）是时间序列中最为重要的模型之一，主要由两部分组成：AR代表 p 阶自回归过程；MA代表 q 阶移动平均过程。其公式如下：

$$Z_t = \varphi_1 Z_{t-1} + \varphi_2 Z_{t-2} + \cdots + \varphi_p Z_{t-p} + a_t - \theta_1 a_{t-1} - \cdots - \theta_q a_{t-q} \quad (8.26)$$

在时间序列中，ARIMA模型在ARMA模型的基础上多了差分的操作。

2. 时间序列建模的基本步骤

（1）获取被观测系统的时间序列数据。

（2）对数据绘图，观测是否为平稳时间序列。对于非平稳时间序列，要先进行 d 阶差分运算，化为平稳时间序列。

（3）经过第二步处理，已经得到平稳时间序列。要对平稳时间序列分别求得其自相关系数ACF和偏自相关系数PACF，通过对自相关图和偏自相关图的分析得到最佳的阶层 p 和阶数 q 。

（4）由以上得到ARIMA模型。然后开始对得到的模型进行模型检验。

3. ARIMA实战解剖

【例8.22】

```
from __future__ import print_function
import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.graphics.api import qqplot
```

这里我们使用一个具有周期性的测试数据进行分析。

```
# 获取数据
dta=
[10930,10318,10595,10972,7706,6756,9092,10551,9722,10913,11151,
8186,6422,

6337,11649,11652,10310,12043,7937,6476,9662,9570,9981,9331,9449
,6773,6304,9355,

10477,10148,10395,11261,8713,7299,10424,10795,11069,11602,11427
,9095,7707,10767,

12136,12812,12006,12528,10329,7818,11719,11683,12603,11495,1367
0,11337,10232,

13261,13230,15535,16837,19598,14823,11622,19391,18177,19994,147
23,15694,13248,

9543,12872,13101,15053,12619,13749,10228,9725,14729,12518,14564
,15085,14722,
11999,9390,13481,14795,15845,15271,14686,11054,10395]
dta=pd.Series(dta)
dta.index =
pd.Index(sm.tsa.datetools.dates_from_range('2001','2090'))
dta.plot(figsize=(12,8))
```

数据绘图如图8.11所示。

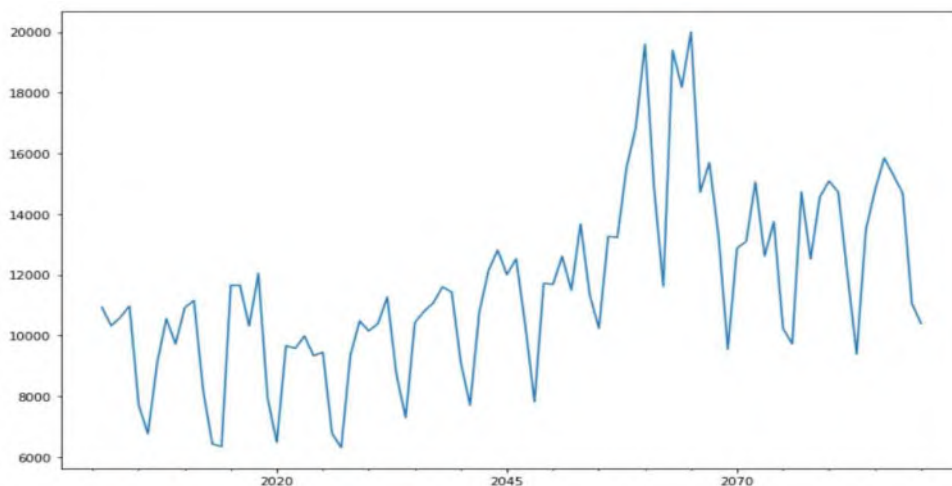


图8.11 数据绘图

ARIMA模型对时间序列的要求是平稳型。因此，当得到一个非平稳的时间序列时，首先要做的是进行时间序列的差分，直到得到一个平稳时间序列。如果对时间序列进行 d 次差分才能得到一个平稳序列，那么可以使用 $ARIMA(p, d, q)$ 模型，其中 d 是差分次数。

```
#时间序列的差分
fig = plt.figure(figsize=(12,8))
ax1= fig.add_subplot(111)
diff1 = dta.diff(1)
diff1.plot(ax=ax1)
```

时间序列一阶差分图如图8.12所示。

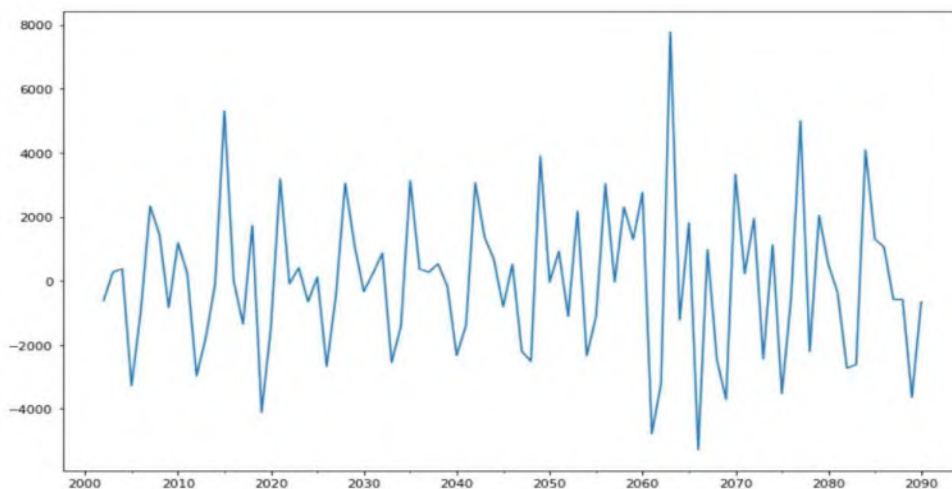


图8.12 时间序列一阶差分

一阶差分的时间序列的均值和方差已经基本平稳，不过我们还是可以比较一下二阶差分的效果。

```
#二阶差分
fig = plt.figure(figsize=(12,8))
ax2= fig.add_subplot(111)
diff2 = dta.diff(2)
diff2.plot(ax=ax2)
```

时间序列二阶差分图如图8.13所示。

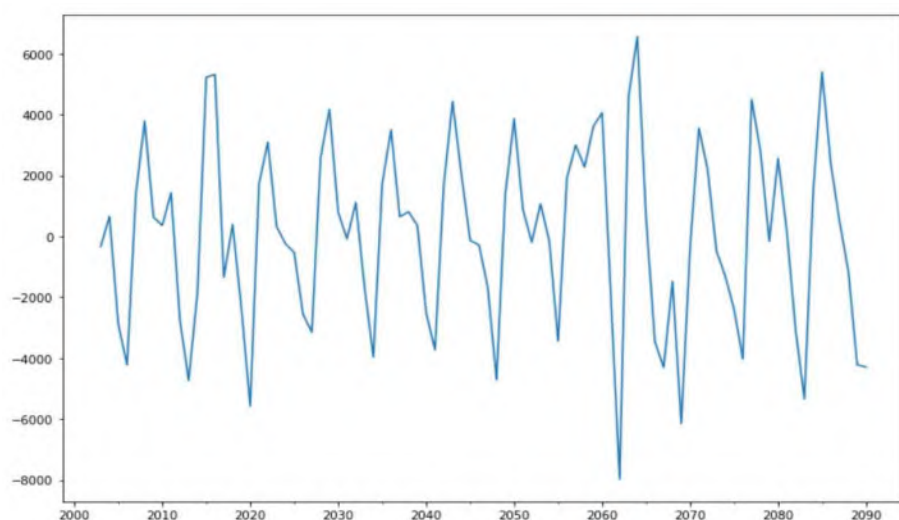


图8.13 时间序列二阶差分

可以看出二阶差分后的时间序列与一阶差分相差不大，并且二者随着时间的推移，时间序列的均值和方差保持不变。因此，可以将差分次数 d 设置为1。

8.13 生存分析

什么是生存？生存的意义很广泛，可以指人或动物的存活（相对于死亡），可以是患者的病情正处于缓解状态（相对于再次复发或恶

化），还可以是某个系统或产品正常工作（相对于失效或故障），甚至可是是客户的流失与否，等等。在生存分析中，研究的主要对象是寿命超过某一时间的概率。还可以描述其他一些事情发生的概率，例如产品的失效、出狱犯人第一次犯罪、失业人员第一次找到工作等。在某些领域的分析中，经常用追踪的方式来研究事物的发展规律，比如研究某种药物的疗效、手术后的存活时间、某件机器的使用寿命等。

1. 概念

生存分析是对一个或多个非负随机变量进行统计推断，研究生存现象和响应时间数据及其统计规律的一门学科。

生存分析是既考虑结果又考虑生存时间，并可充分利用截尾数据所提供的不完全信息对生存时间的分布特征进行描述，对影响生存时间的主要因素进行分析。

2. 生存分析研究的内容

（1）描述生存过程

研究生存时间的分布特点，估计生存率及平均存活时间，绘制生存曲线等，根据生存时间的长短可以估算出各个时点的生存率，并根据生存率来估计中位生存时间，也可以根据生存曲线分析其生存特点，一般使用Kaplan-Meier法和寿命表法。

（2）比较生存过程

可通过生存率及其标准误对各样本的生存率进行比较，以探讨各组间的生存过程是否存在差异，一般使用Log-rank检验和Breslow检验。

（3）分析危险因素

通过生存分析模型来探讨影响生存时间和终点事件的保护因素和不利因素、因素作用的大小及方向、相对危险度的大小，基本使用Cox回归模型。

（4）建立数学模型

建立最终的数学模型通过Cox回归模型完成。

3. 生存分析对资料的基本要求

- 样本由随机抽样方法获得，要有一定的数量，死亡例数和比例不能太少。
- 完整数据所占的比例不能太少，即截尾值不宜太多。
- 截尾值出现的原因无偏性，为防止偏性，经常对被截尾的研究对象的年龄、职业、地区、病情轻重等情况进行分析。
- 生存时间尽可能精确。
- 缺项要尽量补齐。

4. 生存资料的共同特点

- 蕴含结局和时间两个方面的信息。
- 结局为两分类事件。
- 一般通过随访收集得到，随访观察往往是从某统一时间点（如入院或实施手术等某种处理措施后）开始，观察到某规定时间点截止。
- 常因失访等原因造成研究对象的生存时间数据不完整，分布类型复杂，不能简单地套用以前的方法。

5. 一些相关的基本概念

- 起始事件：反映研究对象开始生存过程的起始特征事件，如研究某一治疗对病人生存的影响的起始时间是“开始接受该治疗”。
- 终点事件（死亡事件）：出现研究者所关心的特定结局，如“病人因该疾病死亡”。
- 观察时间：从研究开始观察到研究观察结束的时间。由于研究时长无法无限延伸下去，因此研究一定会在某个特定时刻截止，而研究截止时，所有观察对象并不一定全都出现终点事件。换言之，有的研究对象在观察结束之前出现终点事件，有的直到观察结束时也没有出现终点事件，还有一些特例中途因为某些原因（如失访、意外死亡等）被迫提前结束了观察研究。
- 生存时间：观察到的存活时间，用符号 t 表示。
- 完全数据：从观察起点到死亡事件所经历的时间，生存时间是完整的。
- 截尾数据（删失值）：观察时间不是由于终点事件而结束的，而是由于失访、死于非研究因素、观察结束而对象仍存活3种原因结束的。常在截尾数据的右上角放一个“+”表示其实该对象可能活得更久。

6. 生存分析的主要方法

（1）非参数法

这类方法的特点是，无论分布形式如何，只根据样本的顺序统计量对生存率进行估计。对于两个及多个生存率进行比较，其无效假设只是假定两组或多组生存时间分布相同，而不对其具体的分布形式和参数进行推断。log-rank乘法极限法和寿命表法都是非参数法。

（2）参数法

特点是假定生存时间服从特定的参数分布，然后根据已知的分布特点对生存时间进行分析，如指数分布法、Weibull分布法、对数正态回归分布法和Logistic回归法。

(3) 半参数法

Cox比例风险回归模型就是半参数法，具体介绍它时再说明为什么叫半参数法。

7. 生存分析公式模型

(1) 概率密度函数 $f(t)$

表示每时刻死亡的概率：

$$f(t) = \lim_{\Delta t \rightarrow 0} \frac{P(\text{The probability that a person will die in } (t, t+\Delta t) \text{ time})}{\Delta t} \quad (8.27)$$

分布函数 $F(t)$ ：

$$F(t) = \int_0^t f(u) du \quad P(T \leq t) \quad (8.28)$$

(2) 生存函数 $S(t)$

生存函数 $S(t)$ ： $P(T \geq t)$ 个体生存时间大于 t 的概率。

$$S(t) = 1 - F(t) = \int_0^t f(u) du \quad (8.29)$$

(3) 危险函数 $h(t)$

$$h(t) = \lim_{\Delta t \rightarrow 0} \frac{P(\text{The probability that a patient who has lived beyond } t \text{ years will die within } (t, t + \Delta t) \text{ time})}{\Delta t}$$

$$S(t) = e^{-\int_0^t h(u) du} \quad h(t) = \frac{f(t)}{S(t)} \quad (8.30)$$

(4) 累计危险函数 $H(t)$

$$H(t) = \int_0^t h(u) du = -\ln S(t)$$
$$S(t) = e^{-\int_0^t h(u) du} = e^{-H(t)} \quad (8.31)$$

8. 生存分析的目的

估计：根据样本生存资料估计总体生存率及其他有关指标（如中位生存期等），估计不同时间的生存率、生存曲线以及中位生存期等。

比较：对不同处理组的生存率进行比较，以了解哪种方案较优。

影响因素分析：目的是为了探索 and 了解影响生存时间长短的因素，或平衡某些因素的影响后，研究某个或某些因素对生存率的影响。

预测：具有不同因素水平的个体生存预测。

9. 模拟实验代码实现

【例8.23】

问题描述：泰坦尼克号的沉没是历史上非常有名的沉船之一。1912年4月15日，泰坦尼克号在处女航时与冰山相撞沉没，2224名乘客和船员中有1502人遇难。这一耸人听闻的悲剧震惊了国际社会，并导致制定了更好的船舶安全法规。船难造成如此巨大的人员伤亡的原因之一是船上没有足够的救生艇供乘客和船员使用。虽然在沉船事件中幸存下来有运气因素，但有些人比其他人更有可能存活下来，比如妇女、儿童和上层阶级。

模拟实验分析中需要预测哪一类人更有可能存活下来。需要用机器学习的工具去预测哪些乘客在这次灾难中幸存。本例采用Anaconda 3下的Jupyter Notebook开发环境。

内容：

- 导入必要的库。
- 阅读并研究数据。
- 数据分析。
- 数据可视化。
- 清理数据。
- 选择最佳模式。
- 创建提交文件。

```
#导入必要的库
In[1]:
#data analysis libraries
import numpy as np
import pandas as pd

#visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

#ignore warnings
import warnings
warnings.filterwarnings('ignore')
#读取并探索数据
In[2]:
#import train and test CSV files
train = pd.read_csv("D:\Anaconda3\workspace/train.csv")
test = pd.read_csv("D:\Anaconda3\workspace/test.csv")

#take a look at the training data
train.describe(include="all")
```

输出：

```

Out[2]:
Passenger  Id      Survived    Pclass    Name      Sex Age      SibSp
Parch      Ticket  Fare      Cabin    Embarked
count      891.000000  891.000000  891.000000  891 891 714.000000  891.000000
891.000000  891 891.000000  204 889
unique  NaN      NaN      NaN      891 2      NaN      NaN      NaN      681      NaN
147 3
top  NaN      NaN      NaN      Vande Velde, CA.      B96
Mr. Johannes Joseph  male  NaN NaN
NaN      2343      NaN      B98 S
freq NaN NaN NaN 1 577 NaN NaN NaN 7      NaN 4      644
mean 446.000000  0.383838  2.308642  NaN NaN 29.699118  0.523008
0.381594  NaN 32.204208  NaN NaN
std 257.353842  0.486592  0.836071  NaN NaN 14.526497  1.102743
0.806057  NaN 49.693429  NaN NaN
min 1.000000  0.000000  1.000000  NaN NaN 0.420000  0.000000
0.000000  NaN 0.000000  NaN NaN
25% 223.500000  0.000000  2.000000  NaN NaN 20.125000  0.000000
0.000000  NaN 7.910400  NaN NaN
50% 446.000000  0.000000  3.000000  NaN NaN 28.000000  0.000000
0.000000  NaN 14.454200  NaN NaN
75% 668.500000  1.000000  3.000000  NaN NaN 38.000000  1.000000
0.000000  NaN 31.000000  NaN NaN
max 891.000000  1.000000  3.000000  NaN NaN 80.000000  8.000000
6.000000  NaN 512.329200  NaN NaN

```

数据分析，将考虑数据集中的特性以及它们的完整性：

```

In[3]:
#get a list of the features within the dataset
print(train.columns)

```

输出训练数据的列属性：

```

Out[3]:
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
      'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')

```

参阅数据集示例了解变量：

```

In[4]:
#see a sample of the dataset to get an idea of the variables
train.sample(5)

Out[4]:

```

Passenger	Id	Survived	Pclass	Name	Sex	Age	SibSp
Parch	Ticket	Fare	Cabin	Embarked			
609	610	1	1	Shutes, Miss. Elizabeth W	female	40.0	0
17582	153.4625	C125	S				PC
227	228	0	3	Lovell, Mr. John Hall ("Henry")	male	20.5	0
A/5	21173	7.2500	NaN	S			0
204	205	1	3	Cohen, Mr. Gurshon "Gus"	male	18.0	0
3540	8.0500	NaN	S				A/5
839	840	1	1	Marechal, Mr. Pierre	male	NaN	0
11774	29.7000	C47	C				0
514	515	0	3	Coleff, Mr. Satio	male	24.0	0
349209	7.4958	NaN	S				0

训练数据集的摘要:

```

In[5]:
#see a summary of the training dataset
train.describe(include = "all")

Out[5]:

```

	Passenger Id	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.000000	891.000000	891.000000	891	891
unique	NaN	NaN	NaN	891	2	NaN	NaN	NaN	681	NaN	147	3
top	NaN	NaN	NaN	Vande Velde,				CA.		B96		
					Mr. Johannes Joseph	male	NaN	NaN				
	NaN		2343	NaN	B98	S						
freq	NaN	NaN	NaN	1	577	NaN	NaN	NaN	7	NaN	4	644
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008					
	0.381594	NaN	32.204208	NaN	NaN							
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743					
	0.806057	NaN	49.693429	NaN	NaN							
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000					
	0.000000	NaN	0.000000	NaN	NaN							
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000					
	0.000000	NaN	7.910400	NaN	NaN							
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000					
	0.000000	NaN	14.454200	NaN	NaN							
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000					
	0.000000	NaN	31.000000	NaN	NaN							
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000					
	6.000000	NaN	512.329200	NaN	NaN							

检查其他不可用的值:

```
In[6]:
#check for any other unusable values
print(pd.isnull(train).sum())
```

```
Out[5]:
PassengerId      0
Survived          0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch           0
Ticket           0
Fare             0
Cabin           687
Embarked         2
dtype: int64
```

画一个以性别做对比条件的生存几率图，这是因为全球都说lady first，女士优先：

```
In[7]:
#draw a bar plot of survival by sex
sns.barplot(x="Sex", y="Survived", data=train)

#print percentages of females vs. males that survive
print("Percentage of females who survived:", train["Survived"][train["Sex"] ==
'female'].value_counts(normalize = True)[1]*100)

print("Percentage of males who survived:", train["Survived"][train["Sex"] ==
'male'].value_counts(normalize = True)[1]*100)
```

```
Out[7]:
Percentage of females who survived: 74.20382165605095
Percentage of males who survived: 18.890814558058924
```

性别生存率对比图如图8.14所示。

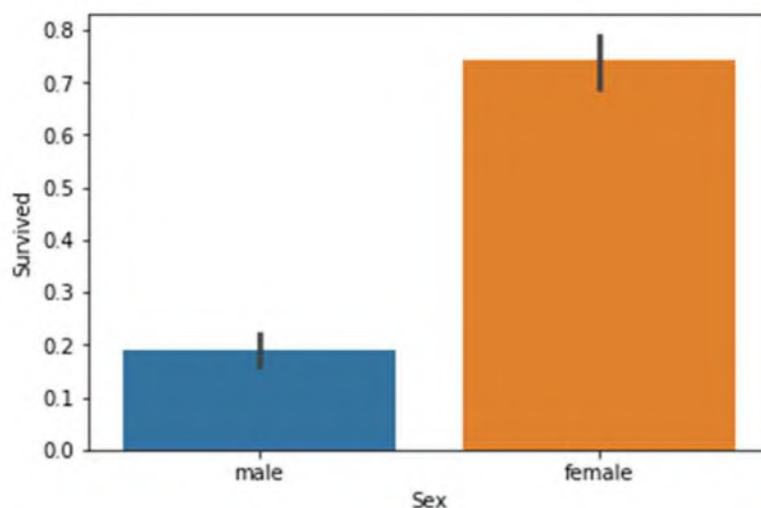


图8.14 性别生存率对比图

按阶层生存率对比，坐船有等级之分，像高铁、飞机都有。这个属性会对生存率产生影响。因为一般有钱人、权贵才会住头等舱。

```
In[8]:
#draw a bar plot of survival by Pclass
sns.barplot(x="Pclass", y="Survived", data=train)

#print percentage of people by Pclass that survived
print("Percentage of Pclass = 1 who survived:", train["Survived"]
[train["Pclass"] == 1].value_counts(normalize = True)[1]*100)

print("Percentage of Pclass = 2 who survived:", train["Survived"]
[train["Pclass"] == 2].value_counts(normalize = True)[1]*100)

print("Percentage of Pclass = 3 who survived:", train["Survived"]
[train["Pclass"] == 3].value_counts(normalize = True)[1]*100)

Out[8]:
Percentage of Pclass = 1 who survived: 62.96296296296296
Percentage of Pclass = 2 who survived: 47.28260869565217
Percentage of Pclass = 3 who survived: 24.236252545824847
```

按阶层生存率对比图如图8.15所示。

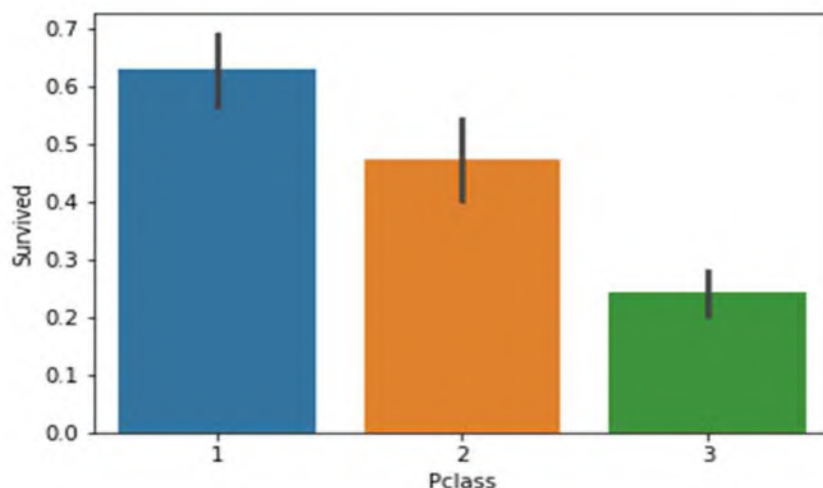


图8.15 按阶层生存率对比图

兄弟姐妹生存率对比，一些人是和兄弟姐妹一起上船的。这个会有影响，因为有可能因为救兄弟姐妹而导致自己没有上救生船。

```
In[9]:
#draw a bar plot for SibSp vs. survival
sns.barplot(x="SibSp", y="Survived", data=train)

#I won't be printing individual percent values for all of these.
print("Percentage of SibSp = 0 who survived:", train["Survived"][train["SibSp"]
== 0].value_counts(normalize = True)[1]*100)

print("Percentage of SibSp = 1 who survived:", train["Survived"][train["SibSp"]
== 1].value_counts(normalize = True)[1]*100)

print("Percentage of SibSp = 2 who survived:", train["Survived"][train["SibSp"]
== 2].value_counts(normalize = True)[1]*100)

Out[9]:
Percentage of SibSp = 0 who survived: 34.53947368421053
Percentage of SibSp = 1 who survived: 53.588516746411486
Percentage of SibSp = 2 who survived: 46.42857142857143
```

兄弟姐妹生存率对比图如图8.16所示。

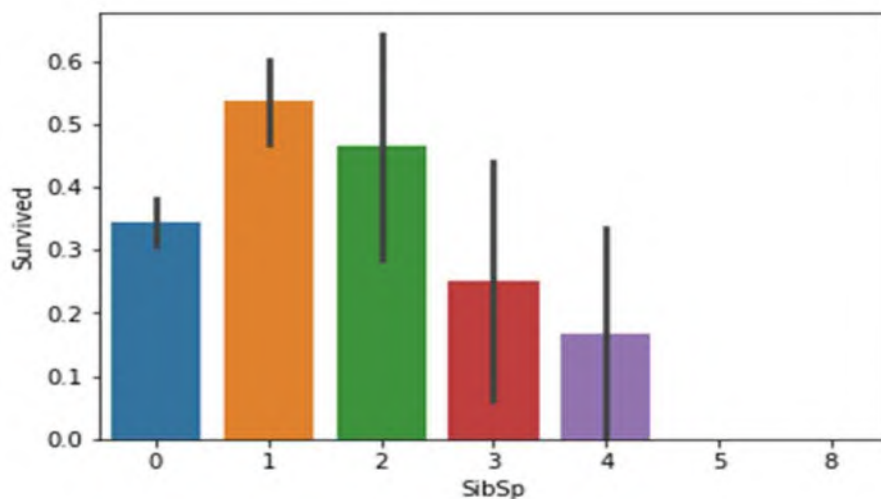


图8.16 兄弟姐妹生存率对比图

父母和小孩生存率对比，有些人是带着父母、小孩上船的，有可能因为要救父母、小孩耽误上救生船。

```
In[10]:  
#draw a bar plot for Parch vs. survival  
sns.barplot(x="Parch", y="Survived", data=train)  
plt.show()
```

父母和小孩生存率对比图如图8.17所示。

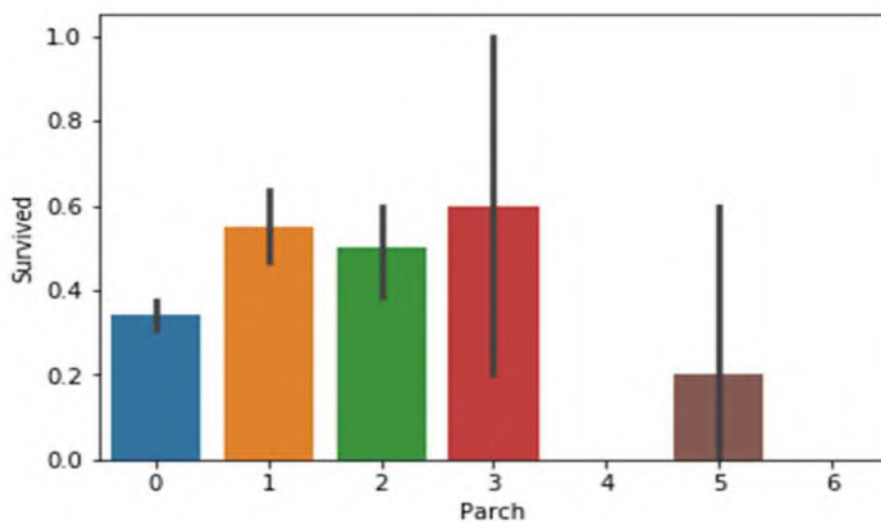


图8.17 父母和小孩生存率对比图

按年龄逻辑分类：

```
In[11]:
#sort the ages into logical categories
train["Age"] = train["Age"].fillna(-0.5)
test["Age"] = test["Age"].fillna(-0.5)
bins = [-1, 0, 5, 12, 18, 24, 35, 60, np.inf]
labels = ['Unknown', 'Baby', 'Child', 'Teenager', 'Student', 'Young Adult',
'Adult', 'Senior']
train['AgeGroup'] = pd.cut(train["Age"], bins, labels = labels)
test['AgeGroup'] = pd.cut(test["Age"], bins, labels = labels)

#draw a bar plot of Age vs. survival
sns.barplot(x="AgeGroup", y="Survived", data=train)
plt.show()
```

按年龄逻辑分类图如图8.18所示。

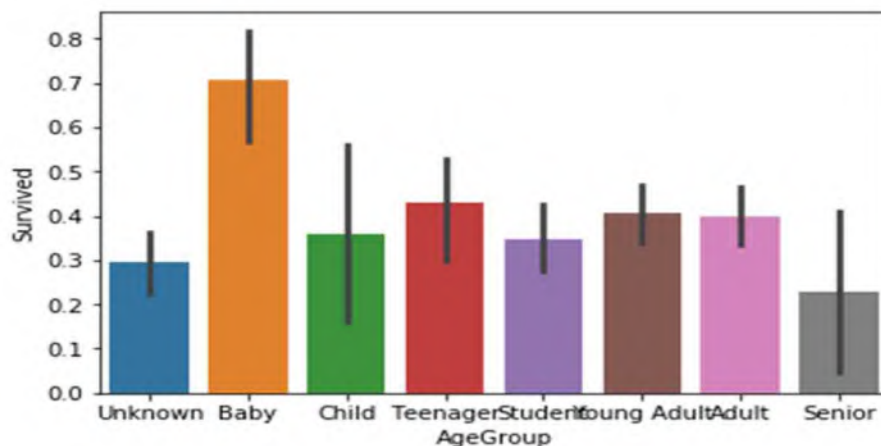


图8.18 按年龄逻辑分类图

计算CabinBool与存活的百分比：

```

In[12]:
train["CabinBool"] = (train["Cabin"].notnull().astype('int'))
test["CabinBool"] = (test["Cabin"].notnull().astype('int'))

#calculate percentages of CabinBool vs. survived
print("Percentage of CabinBool = 1 who survived:", train["Survived"][train
["CabinBool"] == 1].value_counts(normalize = True)[1]*100)

print("Percentage of CabinBool = 0 who survived:", train["Survived"][train
["CabinBool"] == 0].value_counts(normalize = True)[1]*100)
#draw a bar plot of CabinBool vs. survival
sns.barplot(x="CabinBool", y="Survived", data=train)
plt.show()

```

```

Percentage of CabinBool = 1 who survived: 66.66666666666666
Percentage of CabinBool = 0 who survived: 29.985443959243085

```

CabinBool与存活的百分比图如图8. 19所示。

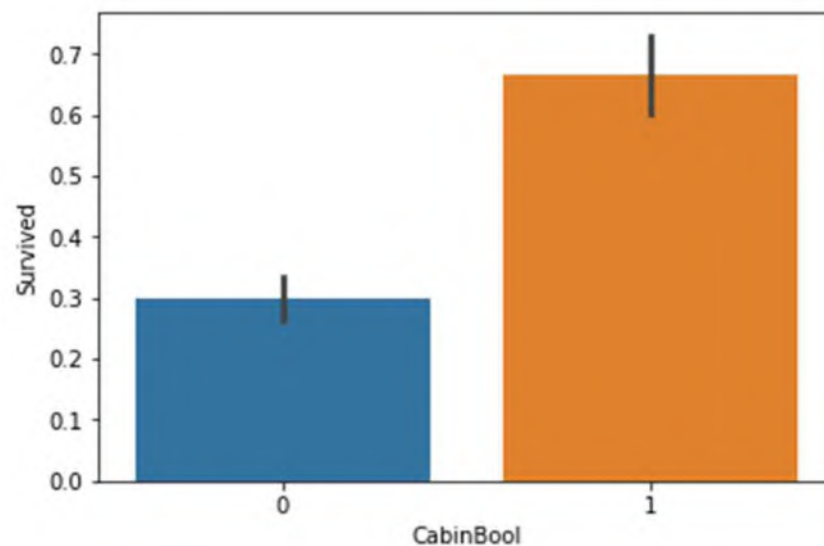


图8. 19 CabinBool与存活的百分比

填充登录地点缺失的值：

```
In[13]:
#we'll start off by dropping the Cabin feature since not a lot more useful
information can be extracted from it.
train = train.drop(['Cabin'], axis = 1)
test = test.drop(['Cabin'], axis = 1)
#we can also drop the Ticket feature since it's unlikely to yield any useful
information
train = train.drop(['Ticket'], axis = 1)
test = test.drop(['Ticket'], axis = 1)
#now we need to fill in the missing values in the Embarked feature
print("Number of people embarking in Southampton (S):")
southampton = train[train["Embarked"] == "S"].shape[0]
print(southampton)

print("Number of people embarking in Cherbourg (C):")
cherbourg = train[train["Embarked"] == "C"].shape[0]
print(cherbourg)

print("Number of people embarking in Queenstown (Q):")
queenstown = train[train["Embarked"] == "Q"].shape[0]
print(queenstown)

Out[13]:
Number of people embarking in Southampton (S):
644
Number of people embarking in Cherbourg (C):
168
Number of people embarking in Queenstown (Q):
77
```

创建两个数据集（train & test）组合：

```

In[14]:
#replacing the missing values in the Embarked feature with S
train = train.fillna({"Embarked": "S"})
#create a combined group of both datasets
combine = [train, test]
#extract a title for each Name in the train and test datasets
for dataset in combine:
    dataset['Title'] = dataset.Name.str.extract(' ([A-Za-z]+)\.', expand=False)

pd.crosstab(train['Title'], train['Sex'])

Out[14]:
Sex      female  male
Title
Capt         0      1
Col           0      2
Countess      1      0
Don           0      1
Dr            1      6
Jonkheer      0      1
Lady          1      0
Major         0      2
Master        0     40
Miss         182      0
Mlle          2      0
Mme           1      0
Mr            0     517
Mrs          125      0
Ms            1      0
Rev           0      6
Sir           0      1

```

用常用名称替换标题:

```

In[15]:
#replace various titles with more common names
for dataset in combine:
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Capt', 'Col',
        'Don', 'Dr', 'Major', 'Rev', 'Jonkheer', 'Dona'], 'Rare')

    dataset['Title'] = dataset['Title'].replace(['Countess', 'Lady', 'Sir'],
        'Royal')
    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')

train[['Title', 'Survived']].groupby(['Title'], as_index=False).mean()

Out[15]:
   Title  Survived
0  Master    0.575000
1   Miss    0.702703
2    Mr     0.156673
3   Mrs     0.793651
4   Rare     0.285714
5  Royal     1.000000

```

将每个标题组映射到一个数值：


```

In[16]:
#map each of the title groups to a numerical value
title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Royal": 5, "Rare":
6}

for dataset in combine:
    dataset['Title'] = dataset['Title'].map(title_mapping)
    dataset['Title'] = dataset['Title'].fillna(0)

train.head()

Out[16]:
Passenger Id Survived Pclass          Name
Sex  Age  SibSp  Parch  Fare  Embarked  AgeGroup  CabinBool  Title
0  1  0          3  Braund, Mr. Owen Harris
male 22.0  1          0    7.2500      S  Student 0    1
1  2  1          1  Cumings, Mrs. John Bradley (Florence Briggs Th... female
38.0  1    0    71.2833      C  Adult  1    3
2  3  1          3  Heikkinen, Miss. Laina
female 26.0  0    0    7.9250      S  Young Adult 0    2
3  4  1          1  Futrelle, Mrs. Jacques Heath (Lily May Peel)
female 35.0  1    0   53.1000      S  Young Adult 1    3
4  5  0          3  Allen, Mr. William Henry
male 35.0  0    0    8.0500      S  Young Adult 0    1

```

为每一个标题填充缺失的年龄与模式年龄组：

```

In[17]:
# fill missing age with mode age group for each title
mr_age = train[train["Title"] == 1]["AgeGroup"].mode() #Young Adult
miss_age = train[train["Title"] == 2]["AgeGroup"].mode() #Student
mrs_age = train[train["Title"] == 3]["AgeGroup"].mode() #Adult
master_age = train[train["Title"] == 4]["AgeGroup"].mode() #Baby
royal_age = train[train["Title"] == 5]["AgeGroup"].mode() #Adult
rare_age = train[train["Title"] == 6]["AgeGroup"].mode() #Adult

age_title_mapping = {1: "Young Adult", 2: "Student", 3: "Adult", 4: "Baby", 5:
"Adult", 6: "Adult"}

#I tried to get this code to work with using .map(), but couldn't.
#I've put down a less elegant, temporary solution for now.
#train = train.fillna({"Age": train["Title"].map(age_title_mapping)})
#test = test.fillna({"Age": test["Title"].map(age_title_mapping)})

for x in range(len(train["AgeGroup"])):
    if train["AgeGroup"][x] == "Unknown":
        train["AgeGroup"][x] = age_title_mapping[train["Title"][x]]

for x in range(len(test["AgeGroup"])):
    if test["AgeGroup"][x] == "Unknown":
        test["AgeGroup"][x] = age_title_mapping[test["Title"][x]]

```

将每个年龄值映射到一个数值：

```

In[18]:
#map each Age value to a numerical value
age_mapping = {'Baby': 1, 'Child': 2, 'Teenager': 3,
'Student': 4, 'Young Adult':
5, 'Adult': 6, 'Senior': 7}
train['AgeGroup'] = train['AgeGroup'].map(age_mapping)
test['AgeGroup'] = test['AgeGroup'].map(age_mapping)

train.head()

#dropping the Age feature for now, might change
train = train.drop(['Age'], axis = 1)
test = test.drop(['Age'], axis = 1)

```

删除name特性，因为它不包含更多有用的信息：

```

In[19]:
#drop the name feature since it contains no more useful

```

```
information.  
train = train.drop(['Name'], axis = 1)  
test = test.drop(['Name'], axis = 1)
```

将每个性别值映射到一个数值：

```
In[20]:  
#map each Sex value to a numerical value  
sex_mapping = {"male": 0, "female": 1}  
train['Sex'] = train['Sex'].map(sex_mapping)  
test['Sex'] = test['Sex'].map(sex_mapping)  
  
train.head()  
  
Out[20]:  
Passenger  Id    Survived  Pclass  Sex SibSp  Parch  Fare  Embarked  
AgeGroup  CabinBool  Title  
0    1      0    3    0    1    0    7.2500    S      4      0      1  
1    2      1    1    1    1    0   71.2833    C      6      1      3  
2    3      1    3    1    0    0    7.9250    S      5      0      2  
3    4      1    1    1    1    0   53.1000    S      5      1      3  
4    5      0    3    0    0    0    8.0500    S      5      0      1
```

将每个登船值映射到一个数值：

```
In[21]:  
#map each Embarked value to a numerical value  
embarked_mapping = {"S": 1, "C": 2, "Q": 3}  
train['Embarked'] = train['Embarked'].map(embarked_mapping)  
test['Embarked'] = test['Embarked'].map(embarked_mapping)  
  
train.head()  
  
Out[21]:  
Passenger  Id  Survived  Pclass  Sex SibSp  Parch  Fare  Embarked  
AgeGroup  CabinBool  Title  
0    1      0      3    0    1    0    7.2500    1      4      0      1  
1    2      1      1    1    1    0   71.2833    2      6      1      3  
2    3      1      3    1    0    0    7.9250    1      5      0      2  
3    4      1      1    1    1    0   53.1000    1      5      1      3  
4    5      0      3    0    0    0    8.0500    1      5      0      1
```

根据舱级的平均票价，在test集中填写缺失的票价值：

```

In[22]:
#fill in missing Fare value in test set based on mean fare for that Pclass
for x in range(len(test["Fare"])):
    if pd.isnull(test["Fare"][x]):
        pclass = test["Pclass"][x] #Pclass = 3
        test["Fare"][x] = round(train[train["Pclass"] == pclass]["Fare"].mean(),
4)

#map Fare values into groups of numerical values
train['FareBand'] = pd.qcut(train['Fare'], 4, labels = [1, 2, 3, 4])
test['FareBand'] = pd.qcut(test['Fare'], 4, labels = [1, 2, 3, 4])

#drop Fare values
train = train.drop(['Fare'], axis = 1)
test = test.drop(['Fare'], axis = 1)

```

检测train集的数据:

```

In[23]:
#check train data
train.head()

Out[23]:

```

Passenger	Id	Survived	Pclass	Sex	SibSp	Parch	Embarked	AgeGroup
CabinBool	Title	FareBand						
0	1	0	3	0	1	0	1	1
1	2	1	1	1	1	0	2	4
2	3	1	3	1	0	0	1	5
3	4	1	1	1	1	0	1	5
4	5	0	3	0	0	0	1	5

检测test集的数据:

```

In[24]:
#check test data
test.head()

Out[24]:

```

Passenger	Id	Pclass	Sex	SibSp	Parch	Embarked	AgeGroup
CabinBool	Title	FareBand					
0	892	3	0	0	0	3	5
1	893	3	1	1	0	1	6
2	894	2	0	0	0	3	7
3	895	3	0	0	0	1	5
4	896	3	1	1	1	1	4

生存预测模型分析:

```
In[25]:
from sklearn.model_selection import train_test_split

predictors = train.drop(['Survived', 'PassengerId'],
axis=1)
target = train["Survived"]
x_train, x_val, y_train, y_val =
train_test_split(predictors, target, test_size
= 0.22, random_state = 0)
```

高斯朴素贝叶斯:

```
In[26]:
# Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

gaussian = GaussianNB()
gaussian.fit(x_train, y_train)
y_pred = gaussian.predict(x_val)
acc_gaussian = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_gaussian)
Out[26]:
78.68
```

逻辑回归:

```
In[27]:
# Logistic Regression
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit(x_train, y_train)
y_pred = logreg.predict(x_val)
acc_logreg = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_logreg)

Out[27]:
79.19
```

支持向量机:

```
In[28]:
# Support Vector Machines
from sklearn.svm import SVC

svc = SVC()
svc.fit(x_train, y_train)
y_pred = svc.predict(x_val)
acc_svc = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_svc)

Out[28]:
82.74
```

线性支持向量机:

```
In[29]:
# Linear SVC
from sklearn.svm import LinearSVC

linear_svc = LinearSVC()
linear_svc.fit(x_train, y_train)
y_pred = linear_svc.predict(x_val)
acc_linear_svc = round(accuracy_score(y_pred, y_val) * 100,
2)
print(acc_linear_svc)

Out[29]:
78.17
```

感知机:

```
In[30]:
# Perceptron
from sklearn.linear_model import Perceptron

perceptron = Perceptron()
perceptron.fit(x_train, y_train)
y_pred = perceptron.predict(x_val)
acc_perceptron = round(accuracy_score(y_pred, y_val) * 100,
2)
print(acc_perceptron)

Out[30]:
79.19
```

决策树:

```
In[31]:
#Decision Tree
from sklearn.tree import DecisionTreeClassifier

decisiontree = DecisionTreeClassifier()
decisiontree.fit(x_train, y_train)
y_pred = decisiontree.predict(x_val)
acc_decisiontree = round(accuracy_score(y_pred, y_val) *
100, 2)
print(acc_decisiontree)

Out[31]:
81.22
```

随机森林:

```
In[32]:
# Random Forest
from sklearn.ensemble import RandomForestClassifier

randomforest = RandomForestClassifier()
randomforest.fit(x_train, y_train)
y_pred = randomforest.predict(x_val)
acc_randomforest = round(accuracy_score(y_pred, y_val) *
100, 2)
print(acc_randomforest)

Out[32]:
82.74
```

K-紧邻:

```
In[33]:
# KNN or k-Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(x_train, y_train)
y_pred = knn.predict(x_val)
acc_knn = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_knn)
```

```
Out[33]:  
77.66
```

随机梯度下降:

```
In[34]:  
# Stochastic Gradient Descent  
from sklearn.linear_model import SGDClassifier  
  
sgd = SGDClassifier()  
sgd.fit(x_train, y_train)  
y_pred = sgd.predict(x_val)  
acc_sgd = round(accuracy_score(y_pred, y_val) * 100, 2)  
print(acc_sgd)  
  
Out[34]:  
80.71
```

梯度提升分类器:

```
In[35]:  
# Gradient Boosting Classifier  
from sklearn.ensemble import GradientBoostingClassifier  
  
gbk = GradientBoostingClassifier()  
gbk.fit(x_train, y_train)  
y_pred = gbk.predict(x_val)  
acc_gbk = round(accuracy_score(y_pred, y_val) * 100, 2)  
print(acc_gbk)  
  
Out[35]:  
84.77
```

预测模型算法得分统计:


```

In[36]:
models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes', 'Perceptron', 'Linear SVC',
              'Decision Tree', 'Stochastic Gradient Descent', 'Gradient Boosting
Classifier'],
    'Score': [acc_svc, acc_knn, acc_logreg,
              acc_randomforest, acc_gaussian, acc_perceptron, acc_linear_svc,
              acc_decisiontree, acc_sgd, acc_gbk]})
models.sort_values(by='Score', ascending=False)

Out[36]:

```

	Model	Score
9	Gradient Boosting Classifier	84.77
0	Support Vector Machines	82.74
3	Random Forest	82.74
7	Decision Tree	81.22
8	Stochastic Gradient Descent	80.71
2	Logistic Regression	79.19
5	Perceptron	79.19
4	Naive Bayes	78.68
6	Linear SVC	78.17
1	KNN	77.66

创建生存预测文件，将id设置为乘客id并预测生存：

```

In[37]:
#set ids as PassengerId and predict survival
ids = test['PassengerId']
predictions = gbk.predict(test.drop('PassengerId', axis=1))

#set the output as a dataframe and convert to csv file named
submission.csv
output = pd.DataFrame({ 'PassengerId' : ids, 'Survived':
predictions })
output.to_csv('submission.csv', index=False)

Out[37]:

```

预测数据片段：

PassengerId	Survived
892	0
893	1
894	0
895	0
896	1
897	0
898	1
899	0
900	1
901	0
902	0
903	0
904	1
905	0
906	1
907	1
908	0

本次实验的数据集 `titanic.csv`、`test.csv`、`train.csv` 从 Kaggle（<https://www.kaggle.com/>）平台下载，软件平台采用 Jupyter Notebook。

8.14 典型相关分析

简单相关系数描述两组变量的相关关系存在一个缺点：只是孤立考虑单个X与单个Y间的相关，没有考虑X、Y变量组内部各变量间的相关。两组间有许多简单相关系数，使问题显得复杂，难以从整体描述。本节讲述的典型相关是简单相关、多重相关的推广。典型相关是研究两组变量之间相关性的一种统计分析方法，也是一种降维技术。

1. 典型相关分析

CCA（Canonical Correlation Analysis，典型相关分析）是利用综合变量对之间的相关关系来反映两组指标之间的整体相关性的多元统计分析方法。它的基本原理是：为了从总体上把握两组指标之间的

相关关系，分别在两组变量中提取有代表性的两个综合变量 U_1 和 V_1 （分别为两个变量组中各变量的线性组合），利用这两个综合变量之间的相关关系来反映两组指标之间的整体相关性。

典型相关分析是常用的挖掘数据关联关系的算法之一。比如我们拿到两组数据，第一组是人身高和体重的数据，第二组是对应的跑步能力和跳远能力的的数据。那么我们能不能说这两组数据是相关的呢？典型相关分析可以帮助我们分析这个问题。

2. 典型相关分析概述

在数理统计中，对相关系数这个概念应该很熟悉。假设有两组一维的数据集 X 和 Y ，则相关系数 ρ 的定义为：

$$\rho(X,Y)=\frac{\text{cov}(X,Y)}{\sqrt{D(X)}\sqrt{D(Y)}} \quad (8.32)$$

其中， $\text{cov}(X,Y)$ 是 X 和 Y 的协方差，而 $D(X)$ 、 $D(Y)$ 分别是 X 和 Y 的方差。相关系数 ρ 的取值为 $[-1,1]$ ， ρ 的绝对值越接近1，则 X 和 Y 的线性相关性越高；越接近于0，则 X 和 Y 的线性相关性越低。

虽然相关系数可以很好地帮助我们分析一维数据的相关性，但是对于高维数据，就不能直接使用相关系数了。如上所述，如果 X 是包括人身高和体重两个维度的数据，而 Y 是包括跑步能力和跳远能力两个维度的数据，就不能直接使用相关系数的方法。那么我们能不能变通一下呢？典型相关分析给了我们变通的方法。

典型相关分析使用的方法是将多维的 X 和 Y 都用线性变换为1维的 X' 和 Y' ，然后使用相关系数来看 X' 和 Y' 的相关性。将数据从多维变到1维，也可以理解为典型相关分析是在降维，将高维数据降到1维，再用相关系数进行相关性的分析。

3. 典型相关分析算法思想

上面提到典型相关分析是将高维的两组数据分别降维到1维，然后用相关系数分析相关性。但是有一个问题是，降维的标准是如何选择的呢？回想一下主成分分析（PCA），降维的原则是投影方差最大；再回想一下线性判别分析（LDA），降维的原则是同类的投影方差小，异类间的投影方差大。对于典型相关分析，它选择的投影标准是降维到1维后，两组数据的相关系数最大。

假设数据集是 X 和 Y ， X 为 $n_1 \times m$ 的样本矩阵， Y 为 $n_2 \times m$ 的样本矩阵，其中 m 为样本个数，而 n_1 、 n_2 分别为 X 和 Y 的特征维度。对于 X 矩阵，将其投影到1维，对应的投影向量为 a ，对于 Y 矩阵，将其投影到1维，对应的投影向量为 b ，这样 X 、 Y 投影后得到的一维向量分别为 X' 、 Y' 。则有：

$$X' = a^T X, Y' = b^T Y \quad (8.33)$$

典型相关分析的优化目标是最大化 $\rho(X', Y')$ ，得到对应的投影向量 a 和 b ，即：

$$\underbrace{\arg \max}_{a, b} \frac{\text{cov}(X', Y')}{\sqrt{D(X')} \sqrt{D(Y')}} \quad (8.34)$$

在投影前，一般会把原始数据标准化，得到均值为0而方差为1的数据 X 和 Y 。这样有：

$$\begin{aligned} \text{cov}(X', Y') &= \text{cov}(a^T X, b^T Y) = E(<a^T X, b^T Y>) = E((a^T X)(b^T Y)^T) = a^T E(XY^T)b \\ D(X') &= D(a^T X) = a^T E(XX^T)a \\ D(Y') &= D(b^T Y) = b^T E(YY^T)b \end{aligned} \quad (8.35)$$

由于 X 、 Y 的均值均为0，则：

$$\begin{aligned}
D(X) &= \text{cov}(X, X) = E(XX^T) \\
D(Y) &= \text{cov}(Y, Y) = E(YY^T) \\
\text{cov}(X, Y) &= E(XY^T) \\
\text{cov}(Y, X) &= E(YX^T)
\end{aligned} \tag{8.36}$$

令 $S_{XY} = \text{cov}(X, Y)$ ，则优化目标可以转化为：

$$\underbrace{\arg \max}_{a,b} \frac{a^T S_{XY} b}{\sqrt{a^T S_{XX} a} \sqrt{b^T S_{YY} b}} \tag{8.37}$$

由于分子、分母增大相同的倍数，优化目标结果不变，因此可以采用和SVM（支持向量机）类似的优化方法，固定分母，优化分子，具体的转化为：

$$\underbrace{\arg \max}_{a,b} a^T S_{XY} b \quad s.t. \quad a^T S_{XX} a = 1, b^T S_{YY} b = 1 \tag{8.38}$$

4. 特征值分解求典型相关分析

特征值分解方式比较传统，利用拉格朗日函数优化目标转化为最大化，如：

$$J(a, b) = a^T S_{XY} b - \frac{\lambda}{2} (a^T S_{XX} a - 1) - \frac{\theta}{2} (b^T S_{YY} b - 1) \tag{8.39}$$

分别对 a 、 b 求导并令结果为0，得：

$$\begin{aligned}
S_{XY} b - \lambda S_{XX} a &= 0 \\
S_{YX} a - \lambda S_{YY} b &= 0
\end{aligned} \tag{8.40}$$

进而：

$$\lambda = \theta = a^T S_{XY} b \tag{8.41}$$

现在拉格朗日系数就是我们要优化的目标。继续将上面的两个式子（8.40）进行整理得：

$$\begin{aligned} S_{XX}^{-1} S_{XY} b &= \lambda a \\ S_{YY}^{-1} S_{YX} a &= \lambda b \end{aligned} \quad (8.42)$$

将式（8.33）的下式代入上式得到：

$$S_{XX}^{-1} S_{XY} S_{YY}^{-1} S_{YX} a = \lambda^2 a \quad (8.43)$$

要求最大的相关系数 λ ，只需要对上面的矩阵进行特征分解，找出最大的特征值取平方根即可，此时最大特征值对应的特征向量即为 X 的线性系数 a 。使用同样的办法，可以找到最大特征值对应的特征向量即为 Y 的线性系数 b 。

【例8.24】

使用Python实现典型相关分析：

```
import numpy as np
from numpy import dot
import scipy.linalg as la

def clean_and_sort_eigenvalues(eigenvalues, eigenvectors):
    evs = [(va, ve) for va, ve in zip(eigenvalues, eigenvectors.T) if va.imag ==
0]

    evs.sort(key=lambda evv: evv[0], reverse=True)
    sevals = np.array([va.real for va, _ in evs])
    sevecs = np.array([ve for _, ve in evs]).T
    return sevals, sevecs
```

```

def cca(X,Y):
    """Canonical Correlation Analysis

    :param X: observation matrix in X space, every column is one data point
    :param Y: observation matrix in Y space, every column is one data point

    :returns: (basis in X space, basis in Y space, correlation)
    """

    N = X.shape[1]
    Sxx = 1.0/N * dot(X, X.T)
    Sxy = 1.0/N * dot(X, Y.T)
    Syy = 1.0/N * dot(Y, Y.T)

    epsilon = 1e-6
    rSyy = Syy + epsilon * np.eye(Syy.shape[0])
    rSxx = Sxx + epsilon * np.eye(Sxx.shape[0])
    irSyy = la.inv(rSyy)

    L = dot(Sxy, dot(irSyy, Sxy.T))
    lambda2s,A = la.eig(L, rSxx)
    lambdas = np.sqrt(lambda2s)
    clambdas, cA = clean_and_sort_eigenvalues(lambdas, A)
    B = dot(irSyy, dot(Sxy.T, dot(cA, np.diag(1.0 / clambdas))))

    return (cA, B, clambdas)

# test
if __name__ == "__main__":

    if True:
        # 3d test
        baseA = np.array([[0, 1, 0],
                           [1, 0, 0],
                           [0, 0, 1]]).T
        baseB = np.array([[0, 0, 1],
                           [0, 1, 0],
                           [1, 0, 0]]).T
        latent = np.random.random((3,1000))
    else:
        # 1d test
        baseA = np.array([[0],
                           [1],
                           [2]])
        baseB = np.array([[1],
                           [0],
                           [1]])
        latent = np.random.random((1,1000))

```

```

x = dot(baseA, latent)
y = dot(baseB, latent)

(A,B,lambdas) = cca(x,y)

#print "latent=\n",latent
#print "x=\n",x
#print "y=\n",y
print ("lambdas=\n",lambdas)
print ("A=\n",A)
print ("B=\n",B)
atx = dot(A.T,x[:,0:5])
aty = dot(B.T,y[:,0:5])
diff = la.norm(atx-aty,'fro')
print ("A^T * x=\n",atx)
print ("B^T * y=\n",aty)
print ("diff=",diff)
assert diff <= 1e-10, 'Test failed'

```

输出：

```

lambdas=
[0.99999882 0.99998803 0.99998744]
A=
[[-0.56807901 0.56701549 -0.59647269]
 [-0.59828792 0.2131334 0.7724155 ]
 [-0.56509981 -0.79565544 -0.21816191]]
B=
[[-0.56509981 -0.79565544 -0.21816191]
 [-0.56807901 0.56701549 -0.59647269]
 [-0.59828792 0.2131334 0.7724155 ]]
A^T * x=
[[-1.02745106 -1.14835092 -0.78623855 -0.83508948 -0.69155825]
 [ 0.42762923 0.26590766 0.23864677 -0.10093813 -0.56404227]
 [ 0.05990267 0.28762823 -0.29587824 -0.61157852 -0.30903221]]
B^T * y=
[[-1.02745106 -1.14835092 -0.78623855 -0.83508948 -0.69155825]
 [ 0.42762923 0.26590766 0.23864677 -0.10093813 -0.56404227]
 [ 0.05990267 0.28762823 -0.29587824 -0.61157852 -0.30903221]]
diff= 8.635787298882178e-16

```

典型相关分析算法广泛地应用于数据相关度的分析，同时还是偏最小二乘法的基础。但是由于它依赖于数据的线性表示，当我们的数

据无法线性表示时，典型相关分析就无法使用，此时我们可以利用核函数的思想将数据映射到高维后，再利用典型相关分析的思想降维到1维，求对应的相关系数和线性关系，这个算法一般称为核典型相关分析（KCCA）。深度典型相关性分析（DCCA），在线性的典型相关分析上增加了深度网络，以此来学习新特征并提高多模态数据之间的相关性。

8.15 RoC分析

在机器学习的算法评估中，尤其是分类算法评估中，经常听到精确率（Precision）与召回率（Recall）、RoC曲线与PR曲线这些概念，这些概念到底有什么用处呢？首先，需要理解几个拗口的概念。

1. TP、FP、TN和FN

- True Positives (TP)：预测为正样本，实际也为正样本的特征数。
- False Positives (FP)：预测为正样本，实际为负样本的特征数。
- True Negatives (TN)：预测为负样本，实际也为负样本的特征数。
- False Negatives (FN)：预测为负样本，实际为正样本的特征数。

听起来还是很费劲，不过我们用一幅图就很容易理解了。如图8.20所示（参看下载资源中给出的图），左侧的半圆就是TP，右侧的半圆就是FP，左侧的灰色长方形（不包括左侧的半圆）就是FN。右侧的浅灰色长方形（不包括右侧的半圆）就是TN。两个半圆组成的圆内代表我们分类得到模型结果，右侧的是正值的样本。

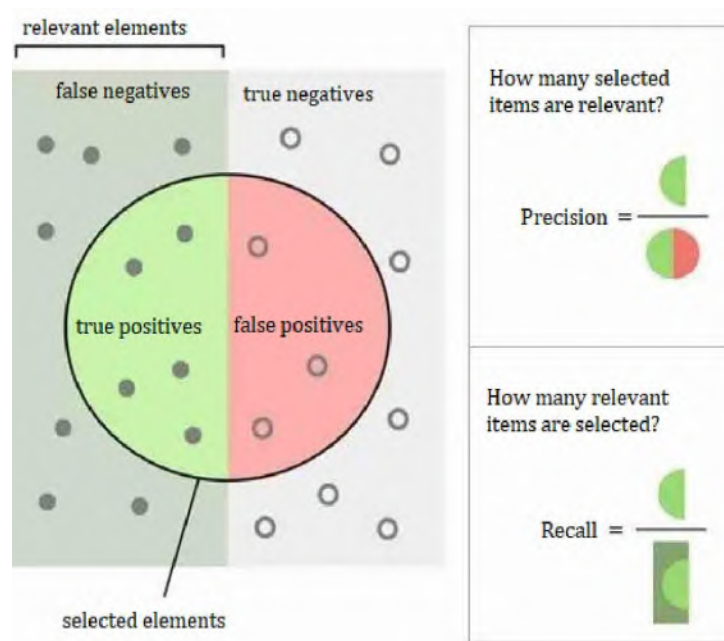


图8.20 TP、FP、TN和FN示意图

2. 精确率、召回率与特异性

精确率的定义从图8.20可以看出，是左侧的半圆除以两个半圆组成的圆。严格的数学定义如下：

$$P = \frac{TP}{TP + FP} \quad (8.44)$$

召回率的定义也可以从图8.20看出，是左边半圆除以左边的长方形。严格的数学定义如下：

$$R = \frac{TP}{TP + FN} \quad (8.45)$$

特异性 (Specificity) 的定义不能从图8.20看出，这里给出，是右边长方形去掉右边红色半圆部分后除以右边的长方形。严格的数学定义如下：

$$S = \frac{TN}{FP + TN} \quad (8.46)$$

有时用一个 F_1 值来综合评估精确率和召回率，它是精确率和召回率的调和均值。当精确率和召回率都高时， F_1 值也会高。严格的数学定义如下：

$$\frac{2}{F_1} = \frac{1}{P} + \frac{1}{R} \quad (8.47)$$

有时我们对精确率和召回率并不是一视同仁，比如有时更加重视精确率。我们用一个参数 β 来度量两者之间的关系。如果 $\beta > 1$ ，召回率就有更大影响；如果 $\beta < 1$ ，精确率就有更大影响。自然，当 $\beta = 1$ 的时候，精确率和召回率的影响力相同，和 F_1 的形式一样。含有度量参数 β 的 F_1 记为 F_β ，严格的数学定义如下：

$$F_\beta = \frac{(1 + \beta^2) * P * R}{\beta^2 * P + R} \quad (8.48)$$

此外，还有灵敏度（True Positive Rate, TPR），它是所有实际正例中正确识别的正例比例，和召回率的表达式没有区别。严格的数学定义如下：

$$TPR = \frac{TP}{TP + FN} \quad (8.49)$$

特异度（False Positive Rate, FPR）是实际负例中错误地识别为正例的负例比例。严格的数学定义如下：

$$FPR = \frac{FP}{FP + TN} \quad (8.50)$$

熟悉了精确率、召回率和特异性，以及TPR和FPR，后面的RoC曲线和PR曲线就好理解了。

3. ROC曲线

ROC (Receiver Operating Characteristic) 曲线和AUC常被用来评价一个二值分类器 (Binary Classifier) 的优劣。

需要提前说明的是，我们这里只讨论二值分类器。对于分类器，或者说分类算法，评价指标主要有Precision、Recall、F-score以及ROC和AUC。图8.21所示是一个ROC曲线的示例。

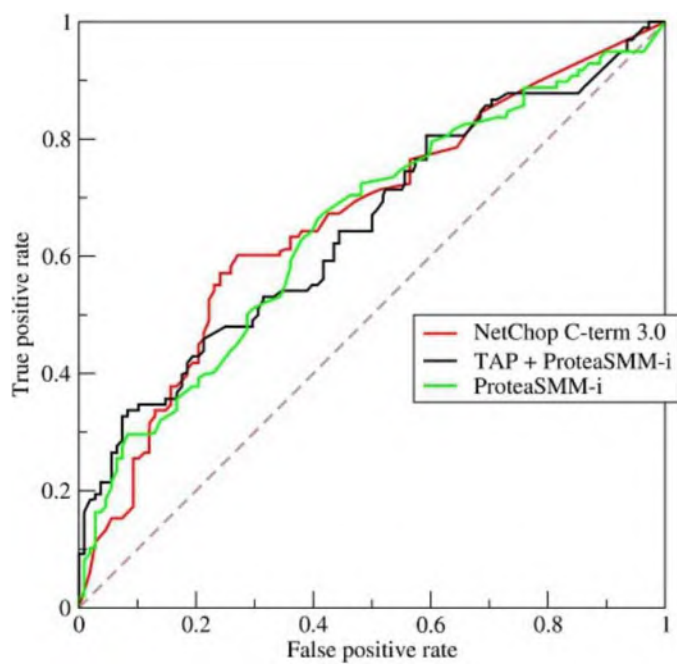


图8.21 ROC曲线示例

正如上面ROC曲线的示例图中看到的那样，ROC曲线的横坐标为FPR (False Positive Rate)，纵坐标为TPR (True Positive Rate)。以TPR为 y 轴，以FPR为 x 轴，就直接得到了ROC曲线。从FPR和TPR的定义可以理解，TPR越高，FPR越小，模型和算法就越高效。也就是画出来的ROC曲线越靠近左上越好。图8.22详细说明了FPR和TPR是如何定义的。

		True class			
		P	n		
Hypothesized class	Y	True Positives	False Positives	fp rate = $\frac{FP}{N}$	tp rate = $\frac{TP}{P}$
	N	False Negatives	True Negatives	precision = $\frac{TP}{TP+FP}$	recall = $\frac{TP}{P}$
Column totals:		P	N	accuracy = $\frac{TP+TN}{P+N}$	
				F-measure = $\frac{2}{1/precision+1/recall}$	

图8.22 混淆矩阵和通用性能指标

接下来考虑ROC曲线图中的4个点和一条线。第一个点：(0, 1)，即FPR=0，TPR=1，这意味着FN=0，并且FP=0。这是一个完美的分类器，它将所有的样本都正确分类。第二个点：(1, 0)，即FPR=1，TPR=0，分析发现这是一个最糟糕的分类器，因为它成功避开了所有的正确答案。第三个点：(0, 0)，即FPR=TPR=0，也就是FP=TP=0，可以发现该分类器预测所有的样本都为负样本（Negative）。类似的，第四个点：(1, 1)，分类器实际上预测所有的样本都为正样本。经过以上分析，我们可以断言，ROC曲线越接近左上角，该分类器的性能越好。考虑ROC曲线图中的虚线 $y = x$ 上的点，这条对角线上的点其实表示的是一个采用随机猜测策略的分类器的结果，例如(0.5, 0.5)，表示该分类器随机对于一半的样本猜测其为正样本，另一半的样本为负样本。

对于一个特定的分类器和测试数据集，显然只能得到一个分类结果，即一组FPR和TPR结果，而要得到一个曲线，实际上需要一系列FPR和TPR的值。

既然已经有这么多评价标准，为什么还要使用ROC和AUC呢？因为ROC曲线有一个很好的特性：当测试集中的正负样本的分布变化的时候，ROC曲线能够保持不变。在实际的数据集中经常会出现类不平衡（Class Imbalance）现象，即负样本比正样本多很多（或者相反），而且测试数据中的正负样本的分布也可能随着时间发生变化。

【例8.25】二分类问题中的ROC曲线。

```

#本实例中的数据来源于 sklearn 中的鸢尾花 (iris) 数据
# -*- coding: utf-8 -*-

import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.metrics import roc_curve, auc ###计算 roc 和 auc
from sklearn import cross_validation

# Import some data to play with
iris = datasets.load_iris()
X = iris.data
y = iris.target

##变为2分类
X, y = X[y != 2], y[y != 2]

# Add noisy features to make the problem harder
random_state = np.random.RandomState(0)
n_samples, n_features = X.shape
X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]

# shuffle and split training and test sets
X_train, X_test, y_train, y_test = cross_validation.train_test_split(X, y,
test_size=.3, random_state=0)

# Learn to predict each class against the other
svm = svm.SVC(kernel='linear', probability=True, random_state=random_state)

###通过 decision_function() 计算得到的 y_score 的值, 用在 roc_curve() 函数中
y_score = svm.fit(X_train, y_train).decision_function(X_test)

# Compute ROC curve and ROC area for each class
fpr, tpr, threshold = roc_curve(y_test, y_score) ###计算真正率和假正率
roc_auc = auc(fpr, tpr) ###计算 auc 的值

plt.figure()
lw = 2
plt.figure(figsize=(10,10))
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc) ###以假正率为横坐标,
真正率为纵坐标绘制曲线
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()

```

输出结果如图8.23所示。

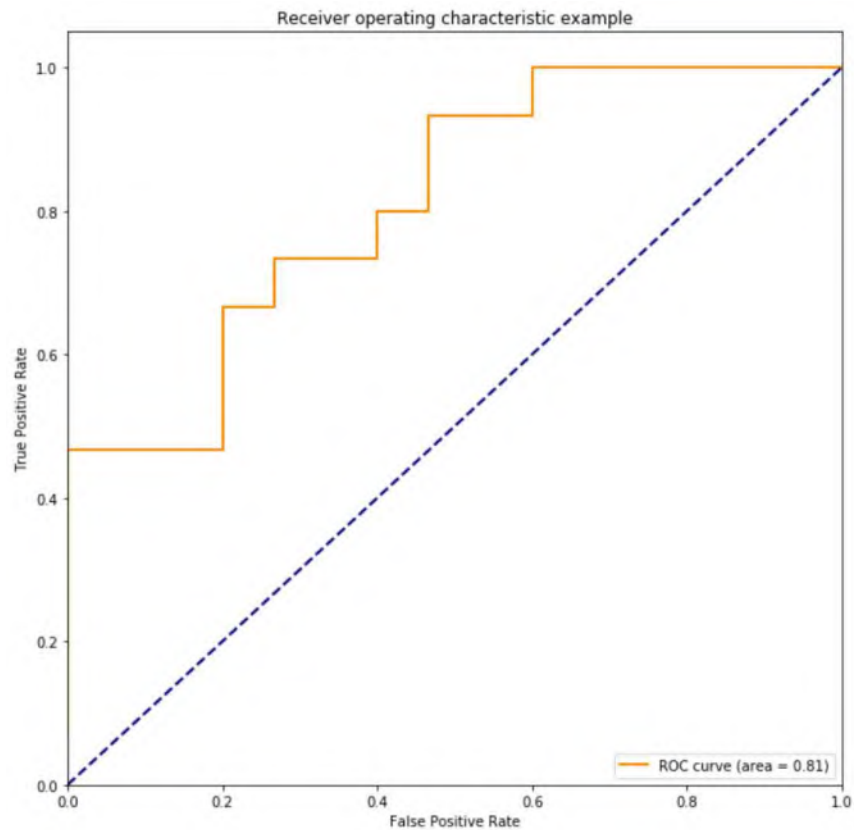


图8.23 鸢尾花 (iris) 数据二分类问题中的ROC曲线

实验环境是Anaconda 3 Spyder (Python 3.6)。

8.16 距离分析

在进行分类时常常需要估算不同样本之间的相似性度量 (Similarity Measurement)，这时通常采用的方法就是计算样本间的“距离” (Distance)。采用什么样的方法计算距离是很讲究的，甚至关系到分类正确与否。

距离是对观察变量之间的相似或者不相似程度的一种测度，它计算的是一对变量之间或一对观测变量之间的广义距离。这些相似性或

距离测度可以应用于其他分析过程，例如因子分析、聚类分析或多维尺度分析等，这样做有助于对复杂数据集的深入分析。

1. 欧氏距离

欧氏距离（Euclidean Distance）是最易于理解的一种距离计算方法，源自欧氏空间中两点间的距离公式。

（1）二维平面上的两点 $a(x_1, y_1)$ 与 $b(x_2, y_2)$ 间的欧氏距离：

$$d_{12} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (8.51)$$

（2）三维空间上的两点 $a(x_1, y_1, z_1)$ 与 $b(x_2, y_2, z_2)$ 间的欧氏距离：

$$d_{12} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \quad (8.52)$$

（3）两个 n 维向量 $a(x_{11}, x_{12}, \dots, x_{1n})$ 与 $b(x_{21}, x_{22}, \dots, x_{2n})$ 间的欧氏距离：

$$d_{12} = \sqrt{\sum_{k=1}^n (x_{1k} - x_{2k})^2} \quad (8.53)$$

（4）表示成向量运算的形式：

$$d_{12} = \sqrt{(a-b)(a-b)^T} \quad (8.54)$$

【例8.26】

```
import numpy as np
x=np.random.random(10)
y=np.random.random(10)
```

```
#方法一：根据公式求解
d1=np.sqrt(np.sum(np.square(x-y)))

#方法二：根据scipy库求解
from scipy.spatial.distance import pdist
X=np.vstack([x,y])
d2=pdist(X)
```

2. 曼哈顿距离

从名字就可以猜出曼哈顿距离（Manhattan Distance）的计算方法了。想象你在曼哈顿要从一个十字路口开车到另一个十字路口，驾驶距离是两点间的直线距离吗？显然不是，除非你能穿越大楼。实际驾驶距离就是这个“曼哈顿距离”。而这也是曼哈顿距离名称的来源，曼哈顿距离也称为城市街区距离（City Block Distance）。

（1）二维平面上的两点 $a(x_1, y_1)$ 与 $b(x_2, y_2)$ 间的曼哈顿距离：

$$d_{12} = |x_1 - x_2| + |y_1 - y_2| \quad (8.55)$$

（2）两个 n 维向量 $a(x_{11}, x_{12}, \dots, x_{1n})$ 与 $b(x_{21}, x_{22}, \dots, x_{2n})$ 间的曼哈顿距离：

$$d_{12} = \sum_{k=1}^n |x_{1k} - x_{2k}| \quad (8.56)$$

【例8.27】

```
import numpy as np
x=np.random.random(10)
y=np.random.random(10)

#方法一：根据公式求解
d1=np.sum(np.abs(x-y))

#方法二：根据scipy库求解
from scipy.spatial.distance import pdist
```

```
X=np.vstack([x,y])
d2=pdist(X,'cityblock')
```

3. 切比雪夫距离

玩过国际象棋吗？国王走一步能够移动到相邻的8个方格中的任意一个。那么国王从格子 (x_1, y_1) 走到格子 (x_2, y_2) 最少需要多少步？你会发现最少需要的步数总是 $\max(|x_2 - x_1|, |y_2 - y_1|)$ 步。类似这样的距离度量方法叫切比雪夫距离（Chebyshev Distance）。

(1) 二维平面上的两点 $a(x_1, y_1)$ 与 $b(x_2, y_2)$ 间的切比雪夫距离：

$$d_{12} = \max(|x_1 - x_2|, |y_1 - y_2|) \quad (8.57)$$

(2) 两个 n 维向量 $a(x_{11}, x_{12}, \dots, x_{1n})$ 与 $b(x_{21}, x_{22}, \dots, x_{2n})$ 间的切比雪夫距离：

$$d_{12} = \max_i |x_{1i} - x_{2i}| \quad (8.58)$$

这个公式的另一种等价形式是：

$$d_{12} = \lim_{k \rightarrow \infty} \left(\sum_{i=1}^n |x_{1i} - x_{2i}|^k \right)^{1/k} \quad (8.59)$$

可以用放缩法和夹逼法则来证明式（8.59）和式（8.58）等价。

【例8.28】

```
import numpy as np
x=np.random.random(10)
y=np.random.random(10)
```

#方法一：根据公式求解

```
d1=np.max(np.abs(x-y))

#方法二：根据scipy库求解
from scipy.spatial.distance import pdist
X=np.vstack([x,y])
d2=pdist(X,'chebyshev')
```

4. 闵可夫斯基距离

闵可夫斯基距离（Minkowski Distance）又叫闵氏距离，闵氏距离不是一种距离，而是一组距离的定义。

两个 n 维变量 $a(x_{11}, x_{12}, \dots, x_{1n})$ 与 $b(x_{21}, x_{22}, \dots, x_{2n})$ 间的闵可夫斯基距离定义为：

$$d_{12} = \sqrt[p]{\sum_{k=1}^n |x_{1k} - x_{2k}|^p} \quad (8.60)$$

其中， p 是一个变参数。

- 当 $p=1$ 时，就是曼哈顿距离。
- 当 $p=2$ 时，就是欧氏距离。
- 当 $p \rightarrow \infty$ 时，就是切比雪夫距离。

根据变参数的不同，闵氏距离可以表示一类距离。

【例8.29】

```
import numpy as np
x=np.random.random(10)
y=np.random.random(10)

#方法一：根据公式求解，p=2
d1=np.sqrt(np.sum(np.square(x-y)))

#方法二：根据scipy库求解
from scipy.spatial.distance import pdist
X=np.vstack([x,y])
d2=pdist(X,'minkowski',p=2)
```

5. 标准化欧氏距离

标准化欧氏距离 (Standardized Euclidean Distance) 是针对简单欧氏距离的缺点而做的一种改进方案。标准欧氏距离的思路：既然数据各维分量的分布不一样，就先将各个分量都“标准化”到均值、方差相等。均值和方差标准化到多少呢？假设样本集 X 的均值为 m ，标准差为 s ，那么 X 的“标准化变量”表示为：

$$X^* = \frac{X - m}{s} \quad (8.61)$$

标准化后的值 = (标准化前的值 - 分量的均值) / 分量的标准差

经过简单的推导就可以得到两个 n 维向量 $a(x_{11}, x_{12}, \dots, x_{1n})$ 与 $b(x_{21}, x_{22}, \dots, x_{2n})$ 间的标准化欧氏距离的公式：

$$d_{12} = \sqrt{\sum_{k=1}^n \left(\frac{x_{1k} - x_{2k}}{s_k} \right)^2} \quad (8.62)$$

如果将方差的倒数看成是一个权重，这个公式可以看成是一种加权欧氏距离 (Weighted Euclidean Distance)。

【例8.30】

```
import numpy as np
x=np.random.random(10)
y=np.random.random(10)

X=np.vstack([x,y])

#方法一：根据公式求解
sk=np.var(X,axis=0,ddof=1)
d1=np.sqrt(((x - y) ** 2 /sk).sum())

#方法二：根据scipy库求解
from scipy.spatial.distance import pdist
d2=pdist(X,'seuclidean')
```

6. 马氏距离

有M个样本向量 $X_1 \sim X_m$ ，协方差矩阵记为 S ，均值记为向量 μ ，则其中样本向量 X 到 μ 的马氏距离（Mahalanobis Distance）表示为：

$$D(X) = \sqrt{(X - \mu)^T S^{-1} (X - \mu)} \quad (8.63)$$

而其中向量 X_i 与 X_j 之间的马氏距离定义为：

$$D(X_i, X_j) = \sqrt{(X_i - X_j)^T S^{-1} (X_i - X_j)} \quad (8.64)$$

若协方差矩阵是单位矩阵（各个样本向量之间独立同分布），则公式就成了：

$$D(X_i, X_j) = \sqrt{(X_i - X_j)^T (X_i - X_j)} \quad (8.65)$$

也就是欧氏距离了。若协方差矩阵是对角矩阵，公式变成标准化欧氏距离。

【例8.31】

```

import numpy as np
x=np.random.random(10)
y=np.random.random(10)

#马氏距离要求样本数大于维数，否则无法求协方差矩阵
#此处进行转置，表示10个样本，每个样本2维
X=np.vstack([x,y])
XT=X.T

#方法一：根据公式求解
S=np.cov(X) #两个维度之间的协方差矩阵
SI = np.linalg.inv(S) #协方差矩阵的逆矩阵
#马氏距离计算两个样本之间的距离，此处共有10个样本，两两组合，共有45个距离
n=XT.shape[0]
d1=[]
for i in range(0,n):
    for j in range(i+1,n):
        delta=XT[i]-XT[j]
        d=np.sqrt(np.dot(np.dot(delta,SI),delta.T))
        d1.append(d)

#方法二：根据 scipy 库求解
from scipy.spatial.distance import pdist
d2=pdist(XT,'mahalanobis')

```

优点：它不受量纲的影响，两点之间的马氏距离与原始数据的测量单位无关。由标准化数据和中心化数据（原始数据与均值之差）计算出的两点之间的马氏距离相同。马氏距离还可以排除变量之间的相关性的干扰。

缺点：它夸大了变化微小的变量的作用。

7. 夹角余弦

夹角余弦（Cosine）也可以叫余弦相似度。在几何中，夹角余弦可用来衡量两个向量方向的差异，机器学习中借用这一概念来衡量样本向量之间的差异。

(1) 在二维空间中, 向量 $A(x_1, y_1)$ 与向量 $B(x_2, y_2)$ 的夹角余弦公式:

$$\cos \theta = \frac{x_1 x_2 + y_1 y_2}{\sqrt{x_1^2 + y_1^2} \sqrt{x_2^2 + y_2^2}} \quad (8.66)$$

(2) 两个 n 维样本点 $a(x_{11}, x_{12}, \dots, x_{1n})$ 和 $b(x_{21}, x_{22}, \dots, x_{2n})$ 的夹角余弦类似, 对于两个 n 维样本点 $a(x_{11}, x_{12}, \dots, x_{1n})$ 和 $b(x_{21}, x_{22}, \dots, x_{2n})$, 可以使用类似于夹角余弦的概念来衡量它们间的相似程度。

$$\cos \theta = \frac{a \cdot b}{|a||b|}$$

即:

$$\cos(\theta) = \frac{\sum_{k=1}^n x_{1k} x_{2k}}{\sqrt{\sum_{k=1}^n x_{1k}^2} \sqrt{\sum_{k=1}^n x_{2k}^2}} \quad (8.67)$$

余弦取值范围为 $[-1, 1]$ 。求得两个向量的夹角, 并得出夹角对应的余弦值, 该余弦值就可以用来表征这两个向量的相似性。夹角越小, 越趋近于0度, 余弦值越接近于1, 它们的方向更加吻合, 就越相似。当两个向量的方向完全相反时, 夹角余弦取最小值-1。当余弦值为0时, 两个向量正交, 夹角为90度。因此可以看出, 余弦相似度与向量的幅值无关, 只与向量的方向相关。

【例8.32】

```
import numpy as np
x=np.random.random(10)
y=np.random.random(10)

#方法一: 根据公式求解
dl=np.dot(x,y)/(np.linalg.norm(x)*np.linalg.norm(y))
```



```
#方法二：根据scipy库求解
from scipy.spatial.distance import pdist
X=np.vstack([x,y])
d2=1-pdist(X,'cosine')
#两个向量完全相等时，余弦值为1，以下代码计算出来的d=1。
d=1-pdist([x,x],'cosine')
```

8. 皮尔逊相关系数

皮尔逊相关系数（Pearson Correlation）的定义：

$$\rho_{XY} = \frac{Cov(X,Y)}{\sqrt{D(X)}\sqrt{D(Y)}} = \frac{E((X-EX)(Y-EY))}{\sqrt{D(X)}\sqrt{D(Y)}} \quad (8.68)$$

前面提到的余弦相似度只与向量方向有关，但它会受向量的平移影响，在夹角余弦公式中，如果将 x 平移到 $x+1$ ，余弦值就会改变。怎样才能实现平移不变性？这就要用到皮尔逊相关系数，有时也直接叫相关系数。

若将夹角余弦公式写成：

$$\text{Cos Sim}(x,y) = \frac{\sum_i x_i y_i}{\sum_i x_i^2 \sum_i y_i^2} = \frac{\langle x, y \rangle}{\|x\| \|y\|} \quad (8.69)$$

表示向量 x 和向量 y 之间的夹角余弦，则皮尔逊相关系数可表示为：

$$\text{Corr}(x,y) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2} \sqrt{\sum_i (y_i - \bar{y})^2}} = \frac{\langle x - \bar{x}, y - \bar{y} \rangle}{\|x - \bar{x}\| \|y - \bar{y}\|} = \text{Cos Sim}(x - \bar{x}, y - \bar{y}) \quad (8.70)$$

皮尔逊相关系数具有平移不变性和尺度不变性，计算出了两个向量（维度）的相关性。

【例8.33】

```

import numpy as np
x=np.random.random(10)
y=np.random.random(10)

#方法一：根据公式求解
x_=x-np.mean(x)
y_=y-np.mean(y)
d1=np.dot(x_,y_)/(np.linalg.norm(x_)*np.linalg.norm(y_))

#方法二：根据Numpy库求解
X=np.vstack([x,y])
d2=np.corrcoef(X)[0][1]

```

相关系数是衡量随机变量 X 与 Y 相关程度的一种方法，相关系数的取值范围是 $[-1, 1]$ 。相关系数的绝对值越大，就表明 X 与 Y 的相关度越高。当 X 与 Y 线性相关时，相关系数取值为1（正线性相关）或-1（负线性相关）。

9. 汉明距离

汉明距离（Hamming Distance）的定义：两个等长字符串 s_1 与 s_2 之间的汉明距离定义为将其中一个变为另一个所需要做的最小替换次数。例如字符串1111与1001之间的汉明距离为2。

应用：信息编码（为了增强容错性，应使得编码间的最小汉明距离尽可能大）。

【例8.34】

```

import numpy as np
from scipy.spatial.distance import pdist
x=np.random.random(10)>0.5
y=np.random.random(10)>0.5

x=np.asarray(x,np.int32)
y=np.asarray(y,np.int32)

#方法一：根据公式求解
d1=np.mean(x!=y)

```

```
#方法二：根据scipy库求解
X=np.vstack([x,y])
d2=pdist(X,'hamming')
```

10. 杰卡德相似系数

(1) 杰卡德相似系数的定义

两个集合A和B的交集元素在A、B的并集中所占的比例称为两个集合的杰卡德相似系数（Jaccard Similarity Coefficient），用符号 $J(A, B)$ 表示。

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (8.71)$$

杰卡德相似系数是衡量两个集合的相似度的一种指标。

(2) 杰卡德距离

与杰卡德相似系数相反的概念是杰卡德距离（Jaccard Distance）。杰卡德距离可用如下公式表示：

$$J_{\delta}(A, B) = 1 - J(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \quad (8.72)$$

杰卡德距离用两个集合中不同元素占所有元素的比例来衡量两个集合的区分度。

(3) 杰卡德相似系数与杰卡德距离的应用

可将杰卡德相似系数用在衡量样本的相似度上。

样本A与样本B是两个 n 维向量，而且所有维度的取值都是0或1，例如A(0111)和B(1011)。我们将样本看成是一个集合，1表示集合包含该元素，0表示集合不包含该元素。

【例8.35】

```
import numpy as np
from scipy.spatial.distance import pdist
x=np.random.random(10)>0.5
y=np.random.random(10)>0.5

x=np.asarray(x,np.int32)
y=np.asarray(y,np.int32)

#方法一：根据公式求解
up=np.double(np.bitwise_and((x != y),np.bitwise_or(x != 0, y
!= 0)).sum())
down=np.double(np.bitwise_or(x != 0, y != 0).sum())
d1=(up/down)

#方法二：根据scipy库求解
X=np.vstack([x,y])
d2=pdist(X,'jaccard')
```

11. 布雷柯蒂斯距离

布雷柯蒂斯距离（Bray Curtis Distance）主要用于生态学和环
境科学，计算坐标之间的距离。该距离取值在[0,1]之间。它也可以用来
计算样本之间的差异。

$$d_{ij} = \frac{\sum_{k=1}^n |x_{ik} - y_{jk}|}{\sum_{k=1}^n x_{jk} + \sum_{k=1}^n y_{jk}} \quad (8.73)$$

样本数据：

	a	b	c	d	e	sum
s29	11	0	7	8	0	26
s30	24	37	5	18	1	85

计算：

$$b_{s_{29},s_{30}} = \frac{|11-24|+|0-37|+|7-5|+|8-18|+|0-1|}{26+85} = \frac{63}{111} = 0.568$$

【例8.36】

```
import numpy as np
from scipy.spatial.distance import pdist
x=np.array([11,0,7,8,0])
y=np.array([24,37,5,18,1])

#方法一：根据公式求解
up=np.sum(np.abs(y-x))
down=np.sum(x)+np.sum(y)
d1=(up/down)

#方法二：根据scipy库求解
X=np.vstack([x,y])
d2=pdist(X,'braycurtis')
```

现在，我们可以为矩阵中对象间的相似程度（接近与远离）提供各种度量方法以及编码实现了。

8.17 对应分析

对应分析也称关联分析、R-Q型因子分析，是近年来新发展起来的一种多元相依变量统计分析技术，通过分析由定性变量构成的交互汇总表来揭示变量间的联系，可以揭示同一变量的各个类别之间的差异，以及不同变量各个类别之间的对应关系。

对应分析主要应用在市场细分、产品定位、地质研究以及计算机工程等领域中。原因在于，它是一种视觉化的数据分析方法，能够将几组看不出任何联系的数据通过视觉上可以接受的定位图展现出来。

对应分析的基本思想是将一个列联表的行和列中各元素的比例结构以点的形式在较低维的空间中表示出来。

对应分析的最大特点是能把众多的样品和众多的变量同时展示在同一张图解上，将样品的大类及其属性在图上直观而又明了地表示出来，具有直观性。另外，它还省去了因子选择和因子轴旋转等复杂的数学运算及中间过程，可以从因子载荷图上对样品进行直观的分类，而且能够指示分类的主要参数（主因子）和分类的依据，是一种直观、简单、方便的多元统计方法。

对应分析法整个处理过程由两部分组成：表格和关联图。对应分析法中的表格是一个二维的表格，由行和列组成。每一行代表事物的一个属性，依次排开。列则代表不同的事物本身，由样本集合构成，排列顺序并没有特别的要求。在关联图上，各个样本都浓缩为一个点集合，而样本的属性变量在图上同样是以点集合的形式显示出来的。

【例8.37】

```

import ca
import pandas as pd
import seaborn as sns

transfrmr = ca.CA()
children = pd.read_csv('./datasets/children.csv', index_col=0)
X = children.loc[:, 'work', : 'university']

transfrmr.fit(X)

pcs_row, pcs_col = \
    transfrmr.get_prncpl_coords_df(row_categories=X.index,
                                   col_categories=X.columns)
pcs_row['Dim 1'] = -pcs_row['Dim 1']
pcs_col['Dim 1'] = -pcs_col['Dim 1']
print('Principal coordinates of row variables in DataFrame:')
print(pcs_row)
print(pcs_col)

supp_rows = children.loc['comfort':, : 'university']
supp_cols = children.loc[:, 'work', 'thirty':]
new_supp_rows = transfrmr.transform(supp_rows)
new_supp_cols = transfrmr.transform(supp_cols, row=False)
new_supp_rows[:, 1] = -new_supp_rows[:, 1]
new_supp_cols[:, 1] = -new_supp_cols[:, 1]
print(new_supp_rows)
print(new_supp_cols)

fig, ax = sns.plt.subplots()
sns.regplot('Dim 0', 'Dim 1', data=pcs_row, ax=ax, fit_reg=False, label='rows')
sns.regplot('Dim 0', 'Dim 1', data=pcs_col, ax=ax, fit_reg=False, label='cols')
sns.regplot(new_supp_rows[:, 0], new_supp_rows[:, 1],
            ax=ax, fit_reg=False, label='supp rows')
sns.regplot(new_supp_cols[:, 0], new_supp_cols[:, 1],
            ax=ax, fit_reg=False, label='supp cols')
for i, txt in enumerate(list(X.index)):
    ax.annotate(txt, (pcs_row.iloc[i]['Dim 0'], pcs_row.iloc[i]['Dim 1']))
for i, txt in enumerate(list(X.columns)):
    ax.annotate(txt, (pcs_col.iloc[i]['Dim 0'], pcs_col.iloc[i]['Dim 1']))
for i, txt in enumerate(list(supp_rows.index)):
    ax.annotate(txt, (new_supp_rows[i, 0], new_supp_rows[i, 1]))
for i, txt in enumerate(list(supp_cols.columns)):
    ax.annotate(txt, (new_supp_cols[i, 0], new_supp_cols[i, 1]))
ax.legend()
sns.plt.show()

```

这里使用的数据是一个列联表，描述了不同类别的人有以下问题：什么原因会让一个女人或一对夫妇犹豫要不要孩子？

children.csv 数据 可以 从 <https://github.com/YiChanLee/correspondence-analysis> 下载。

8.18 决策树分析

本节只讲述在Sklearn中如何使用决策树。开发环境需要安装NumPy、Pandas、Matplotlib、sklearn、pydotplus、Graphviz库。

安装Graphviz软件后，需要进入系统环境变量设置。在系统变量的Path变量中，添加Graphviz的环境变量，比如Graphviz安装在了D盘的根目录，则添加D:\Graphviz\bin。

设置完环境变量，记得要重启IDE。

sklearn使用DecisionTreeClassifier构建决策树，这个函数一共有12个参数：

参数说明如下：

- **criterion** : 特征选择标准，可选参数，默认是gini，可以设置为entropy。gini是基尼不纯度，它将来自集合的某种结果随机应用于某一数据项的预期误差率，是一种基于统计的思想。entropy是香农熵，是一种基于信息论的思想。sklearn把gini设为默认参数，应该是做了相应斟酌的，精度也许更高些，ID3算法使用的是entropy，CART算法使用的则是gini。
- **splitter** : 特征划分点选择标准，可选参数，默认是best，可以设置为random。每个节点的选择策略。best参数是根据算法选择最佳的切分特征，例如gini、entropy。random随机地在部分划分点中找局部最优的划分点。默认的best适合样

本量不大的时候，而如果样本数据量非常大，此时决策树构建推荐random。

- **max_features** : 划分时考虑的最大特征数，可选参数，默认是None。寻找最佳切分时考虑的最大特征数（n_features为总共的特征数）有如下6种情况：

- 若max_features是整型的数，则考虑max_features个特征。
- 若 max_features 是浮点型的数，则考虑 $\text{int}(\text{max_features} * \text{n_features})$ 个特征。
- 若 max_features 设为 auto，则 $\text{max_features} = \text{sqrt}(\text{n_features})$ 。
- 若 max_features 设为 sqrt，则 $\text{max_features} = \text{sqrt}(\text{n_features})$ ，跟auto一样。
- 若 max_features 设为 log2，则 $\text{max_features} = \text{log2}(\text{n_features})$ 。
- 若 max_features 设为 None，则 $\text{max_features} = \text{n_features}$ ，也就是所有特征都用。

一般来说，如果样本特征数不多，比如小于50，用默认的None就可以了，如果特征数非常多，可以灵活使用刚才描述的其他取值来控制划分时考虑的最大特征数，以控制决策树的生成时间。

- **max_depth** : 决策树最大深度，可选参数，默认是None。该参数是树的层数。层数就是决策树的层数。如果这个参数设置为None，那么决策树在建立子树的时候不会限制子树的深度。一般来说，数据少或者特征少的时候可以不管这个值。或者如果设置了 min_samples_split 参数，那么直到少于 min_samples_split 个样本为止。在模型样本量多，特征也多的

情况下，推荐限制这个最大深度，具体的取值取决于数据的分布，常用的可以取值10~100。

- **min_samples_split** : 内部节点再划分所需的最小样本数，可选参数，默认是2。这个值限制了子树继续划分的条件。如果min_samples_split为整数，那么在切分内部节点的时候，min_samples_split作为最小的样本数，也就是说，如果已经少于min_samples_split个样本，就停止继续切分。如果min_samples_split为浮点数，min_samples_split就是一个百分比， $\text{ceil}(\text{min_samples_split} * \text{n_samples})$ ，数是向上取整的。如果样本量不大，就不需要管这个值。如果样本量的数量级非常大，就推荐增大这个值。
- **min_weight_fraction_leaf** : 叶子节点最小的样本权重和，可选参数，默认是0。这个值限制了叶子节点所有样本权重和的最小值，如果小于这个值，就会和兄弟节点一起被剪枝。一般来说，如果有较多样本有缺失值，或者分类树样本的分布类别偏差很大，就会引入样本权重，这时我们就要注意这个值了。
- **max_leaf_nodes** : 最大叶子节点数，可选参数，默认是None。通过限制最大叶子节点数可以防止过拟合。如果加了限制，算法就会建立在最大叶子节点数内最优的决策树。如果特征不多，就可以不考虑这个值，但是如果特征很多，就需要加以限制，具体的值可以通过交叉验证得到。
- **class_weight** : 类别权重，可选参数，默认是None，也可以是字典、字典列表、balanced（保持平衡）。指定样本各类别的权重，主要是为了防止训练集某些类别的样本过多，导致训练的决策树过于偏向这些类别。类别的权重可以通过{class_label: weight}这样的格式给出，这里可以自己指定各个样本的权重，或者用“balanced”。如果使用balanced，算

法就会自己计算权重，样本量少的类别所对应的样本权重会高。当然，如果样本类别分布没有明显的偏倚，就可以不管这个参数，选择默认的None。

- **random_state** : 可选参数，默认是None。如果是证书，random_state就会作为随机数生成器的随机数种子。如果没有设置随机数，随机出来的数就与当前系统时间有关，每个时刻都是不同的。如果设置了随机数种子，那么相同随机数种子不同时刻产生的随机数是相同的。如果是RandomState instance（随机数实例），那么random_state是随机数生成器。如果为None，那么随机数生成器使用np.random。
- **min_impurity_split** : 节点划分最小不纯度，可选参数，默认是1e-7。这是个阈值，该值限制了决策树的生长。如果某节点的不纯度（基尼系数、信息增益、均方差、绝对差）小于这个阈值，那么该节点不再生成子节点，即为叶子节点。
- **presort** : 数据是否预排序，可选参数，这个值是布尔值，默认是False（不排序）。一般来说，如果样本量少或者限制了一个深度很小的决策树，设置为True可以让划分点选择得更加快，决策树建立得更加快。如果样本量太大，反而没有什么好处。问题是样本量少的时候，速度本来就不慢。所以该值一般不考虑。
- **max_features**: 划分考虑最大特征数，默认为None。不输入则默认全部特征，可以选log2N、sqrt(N)、auto或者是小于1的浮点数（百分比）或整数（具体数量的特征）。如果特征特别多时，比如大于50，可以考虑选择auto来控制决策树的生成时间。

当样本数量少但是样本特征非常多的时候，决策树很容易过拟合。一般来说，样本数比特征数多一些会比较容易建立健壮模型。如果样本数量少但是样本特征非常多，在拟合决策树模型前，推荐先

做维度规约，比如主成分分析（PCA）、特征选择（LOSS0）或者独立成分分析（ICA）。这样特征的维度会大大减小，再来拟合决策树模型效果会更好。多用决策树的可视化，同时先限制决策树的深度，这样可以先观察生成的决策树里数据的初步拟合情况，再决定是否要增加深度。在训练模型时，注意观察样本的类别情况（主要指分类树），如果类别分布非常不均匀，就要考虑用`class_weight`来限制模型过于偏向样本多的类别。决策树的数组使用的是NumPy的float32类型，如果训练数据不是这样的格式，算法就会先做copy（复制）再运行。如果输入的样本矩阵是稀疏的，那么推荐在拟合前调用`csc_matrix`稀疏化，在预测前调用`csr_matrix`稀疏化。

【例8.38】

```

# -*- coding: UTF-8 -*-

import numpy as np          # 快速操作结构数组的工具
import pandas as pd         # 数据分析处理工具
import matplotlib.pyplot as plt  # 画图工具
from sklearn import datasets # 机器学习库
from sklearn.preprocessing import LabelEncoder
from sklearn import tree

# 下面的数据分别为每个用户的来源网站、位置、是否阅读 FAQ、浏览网页数目、选择的服务类型（目标结果）
attr_arr=[['slashdot','USA','yes',18,'None'],
          ['google','France','yes',23,'Premium'],
          ['digg','USA','yes',24,'Basic'],
          ['kiwitobes','France','yes',23,'Basic'],
          ['google','UK','no',21,'Premium'],
          ['(direct)','New Zealand','no',12,'None'],
          ['(direct)','UK','no',21,'Basic'],
          ['google','USA','no',24,'Premium'],
          ['slashdot','France','yes',19,'None'],
          ['digg','USA','no',18,'None'],
          ['google','UK','no',18,'None'],
          ['kiwitobes','UK','no',19,'None'],
          ['digg','New Zealand','yes',12,'Basic'],
          ['slashdot','UK','no',21,'None'],
          ['google','UK','yes',18,'Basic'],
          ['kiwitobes','France','yes',19,'Basic']]

#生成属性数据集和结果数据集
dataMat = np.mat(attr_arr)
arrMat = dataMat[:,0:4]
resultMat = dataMat[:,4]

# 构造数据集成 Pandas 结构
attr_names = ['src', 'address', 'FAQ', 'num'] #特征属性的名称
attr_pd = pd.DataFrame(data=arrMat,columns=attr_names) #每行为一个对象，每列为一种属性，最后一个为结果值
print(attr_pd)

#将数据集中的字符串转化为代表类别的数字。因为 Sklearn 的决策树只识别数字
le = LabelEncoder()
for col in attr_pd.columns: #为每一列序列化，就是将每种字符串转化为对应的数字。
    用数字代表类别

```

```
    attr_pd[col] = le.fit_transform(attr_pd[col])
print(attr_pd)

# 构建决策树
clf = tree.DecisionTreeClassifier()
clf.fit(attr_pd, resultMat)
print(clf)

# 使用决策树进行预测
result = clf.predict([[1,1,1,0]])    # 输入也必须是数字，分别代表每个数字所代表的属
性的字符串值
print(result)

# 将决策树保存成图片
from sklearn.externals.six import StringIO
import pydotplus

dot_data = StringIO()
target_name=['None','Basic','Premium']
tree.export_graphviz(clf, out_file=dot_data,feature_names=attr_names,
                     class_names=target_name, filled=True, rounded=True,
                     special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('tree.png')
```

输出：

	src	address	FAQ	num
0	lashdot	USA	yes	18
1	google	France	yes	23
2	digg	USA	yes	24
3	kiwitobes	France	yes	23
4	google	UK	no	21
5	(direct)	New Zealand	no	12
6	(direct)	UK	no	21
7	google	USA	no	24
8	slashdot	France	yes	19
9	digg	USA	no	18
10	google	UK	no	18
11	kiwitobes	UK	no	19
12	digg	New Zealand	yes	12
13	slashdot	UK	no	21
14	google	UK	yes	18
15	kiwitobes	France	yes	19

	src	address	FAQ	num
0	4	3	1	1
1	2	0	1	4
2	1	3	1	5
3	3	0	1	4
4	2	2	0	3
5	0	1	0	0
6	0	2	0	3
7	2	3	0	5
8	4	0	1	2
9	1	3	0	1
10	2	2	0	1
11	3	2	0	2
12	1	1	1	0
13	4	2	0	3
14	2	2	1	1
15	3	0	1	2

```

DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                        splitter='best')

['Basic']

```

自动生成的可视化决策树被保存在当前目录下的tree.png文件中，如图8.24所示。可以看到，我们使用gini指数来选择最优划分属

性，并且经剪枝后的决策树只用到了数据13个属性中的num、FAQ、src三个属性，已经非常简洁了，如图8.24所示。

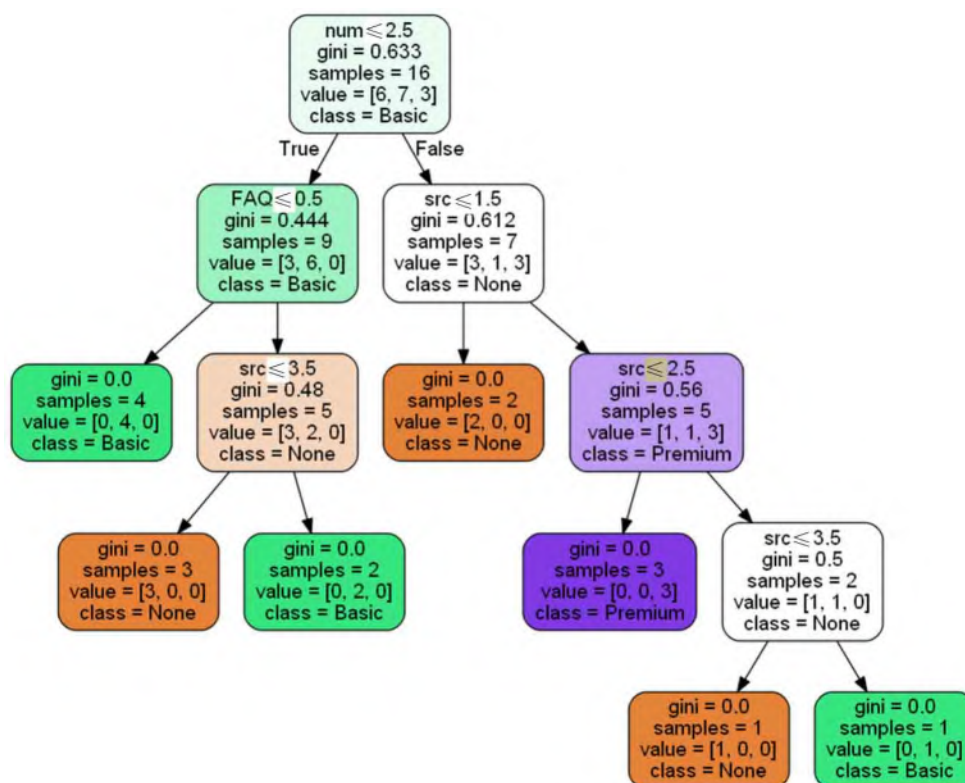


图8.24 sklearn一生成决策树分类模型

8.19 神经网络—深度学习

深度学习（Deep Learning）是机器学习的一个分支，可以理解为具有多层结构的模型。具体来说，深度学习是机器学习中的具有深层结构的神经网络算法，即机器学习>神经网络>深度神经网络（深度学习）。

神经网络技术起源于20世纪五六十年代，当时叫感知器（Perceptron），拥有输入层、输出层和一个隐含层。输入的特征向量通过隐含层变换达到输出层，在输出层得到分类结果。多层感知器可以摆脱早期离散传输函数的束缚，使用Sigmoid或tanh等连续函数模

拟神经元对激励的响应，在训练算法上则使用Werbos发明的反向传播BP算法。随着神经网络层数的加深，有两个重大问题：一是优化函数越来越容易陷入局部最优解，并且这个“陷阱”越来越偏离真正的全局最优，利用有限数据训练的深层网络，性能还不如较浅层网络；二是“梯度消失”现象更加严重。2006年，Hinton利用预训练方法缓解了局部最优解问题，将隐含层推动到了7层，神经网络真正意义上有了“深度”，由此揭开了深度学习的热潮，随后的DBN、CNN、RNN、LSTM等才逐渐出现。这里的“深度”并没有固定的定义——在语音识别中4层网络就能够被认为是“较深的”，而在图像识别中20层以上的网络屡见不鲜。

随着神经网络层数的加深，有三个重大问题：一是非凸优化问题，即优化函数越来越容易陷入局部最优解；二是梯度消失（Gradient Vanish）问题；三是过拟合问题。

8.19.1 深度学习的基本模型

深度学习里面的基本模型大致分为3类：多层感知器模型、深度神经网络模型和递归神经网络模型。其代表分别是深度信念网络（Deep Belief Network, DBN）、卷积神经网络、递归神经网络。

1. 深度信念网络

2006年，Geoffrey Hinton提出深度信念网络及其高效的学习算法，即Pre-Training+Fine Tuning，并发表于《Science》上，成为其后深度学习算法的主要框架。深度信念网络是一种生成模型，通过训练其神经元间的权重，我们可以让整个神经网络按照最大概率来生成训练数据。所以，我们不仅可以使深度信念网络识别特征、分类数据，还可以用它来生成数据。

（1）深度信念网络结构

深度信念网络由若干层受限玻尔兹曼机（Restricted Boltzmann Machine, RBM）堆叠而成，上一层受限玻尔兹曼机的隐层作为下一层受限玻尔兹曼机的可见层，如图8.25所示。

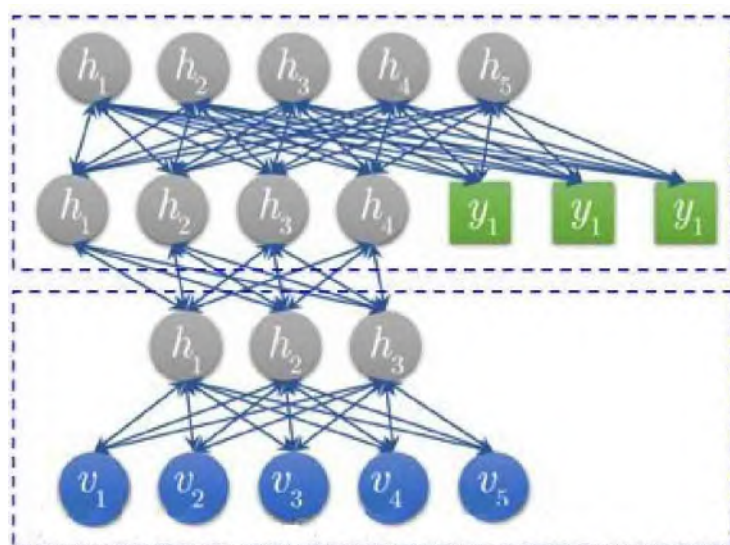


图8.25 深度信念网络结构

深度信念网络模型由若干层受限玻尔兹曼机堆叠而成，如果在训练集中有标签数据，那么最后一层受限玻尔兹曼机的可见层中既包含前一层受限玻尔兹曼机的隐层单元，又包含标签层单元。假设顶层受限玻尔兹曼机的可见层有500个神经元，训练数据的分类一共分成了10类，那么顶层受限玻尔兹曼机的可见层有510个显性神经元，对每一训练数据，相应的标签神经元被打开设为1，而其他的则被关闭设为0。

（2）训练过程

深度信念网络的训练包括Pre-Training和Fine Tuning两步，其中Pre-Training过程相当于逐层训练每一个受限玻尔兹曼机，经过Pre-Training的深度信念网络已经可用于模拟训练数据，而为了进一步提高网络的判别性能，Fine Tuning过程利用标签数据通过BP算法对网络参数进行微调。

（3）改进模型

深度信念网络的变体比较多，它的改进主要集中于其组成“零件”受限玻尔兹曼机的改进，有卷积深度信念网络（CDBN）和条件受限玻尔兹曼机（Conditional RBM）等。深度信念网络并没有考虑到图像的二维结构信息，因为输入只是简单地将一个图像矩阵转换为一维向量。而CDBN利用邻域像素的空域关系，通过一个称为卷积受限玻尔兹曼机（CRBM）的模型达到生成模型的变换不变性，而且容易变换到高维图像。深度信念网络并没有明确地处理对观察变量的时间联系的学习，条件受限玻尔兹曼机通过考虑前一时刻的可见层单元变量作为附加的条件输入，以模拟序列数据，这种变体在语音信号处理领域应用较多。

2. 卷积神经网络

卷积神经网络是人工神经网络的一种，已成为当前语音分析和图像识别领域的研究热点。它的权值共享网络结构使之更类似于生物神经网络，降低了网络模型的复杂度，减少了权值的数量。该优点在网络的输入是多维图像时表现得更为明显，使图像可以直接作为网络的输入，避免了传统识别算法中复杂的特征提取和数据重建过程。卷积网络是为识别二维形状而特殊设计的一个多层感知器，这种网络结构对平移、比例缩放、倾斜或者其他形式的变形具有高度不变性。

（1）网络结构

卷积神经网络是一个多层的神经网络，其基本运算单元包括：卷积运算、池化运算、全连接运算和识别运算，如图8.26所示。

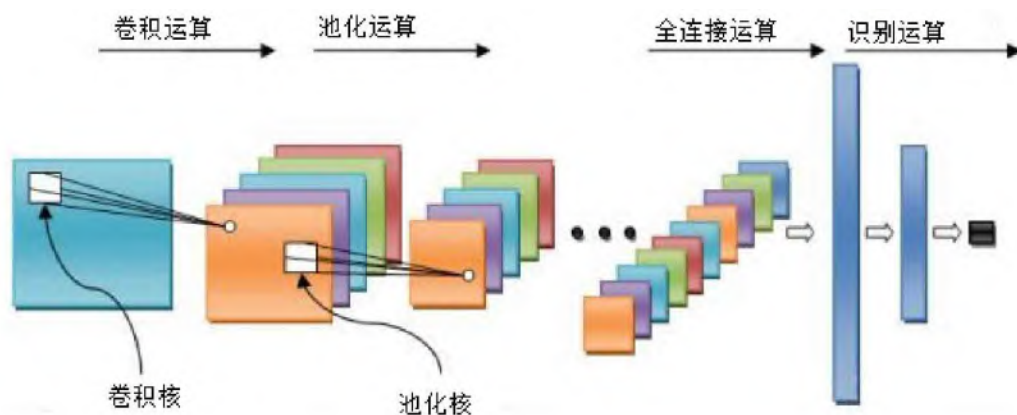


图8.26 卷积神经网络结构

卷积神经网络与常规的神经网络十分相似，它们都由可以对权重和偏置进行学习的神经元构成。每个神经元接收一些输入，然后执行点积操作，再紧接一个可选的非线性函数。整个网络仍然表示为单可微分的评估函数，整个网络从一端输入原始图像像素，另一端输出类别的概率。其最后一层（全连接层）同样有损失函数，并且学习常规神经网络的方法和技巧在这里仍然奏效。卷积网络很明确地假设所有输入都为图像，这就允许在结构中对明确的属性进行编码。这使得前向函数的实现更加高效，并且极大地减少了网络中参数的数量。

一个简单的卷积网络由一系列的层构成，并且卷积网络的每一层通过一个可微函数将一个激活量转换成另一个激活量。用3类主要的层来构造卷积网络：卷积层、池化层以及全连接层。我们将通过堆叠这些层来形成一个完整的卷积网络结构。

三维神经元：卷积神经网络充分利用了输入是由图像组成的这一事实，并以更加合理的方式对结构进行了约束。特别是，与常规网络不同，卷积网络各层的神经元具有3个维度，即宽度、高度、深度。每一层的神经元仅与前一层的局部相连，而不是采用全连接的方式。

(2) 卷积层

卷积层是一个卷积神经网络的核心组成部分，它负责完成大部分计算繁重的工作。首先，卷积层的参数由一组可以进行学习的滤波器组成。每一个滤波器都是一个小的二维面区域（沿宽度和高度），如图8.27所示，但却延伸到输入激活量的全部深度。在正向传播期间，沿着输入方柱体的宽度和高度方向滑动（更确切地说是卷积）每个滤波器，并计算滤波器的端口与滤波器处于每个位置上的输入的点积。当将滤波器沿着输入激活量的高度和宽度滑过之后，将会得到一个二维的激活图，该图给出了滤波器在滑动过的每个位置上得到的结果。直观地说，网络将对滤波器进行学习，当看到某种类型的视觉特征时激活，例如第一层上某种方向的条棱或某种颜色的斑点，或者最终在网络的更高层上形成的整个蜂窝或轮状图案。接下来，将在每个卷积层上使用一整组的滤波器，其中每一个滤波器都将产生一个二维激活图。将这些激活图沿着深度堆叠起来，得到一个输出激活量。

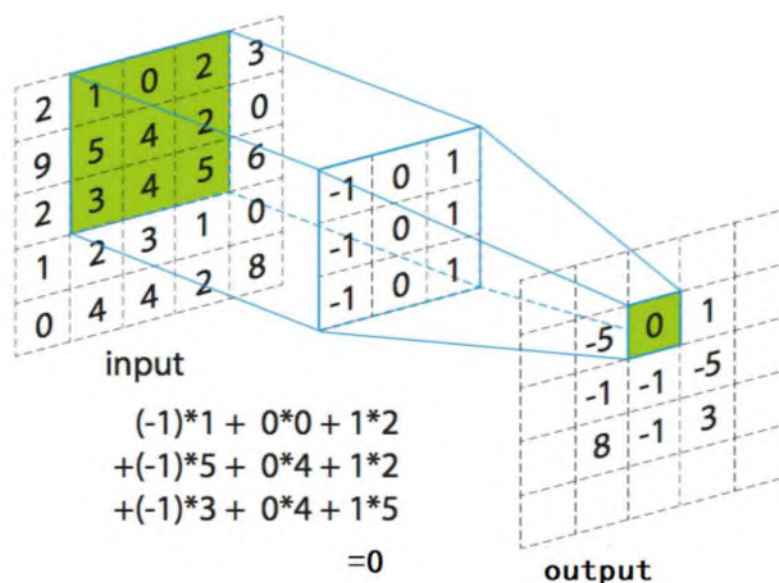


图8.27 典型滤波器（3）训练过程

卷积网络在本质上是一种输入到输出的映射，它能够学习大量的输入与输出之间的映射关系，而不需要任何输入和输出之间的精确的数学表达式，只要用已知的模式对卷积网络加以训练，网络就具有输

入输出对之间的映射能力。卷积网络执行的是有监督训练，所以其样本集是由形如（输入信号，标签值）的向量对构成的。

3. 递归神经网络

全连接的深度神经网络存在一个问题—无法对时间序列上的变化进行建模。然而，样本出现的时间顺序对于自然语言处理、语音识别、手写体识别等应用非常重要。为了适应这种需求，就出现了递归神经网络。在普通的全连接网络或卷积神经网络中，每层神经元的信号只能向上一层传播，样本的处理在各个时刻独立，因此又被称为前向神经网络（Feed-Forward Neural Networks）。而在递归神经网络中，神经元的输出可以在下一个时间戳直接作用到自身。递归神经网络可以看成是一个在时间上传递的神经网络，它的深度是时间的长度。正如前面介绍的，“梯度消失”现象又要出现了，只不过这次发生在时间轴上。为了解决时间上的梯度消失，机器学习领域发展出了长短时记忆单元（LSTM），通过门的开关实现时间上的记忆功能，并防止梯度消失。

（1）训练过程

递归神经网络中由于输入时叠加了之前的信号，因此反向传导时不同于传统的神经网络，因为对于时刻 t 的输入层，其残差不仅来自于输出，还来自于之后的隐层。通过反向传递算法，利用输出层的误差求解各个权重的梯度，然后利用梯度下降法更新各个权重。

（2）改进模型

递归神经网络模型可以用来处理序列数据，递归神经网络包含大量参数，且难于训练（时间维度的梯度消散或梯度爆炸），所以出现了一系列对递归神经网络的优化，比如网络结构、求解算法与并行化。近年来，双向循环神经网络（BRNN）与长短时记忆单元在image

captioning (图题)、language translation (翻译)、handwriting recognition (手写识别) 这几个方向上有了突破性进展。

8.19.2 新闻分类实例

【例8.39】新闻分类（多分类问题）。

在本例题中，代码示例使用Keras (<https://keras.io>)。Keras是Python的一个深度学习框架，它提供了一种方便的方式来定义和训练几乎任何类型的深度学习模型。Keras最初是为研究人员开发的，目的是实现快速实验。Keras是以MIT License发布的，这意味着它可以在商业项目中自由使用。它兼容任何版本的Python，从2.7到3.6（2017年中期）。

Keras是一个模型级库，提供了开发深度学习模型的高级构建模块。它不处理低级操作，如张量操作和微分。相反，它依赖于一个专门的、经过良好优化的张量库，充当Keras的后端引擎。它不是选择单个张量库并将Keras的实现与该库绑定在一起，而是以模块化的方式处理问题，因此可以将几个不同的后端引擎无缝地插入Keras中。目前，现有的3个后端实现是TensorFlow后端、Theano后端和Microsoft Cognitive工具包（CNTK）的后端。TensorFlow、CNTK和Theano是当今深度学习的主要平台。Theano (<http://deeplearning.net/software/theano>) 由蒙特利尔大学的MILA实验室开发，TensorFlow (www.tensorflow.org) 由谷歌开发，CNTK (<https://github.com/Microsoft/CNTK>) 由微软开发。用户使用Keras编写的任何代码片段都可以与这些后端一起运行，而无须更改代码中的任何内容。

本节示例构建一个网络，将路透社新闻专线分为46个相互排斥的主题。因为有很多类，所以这个问题是一个多分类的实例；而且，由

于每个数据点都应该被划分为一个类别，因此这个问题更具体地说就是一个单标签、多分类（Single-Label，Multiclass Classification）的实例。如果每个数据点可能属于多个类别（在本例中是主题），那么将面临一个多标签、多分类（Multi-Label，Multiclass Classification）的问题。

本例题实验环境为Anaconda 3 + Jupyter Notebook。

1. 加载数据集

```
In[1]: from keras.datasets import reuters
      (train_data, train_labels), (test_data, test_labels) =
reuters.load_data
(num_words=10000)
#将数据限定在10000个常见的单词、8982个训练样本和2264个测试样本
In[2]: len(train_data)
Out[2]: 8982
In[3]: len(test_data)
Out[3]: 2246
#每个示例都是一个整数列表(单词索引)
In[4]: train_data[10]
Out[4]: [1, 245, 273, 207, 156,
53, 74, 160, 26, 14, 46,
296, 26, 39, 74, 2979, 3554, 14, 46, 4689, 4329, 86, 61, 3499,
4795, 14, 61, 451, 4329, 17, 12]
```

2. 将索引解码为新闻文本

```
In[5]: word_index = reuters.get_word_index()
      reverse_word_index = dict([(value, key) for (key, value) in
word_index.items()])
# Note that our indices were offset by 3
# because 0, 1 and 2 are reserved indices for "padding",
"start of sequence",
and "unknown".
      decoded_newswire = ' '.join([reverse_word_index.get(i - 3,
'?') for i in
train_data[0]])
In[6]: train_labels[10]
Out[6]: 3
```


3. 编码数据

```
In[7]: import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1
    return results

# 将训练数据向量化
x_train = vectorize_sequences(train_data)
# 将测试数据向量化
x_test = vectorize_sequences(test_data)
In[8]: # 将标签向量化, 将标签转化为 one-hot
def to_one_hot(labels, dimension=46):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1
    return results

one_hot_train_labels = to_one_hot(train_labels)
one_hot_test_labels = to_one_hot(test_labels)
from keras.utils.np_utils import to_categorical
one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
```

4. 模型定义

```
In[9]: from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))
```

5. 编译模型

#对于这个例子, 最好的损失函数是 `categorical_crossentropy` (分类交叉熵), 用于衡量两个概率分布之间的距离

```
In[10]: model.compile(optimizer='rmsprop',
```

```
loss='categorical_crossentropy',  
metrics=['accuracy'])
```

6. 留出验证集

```
#留出1000个样本作为验证集  
In[11]:x_val = x_train[:1000]  
partial_x_train = x_train[1000:]  
  
y_val = one_hot_train_labels[:1000]  
partial_y_train = one_hot_train_labels[1000:]
```

7. 训练模型

```
In[12]: history = model.fit(partial_x_train,  
partial_y_train, epochs=20,  
batch_size = 512, validation_data = (x_val, y_val))
```

8. 绘制训练损失和验证损失

```
In[13]: import matplotlib.pyplot as plt  
loss = history.history['loss']  
val_loss = history.history['val_loss']  
  
epochs = range(1, len(loss) + 1)  
plt.plot(epochs, loss, 'bo', label = 'Training loss')  
plt.plot(epochs, val_loss, 'b', label = 'Validation loss')  
plt.title('Training and validation loss')  
plt.xlabel('Epochs')  
plt.ylabel('Loss')  
plt.legend()  
plt.show()
```

训练损失和验证损失如图8. 28所示。

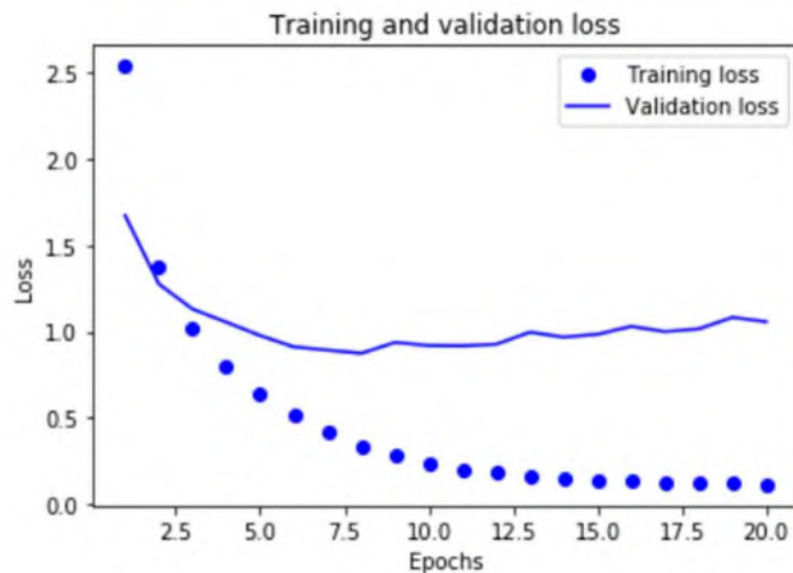


图8.28 训练损失和验证损失

9. 绘制训练精度和验证精度

```
In[14]:plt.clf()      # 清除图像
acc = history.history['acc']
val_acc = history.history['val_acc']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

训练精度和验证精度如图8.29所示。

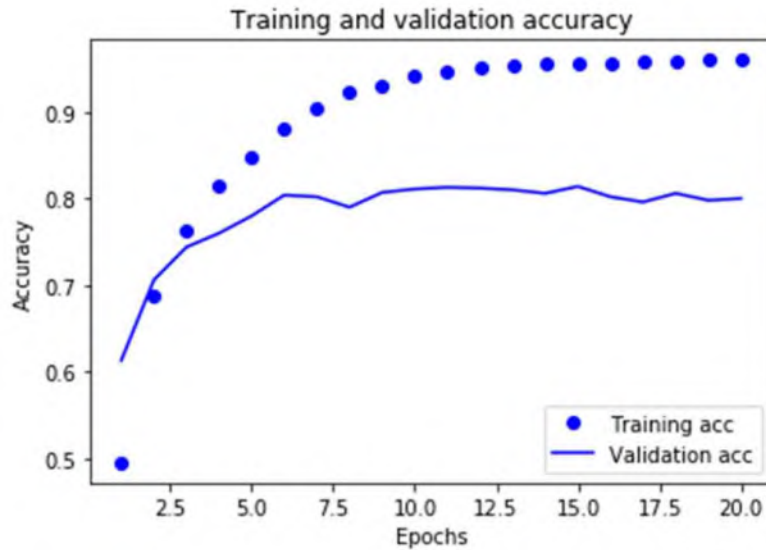


图8.29 训练精度和验证精度

10. 从头开始重新训练一个模型

```
#中间层有64个隐藏神经元
In[15]:# 从头开始训练一个新的模型
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=
(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(partial_x_train, partial_y_train, epochs=9,
batch_size = 512,
validation_data = (x_val, y_val))
results = model.evaluate(x_test, one_hot_test_labels)
In[16]: results
Out[6]: [1.0231964622134093, 0.7756010686194165]
#这种方法可以得到77%的精度
In[17]: import copy

test_labels_copy = copy.copy(test_labels)
np.random.shuffle(test_labels_copy)
float(np.sum(np.array(test_labels) ==
np.array(test_labels_copy))) /
len(test_labels)
```

```

Out[17]:0.182546749777382
#完全随机的精度约为18%
In[18]: # 在新数据上生成预测结果
predictions = model.predict(x_test)
predictions[0].shape
Out[18]: (46,)
In[19]:np.sum(predictions[0])
Out[19]: 0.99999994
In[20]:np.argmax(predictions[0])
Out[20]:3

```

11. 处理标签和损失的另一种方法

```

In[21]:y_train = np.array(train_labels)
y_test = np.array(test_labels)
model.compile(optimizer='rmsprop',
loss='sparse_categorical_crossentropy',
metrics=['acc'])

```

12. 中间层维度足够大的重要性

```

#最终输出是46维的，本代码中间层只有4个隐藏单元，中间层的维度远远小于46
In[22]:model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=
(10000,)))
model.add(layers.Dense(4, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))
model.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(partial_x_train, partial_y_train, epochs=20,
batch_size = 128,
validation_data = (x_val, y_val))
Out[22]: Epoch 20/20
7982/7982 [=====] - 2s 251us/step
- loss: 0.3668 - acc:
0.8959 - val_loss: 1.7518 - val_acc: 0.7120
#验证精度最大约为71%，比前面下降了8%。导致这一下降的主要原因在于，试图
将大量信息（这些信息
足够回复46个类别的分割超平面）压缩到维度很小的中间空间

```

13. 中间层128个

```

In[23]:model = models.Sequential()
        model.add(layers.Dense(64, activation='relu', input_shape=
(10000,)))
        model.add(layers.Dense(128, activation='relu'))
        model.add(layers.Dense(46, activation='softmax'))

        model.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
metrics=['accuracy'])
        model.fit(partial_x_train, partial_y_train, epochs=9,
batch_size = 128,
validation_data = (x_val, y_val))
        results = model.evaluate(x_test, one_hot_test_labels)
        results

Epoch 9/9
7982/7982 [=====] - 2s 237us/step
- loss: 0.1515 - acc:
0.9551 - val_loss: 0.9722 - val_acc: 0.8220
2246/2246 [=====] - 0s 160us/step
Out[23]:[1.1663296235425071, 0.7938557435440784]
#精度大约在79%
#尝试了中间层128个，但是迭代20轮，准确率却只有77%，说明迭代次数过高，
出现了过拟合

```

在单标签、多类分类问题中，网络应该以Softmax激活函数结束，这样它就会输出一个在N输出类上的概率分布。如果需要将数据分类为大量的类别，就应该避免由于中间层太小而在网络中造成信息瓶颈。

8.20 蒙特·卡罗模拟

蒙特·卡罗方法（Monte Carlo Method）也称统计模拟方法，是20世纪40年代中期由于科学技术的发展和电子计算机的发明而提出的一种以概率统计理论为指导的、非常重要的数值计算方法，是使用随机数（或更常见的伪随机数）来解决很多计算问题的方法。与它对应的是确定性算法。蒙特·卡罗方法在金融工程学、宏观经济学、计算物理学（如粒子输运计算、量子热力学计算、空气动力学计算）等领域应用广泛。

8.20.1 蒙特·卡罗模拟基本模型

1. 基本思想

当所求解的问题是某种随机事件出现的概率或者某个随机变量的期望值时，通过某种“实验”的方法，以这种事件出现的频率估计这一随机事件的概率，或者得到这个随机变量的某些数字特征，并将其作为问题的解。

2. 工作过程

蒙特·卡罗方法的解题过程中可以归结为3个主要步骤：构造或描述概率的过程、实现从已知概率分布抽样以及建立各种估计量。

3. 数学应用

通常蒙特·卡罗方法通过构造符合一定规则的随机数来解决数学上的各种问题。对于那些由于计算过于复杂而难以得到解析解或者根本没有解析解的问题，蒙特·卡罗方法是一种有效的求出数值解的方法。一般蒙特·卡罗方法在数学中常见的应用就是蒙特·卡罗积分。

4. 应用领域

蒙特·卡罗方法在金融工程学、宏观经济学、生物医学、计算物理学（如粒子输运计算、量子热力学计算、空气动力学计算、核工程）等领域应用广泛。

5. 工作过程

在解决实际问题的時候應用蒙特·卡羅方法主要有兩部分工作：

- 用蒙特·卡羅方法模擬某一過程時，需要產生某一概率分布的隨機變量。
- 用統計方法把模型的數字特徵估計出來，從而得到實際問題的數值解。

8.20.2 蒙特·卡羅模擬計算看漲期權實例

【例8.40】蒙特·卡羅模擬計算看漲期權。

1. 方法一：使用SciPy

歐式看漲期權的定價公式Black-Scholes-Merton：

$$C(S_t, K, t, T, r, \sigma) = S_t * N(d_1) - e^{-r(T-t)} * K * N(d_2)$$

$$N(d) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^d e^{-\frac{1}{2}x^2} dx$$

$$d_1 = \frac{\log\left(\frac{S_t}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}} \quad (8.74)$$

$$d_2 = \frac{\log\left(\frac{S_t}{K}\right) + \left(r - \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$

- S_t ： t 時刻股票或指數的水平。
- K ：行權價格。
- T ：期權到期年限（距離到期日的时间間隔）。
- r ：無風險利率。
- σ ：波動率（收益標準差）。

【例8.41】

代码实施环境是Anaconda 3+Jupyter Notebook。

```
In[1]: # 导入用到的库
from math import log, sqrt, exp
from scipy import stats

In[2]: # 期权的定价计算, 根据公式 (8.74)
def bsm_call_value(S_0, K, T, r, sigma):
    S_0 = float(S_0)
    d_1 = (log(S_0 / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * sqrt(T))
    d_2 = (log(S_0 / K) + (r - 0.5 * sigma ** 2) * T) / (sigma * sqrt(T))
    C_0 = (S_0 * stats.norm.cdf(d_1, 0.0, 1.0) - K * exp(-r * T) *
stats.norm.cdf(d_2, 0.0, 1.0))
    return C_0

In[3]:          # 计算的一些初始值
S_0 = 100.0      # 股票或指数初始的价格
K = 105          # 行权价格
T = 1.0          # 期权的到期年限 (距离到期日的时间间隔)
r = 0.05         # 无风险利率
sigma = 0.2      # 波动率 (收益标准差)

In[4]: # 到期期权价值
%time print (bsm_call_value(S_0, K, T, r, sigma))
#估值结果
8.021352235143176
Wall time: 34.9 ms
8.0213522作为蒙特卡罗估值的基准值, 计算耗时34.9ms
```

2. 方法二：使用Python

下面的计算仍基于BSM (Black-Scholes-Merton) 模型, 模型中的高风险标识 (股票指数) 在风险中立的情况下遵循以随机微分方程 (SDE) 表示的布朗运动。

随机微分方程:

$$dS_t = rS_t dt + \sigma S_t dZ_t \quad (8.75)$$

Z_t 是一个服从布朗运动的随机变量。

蒙特·卡罗估值公式（8.76）中的参数与公式（8.74）中的定义相同：

$$S_t = S_t - \Delta t \exp\left(\left(r - \frac{1}{2}\sigma^2\right)\Delta t + \sigma\sqrt{\Delta t}z_t\right) \quad (8.76)$$

变量 z_t 是服从正态分布的随机变量， Δt 是一个足够小的时间间隔。期权的到期年限 T 满足 $0 < T \leq 5$ （Hilpisch的著作中）。

蒙特·卡罗数值化计算方法：

（1）将到期日之前的时间间隔 $[0, T]$ 分割为多个等距的 Δt 子间隔。

（2）进行 I 次模拟， $i \in \{1, 2, \dots, I\}$ 。

每次模拟循环 M 个子间隔， $t \in \{\Delta t, 2\Delta t, 3\Delta t, \dots, T\}$ 。

①每个间隔点取一个随机数 $z_t(i)$ 。

②根据离散化公式（8.76），计算 S_T 。

③计算 T 时刻期权的内在价值 h_T ：

$$h_T(S_T(i)) = \max(S_T(i) - K, 0) \quad (8.77)$$

（3）根据 I 次的模拟计算期权到期价值：

$$C_0 = e^{-rT} \frac{1}{I} \sum_{i=1}^I h_T(S_T(i)) \quad (8.78)$$

【例8.42】

```

# 纯Python实现
from time import time
from math import exp, sqrt, log
from random import gauss, seed
seed(2000)
# 计算的一些初始值
S_0 = 100.0      # 股票或指数初始的价格
K = 105          # 行权价格
T = 1.0          # 期权的到期年限(距离到期日的时间间隔)
r = 0.05         # 无风险利率
sigma = 0.2      # 波动率(收益标准差)
M = 50           # number of time steps
dt = T/M         # time interval
I = 20000        # number of simulation
start = time()
S = []           #
for i in range(I):
    path = []     # 时间间隔上的模拟路径
    for t in range(M+1):
        if t==0:
            path.append(S_0)
        else:
            z = gauss(0.0, 1.0)
            S_t = path[t-1] * exp((r-0.5*sigma**2) * dt + sigma * sqrt(dt) * z)
            path.append(S_t)
    S.append(path)
# 计算期权现值
C_0 = exp(-r * T) * sum([max(path[-1] -K, 0) for path in S])/I
total_time = time() - start
print ('European Option value %.6f'% C_0)
print ('total time is %.6f seconds'% total_time)

```

估值结果:

```

European Option value 8.159995
total time is 1.513183 seconds

```

前30条模拟路径:

```

# 选取部分模拟路径可视化
import matplotlib.pyplot as plt
plt.figure(figsize=(10,7))
plt.grid(True)
plt.xlabel('Time step')
plt.ylabel('index level')

```

```
for i in range(30):  
    plt.plot(S[i])
```

前30条模拟路径如图8.30所示。

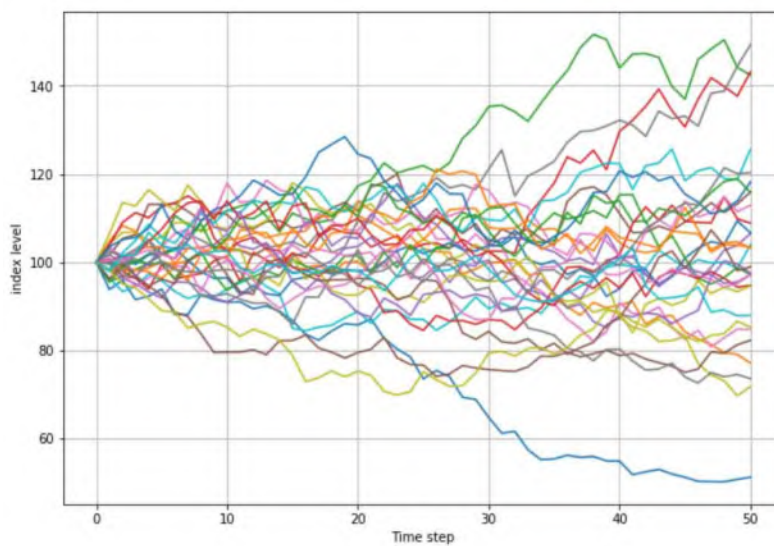


图8.30 前30条模拟路径

3. 方法三：使用NumPy

还有其它方法，如NumPy，使用一些数组计算，减少for循环的使用。

【例8.43】

```

import numpy as np
from time import time
# 计算的一些初始值
S_0 = 100.0      # 股票或指数初始的价格
K = 105          # 行权价格
T = 1.0          # 期权的到期年限(距离到期日的时间间隔)
r = 0.05         # 无风险利率
sigma = 0.2      # 波动率(收益标准差)
M = 50           # number of time steps
dt = T/M         # time interval
I = 20000        # number of simulation
# 20000条模拟路径, 每条路径50个时间步数
S = np.zeros((M+1, I))
S[0] = S_0
np.random.seed(2000)
start = time()
for t in range(1, M+1):
    z = np.random.standard_normal(I)
    S[t] = S[t-1] * np.exp((r- 0.5 * sigma **2)* dt + sigma * np.sqrt(dt)*z)
C_0 = np.exp(-r * T) * np.sum(np.maximum(S[-1] - K, 0))/I
end = time()
# 估值结果
print ('total time is %.6f seconds'%(end-start))
print ('European Option Value %.6f'%C_0)
估值结果:
total time is 0.034907 seconds
European Option Value 7.993282

前20条模拟路径:
import matplotlib.pyplot as plt

plt.figure(figsize=(10,7))
plt.grid(True)
plt.xlabel('Time step')
plt.ylabel('index level')
for i in range(20):
    plt.plot(S.T[i])

```

前20条模拟路径如图8.31所示。

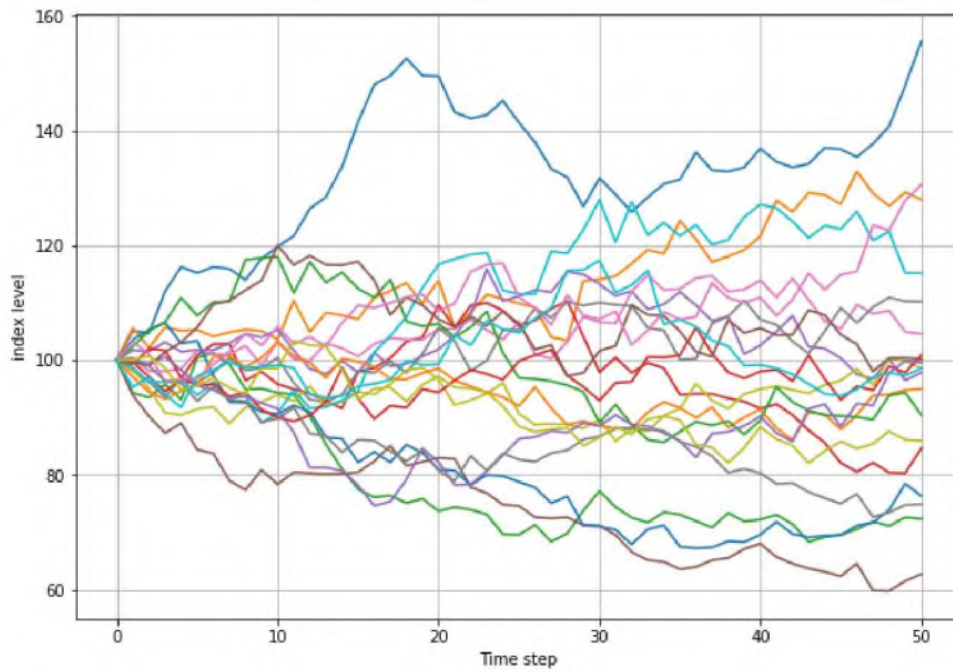


图8.31 前20条模拟路径

到期指数模拟水平：

```
# 到期时所有模拟指数水平的频率直方图
%matplotlib inline
plt.hist(S[-1], bins=50)
plt.grid(True)
plt.xlabel('index level')
plt.ylabel('frequency')
```

到期时所有模拟指数水平的频率直方图如图8.32所示。

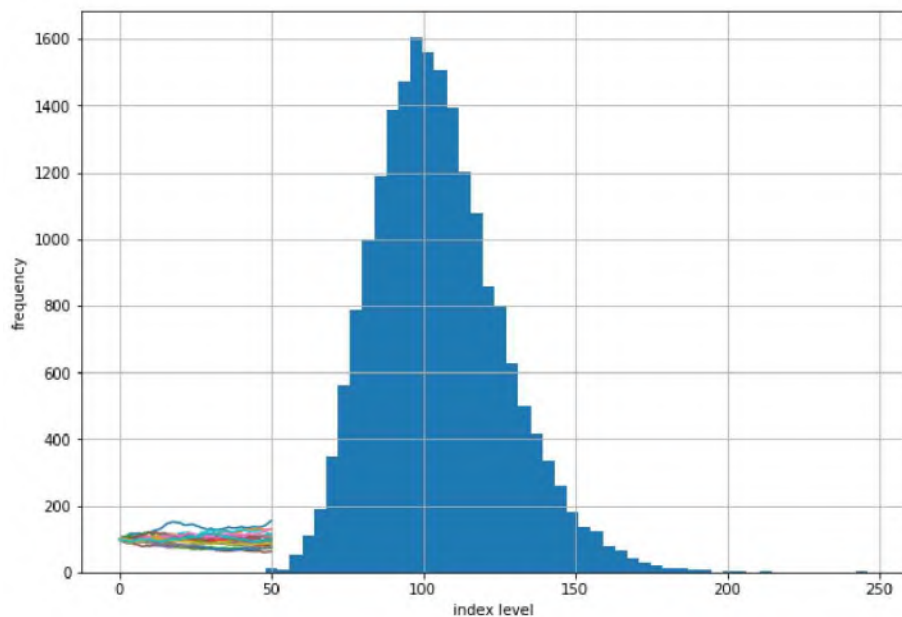


图8.32 到期时所有模拟指数水平的频率直方图

到期期权内在价值：

```
# 模拟期权到期日的内在价值
%matplotlib inline
plt.hist(np.maximum(S[-1]-K, 0), bins=50)
plt.grid(True)
plt.xlabel('option inner value')
plt.ylabel('frequency')
```

模拟期权到期日的内在价值如图8.33所示。

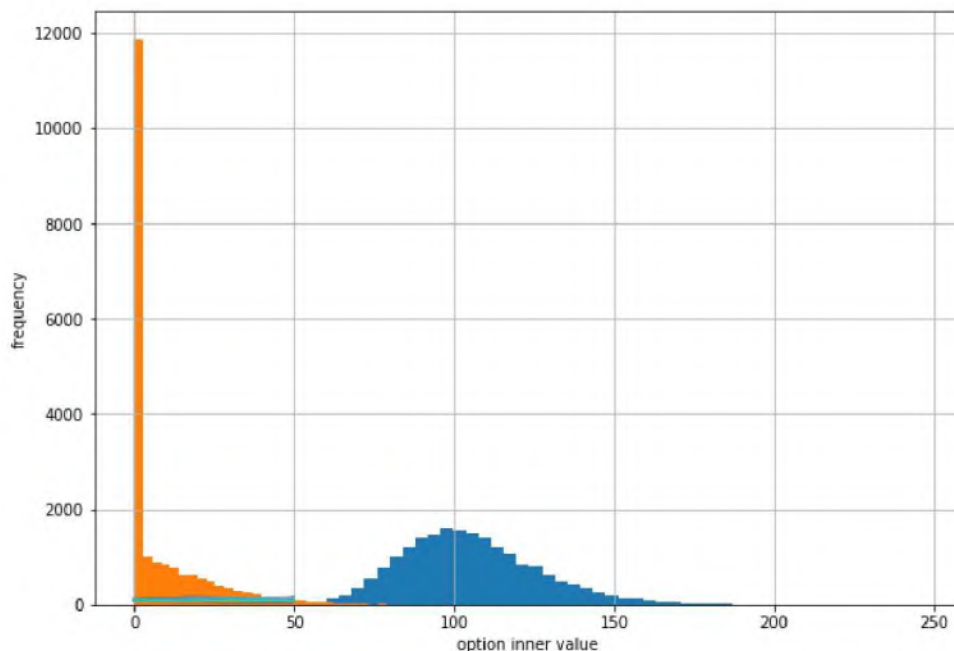


图8.33 模拟期权到期日的内在价值

结果对比：

- SciPy估值结果：8.021352235143176，耗时34.9 ms。
- Python估值结果：8.159995，耗时1.513183 seconds。
- NumPy估值结果：7.993282，耗时0.034907 seconds。

SciPy用时最短是因为没有进行20 000次的模拟估值，其他两个方法进行了20 000次的模拟，基于NumPy的计算方法速度比较快。

8.21 关联规则

关联规则（Association Rules）反映一个事物与其他事物之间的相互依存性和关联性，如果两个或多个事物之间存在一定的关联关系，其中一个事物就能通过其他事物预测到。关联规则是数据挖掘的一个重要技术，用于从大量数据中挖掘出有价值的数据项之间的相关关系。

关联规则挖掘的经典例子就是沃尔玛的啤酒与尿布的故事，通过对超市购物篮数据进行分析，即顾客放入购物篮中不同商品之间的关系来分析顾客的购物习惯，发现美国妇女经常会叮嘱丈夫下班后为孩子买尿布，30%~40%的丈夫会顺便购买喜爱的啤酒，超市就把尿布和啤酒放在一起销售增加销售额。有了这个发现后，超市调整了货架的设置，把尿布和啤酒摆放在一起销售，从而大大增加了销售额。

关联规则挖掘是一种基于规则的机器学习算法，该算法可以在大数据库中发现感兴趣的关系。它的目的是利用一些度量指标来分辨数据库中存在的强规则。也就是说关联规则挖掘是用于知识发现而非预测，所以是属于无监督的机器学习方法。

8.21.1 关联规则的概念

- 关联分析 (Association Analysis)：在大规模数据集中寻找有趣的关系。
- 频繁项集 (Frequent Item Sets)：经常出现在一块的物品的集合，包含0个或者多个项的集合称为项集。
- 支持度 (Support)：数据集中包含该项集的记录所占的比例，是针对项集来说的。
- 置信度 (Confidence)：出现某些物品时，另外一些物品必定出现的概率，针对规则而言。
- 关联规则 (Association Rules)：暗示两个物品之间可能存在很强的关系。形如 $A \rightarrow B$ 的表达式，规则 $A \rightarrow B$ 的度量包括支持度和置信度。
- 项集支持度：一个项集出现的次数与数据集所有事物数的百分比称为项集的支持度。
- 项集置信度：包含A的数据集中包含B的百分比。

1. 支持度

支持度揭示了 A 与 B 同时出现的概率。如果 A 与 B 同时出现的概率小，就说明 A 与 B 的关系不大；如果 A 与 B 同时出现得非常频繁，就说明 A 与 B 总是相关的。

支持度： $P(A \cup B)$ ，即 A 和 B 这两个项集在事务集 D 中同时出现的概率。

$$\text{support}(A \Rightarrow B) = P(A \cup B) \quad (8.79)$$

2. 置信度

置信度揭示了 A 出现时， B 是否也会出现或有多大概率出现。如果置信度为100%， A 和 B 就可以捆绑销售了。如果置信度太低，就说明 A 的出现与 B 是否出现关系不大。

置信度： $P(B | A)$ ，即在出现项集 A 的事务集 D 中，项集 B 同时出现的概率。

$$\text{confidence}(A \Rightarrow B) = P(B|A) = \frac{\text{support}(A \cup B)}{\text{support}(A)} = \frac{\text{support_count}(A \cup B)}{\text{support_count}(A)} \quad (8.80)$$

3. 设定合理的支持度和置信度

对于某条规则： $(A = a) \rightarrow (B = b)$ （支持度=30%，置信度=60%），其中支持度=30%表示在所有的数据记录中，同时出现 $A = a$ 和 $B = b$ 的概率为30%；置信度=60%表示在所有的数据记录中，在出现 $A = a$ 的情况下出现 $B = b$ 的概率为60%，也就是条件概率。支持度揭示了

$A = a$ 和 $B = b$ 同时出现的概率，置信度揭示了当 $A = a$ 出现时， $B = b$ 一定会出现的概率。

- 支持度和置信度闭值设置得过高，虽然可以减少挖掘时间，但是容易造成一些隐含在数据中非频繁的特征项被忽略掉，难以发现足够有用的规则。
- 支持度和置信度闭值设置得过低，有可能产生过多的规则，甚至产生大量冗余和无效的规则，同时由于算法存在的固有问题会导致高负荷的计算量，因此大大增加挖掘时间。

8.21.2 Apriori算法及实例

关联规则算法的主要应用是购物篮分析，是为了从大量的订单中发现商品潜在的关联。其中常用的一个算法叫Apriori算法。

Apriori算法是一种很有影响的、挖掘布尔关联规则频繁项集的算法，它是基于两个阶段频集思想的递推算法。该关联规则在分类上属于单维、单层、布尔关联规则。在这里，所有支持度大于最小支持度的项集称为频繁项集，简称频集。

关联规则的一般步骤：

- 找到频繁集。
- 在频繁集中通过可信度筛选获得关联规则。

1. Apriori的原理

如果某个项集是频繁的，那么它的所有子集也是频繁的。该定理的逆反定理为：如果某一个项集是非频繁的，那么它的所有超集（包含该集合的集合）也是非频繁的。Apriori原理的出现使得我们可以在

得知某些项集是非频繁之后，不需要计算该集合的超集，有效地避免项集数目的指数增长，从而在合理时间内计算出频繁项集。

2. 算法的思想

- 找出所有的频集，这些项集出现的频繁性至少和预定义的最小支持度一样。
- 由频集产生强关联规则，这些规则必须满足最小支持度和最小可信度。

Apriori算法采用了迭代的方法，先搜索出候选1项集及对应的支持度，剪枝去掉低于支持度的1项集，得到频繁1项集。然后对剩下的频繁1项集进行连接，得到候选的频繁2项集，筛选去掉低于支持度的候选频繁2项集，得到真正的频繁二项集，以此类推，迭代下去，直到无法找到频繁 $k+1$ 项集为止，对应的频繁 k 项集的集合即为算法的输出结果。

Apriori算法示例如图8.34所示。

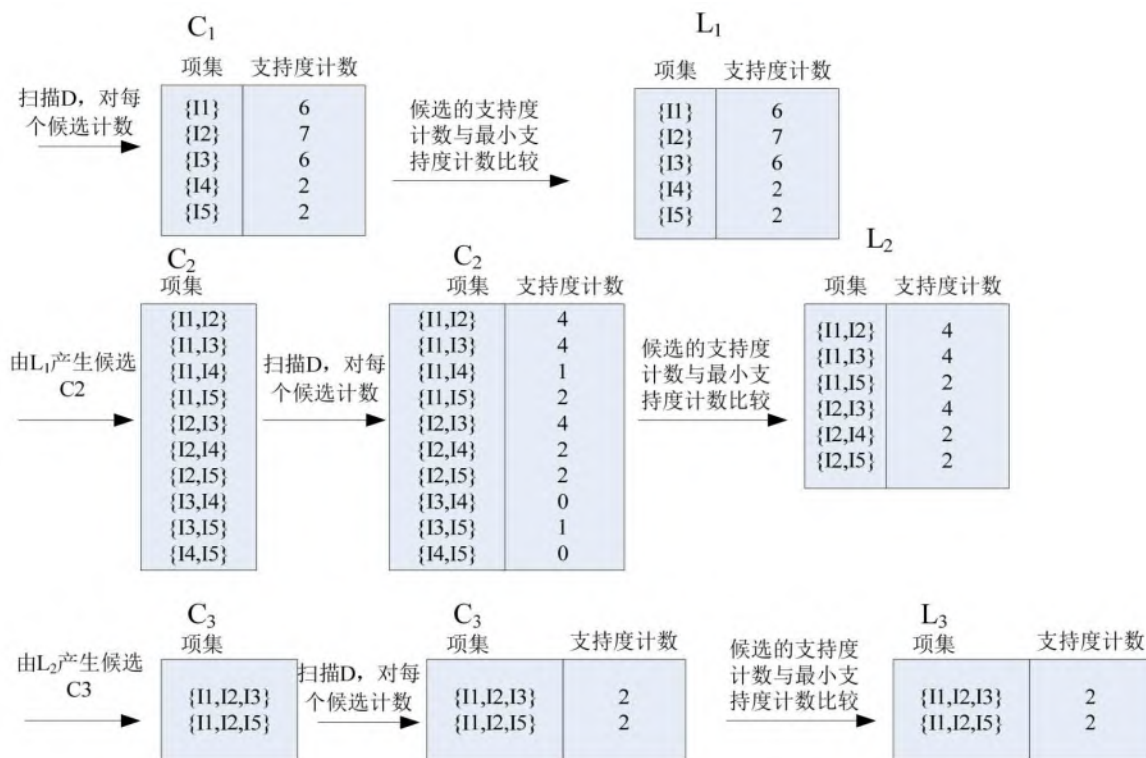


图8.34 Apriori算法示例

3. 算法的步骤

- 连接（将项集进行两两连接形成新的候选集）：

利用已经找到的 k 个项的频繁项集 L_k ，通过两两连接得出候选集 C_{k+1} ，注意进行连接的 $L_k[j]$ 、 $L_k[j]$ 必须有 $k-1$ 个属性值相同，然后另外两个不同的频繁项集分别分布在 $L_k[j]$ 中，这样求出的 C_{k+1} 为 L_{k+1} 的候选集。

- 剪枝（去掉非频繁项集）：

候选集 C_{k+1} 中的并不都是频繁项集，必须剪枝去掉非频繁项集，越早越好，以防止所处理的数据无效项越来越多。只有子集都是频繁集的候选集才是频繁集，这是剪枝的依据。

【例8.44】

```
# -*- coding: utf-8 -*-  
#Apriori 算法实现  
from numpy import *  
  
def loadDataSet():  
    return [[1, 3, 4], [2, 3, 5], [1, 2, 3, 5], [2, 5]]
```

```

# 获取候选1项集，dataSet 为事务集。返回一个 list，每个元素都是 set 集合
def createC1(dataSet):
    C1 = [] # 元素个数为1的项集（非频繁项集，因为还没有同最小支持度比较）
    for transaction in dataSet:
        for item in transaction:
            if not [item] in C1:
                C1.append([item])
    C1.sort() # 这里排序是为了生成新的候选集时可以直接认为两个 n 项候选集前面的部分相同
    # 因为除了候选1项集外，其他的候选 n 项集都是以二维列表的形式存在的，所以要将候选1项集的
    # 每一个元素都转化为一个单独的集合
    return list(map(frozenset, C1)) #map(frozenset, C1)的语义是将 C1由 Python
    # 列表转换为不变集合（frozenset, Python 中的数据结构）

# 找出候选集中的频繁项集
# dataSet 为全部数据集，Ck 为大小为 k（包含 k 个元素）的候选项集，minSupport 为设定的最小
# 支持度
def scanD(dataSet, Ck, minSupport):
    ssCnt = {} # 记录每个候选项的个数
    for tid in dataSet:
        for can in Ck:
            if can.issubset(tid):
                ssCnt[can] = ssCnt.get(can, 0) + 1 # 计算每一个项集出现的频率
    numItems = float(len(dataSet))
    retList = []
    supportData = {}
    for key in ssCnt:
        support = ssCnt[key] / numItems
        if support >= minSupport:
            retList.insert(0, key) #将频繁项集插入返回列表的首部
            supportData[key] = support
    return retList, supportData #retList 为在 Ck 中找出的频繁项集（支持度大于
    # minSupport 的），supportData 记录各频繁项集的支持度

# 通过频繁项集列表 Lk 和项集个数 k 生成候选项集 C(k+1)
def aprioriGen(Lk, k):
    retList = []
    lenLk = len(Lk)
    for i in range(lenLk):
        for j in range(i + 1, lenLk):
            # 前 k-1项相同时，才将两个集合合并，合并后才能生成 k+1项
            L1 = list(Lk[i])[:k-2]; L2 = list(Lk[j])[:k-2] # 取出两个集合的前 k-1
            # 个元素
            L1.sort(); L2.sort()
            if L1 == L2:
                retList.append(Lk[i] | Lk[j])
    return retList

```

```

# 获取事务集中所有的频繁项集
# Ck 表示项数为 k 的候选项集, 最初的 C1 通过 createC1() 函数生成。Lk 表示项数为 k 的频繁项集,
supK 为其支持度, Lk 和 supK 由 scanD() 函数通过 Ck 计算而来
def apriori(dataSet, minSupport=0.5):
    C1 = createC1(dataSet) # 从事务集中获取候选1项集
    D = list(map(set, dataSet)) # 将事务集的每个元素转化为集合
    L1, supportData = scanD(D, C1, minSupport) # 获取频繁1项集和对应的支持度
    L = [L1] # L 用来存储所有的频繁项集
    k = 2
    while (len(L[k-2]) > 0): # 一直迭代到项集数目过大而在事务集中不存在这种 n 项集
        Ck = aprioriGen(L[k-2], k) # 根据频繁项集生成新的候选项集。Ck 表示项数为 k
        # 的候选项集
        Lk, supK = scanD(D, Ck, minSupport) # Lk 表示项数为 k 的频繁项集, supK 为其
        # 支持度
        L.append(Lk); supportData.update(supK) # 添加新频繁项集和它们的支持度
        k += 1
    return L, supportData

if __name__ == '__main__':
    dataSet = loadDataSet() # 获取事务集。每个元素都是列表
    # C1 = createC1(dataSet) # 获取候选1项集。每个元素都是集合
    # D = list(map(set, dataSet)) # 转化事务集的形式, 每个元素都转化为集合
    # L1, suppDat = scanD(D, C1, 0.5)
    # print(L1, suppDat)

    L, suppData = apriori(dataSet, minSupport=0.7)
    print(L, suppData)

```

结果输出:

```

[[frozenset({5}), frozenset({2}), frozenset({3})],
[frozenset({2, 5})], []]
{frozenset({1}): 0.5, frozenset({3}): 0.75, frozenset({4}):
0.25, frozenset({2}):
0.75, frozenset({5}): 0.75, frozenset({2, 5}): 0.75,
frozenset({3, 5}): 0.5,
frozenset({2, 3}): 0.5}

```

8.21.3 FP树频集算法

在关联分析中, 频繁项集的挖掘经常用到Apriori算法。Apriori算法是一种先产生候选项集再检验是否频繁的“产生-测试”的方法。

这种方法有种弊端：当数据集很大的时候，需要不断扫描数据集，造成运行效率很低。

而FP树频集（FP-Growth）算法就很好地解决了这个问题。它的思路是把数据集中的事务映射到一棵FP树（FP-Tree）上，再根据这棵树找出频繁项集。FP树的构建过程只需要扫描两次数据集。

1. 算法步骤

- 创建FP树。
- 从FP树中挖掘频繁项集。

2. 实现流程

输入：数据集、最小值尺度。

输出：FP树、头指针表。

流程：

（A）遍历数据集，统计各元素项出现的次数，创建头指针表。

（B）移除头指针表中不满足最小值尺度的元素项。

（C）第二次遍历数据集，创建FP树。对每个数据集中的项集：

（C.1）初始化空FP树。

（C.2）对每个项集进行过滤和重排序。

（C.3）使用这个项集更新FP树，从FP树的根节点开始：

（C.3.1）如果当前项集的第一个元素项存在于FP树当前节点的子节点中，就更新这个子节点的计数值。

（C.3.2）否则创建新的子节点，更新头指针表。

(C.3.3) 对当前项集的其余元素项和当前元素项的对应子节点递归C.3的过程。

3. 算法详解

(1) 用FP树编码数据集

FP树频集算法将数据存储在一个称为FP树的紧凑数据结构中，它与计算机科学中的其他树的结构类似，但是通过链接来链接相似元素，被连起来的元素可以看作一个链表，如图8.35所示。

FP树会存储项集出现的频率，每个项集都会以路径的形式存储在树中，存在相似元素的集合会共享树的一部分。只有当集合之间完全不同时，树才会分叉，树节点上给出集合中单个元素及其在序列中出现的次数，路径会给出该序列出现的次数。相似项之间的链接即节点链接，用于快速发现相似项的位置，如表8.1所示。

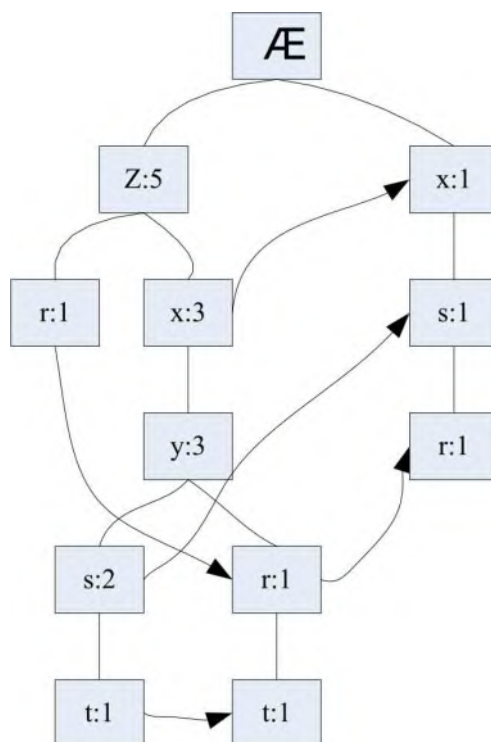


图8.35 FP树的结构示意图

表8.1 图8.35中FP树的事务数据样例

TID	事务频繁项
001	r,z,h,j,p
002	z,y,x,w,v,u,t,s
003	z
004	r,x,n,o,s
005	y,r,x,z,q,t,p
006	y,z,x,e,q,s,t,m

第一列是事务的ID，第二列是事务中的元素项，在图8.35中z出现了5次，而{r,z}项只出现了一次，所以z一定自己本身或者和其他的符号一起出现了4次，由图8.35同样可知：集合{t,s,y,x,z}出现了两次，集合{t,r,y,x,z}出现了一次，所以z一定本身出现了一次。看了表8.1可能会有疑问，为什么在图8.35中没有p、q、w、v等元素呢？这是因为通常会给所有的元素设置一个阈值（Apriori里的支持度），低于这个阈值的元素不加以研究。

（2）构建FP树

构建FP树是算法的第一步，在FP树的基础之上再对频繁项集进行挖掘。为了构建FP树，要对数据集扫描两次，第一次对所有元素项出现的次数进行计数，记住如果一个元素不是频繁的，那么包含这个元素的超集也不是频繁的，所以不需要考虑这些超集，第二遍的扫描只考虑那些频繁元素。

除了图8.35给出的FP树之外，还需要一个头指针表来指向给定类型的第一个实例。利用头指针表可以快速访问FP树中一个给定类型的所有元素，发现相似元素项，如图8.36所示。

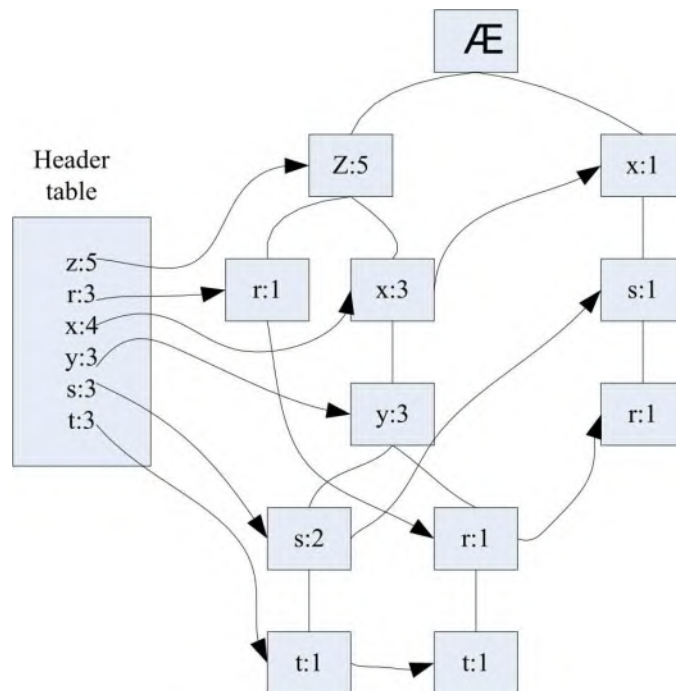


图8.36 带头指针的FP树

头指针表的数据结构是字典，除了存放头指针元素之外，还可以存放FP树中每类元素的个数。第一次遍历数据集得到每个元素项出现的频率，接下来去掉不满足最小值支持度的元素项，就可以创建FP树了，构建时将每个项集添加到一个已经存在的路径中，如果该路径不存在，就创建一个新的路径。每个事务都是一个无序的集合，然而在FP树中相同项只会出现一次， $\{x, y, z\}$ 和 $\{y, z, x\}$ 应该在同一个路径上，所以在将集合添加到树之前要对每个集合进行排序，排序是基于各个元素出现的频率来进行的，使用图8.36头指针表中单个元素出现的值对表8.1中的数据进行过滤，重排后的新数据如表8.2所示。

表8.2 移除非频繁项重新排序后的事务表

TID	事务频繁项	事务频繁项过滤和排序
001	r,z,h,j,p	z,r
002	z,y,x,w,v,u,t,s	z,x,y,s,t
003	z	z
004	r,x,n,o,s	x,s,r
005	y,r,x,z,q,t,p	z,x,y,r,t
006	y,z,x,e,q,s,t,m	z,x,y,s,t

现在就可以构建FP树了，从空集开始，不断向其中添加频繁项集。过滤排序后的事务依次添加到树中，如果树中已有现有元素，就增加该元素的值；如果元素不存在，就添加新分枝。表8.2中前两条事务添加的过程如图8.37所示。

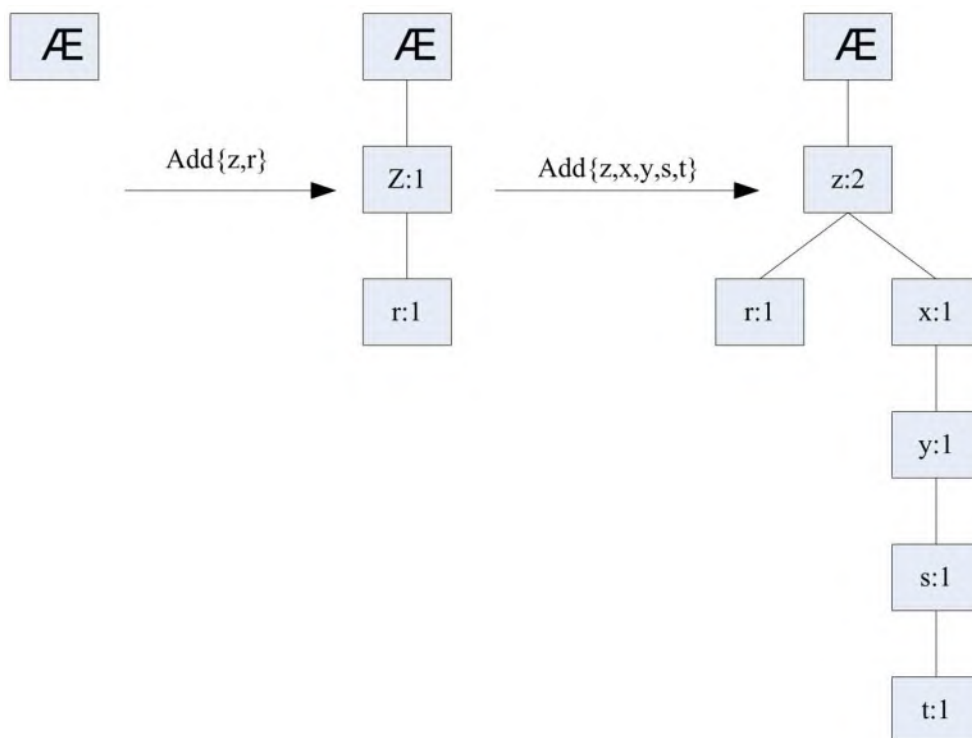


图8.37 FP构建过程示例图

(3) 从FP树中挖掘频繁项

有了FP树之后就可以抽取频繁项集了，思想与Apriori算法大致一样，从单元素项集开始，逐步构建更大的集合，只不过不需要原始的数据集了。

从FP树中抽取频繁项集的3个基本步骤：

- ①从FP树中获得条件模式基。
- ②利用条件模式基构建一个条件FP树。
- ③迭代重复步骤①和②直到树只包含一个元素项为止。

（4）抽取条件模式基

条件模式基是以所查找元素项为结尾的路径集合，每一条路径包含一条前缀路径和结尾元素。在图8.36中，符号r的前缀路径有{x, s}、{z, x, y}和{z}，每一条前缀路径都与一个数据值关联，这个值等于路径上r的数目。表8.3中列出了单元素频繁项的所有前缀路径。

表8.3 每个频繁项的前缀路径

频繁项	前导路径
s	{ }5
r	{x,s}1,{z,x,y}1,{z}1
x	{z}3,{ }1
y	{z,x}3
s	{z,x,y}2,{x}1
t	{z,x,y,s}2,{z,x,y,r}1

前缀路径将被用于构建条件FP树。为了获得这些路径，可以对FP树进行穷举式搜索，直到获得想要的频繁项为止。但可以使用一个更为有效的方法加速搜索过程。可以用先前的头指针表来创建更为有效的方法，头指针表中包含相同类型元素链表的起始指针，一旦到达每一个元素项，就可以上溯这棵树直到根节点为止。

(5) 创建条件FP树

对于每一个频繁项，都要创建一个条件FP树，将上面的条件模式基作为输入，通过相同的建树方法来构建这些条件树，然后递归地发现频繁项、发现条件模式基以及发现其他的条件树。例如，假定为频繁项 t 创建一个条件FP树，然后对 $\{t, y\}$ 、 $\{t, x\}$... 重复该过程。元素项 t 的条件FP树的构建过程如图8.38所示。

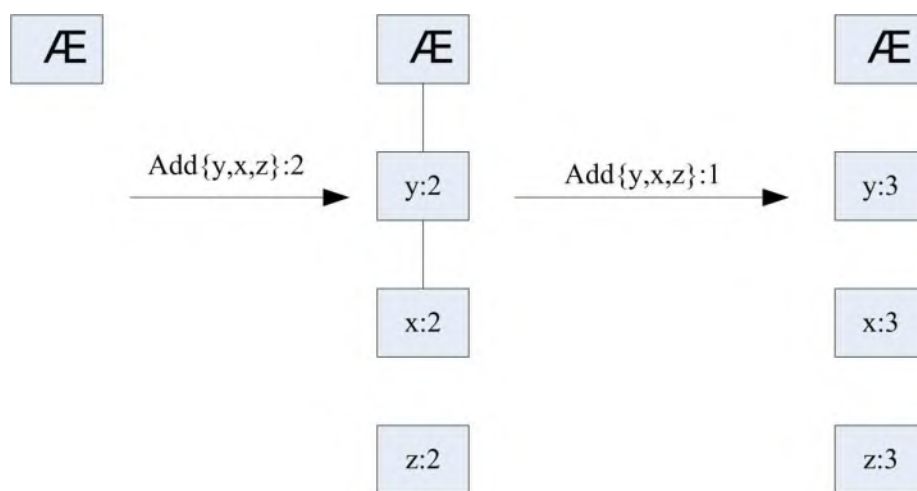


图8.38 构建条件FP树示例图

s 、 r 虽然是条件模式基的一部分，且单独看都是频繁项，但是在 t 的条件树中，它却是不频繁的，分别出现了两次和一次，小于阈值3，所以 $\{t, r\}$ 、 $\{t, s\}$ 不是频繁的。接下来对集合 $\{t, z\}$ 、 $\{t, x\}$ 、 $\{t, y\}$ 来挖掘对应的条件树，会产生更复杂的频率项集，该过程重复进行，直到条件树中没有元素为止。

【例8.45】

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
# FP 树类
class treeNode:
    def __init__(self, nameValue, numOccur, parentNode):
        self.name = nameValue      #节点元素名称，在构造时初始化为给定值
        self.count = numOccur      # 出现次数，在构造时初始化为给定值
        self.nodeLink = None       # 指向下一个相似节点的指针，默认为 None
        self.parent = parentNode   # 指向父节点的指针，在构造时初始化为给定值
        self.children = {}         # 指向子节点的字典，以子节点的元素名称为键，指向子节
点的指针为值，初始化为空字典

    # 增加节点的出现次数值
    def inc(self, numOccur):
        self.count += numOccur

    # 输出节点和子节点的 FP 树结构
    def disp(self, ind=1):
        print(' ' * ind, self.name, ' ', self.count)
        for child in self.children.values():
            child.disp(ind + 1)

# =====构建 FP 树=====

# 对不是第一个出现的节点，更新头指针块。就是添加到相似元素链表的尾部
def updateHeader(nodeToTest, targetNode):

```



```

while (nodeToTest.nodeLink != None):
    nodeToTest = nodeToTest.nodeLink
nodeToTest.nodeLink = targetNode

# 根据一个排序过滤后的频繁项更新 FP 树
def updateTree(items, inTree, headerTable, count):
    if items[0] in inTree.children:
        # 有该元素项时计数值+1
        inTree.children[items[0]].inc(count)
    else:
        # 没有这个元素项时创建一个新节点
        inTree.children[items[0]] = treeNode(items[0], count, inTree)
        # 更新头指针表或前一个相似元素项节点的指针指向新节点
        if headerTable[items[0]][1] == None: # 如果是第一次出现, 就在头指针表中增加
对该节点的指向
            headerTable[items[0]][1] = inTree.children[items[0]]
        else:
            updateHeader(headerTable[items[0]][1], inTree.children[items[0]])

    if len(items) > 1:
        # 对剩下的元素项迭代调用 updateTree 函数
        updateTree(items[1:], inTree.children[items[0]], headerTable, count)

# 主程序. 创建 FP 树, dataSet 为事务集, 为一个字典, 键为每个事物, 值为该事物出现的次数. minSup
为最低支持度
def createTree(dataSet, minSup=1):
    # 第一次遍历数据集, 创建头指针表
    headerTable = {}
    for trans in dataSet:
        for item in trans:
            headerTable[item] = headerTable.get(item, 0) + dataSet[trans]
    # 移除不满足最小支持度的元素项
    keys = list(headerTable.keys()) # 因为字典要求在迭代中不能修改, 所以转化为列表
    for k in keys:
        if headerTable[k] < minSup:
            del(headerTable[k])
    # 空元素集, 返回空
    freqItemSet = set(headerTable.keys())
    if len(freqItemSet) == 0:
        return None, None
    # 增加一个数据项, 用于存放指向相似元素项的指针
    for k in headerTable:
        headerTable[k] = [headerTable[k], None] # 每个键的值, 第一个为个数, 第二个
为下一个节点的位置
    retTree = treeNode('Null Set', 1, None) # 根节点
    # 第二次遍历数据集, 创建 FP 树
    for tranSet, count in dataSet.items():

```

```

        localD = {} # 记录频繁1项集的全局频率，用于排序
        for item in tranSet:
            if item in freqItemSet: # 只考虑频繁项
                localD[item] = headerTable[item][0] # 注意这个[0]，因为之前加过一个
数据项
            if len(localD) > 0:
                orderedItems = [v[0] for v in sorted(localD.items(), key=lambda p:
p[1], reverse=True)] # 排序
                updateTree(orderedItems, retTree, headerTable, count) # 更新 FP 树
        return retTree, headerTable

# =====查找元素条件模式基=====

# 直接修改 prefixPath 的值，将当前节点 leafNode 添加到 prefixPath 的末尾，然后递归添加其
父节点
# prefixPath 就是一条从 treeNode（包括 treeNode）到根节点（不包括根节点）的路径
def ascendTree(leafNode, prefixPath):
    if leafNode.parent != None:
        prefixPath.append(leafNode.name)
        ascendTree(leafNode.parent, prefixPath)

# 为给定元素项生成一个条件模式基（前缀路径）。basePat 表示输入的频繁项，treeNode 为当前 FP
树中对应的第一个节点
# 函数返回值即为条件模式基 condPats，用一个字典表示，键为前缀路径，值为计数值
def findPrefixPath(basePat, treeNode):
    condPats = {} # 存储条件模式基
    while treeNode != None:
        prefixPath = [] # 用于存储前缀路径
        ascendTree(treeNode, prefixPath) # 生成前缀路径
        if len(prefixPath) > 1:
            condPats[frozenset(prefixPath[1:])] = treeNode.count # 出现的数量就
是当前叶子节点的数量
        treeNode = treeNode.nodeLink # 遍历下一个相同元素
    return condPats

# =====递归查找频繁项集=====
# 根据事务集获取 FP 树和频繁项
# 遍历频繁项，生成每个频繁项的条件 FP 树和条件 FP 树的频繁项
# 这样每个频繁项条件 FP 树的频繁项都构成了频繁项集

# inTree 和 headerTable 是由 createTree() 函数生成的事务集的 FP 树
# minSup 表示最小支持度
# preFix 请传入一个空集合 (set([]))，将在函数中用于保存当前前缀
# freqItemList 请传入一个空列表 ([])，将用来储存生成的频繁项集
def mineTree(inTree, headerTable, minSup, preFix, freqItemList):
    # 对频繁项按出现的数量进行排序
    sorted_headerTable = sorted(headerTable.items(), key=lambda p: p[1][0])

```

```

#返回重新排序的列表。每个元素是一个元组，[(key,[num,treeNode],())]
bigL = [v[0] for v in sorted_headerTable] # 获取频繁项
for basePat in bigL:
    newFreqSet = preFix.copy() # 新的频繁项集
    newFreqSet.add(basePat) # 当前前缀添加一个新元素
    freqItemList.append(newFreqSet) # 所有的频繁项集列表
    condPattBases = findPrefixPath(basePat, headerTable[basePat][1])
# 获取条件模式基，就是basePat元素的所有前缀路径。它像一个新的事务集
    myCondTree, myHead = createTree(condPattBases, minSup) # 创建条件FP树

if myHead != None:
    # 用于测试
    print('conditional tree for:', newFreqSet)
    myCondTree.disp()
    mineTree(myCondTree, myHead, minSup, newFreqSet, freqItemList)
# 递归直到不再有元素

# 生成数据集
def loadSimpDat():
    simpDat = [['r', 'z', 'h', 'j', 'p'],
                ['z', 'y', 'x', 'w', 'v', 'u', 't', 's'],
                ['z'],
                ['r', 'x', 'n', 'o', 's'],
                ['y', 'r', 'x', 'z', 'q', 't', 'p'],
                ['y', 'z', 'x', 'e', 'q', 's', 't', 'm']]
    return simpDat

# 将数据集转化为目标格式
def createInitSet(dataSet):
    retDict = {}
    for trans in dataSet:
        retDict[frozenset(trans)] = 1
    return retDict

if __name__ == '__main__':
    minSup = 3
    simpDat = loadSimpDat() # 加载数据集
    initSet = createInitSet(simpDat) # 转化为符合格式的事务集
    myFPtree, myHeaderTab = createTree(initSet, minSup) # 形成FP树
    # myFPtree.disp() # 打印树

    freqItems = [] # 用于存储频繁项集
    mineTree(myFPtree, myHeaderTab, minSup, set([]), freqItems) # 获取频繁项集
    print(freqItems) # 打印频繁项集

```

结果输出：

```
conditional tree for: {'s'}
  Null Set  1
    x        3
conditional tree for: {'y'}
  Null Set  1
    z        3
    x        3
conditional tree for: {'y', 'x'}
  Null Set  1
    z        3
conditional tree for: {'t'}
  Null Set  1
    y        3
    x        3
    z        3
conditional tree for: {'t', 'x'}
  Null Set  1
    y        3
conditional tree for: {'t', 'z'}
  Null Set  1
    y        3
    x        3
conditional tree for: {'t', 'z', 'x'}
  Null Set  1
    y        3
conditional tree for: {'x'}
  Null Set  1
    z        3
  [{'r'}, {'s'}, {'x', 's'}, {'y'}, {'y', 'z'}, {'y', 'x'}, {'y', 'z', 'x'}, {'t'},
{'t', 'y'}, {'t', 'x'}, {'t', 'y', 'x'}, {'t', 'z'}, {'t', 'z', 'y'}, {'t', 'z',
'x'}, {'t', 'z', 'y', 'x'}, {'x'}, {'z', 'x'}, {'z'}]
```

8.22 Uplift Modeling

Uplift Modeling采用随机科学控制，不仅可以衡量事务行为的有效性，还可以建立预测模型、预测行为的增量响应。它是一种数据挖掘技术，主要应用于金融服务、电信和零售直销行业，用于追加销售、交叉销售、客户流失和扣除留置。

通常的Propensity Model和Response Model只是给目标用户打了个分，并没有确保模型的结果可以使得活动的提升最大化，它没有告诉市场营销人员哪个用户最有可能提升活动响应，因此需要另一个统计模型来定位那些可以被营销推广活动明显驱动他们偏好响应的用户，也就是“营销敏感”用户。Uplift Model的最终目标就是找到最有可能被营销活动影响的用户，从而提升活动的反响（ $r(\text{test}) - r(\text{control})$ ），提升ROI（投资回收率），提升整体的市场响应率。

下面说明进行Uplift Modeling的方法。

(1) 建立两个Logistic模型：

$$\text{Logit}(P(\text{test}(\text{response} | X, \text{treatment}=1))) = a + b * X + c * \text{treatment}$$

$$\text{Logit}(P(\text{control}(\text{response} | X, \text{treatment}=0))) = a + b * X$$

(2) 将两个得分相减，计算Uplift Score：

$$\text{Score} = P(\text{test}(\text{response} | X, \text{treatment}=1)) - P(\text{control}(\text{response} | X, \text{treatment}=0))$$

训练样本：

由于强化学习需要用到的是反馈数据，因此训练样本的及时、自动更新是比较重要的方面（尤其是label的更新和实时特征的更新），才能体现出强化学习优于机器学习的地方，使用用户反馈的标注样本来更新训练样本库，可以使得反馈及时地得到学习，从而优化算法效果。

【例8.46】实验环境是Jupyter Notebook。

我们将使用模拟数据，目标是能够预测Uplift，即每个人的治疗产生的结果概率的差异。

```
In [1]:%pylab inline
import warnings
warnings.filterwarnings("ignore")
```

```
import pandas as pd
Populating the interactive namespace from numpy and
matplotlib
```

1. 加载数据 (Loading the data)

```
In [2]:# load dataset
thefile = "/Users /customer_simulation.csv"
df = pd.read_csv(thefile)
In [3]:df.shape
Out[3]:(10000, 24)
This simulated dataset contains 10000 lines.
In [4]:df.head()
Out[4]: 5*24表部分数据截图
```

	customer_id	Node1	Node2	Node3	Node4	No
0	1	Value4	Value1	Value2	Value2	Val
1	2	Value2	Value1	Value1	Value2	Val
2	3	Value2	Value2	Value1	Value3	Val
3	4	Value3	Value1	Value1	Value2	Val
4	5	Value4	Value1	Value1	Value3	Val

5 rows 5*24ad()dat

```
In [5]:df.target_control.value_counts()
Out[5]:control    5063
target    4937
Name: target_control, dtype: int64
In [6]:df.customer_type.value_counts()
Out[6]:lost_cause    2554
sleeping_dog    2528
persuadable    2471
sure_thing    2447
Name: customer_type, dtype: int64
In [7]:df.outcome.value_counts()
Out[7]:
0    5047
1    4953
Name: outcome, dtype: int64
```

2. 矫形 (Reshaping)

```

In [9]:# dummify
feat = [x for x in df.columns if "Node" in x]
features = []
for f in feat :
    dummies = pd.get_dummies(df[f]).rename(columns=lambda x: f +
    "_" + str(x))

    features = features + list(dummies.columns)
    df = pd.concat([df, dummies], axis=1)
    df = df.drop([f], axis=1)
    print "done", f

```

打印结果:

```

done Node1
done Node2
done Node3
done Node4
done Node5
done Node6
done Node7
done Node8
done Node9
done Node10
done Node11
done Node12
done Node13
done Node14
done Node15
done Node17
done Node18
done Node19
done Node20

In [10]:train_df = df[df["train_test"]=="train"]
test_df = df[df["train_test"]=="test"]

print train_df.shape
print test_df.shape

```

打印结果:

```

(7952, 81)
(2048, 81)

```

3. 两类模型 (Two model approach)

用目标数据集的结果概率减去控制数据集的结果概率的差来建模 Uplift:

```
In [11]:target =
train_df[train_df["target_control"]=="target"]
control = train_df[train_df["target_control"]=="control"]

print target.shape
print control.shape
```

打印结果:

```
(3934, 81)
(4018, 81)
In [12]:target_X = target[features]
control_X = control[features]
target_Y = target[['outcome']]
control_Y = control[['outcome']]
test_X = test_df[features]
```

4. 训练 (training)

```
In [13]:from sklearn.ensemble import GradientBoostingClassifier

clf1 = GradientBoostingClassifier(n_estimators = 100,learning_rate =
0.1,max_depth = 3)
clf2 = GradientBoostingClassifier(n_estimators = 100,learning_rate =
0.1,max_depth = 3)
In [14]:clf1.fit(target_X.values,target_Y.values.ravel())
clf2.fit(control_X.values,control_Y.values.ravel())

Out[14]:GradientBoostingClassifier(init=None, learning_rate=0.1,
loss='deviance',
max_depth=3, max_features=None, max_leaf_nodes=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100,
presort='auto', random_state=None, subsample=1.0, verbose=0,
warm_start=False)
```


5. 得分 (scoring)

```
In [36]:test_df["proba_outcome_target"] =
clf1.predict_proba(test_X)[: ,1]
test_df["proba_outcome_control"] =
clf2.predict_proba(test_X)[: ,1]
# uplift is just the difference.
test_df["uplift_1"] =
test_df["proba_outcome_target"] -
test_df["proba_outcome_control"]
```

6. 类别修正 (Class Modification approach)

实现类别修正的方法如下：

- 堆叠目标和控制数据。
- 翻转控制数据集的目标。
- 在这个目标上训练一个模型。
- Uplift是预测概率的2倍减去1。

```
In [21]:train_df['istarget'] = train_df['target_control'].map(lambda x : 1 if
x=='target' else 0)
train_df['modified_outcome'] = train_df['outcome'] * train_df['istarget'] \
+
(1-train_df['outcome'])*(1-train_df['istarget'])
In [27]:from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators = 500,max_depth=10)
In [28]:clf.fit(train_df[features].values,
train_df['modified_outcome'].values.ravel())

Out[28]:RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
max_depth=10, max_features='auto', max_leaf_nodes=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=500, n_jobs=1,
oob_score=False, random_state=None, verbose=0,
warm_start=False)
In [43]:test_df["uplift_2"] = 2*clf.predict_proba(test_X)[: ,1] -1
```

7. 保存预测结果 (Save Predictions)

```
In [44]: test_df.to_csv("/Users/uplift_predictions.csv")
```

8.23 集成方法

1. 集成方法的概念

通过聚合多个分类器的预测来提高分类的准确率，这种技术称为集成方法（Ensemble Method）。集成方法由训练数据构建一组基分类器，然后通过对每个基分类器的预测进行权重控制以作出分类。

集成技术在数据挖掘的3个方向：

- 在样本上做工作，基分类器为同一个分类算法，主要的技术有Bagging和Boosting。
- 在分类算法上做工作，即用于训练基分类器的样本相同，基分类器的算法不同。
- 在样本属性集上做工作，即在不同的属性空间上构建基分类器，比较出名的是randomforestTree算法，这个在weka中也有实现。

集成方法大致包括3种框架：Bagging、Boosting和Stacking。对于Bagging来说，添加随机变量的学习器反而能够提高整体的效果。这3种方法中，Boosting是表现最好的模型，它与有着广泛研究基础的加性模型（Addictive Models）的统计技术有着相近的关系。

2. 集成学习的几种方法

- 在验证数据集上找到表现最好的模型作为最终的预测模型。
- 对多个模型的预测结果进行投票或者取平均值。

- 对多个模型的预测结果做加权平均。

以上几种思路对应了集成学习中的几种主要的学习框架。

(1) 多个模型投票或者取平均值

对于数据集训练多个模型来说，分类问题可以采用投票的方法，选择票数最多的类别作为最终的类别；而回归问题，可以采用取均值的方法，取得的均值作为最终的结果。在这样的思路中，最著名的是Bagging方法，Bagging即Bootstrap Aggregating，其中Bootstrap是一种有放回的抽样方法，其抽样策略是简单的随机抽样。在Bagging方法中，让学习算法训练多次，每次的训练集由初始的训练集中随机取出的训练样本组成，初始的训练样本在某次的训练集中可能出现多次或者根本不出现。最终训练出个预测函数，最终的预测函数对于分类和回归问题可采用如下两种方法：

- 对于分类问题采用投票的方法，得票最多的类别为最终的类别。
- 对于回归问题采用简单的平均方法。

随机森林算法就是基于Bagging思想的学习算法。

(2) 对多个模型的预测结果进行加权平均

在上述的Bagging方法中，其特点在于随机化抽样，通过反复地抽样训练新的模型，最终在这些模型的基础上取平均。而对多个模型的预测结果进行加权平均，则是将多个弱学习模型提升为强学习模型，这就是Boosting的核心思想。在Boosting算法中，初始化时对每个训练样本赋予相等的权重，如 $\frac{1}{n}$ ，然后用该学习算法对训练集训练 G 轮，每次训练后，对训练失败的训练样本赋予更大的权重，也就是让学习算法在后续的学习中对几种比较难学的训练样本进行学习，从而得到一个预测函数序列 $\{h_1, \dots, h_G\}$ ，其中每

个 h_i 都有一个权重，预测效果好的预测函数的权重较大。最终的预测函数为 H 。对于分类和回归问题可采用如下两种方法。

- 分类问题：有权重的投票方式。
- 回归问题：加权平均。

3. 组合分类器的性能优于单个分类器的条件

组合分类器的性能优于单个分类器必须满足两个条件：

- 基分类器之间是相互独立的。
- 基分类器应当优于随机猜测分类器。

实践时很难保证基分类器之间完全独立，但是在基分类器轻微相关的情况下，组合方法可以提高分类的准确率。

Adaboost算法

如果存在一个多项式的学习算法能够学习并且正确率很高，就称为强可学习；相反，弱可学习就是学习的正确率仅比随机猜测稍好。

1. Adaboost 迭代算法

整个Adaboost迭代算法就3步：

(1) 初始化训练数据的权值分布。如果有 N 个样本，那么每一个训练样本最开始时都被赋予相同的权重： $1/N$ 。

(2) 训练弱分类器。在具体训练过程中，如果某个样本点已经被准确地分类，那么在构造下一个训练集时，它的权重就会被降低；相反，如果某个样本点没有被准确地分类，它的权重就会得到提高。然

后，权重更新过的样本集被用于训练下一个分类器，整个训练过程如此迭代地进行下去。

(3) 将各个训练得到的弱分类器组合成强分类器。各个弱分类器的训练过程结束后，加大分类误差率小的弱分类器的权重，使其在最终的分类函数中起着较大的决定作用，而降低分类误差率大的弱分类器的权重，使其在最终的分类函数中起着较小的决定作用。换言之，误差率低的弱分类器在最终分类器中占的权重较大，否则较小。

2. Adaboost算法的流程

给定一个训练数据集 $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ ，其中实例 $x \in \mathcal{X}$ ，而实例空间 $\mathcal{X} \in \mathbb{R}^n$ ， y_i 属于标记集合 $\{-1, +1\}$ ，Adaboost的目的就是从训练数据中学习一系列弱分类器或基本分类器，然后将这些弱分类器组合成一个强分类器。

Adaboost算法的流程说明如下：

(1) 步骤1：初始化训练数据的权值分布。每一个训练样本最开始时都被赋予相同的权重： $1/N$ 。

$$D_1 = (w_{11}, w_{12}, \dots, w_{1i}, \dots, w_{1N}), w_{1i} = \frac{1}{N}, i = 1, 2, \dots, N \quad (8.81)$$

(2) 步骤2：进行多轮迭代，用 $m = 1, 2, \dots, M$ 表示迭代的第多少轮。

①使用具有权值分布 D_m 的训练数据集学习，得到基本分类器：

$$G_m(x): \mathcal{X} \rightarrow \{-1, +1\} \quad (8.82)$$

②计算 $G_m(x)$ 在训练数据集上的分类误差率：

$$e_m = P(G_m(x_i) \neq y_i) = \sum_{i=1}^N w_{mi} I(G_m(x_i) \neq y_i) \quad (8.83)$$

③计算 $G_m(x)$ 的系数， a_m 表示 $G_m(x)$ 在最终分类器中的重要程度（目的是得到基本分类器在最终分类器中所占的权重）：

$$a_m = \frac{1}{2} \log \frac{1 - e_m}{e_m} \quad (8.84)$$

由上述式子可知， $e_m \leq 1/2$ 时， $a_m \geq 0$ ，且 a_m 随着 e_m 的减小而增大，意味着分类误差率越小的基本分类器在最终分类器中的作用越大。

④更新训练数据集的权值分布（目的是得到样本的新的权值分布），用于下一轮迭代：

$$\begin{aligned} D_{m+1} &= (w_{m+1,1}, w_{m+1,2}, \dots, w_{m+1,i}, \dots, w_{m+1,N}) \\ w_{m+1,i} &= \frac{w_{mi}}{Z_m} \exp(-a_m y_i G_m(x_i)), i = 1, 2, \dots, N \end{aligned} \quad (8.85)$$

使得被基本分类器 $G_m(x)$ 误分类样本的权值增大，而被正确分类样本的权值减小。通过这样的方式，Adaboost方法能“聚焦于”那些较难分的样本上。

其中， Z_m 是规范化因子，使得 D_{m+1} 成为一个概率分布：

$$Z_m = \sum_{i=1}^N w_{mi} \exp(-a_m y_i G_m(x_i)) \quad (8.86)$$

（3）步骤3：组合各个弱分类器。

$$f(x) = \sum_{m=1}^M a_m G_m(x) \quad (8.87)$$

从而得到最终分类器：

$$G(x) = \text{sign}(f(x)) = \text{sign}\left(\sum_{m=1}^M a_m G_m(x)\right) \quad (8.88)$$

①Adaboost的误差界：

通过上面的例子可知，Adaboost在学习的过程中不断减少训练误差 e ，直到各个弱分类器组合成最终分类器，Adaboost最终分类器的训练误差的上界为：

$$\frac{1}{N} \sum_{i=1}^N I(G(x_i) \neq y_i) \leq \frac{1}{N} \sum_i \exp(-y_i f(x_i)) \prod_m Z_m \quad (8.89)$$

②对于二分类而言，有如下结果：

$$\prod_{m=1}^M Z_m = \prod_{m=1}^M (2\sqrt{e_m(1-e_m)}) = \prod_{m=1}^M \sqrt{1-4\gamma_m^2} \leq \exp\left(-2\sum_{m=1}^M \gamma_m^2\right) \quad (8.90)$$

其中， $\gamma_m = \frac{1}{2} - e_m$ 。

这个结论表明，Adaboost的训练误差是以指数速率下降的。另外，Adaboost算法不需要事先知道下界 γ ，Adaboost具有自适应性，它能适应弱分类器各自的训练误差率。

【例8.47】 在一个简单数据集上的Adaboost的实现。

```
# -*- coding: utf-8 -*-
from numpy import*
def loadSimpData():
    datMat = matrix([[ 1. ,  2.1],
        [ 2. ,  1.1],
        [ 1.3,  1. ],
        [ 1. ,  1. ],
        [ 2. ,  1. ]])
    classLabels = [1.0, 1.0, -1.0, -1.0, 1.0]
    return datMat,classLabels

def stumpClassify(dataMatrix,dimen,threshVal,threshIneq):
    retArray = ones((shape(dataMatrix)[0],1))
    if threshIneq == 'lt':
        retArray[dataMatrix[:,dimen]<= threshVal] = -1.0
    else:
        retArray[dataMatrix[:,dimen] > threshVal] = -1.0
    return retArray
```



```

def buildStump(dataArr,classLabels,D):
    dataMatrix = mat(dataArr)
    labelMat = mat(classLabels).T
    m,n = shape(dataMatrix)
    numSteps = 10.0 ; bestStump = {} ; bestClasEst = mat(zeros((m,1)))
    minError = inf
    for i in range(n):
        rangeMin = dataMatrix[:,i].min(); rangeMax = dataMatrix[:,i].max()
        stepSize = (rangeMax- rangeMin)/numSteps
        for j in range(-1,int(numSteps)+1):
            for inequal in ['lt','gt']:
                threshVal = (rangeMin + float(j)* stepSize)
                predictedVals = stumpClassify(dataMatrix, i, threshVal, inequal)
                errArr = mat(ones((m,1)))
                errArr[predictedVals == labelMat]=0
                weightedError = D.T *errArr
                # print "split: dim%d, thresh %.2f, thresh inequal: %s, the weighted
error is %.3f" % (i, threshVal, inequal, weightedError)
                if weightedError < minError:
                    minError = weightedError
                    bestClasEst = predictedVals.copy()
                    bestStump['dim'] = i
                    bestStump['thresh'] = threshVal
                    bestStump['ineq'] = inequal
    return bestStump,minError,bestClasEst

def adaBoostTrainDS(dataArr,classLabels,numIt=40):
    weakClassArr = []
    m = shape(dataArr)[0]
    D = mat(ones((m,1))/m) #init D to all equal
    aggClassEst = mat(zeros((m,1)))
    for i in range(numIt):
        bestStump,error,classEst = buildStump(dataArr,classLabels,D)#build
Stump
        print ("D:",D.T)
        alpha = float(0.5*log((1.0-error)/max(error,1e-16)))#calc alpha, throw
in max(error,eps) to account for error=0
        bestStump['alpha'] = alpha
        weakClassArr.append(bestStump) #store Stump Params in Array
        print ("classEst: ",classEst.T)
        expon = multiply(-1*alpha*mat(classLabels).T,classEst) #exponent for D
calc, getting messy
        D = multiply(D,exp(expon)) #Calc New D for next iteration
        D = D/D.sum()
        #calc training error of all classifiers, if this is 0 quit for loop early
(use break)

```

```

        aggClassEst += alpha*classEst
        print ("aggClassEst: ",aggClassEst.T)
        aggErrors = multiply(sign(aggClassEst) != mat(classLabels).T,
ones((m,1)))
        errorRate = aggErrors.sum()/m
        print ("total error: ",errorRate)
        if errorRate == 0.0: break
    return weakClassArr,aggClassEst

if __name__ == "__main__":
    D = mat(ones((5,1))/5)
    datMat,classLabels = loadSimpData()
    buildStump(datMat, classLabels, D)
    adaBoostTrainDS(datMat, classLabels, 10)

```

输出：

```

D: [[0.2 0.2 0.2 0.2 0.2]]
classEst: [[-1.  1. -1. -1.  1.]]
aggClassEst: [[-0.69314718  0.69314718 -0.69314718 -0.69314718  0.69314718]]
total error: 0.2
D: [[0.5  0.125 0.125 0.125 0.125]]
classEst: [[ 1.  1. -1. -1. -1.]]
aggClassEst: [[ 0.27980789  1.66610226 -1.66610226 -1.66610226 -0.27980789]]
total error: 0.2
D: [[0.28571429 0.07142857 0.07142857 0.07142857 0.5      ]]
classEst: [[1.  1.  1.  1.  1.]]
aggClassEst: [[ 1.17568763  2.56198199 -0.77022252 -0.77022252  0.61607184]]
total error: 0.0

```

8.24 异常检测

在数据挖掘中，异常检测（Anomaly Detection）是通过与大多数数据显著不同而引起怀疑的稀有项目、事件或观察的识别。通常情况下，异常项目会转化为某种问题，例如银行欺诈、结构缺陷、医疗问题或文本错误。异常也被称为异常值、新奇、噪声和偏差。

1. 异常检测

所谓异常检测，就是发现与大部分对象不同的对象，其实就是发现离群点。异常检测有时也称偏差检测。异常对象是相对罕见的。

2. 异常检测的应用

数据异常可以转化为各种应用领域中的重要（且常常是关键的）可操作信息。例如，计算机网络中的异常流量模式可能意味着被黑客窃取的计算机正在将敏感数据发送到未经授权的目的地；异常的MRI图像可能表明存在恶性肿瘤；信用卡交易数据的异常可能表明信用卡或身份被盗用；来自航天器传感器的异常读数可能意味着航天器某些部件的故障。

- 欺诈检测：主要通过检测异常行为来检测是否为盗刷他人信用卡。
- 入侵检测：检测入侵计算机系统的行为。
- 医疗领域：检测人的健康是否异常。

异常值检测常用于异常订单识别、风险客户预警、黄牛识别、贷款风险识别、欺诈检测、技术入侵等针对个体的分析场景。

3. 常见的异常检测方法

- 基于统计：基于泊松分布、正态分布找到异常分布点。
- 基于距离：K-means。
- 基于密度：KNN、LOF（Local Outlier Factor）、隔离森林。
- 一类支持向量机（One-Class SVM）。
- 隐马尔可夫模型（HMM）。

4. 新奇检测和离群检测

异常数据根据原始数据集的不同可以分为离群点检测和新奇检测。

(1) 离群点检测 (Outlier Detection)

大多数情况下定义的异常数据都属于离群点检测，对这些数据训练完之后，再在新的数据集中寻找异常点。

(2) 新奇检测

新奇检测 (Novelty Detection) 是识别新的或未知数据模式和规律的检测方法，这些规律只是在已有机器学习系统的训练集中没有被发掘出来。新奇检测的前提是已知训练数据集是“纯净”的，未被真正的“噪音”数据或真实的“离群点”污染，然后针对这些数据训练完成之后再对新的数据进行训练，以寻找新奇数据的模式。

新奇检测主要应用于新的模式、主题、趋势的探索和识别，包括信号处理、计算机视觉、模式识别、智能机器人等技术方向，应用领域有潜在疾病的探索、新物种的发现、新传播主题的获取等。

新奇检测和异常检测有关，一开始的新奇点往往都以一种离群的方式出现在数据中，这种离群方式一般会被认为是离群点，因此二者的检测和识别模式非常类似。但是，当经过一段时间之后，新奇数据一旦被证实为正常模式，例如将新的疾病识别为一种普通疾病，那么新奇模式将被合并到正常模式中，就不再属于异常点的范畴。

5. 处理高维数据异常检测

- 拓展现有的离群点检测模式。
- 发现子空间中的离群点。
- 对高维数据进行建模。

6. 以One-Class SVM算法为例进行异常检测

One-Class SVM是一种无监督的算法，可以用来检查新的样本是否符合以前的数据分布。它有三个主要参数：

(1) Kernel（核函数）和Degree

这两个变量相关，根据经验值Kernel应为rbf，Degree应为3。

(2) Gamma

这是与rbf核相关的参数。建议这个参数设置得越低越好，通常为实例数倒数和变量数倒数之间的最小值。

(3) Nu

它决定模型是必须符合一个精确的分布，还是应该尽量保持某种标准分布而不太注重适应现有数据区间为(0, 1]，默认值为0.5。但在这里0.5的结果并不好，所以可以由以下公式确定：

$$\text{nu_estimate} = 0.95 * \text{outliers_fraction} + 0.05$$

通常outliers_fraction的范围为0.02~0.1。

【例8.48】

```

from sklearn.svm import OneClassSVM
import numpy as np
import plotly.offline as py
import plotly.graph_objs as go
py.init_notebook_mode(connected=True)

# 导入数据
data = np.loadtxt('https://raw.githubusercontent.com/ffzs/dataset/master/outlier.txt', delimiter=' ')

# 分配训练集和测试集
train_set = data[:900, :]
test_set = data[-100:, :]

#### 异常检测 ####
# 创建异常检测模型
one_svm = OneClassSVM(nu=0.1, kernel='rbf', random_state=2018)
# 训练模型
one_svm.fit(train_set)
# 预测异常数据
pre_test_outliers = one_svm.predict(test_set)

### 异常结果统计###
# 合并测试检测结果
total_test_data = np.hstack((test_set, pre_test_outliers.reshape(-1,1)))

# 获取正常数据
normal_test_data = total_test_data[total_test_data[:, -1] == 1]
# 获取异常数据
outlier_test_data = total_test_data[total_test_data[:, -1] == -1]

# 输出异常数据结果
print('异常数据为: {}/{}'.format(len(outlier_test_data), len(total_test_data)))
# 异常数据为: 9/100

# 可视化结果
py.iplot([go.Scatter3d(x=total_test_data[:,0], y=total_test_data[:,1],
z=total_test_data[:,2],
mode='markers',marker=dict(color=total_test_data[:, -1],
size=5))])

```

输出:

异常数据为: 9/100

SVM异常检测分析图如图8.39所示。

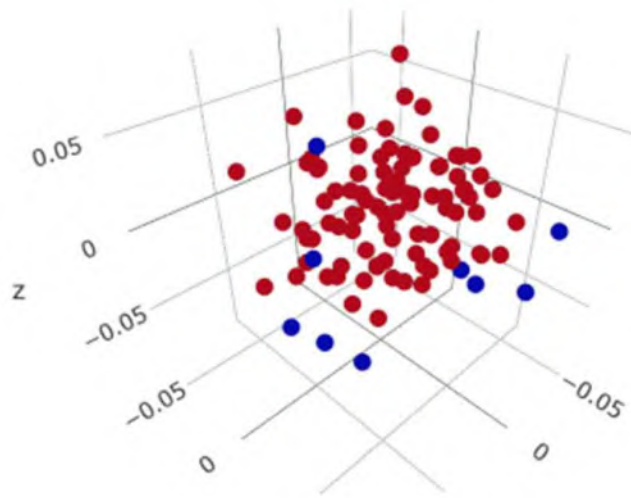


图8.39 One-Class SVM异常检测分析

8.25 文本挖掘

1. 文本挖掘的概念

文本挖掘 (Text Mining) 是抽取有效的、新颖的、有用的、可理解的、散布在文本文件中的有价值知识，并且利用这些知识更好地组织信息的过程。

机器学习算法的空间、时间复杂度依赖于输入数据的规模，维度规约 (Dimensionality Reduction) 是一种用于降低输入数据维数的方法。维度规约可以分为两类：

- 特征选择 (Feature Selection)，从原始的d维空间中选择提供信息最多的k维 (这k个维属于原始空间的子集)。
- 特征提取 (Feature Extraction)，将原始的d维空间映射到k维空间中 (新的k维空间不输入原始空间的子集)。

在文本挖掘与文本分类的有关问题中，常采用特征选择方法。原因是文本的特征一般都是单词（Term），具有语义信息，使用特征选择找出的k维子集仍然是以单词作为特征，保留了语义信息，而特征提取则是找k维新空间，将会丧失语义信息。

2. 文本的特征选择方法

对于一个语料而言，可以统计的信息包括文档频率和文档类比例，所有的特征选择方法均依赖于这两个统计量。目前，文本的特征选择方法主要有：DF、MI、IG、CHI、WLLR、WF0六种。

为了方便描述，我们首先进行一些概率上的定义：

- $p(t)$ ：一篇文档 x 包含特征词 t 的概率。
- $p(\bar{C}_i)$ ：文档 x 不属于 C_i 的概率。
- $p(C_i | t)$ ：已知文档 x 在包括某个特征词 t 的条件下，该文档属于 C_i 的概率。
- $p(\bar{t}|C_i)$ ：已知文档属于 C_i 的条件下，该文档不包括特征词 t 的概率。

其他的一些概率如 $p(C_i)$ 、 $p(\bar{t})$ 、 $p(\bar{C}_i|t)$ 等，有着类似的定义。

下面介绍其他的统计量还包括：

- A_{ij} ：包含特征词 t_i ，并且类别属于 C_j 的文档数量。
- B_{ij} ：包含特征词 t_i ，并且类别不属于 C_j 的文档数量。
- C_{ij} ：不包含特征词 t_i ，并且类别属于 C_j 的文档数量。
- D_{ij} ：不包含特征词 t_i ，并且类别不属于 C_j 的文档数量。
- $A_{ij} + B_{ij}$ ：包含特征词 t_i 的文档数量。

- $C_{ij} + D_{ij}$: 不包含特征词 t_i 的文档数量。
- $A_{ij} + C_{ij}$: C_j 类的文档数量数据。
- $B_{ij} + D_{ij}$: 非 C_j 类的文档数量数据。
- $A_{ij} + B_{ij} + C_{ij} + D_{ij} = N$: 语料中所有文档的数量。

有了这些统计量，有关概率的估算就会变得更容易，如：

$$p(t_i) = (A_{ij} + B_{ij}) / N \quad (8.91)$$

$$p(C_j) = (A_{ij} + C_{ij}) / N \quad (8.92)$$

$$p(C_j|t_i) = A_{ij} / (A_{ij} + B_{ij}) \quad (8.93)$$

3. 常见的6种特征选择方法的计算

(1) DF (Document Frequency)

DF指统计特征词出现的文档数量，用来衡量某个特征词的重要性。DF的定义如下：

$$DF = \sum_{i=1}^m A_i \quad (8.94)$$

DF的原理是：如果某些特征词在文档中经常出现，这个词就很重要。而对于在文档中出现很少（如仅在语料中出现1次有可能）的特征词，携带的信息量很少，甚至是“噪声”，这些特征词对分类器学习的影响也很小。

DF特征选择方法属于无监督的学习算法（也将其改成有监督的算法，但是大部分情况下都作为无监督的算法使用），仅考虑了频率因素而没有考虑类别因素，因此DF算法将会引入一些没有意义的词。例如中文的“的”“是”“个”等常常具有很高的DF得分，但是对分类并没有多大的意义。

(2) MI (Mutual Information)

互信息法用于衡量特征词与文档类别直接的信息量。互信息法的定义如下：

$$I(t_i, C_j) = \log \frac{p(t_i | C_j)}{p(t_i)} \approx \frac{A_{ij} N}{(A_{ij} + C_{ij})(A_{ij} + B_{ij})} \quad (8.95)$$

继续推导MI的定义公式：

$$I(t_i, C_j) = \log \frac{p(t_i | C_j)}{p(t_i)} = \log p(t_i | C_j) - \log p(t_i) \quad (8.96)$$

从上面的公式上可以看出：如果某个特征词的频率很低，互信息得分就会很高，因此互信息法倾向于“低频”的特征词。相对的，词频很高的词，得分就会变低，如果该词携带了很高的信息量，互信息法就会变得低效。

(3) IG (Information Gain)

信息增益法通过某个特征词在缺失与存在两种情况下语料中前后信息的增加来衡量某个特征词的重要性。信息增益的定义如下：

$$G(t_i) \approx \left\{ \frac{A_{ij} + B_{ij}}{N} \left[\sum_{j=1}^m \frac{A_{ij}}{A_{ij} + B_{ij}} \log \frac{A_{ij}}{A_{ij} + B_{ij}} \right] \right\} + \left\{ \frac{C_{ij} + D_{ij}}{N} \left[\sum_{j=1}^m \frac{C_{ij}}{C_{ij} + D_{ij}} \log \frac{C_{ij}}{C_{ij} + D_{ij}} \right] \right\} \quad (8.97)$$

IG与MI存在关系：

$$G(t_i) = \sum_{j=1}^m p(t_i, C_j) I(t_i, C_j) + \sum_{j=1}^m p(\bar{t}_i, C_j) I(\bar{t}_i, C_j) \quad (8.98)$$

因此，IG方式实际上就是互信息 $I(t_i, C_j)$ 与互信息 $I(\bar{t}_i, C_j)$ 加权。

(4) CHI (Chi-square, 卡方检验)

CHI特征选择算法利用了统计学中的“假设检验”的基本思想：首先假设特征词与类别直接是不相关的，利用CHI分布计算出的检验值偏离阈值越大，就越否定原假设，接受原假设的备则假设：特征词与类别有着很高的关联度。CHI的定义如下：

$$\chi(t_i, C_j) = \frac{N(A_{ij}D_{ij} - C_{ij}B_{ij})^2}{(A_{ij} + C_{ij})(B_{ij} + D_{ij})(A_{ij} + B_{ij})(C_{ij} + D_{ij})} \quad (8.99)$$

对于一个给定的语料而言，文档的总数 N 以及 C_j 类文档的数量、非 C_j 类文档的数量都是一个定值。因此，CHI的计算公式可以简化为：

$$\chi(t_i, C_j) = \frac{(A_{ij}D_{ij} - C_{ij}B_{ij})^2}{(A_{ij} + B_{ij})(C_{ij} + D_{ij})} \quad (8.100)$$

CHI特征选择方法综合考虑文档频率与类别比例两个因素。

(5) WLLR (Weighted Log Likelihood Ratio, 加权对数似然比)

WLLR特征选择方法的定义如下：

$$WLLR(t_i, C_j) = p(t_i | C_j) \log \frac{p(t_i | C_j)}{p(t_i | C_i)} \quad (8.101)$$

计算公式如下：

$$WLLR(t_i, C_j) = \frac{A_{ij}}{A_{ij} + C_{ij}} \log \frac{A_{ij}(B_{ij} + D_{ij})}{B_{ij}(A_{ij} + C_{ij})} \quad (8.102)$$

(6) WFO (Weighted Frequency and Odds, 加权频率和比值)

WFO的算法定义如下：

如果 $p(t_i|C_i) / p(t_i|\overline{C_i}) > 1$

$$WFO(t_i, C_j) = p(t_i, C_j)^\lambda \log \frac{p(t_i | C_j)^{1-\lambda}}{p(t_i | \overline{C_j})} \quad (8.103)$$

否则：

$$WFO(t_i, C_j) = 0 \quad (8.104)$$

一般来说，不同的语料文档词频与文档的类别比例起的作用应该是不一样的，WFO方法可以通过调整参数 λ 找出一个较好的特征选择依据。

4. 文本挖掘实例

【例8.49】NLTK文本分析+jieba中文文本挖掘。

NLTK的全称是Natural Language Toolkit，是一套基于Python的自然语言处理工具集。

NLTK的安装：

```
(base) C:\Users\workspace>pip install nltk
```

在NLTK中集成语料与模型等的包管理器，通过在Python解释器中执行：

```
>>> import nltk
>>> nltk.download()
showing info
https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
True
```

便会弹出如图8.40所示的包管理界面，在管理器中可以下载语料、预训练的模型等。

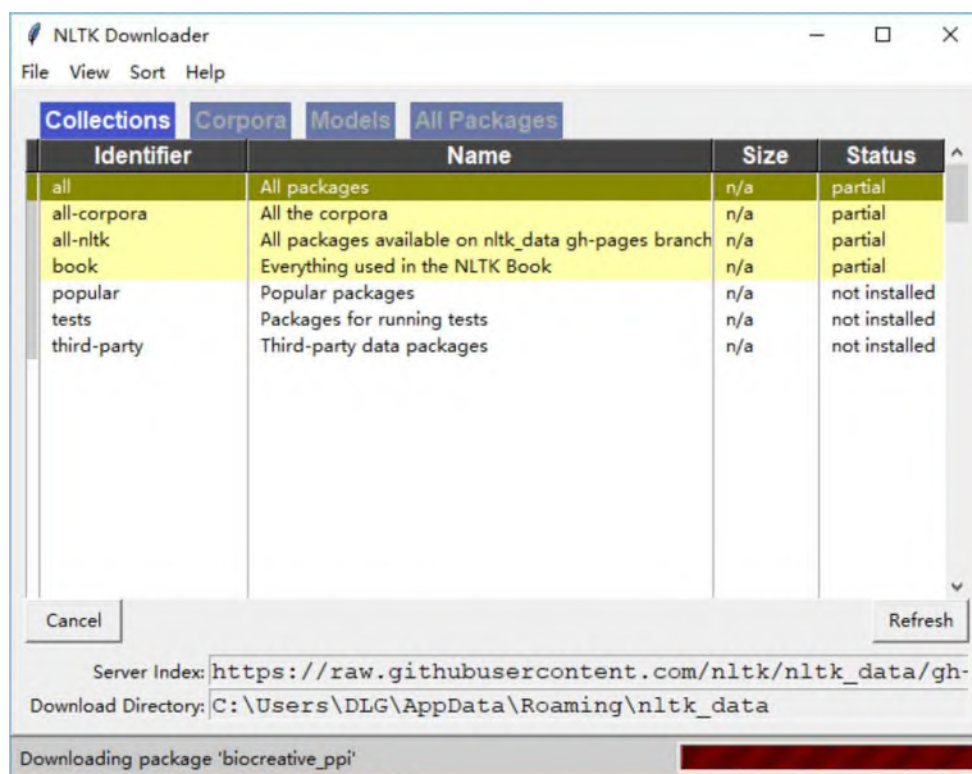


图8.40 NLTK下载

(1) NLTK自带语料库:

```
>>> from nltk.corpus import brown
>>> brown.categories()
['adventure', 'belles_lettres', 'editorial',
'fiction', 'government',
'hobbies', 'humor', 'learned', 'lore', 'mystery', 'news',
'religion', 'reviews',
'romance', 'science_fiction']
>>> len(brown.sents())
57340
>>> len(brown.words())
1161192
```

(2) Tokenize: 把句子分成一个个的小部件，例如:

```
>>> sentence="this is a good student,snoy"
>>> tokens=nltk.word_tokenize(sentence)
>>> tokens
['this', 'is', 'a', 'good', 'student', ',', 'snoy']
```

(3) nltk.text类介绍。

nltk.text.Text()类用于对文本进行初级的统计与分析，它接收一个词的列表作为参数。nltk.text.Text()类提供了下列方法：

- Text(words)：对象构造。
- concordance(word, width=79, lines=25)：显示word出现的上下文。
- common_contexts(words)：显示words出现的相同模式。
- similar(word)：显示word的相似词。
- collocations(num=20, window_size=2)：显示最常见的二词搭配。
- count(word)：word出现的词数。
- dispersion_plot(words)：绘制words文档中出现的位置图。
- vocab()：返回文章去重的词典。

nltk.text.TextCollection类是Text的集合，提供下列方法。

- nltk.text.TextCollection([text1, text2,]): 对象构造。
- idf(term)：计算词term在语料库中的逆文档频率。
- tf(term, text)：统计term在text中的词频。
- tf_idf(term, text)：计算term在句子中的tf_idf，即tf*idf。

(4) 使用结巴分词（中文分词）。

结巴分词的GitHub主页：

<https://github.com/fxsjy/jieba>

安装结巴库文件方法如下：

```
pip install jieba  
(base) C:\Users\workspace>pip install jieba
```

- jieba.cut方法接收3个输入参数：需要分词的字符串；cut_all参数用来控制是否采用全模式；HMM参数用来控制是否使用HMM模型。
- jieba.cut_for_search方法接收两个参数：需要分词的字符串；是否使用HMM模型。该方法适用于搜索引擎构建倒排索引的分词，粒度比较细。
- 待分词的字符串可以是Unicode或UTF-8字符串、GBK字符串。注意：不建议直接输入GBK字符串，可能无法预料的错误解码成UTF-8。
- jieba.cut和jieba.cut_for_search返回的结构都是一个可迭代的generator，可以使用for循环来获得分词后得到的每一个词语（Unicode），或者用jieba.lcut和jieba.lcut_for_search直接返回list。
- jieba.Tokenizer(dictionary=DEFAULT_DICT)新建自定义分词器，可用于同时使用不同的词典。jieba.dt为默认分词器，所有全局分词相关函数都是该分词器的映射。

（5）语料库介绍。在NLTK文本分析中，我们经常用到不同的语料库：

```
>>> from nltk.corpus import (gutenberg, genesis, inaugural,  
                             nps_chat, webtext, treebank, wordnet)
```

这些语料库的文本文件说明如下。

古滕堡语料库：

```
>>> from nltk.corpus import gutenberg
>>> gutenberg.fileids()
['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt',
'bible-kjv.txt', 'blake-poems.txt', 'bryant-stories.txt',
'burgess-busterbrown.txt', 'carroll-alice.txt', 'chesterton-ball.txt',
'chesterton-brown.txt', 'chesterton-thursday.txt', 'edgeworth-parents.txt',
'melville-moby_dick.txt', 'milton-paradise.txt', 'shakespeare-caesar.txt',
'shakespeare-hamlet.txt', 'shakespeare-macbeth.txt', 'whitman-leaves.txt']
```

网络和聊天文本库：

```
>>> from nltk.corpus import webtext
>>> webtext.fileids()
['firefox.txt', 'grail.txt', 'overheard.txt', 'pirates.txt', 'singles.txt',
'wine.txt']
```

即时聊天会话语料库，是美国海军研究生院的研究生收集的。

```
>>> from nltk.corpus import nps_chat
>>> nps_chat.fileids()
['10-19-20s_706posts.xml', '10-19-30s_705posts.xml',
'10-19-40s_686posts.xml', '10-19-adults_706posts.xml', '10-24-40s_706posts.xml',
'10-26-teens_706posts.xml', '11-06-adults_706posts.xml',
'11-08-20s_705posts.xml', '11-08-40s_706posts.xml', '11-08-adults_705posts.xml',
'11-08-teens_706posts.xml', '11-09-20s_706posts.xml', '11-09-40s_706posts.xml',
'11-09-adults_706posts.xml', '11-09-teens_706posts.xml']
```

布朗语料库，布朗语料库是第一个百万词级别的英语电子语料库，这个语料库包含500个不同来源的文本，按文体分类有新闻、社论等完整列表。

```
>>> from nltk.corpus import brown
>>> brown.categories()
['adventure', 'belles_lettres', 'editorial',
'fiction', 'government',
'hobbies', 'humor', 'learned', 'lore', 'mystery', 'news',
'religion', 'reviews',
'romance', 'science_fiction']
#这里注意的是它是按文体分类的，例如要取得news类别的文本
>>> print(brown.words(categories='news'))
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
#PlaintextCorpusReader类
#官方文档: nltk.corpus.reader.plaintext.PlaintextCorpusReader
```



```

#介绍：基于文件系统，用来加载纯文本的类
>>> words = gutenbergr.words('shakespeare-macbeth.txt')
>>> print(words)
[['', 'The', 'Tragedie', 'of', 'Macbeth', 'by', ...]
#标点符号也会被单独分出来
#断句，定义：sents(fileids=None)
>>> print(sents)
[['', 'The', 'Tragedie', 'of', 'Macbeth', 'by', 'William',
'Shakespeare',
'1603', ''], ['Actus', 'Primus', '.'], ...]
#分段，定义：paras(fileids=None)
paras = gutenbergr.paras('shakespeare-macbeth.txt')
print (' '.join(paras[0][0]) #first para, first sentence)
[ The Tragedie of Macbeth by William Shakespeare 1603 ]

```

NLTK数据挖掘工具的其他文本挖掘功能就不一一介绍了，在实践中常用的还有：

- 创建：定义__init__(self, tokens, name=None)。
- 查找单词：定义concordance(word, width=79, lines=25)。
- 搭配词：定义collocations(num=20, window_size=2)。
- 相似词：定义similar(word, num=20)。
- 相同上下文：定义common_contexts(words, num=20)。
- 词分布：定义dispersion_plot(words)。
- 正则表达式匹配单词：定义findall(regex)。
- 词出现次数：定义count(word)。
- 词索引：定义index(word)。
- 文本总长度：len(text)。
- 生成FreqDist类：dist = text.vocab()。
- 打印频率分布图：定义plot(*args)。

8.26 Boosting算法（提升法和Gradient Boosting）

Boosting算法是一种集成学习算法，由一系列基本分类器按照不同的权重组合成为一个强分类器，这些基本分类器之间有依赖关系。包括Adaboost算法、提升树、GBDT算法。

当Adaboost算法中的基本分类器是cart回归树时，就是提升树，同时，损失函数变为平方误差损失函数。在Adaboost算法中通过改变样本的权重来进行每一轮的基本分类器的学习，在提升树算法中，是通过上一轮学习的残差进行本轮的学习。

1. 需要解决的问题

对于Boosting算法，需要解决两个问题：

- 如何调整训练集，使得在训练集上训练的弱分类器得以进行。
- 如何将训练得到的各个弱分类器联合起来形成强分类器。

2. 算法的三个要素

函数模型（Boosting的函数模型是叠加型的），即：

$$F(x) = \sum_{i=1}^k f_i(x; \theta_i) \quad (8.105)$$

目标函数（选定某种损失函数作为优化目标）：

$$E\{F(x)\} = E\left\{\sum_{I=1}^K f_i(x; \theta_i)\right\} \quad (8.106)$$

优化算法（贪婪地逐步优化），即：

$$\theta_m^* = \arg \min_{\theta_i} E \left\{ \sum_{i=1}^{m-1} f_i(x; \theta_i^*) + f_m(x; \theta_m^*) \right\} \quad (8.107)$$

3. Gradient Boosting算法（梯度提升算法）

Gradient Boosting算法的实现：

- 函数模型为CART回归树模型。
- 损失函数一般为“对数损失函数”或“指数损失函数”。
- 优化算法采用梯度下降。
- 针对Boosting需要解决的两个问题，Gradient Boosting算法采用了以下策略：

- 将残差作为下一个弱分类器的训练数据，每个新的弱分类器的建立都是为了使之前弱分类器的残差往梯度方向减少。
- 将弱分类器联合起来，使用累加机制代替平均投票机制。

与Adaboost算法不同，Gradient Boosting算法在迭代的时候选择梯度下降的方向来保证最后的结果最好。损失函数用来描述模型的“靠谱”程度，假设模型没有过拟合，损失函数越大，模型的错误率就越高。如果模型能够让损失函数持续下降，就说明模型在不停地改进，而最好的方式就是让损失函数在其梯度方向上下降。

【例8.50】 GradientBoostingRegressor的实现。

Python中的Scikit-Learn包提供了GradientBoostingRegressor和GBDT的函数接口，可以很方便地调用函数完成模型的训练和预测。

```

# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt

from sklearn import ensemble
from sklearn import datasets
from sklearn.utils import shuffle
from sklearn.metrics import mean_squared_error

#####
# Load data
boston = datasets.load_boston()
X, y = shuffle(boston.data, boston.target, random_state=13)
X = X.astype(np.float32)
offset = int(X.shape[0] * 0.9)
X_train, y_train = X[:offset], y[:offset]
X_test, y_test = X[offset:], y[offset:]

#####
# Fit regression model
params = {'n_estimators': 500, 'max_depth': 4, 'min_samples_split': 1,
          'learning_rate': 0.01, 'loss': 'ls'}
clf = ensemble.GradientBoostingRegressor(**params)

clf.fit(X_train, y_train)
mse = mean_squared_error(y_test, clf.predict(X_test))
print("MSE: %.4f" % mse)

#####
# Plot training deviance

# compute test set deviance
test_score = np.zeros((params['n_estimators'],), dtype=np.float64)

for i, y_pred in enumerate(clf.staged_predict(X_test)):

```

```

test_score[i] = clf.loss_(y_test, y_pred)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('Deviance')
plt.plot(np.arange(params['n_estimators']) + 1, clf.train_score_, 'b-',
         label='Training Set Deviance')
plt.plot(np.arange(params['n_estimators']) + 1, test_score, 'r-',
         label='Test Set Deviance')
plt.legend(loc='upper right')
plt.xlabel('Boosting Iterations')
plt.ylabel('Deviance')

#####
# Plot feature importance
feature_importance = clf.feature_importances_
# make importances relative to max importance
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.subplot(1, 2, 2)
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, boston.feature_names[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()

```

Gradient Boosting算法模型图如图8.41所示。

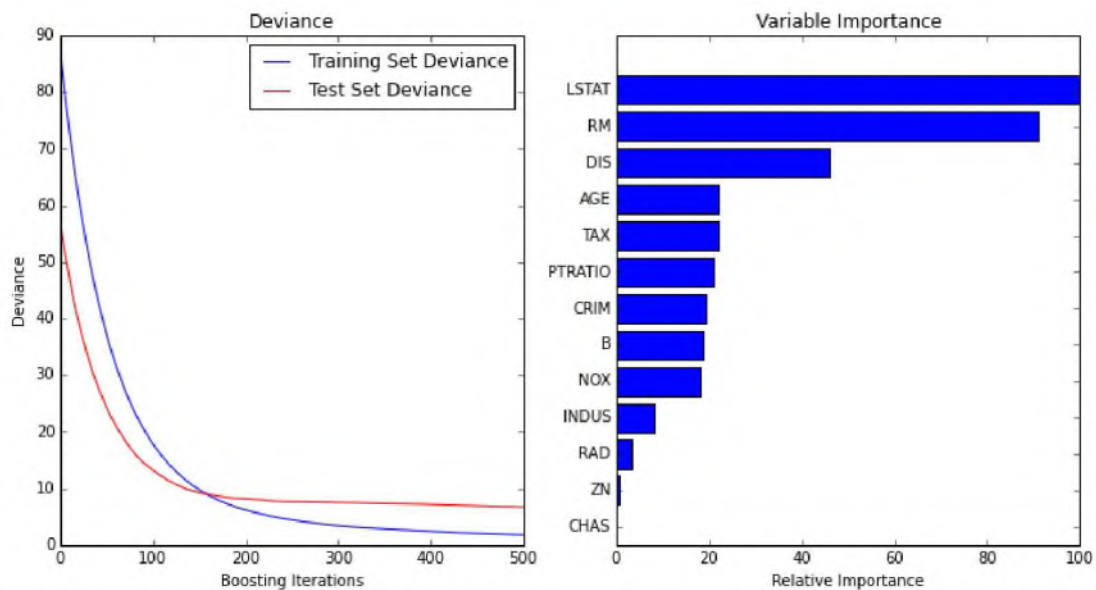


图8.41 Gradient Boosting算法

可以发现，如果要用Gradient Boosting算法，那么在sklearn包里调用还是非常方便的，几行代码即可完成，大部分的工作应该是在特征提取上。

8.27 本章小结

使用Python进行数据分析是最近几年才开始火起来的，之前网上很多的资料都是关于Python网页开发的。但使用Python进行数据分析的侧重点已经完全不一样了。

在数据分析、交互式计算以及数据可视化方面，Python将不可避免地与其他开源和商业领域的特定编程语言 / 工具进行对比，如R、Matlab、SAS、Stata等。近年来，由于Python的库（例如Pandas和Scikit-Learn）不断改良，使其成为数据分析任务的一个优选方案。结合其在通用编程方面的强大实力，我们完全可以只使用Python一种语言构建以数据为中心的应用。

本章主要从数据分析算法的角度出发，介绍如何零基础学习Python语法、数据清洗以及数据建模。同时介绍了Python数据分析、统计模型以及机器学习的各个方面，理论联系实际，内容十分充足。最后举例介绍了基于Python使用频率和效率比较典型的数据分析方法。

参考文献

- [1] Tom Mitchell (曾 华 军 , 张 银 奎 , 等 译) . Machine Learning [M] , 机械工业出版社, 北京, 2003.
- [2] 姚旭, 王晓丹, 张玉玺, 权文. 特征选择方法综述 [J] , 《控制与决策》, 2012, 27 (2) : 161-166.
- [3] Jiawei Han. 数据挖掘概念与技术 [M] , 机械工业出版社, 北京, 2012.
- [4] XU Rui, Donald Wunsch II. survey of clustering algorithm [J]. IEEE. Trans-actions on Neural Networks, 2005, 16(3): 645-678.
- [5] 贺玲, 吴玲达, 蔡益朝. 数据挖掘中的聚类算法综述 [J] . 计算机应用研究, 2007, 24 (1) : 10-13.
- [6] 孙吉贵, 刘杰, 赵连宇. 聚类算法研究 [J] . 软件学报, 2008, 19 (1) : 48-61.
- [7] 马晓艳, 唐雁. 层次聚类算法研究 [J] . 计算机科学, 2008, 34 (7) : 34-36.
- [8] Bart van Merriënboer, Dzmitry Bahdanau, Vincent Dumoulin, Dmitriy Serdyuk, David Ward e-Farley, Jan Chorowski, and Yoshua Bengio, Blocks and Fuel: Frameworks for deeplearning[J], arXiv preprint arXiv:1506.00619 [cs.LG], 2015.
- [9] Hanke, M., Halchenko, Y. O., Sederberg, P. B., Hanson, S. J., Haxby, J. V. & Pollmann, S. Py MVPA: A Python

toolbox for multivariate pattern analysis of fMRI data. Neuroinformatics[J], 2009, 7, 37–53.

- [10] Hanke, M., Halchenko, Y. O., Haxby, J. V., and Pollmann, S. Statistical learning analysis in neuroscience: aiming for transparency[J]. Frontiers in Neuroscience. 2010, 4, 1: 38–43.
- [11] Haxby, J. V., Guntupalli, J. S., Connolly, A. C., Halchenko, Y. O., Conroy, B. R., Gobbini, M. I., Hanke, M. & Ramadge, P. J.. A Common, High-Dimensional Model of the Representational Space in Human Ventral Temporal Cortex. Neuron[J], 2011(72), 404 – 416.
- [12] Bengio Y, Ducharme R, Vincent P, et al. A neural probabilistic language model[J]. journal of machine learning research, 2003, 3(Feb): 1137–1155.
- [13] Mikolov T, Kombrink S, Deoras A, et al. Rnnlm-recurrent neural network language modeling toolkit[C]//Proc. of the 2011 ASRU Workshop. 2011: 196–201.
- [14] Mikolov T, Chen K, Corrado G, et al. Efficient estimation of word representations in vector space[J]. arXiv preprint arXiv:1301.3781, 2013.
- [15] Maaten L, Hinton G. Visualizing data using t-SNE[J]. Journal of Machine Learning Research, 2008, 9(Nov): 2579–2605.

[16] https://en.wikipedia.org/wiki/Singular_value_decomposition.

- [17] https://en.wikipedia.org/wiki/Linear_regression.
- [18] Friedman J, Hastie T, Tibshirani R. The elements of statistical learning[M]. Springer, Berlin: Springer series in statistics, 2001.
- [19] Murphy K P. Machine learning: a probabilistic perspective[M]. MIT press, 2012.
- [20] Bishop C M. Pattern recognition[J]. Machine Learning, 2006, 128.
- [21] LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86, no. 11 (1998): 2278-2324.
- [22] Wejéus, Samuel. "A Neural Network Approach to Arbitrary SymbolRecognition on Modern Smartphones." (2014).
- [23] Decoste, Dennis, and Bernhard Schölkopf. "Training invariant support vector machines." Machine learning 46, no. 1-3 (2002): 161-190.
- [24] Simard, Patrice Y., David Steinkraus, and John C. Platt. "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis." In ICDAR, vol. 3, pp. 958-962. 2003.
- [25] Salakhutdinov, Ruslan, and Geoffrey E. Hinton. "Learning a Nonlinear Embedding by Preserving Class Neighbourhood Structure." In AISTATS, vol. 11. 2007.

- [26] Cireşan, Dan Claudiu, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. "Deep, big, simple neural nets for handwritten digit recognition." *Neural computation* 22, no. 12 (2010): 3207–3220.
- [27] Deng, Li, Michael L. Seltzer, Dong Yu, Alex Acero, Abdel-rahman Mohamed, and Geoffrey E. Hinton. "Binary coding of speech spectrograms using a deep auto-encoder." In *Interspeech*, pp. 1692–1695. 2010.
- [28] Kégl, Balázs, and Róbert Busa-Fekete. "Boosting products of base classifiers." In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 497–504. ACM, 2009.
- [29] Rosenblatt, Frank. "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological review* 65, no. 6 (1958): 386.
- [30] Bishop, Christopher M. "Pattern recognition." *Machine Learning* 128 (2006): 1–58.
- [31] D. G. Lowe, Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.
- [32] N. Dalal, B. Triggs, Histograms of Oriented Gradients for Human Detection, *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2005.
- [33] Ahonen, T., Hadid, A., and Pietikinen, M. (2006). Face description with local binary patterns: Application to face recognition. *PAMI*, 28.

- [34] J. Sivic, A. Zisserman, Video Google: A Text Retrieval Approach to Object Matching in Videos, Proc. Ninth Int'l Conf. Computer Vision, pp. 1470–1478, 2003.
- [35] B. Olshausen, D. Field, Sparse Coding with an Overcomplete Basis Set: A Strategy Employed by V1?, Vision Research, vol. 37, pp. 3311–3325, 1997.
- [36] Wang, J., Yang, J., Yu, K., Lv, F., Huang, T., and Gong, Y. (2010). Locality-constrained Linear Coding for image classification. In CVPR.
- [37] Perronnin, F., Sánchez, J., & Mensink, T. (2010). Improving the fisher kernel for large-scale image classification. In ECCV (4).
- [38] Lin, Y., Lv, F., Cao, L., Zhu, S., Yang, M., Cour, T., Yu, K., and Huang, T. (2011). Large-scale image classification: Fast feature extraction and SVM training. In CVPR.
- [39] Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In NIPS.
- [40] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R.R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580, 2012.
- [41] K. Chatfield, K. Simonyan, A. Vedaldi, A. Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. BMVC, 2014.

- [42] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., Going deeper with convolutions. In: CVPR. (2015)
- [43] Lin, M., Chen, Q., and Yan, S. Network in network. In Proc. ICLR, 2014.
- [44] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In ICML, 2015.
- [45] K. He, X. Zhang, S. Ren, J. Sun. Deep Residual Learning for Image Recognition. CVPR 2016.
- [46] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z. Rethinking the inception architecture for computer vision. In: CVPR. (2016).
- [47] Szegedy, C., Ioffe, S., Vanhoucke, V. Inception-v4, inception-resnet and the impact of residual connections on learning. arXiv:1602.07261 (2016).
- [48] Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J. and Zisserman, A. The Pascal Visual Object Classes Challenge: A Retrospective. International Journal of Computer Vision, 111(1), 98–136, 2015.
- [49] He, K., Zhang, X., Ren, S., and Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. ArXiv e-prints, February 2015.
- [50] <http://deeplearning.net/tutorial/lenet.html>.

- [51] <https://www.cs.toronto.edu/~kriz/cifar.html>.
- [52] <http://cs231n.github.io/classification/>.
- [53] Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.
- [54] Bengio Y, Ducharme R, Vincent P, et al. A neural probabilistic language model[J]. journal of machine learning research, 2003, 3(Feb): 1137-1155.
- [55] Mikolov T, Kombrink S, Deoras A, et al. Rnnlm-recurrent neural network language modeling toolkit[C]//Proc. of the 2011 ASRU Workshop. 2011: 196-201.
- [56] Mikolov T, Chen K, Corrado G, et al. Efficient estimation of word representations in vector space[J]. arXiv preprint arXiv:1301.3781, 2013.
- [57] Maaten L, Hinton G. Visualizing data using t-SNE[J]. Journal of Machine Learning Research, 2008, 9(Nov): 2579-2605.
- [58]
https://en.wikipedia.org/wiki/Singular_value_decomposition.
- [59] P. Resnick, N. Iacovou, etc. "GroupLens: An Open Architecture for Collaborative Filtering of Netnews", Proceedings of ACM Conference on Computer Supported Cooperative Work, CSCW 1994. pp.175-186.
- [60] Sarwar, Badrul, et al. "Item-based collaborative filtering recommendation algorithms." Proceedings of the

10th international conference on World Wide Web. ACM, 2001.

- [61] Kautz, Henry, Bart Selman, and Mehul Shah. "Referral Web: combining social networks and collaborative filtering." *Communications of the ACM* 40.3 (1997): 63-65. APA.
- [62] Peter Brusilovsky (2007). *The Adaptive Web*. p. 325.
- [63] Robin Burke, *Hybrid Web Recommender Systems*, pp. 377-408, *The Adaptive Web*, Peter Brusilovsky, Alfred Kobsa, Wolfgang Nejdl (Ed.), *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, *Lecture Notes in Computer Science*, Vol. 4321, May 2007, 978-3-540-72078-2.
- [64] Yuan, Jianbo, et al. "Solving Cold-Start Problem in Large-scale Recommendation Engines: A Deep Learning Approach." *arXiv preprint arXiv:1611.05480* (2016).
- [65] Covington P, Adams J, Sargin E. Deep neural networks for youtube recommendations[C]//*Pro-ceedings of the 10th ACM Conference on Recommender Systems*. ACM, 2016:191-198.
- [66] Kim Y. Convolutional neural networks for sentence classification[J]. *arXiv preprint arXiv:1408.5882*, 2014.
- [67] Kalchbrenner N, Grefenstette E, Blunsom P. A convolutional neural network for modelling sentences[J]. *arXiv preprint arXiv:1404.2188*, 2014.
- [68] Yann N. Dauphin, et al. Language Modeling with Gated Convolutional Networks[J] *arXiv preprint arXiv:1612.08083*, 2016.

- [69] Siegelmann H T, Sontag E D. On the computational power of neural nets[C]//Proceedings of the fifth annual workshop on Computational learning theory. ACM, 1992: 440-449.
- [70] Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural computation, 1997, 9(8):1735-1780.
- [71] Bengio Y, Simard P, Frasconi P. Learning long-term dependencies with gradient descent is difficult[J]. IEEE transactions on neural networks, 1994, 5(2): 157-166.
- [72] Graves A. Generating sequences with recurrent neural networks[J]. arXiv preprint arXiv:1308.0850, 2013.
- [73] Cho K, Van Merriënboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation[J]. arXiv preprint arXiv:1406.1078, 2014.
- [74] Zhou J, Xu W. End-to-end learning of semantic role labeling using recurrent neural networks[C]//Proceedings of the Annual Meeting of the Association for Computational Linguistics. 2015.
- [75] https://en.wikipedia.org/wiki/Linear_regression.
- [76] Friedman J, Hastie T, Tibshirani R. The elements of statistical learning[M]. Springer, Berlin: Springer series in statistics, 2001.
- [77] Murphy K P. Machine learning: a probabilistic perspective[M]. MIT press, 2012.

- [78] Bishop C M. Pattern recognition[J]. Machine Learning, 2006, 128.
- [79] Sun W, Sui Z, Wang M, et al. Chinese semantic role labeling with shallow parsing[C]//Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3-Volume 3. Association for Computational Linguistics, 2009: 1475-1483.
- [80] Pascanu R, Gulcehre C, Cho K, et al. How to construct deep recurrent neural networks[J].arXiv preprint arXiv:1312.6026, 2013.
- [81] Cho K, Van Merriënboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation[J]. arXiv preprint arXiv:1406.1078, 2014.
- [82] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate[J]. arXiv preprint arXiv:1409.0473, 2014.
- [83] Lafferty J, McCallum A, Pereira F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data[C]//Proceedings of the eighteenth international conference on machine learning, ICML. 2001, 1: 282-289.
- [84] 李航. 统计学习方法 [J] . 清华大学出版社, 北京, 2012.
- [85] Marcus M P, Marcinkiewicz M A, Santorini B. Building a large annotated corpus of English: The Penn Treebank [J]. Computational linguistics, 1993, 19(2): 313-330.

- [86] Palmer M, Gildea D, Kingsbury P. The proposition bank: An annotated corpus of semantic roles [J]. Computational linguistics, 2005, 31(1): 71-106.
- [87] Carreras X, Màrquez L. Introduction to the CoNLL-2005 shared task: Semantic role labeling[C]//Proceedings of the Ninth Conference on Computational Natural Language Learning. Association for Computational Linguistics, 2005: 152-164.
- [88] Zhou J, Xu W. End-to-end learning of semantic role labeling using recurrent neural networks[C]//Proceedings of the Annual Meeting of the Association for Computational Linguistics. 2015.
- [89] Koehn P. Statistical machine translation[M]. Cambridge University Press, 2009.
- [90] Cho K, Van Merriënboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation[C]//Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014: 1724-1734.
- [91] Chung J, Gulcehre C, Cho K H, et al. Empirical evaluation of gated recurrent neural networks on sequence modeling [J]. arXiv preprint arXiv:1412.3555, 2014.
- [92] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate[C]//Proceedings of ICLR 2015, 2015.

- [93] Papineni K, Roukos S, Ward T, et al. BLEU: a method for automatic evaluation of machine translation[C]//Proceedings of the 40th annual meeting on association for computational linguistics. Association for Computational Linguistics, 2002: 311-318.
- [94] alimans, Tim; Goodfellow, Ian; Zaremba, Wojciech; Cheung, Vicki; Radford, Alec; Chen, Xi. Improved Techniques for Training GANs. 2016. arXiv:1606.03498 [cs.LG].
- [95] Radford A, Metz L, Chintala S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks[J]. Computer Science, 2015.

大数据与人工智能技术丛书



Python 机器学习

——数据分析与评分卡建模 微课版

◎ 翟锟 胡锋 周晓然 编著

本书特色

- 零基础入门，注重实战
10个学习实例，3个完整的项目案例
- 视频教学，全程语音讲解
270分钟高品质配套教学视频
- 教学资源丰富
提供教学课件、源代码、数据集

270分钟
微课视频

清华大学出版社

大数据与人工智能技术丛书

Python 机器学习

——数据分析与评分卡建模

(微课版)

翟锐 胡锋 周晓然 编著

清华大学出版社
北 京

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989
13701121933

图书在版编目（CIP）数据

Python机器学习：数据分析与评分卡建模：微课版 / 翟锟，胡
锋，周晓然编著．—北京：清华大学出版社，2019

（大数据与人工智能技术丛书）

ISBN 978-7-302-51684-2

I . ①P… II . ①翟…②胡…③周… III . ①软件工具-程序设计 IV . ①TP311.561

中国版本图书馆CIP数据核字（2018）第264264号

责任编辑：黄 芝

封面设计：刘 键

责任校对：时翠兰

责任印制：沈 露

出版发行：清华大学出版社

网 址：<http://www.tup.com.cn>，<http://www.wqbook.com>

地 址：北京清华大学学研大厦A座

邮 编：100084

社 总 机：010-62770175

邮 购： 010-62786544

投稿与读者服务： 010-62776969, c-service@tup.tsinghua.edu.cn

质量反馈： 010-62772015, zhiliang@tup.tsinghua.edu.cn

课件下载： <http://www.tup.com.cn>, 010-62795954

印 刷 者： 北京富博印刷有限公司

装 订 者： 北京市密云县京文制本装订厂

经 销： 全国新华书店

开 本： 185mm×260mm

印 张： 11.75

字 数： 200千字

版 次： 2019年5月第1版

印 次： 2019年5月第1次印刷

印 数： 1~2000

产品编号： 080057-01

内容简介

本书在Python数据分析与建模方面，既是一本入门书，也是一本提高书，它提炼总结了作者从Python小白到Python建模工程师的历程。如果读者有志于数据分析、建模领域，那么它一定会带给读者惊喜。书中代码具有很高的可移植性，可供读者直接使用。

全书共分为8章，从Python的环境搭建到基本语法结构，从趣味应用到分析与建模，最后以社交网络分析结束。

本书附有教学视频、源代码、课件等配套资源，适用于银行业或互联网金融行业中的风控人员，金融行业中的数据分析师（或想转行数据分析师的学习者），以及正在学习机器学习的从业人员。

前言

Python自20世纪90年代初诞生以来，已逐渐被越来越多的开发者所接受甚至着迷。当然，这与其本身的简洁性、易读性以及可拓展性密不可分。Python具有丰富而强大的库，它能够很轻松地与其他语言的各种模块相结合。如果读者第一次见到用Python编写的语句，估计会被其整齐划一的设计风格所触动，Python的最大优点是易读、易维护。

本书以“零基础”为出发点，直接从实际应用案例入手，在内容方面更注重实战性。希望读者能在学习中不断思考，学以致用。

在本书的编写过程中，结合工作中的一些具体案例，整理成书。由于作者水平有限，疏漏在所难免，读者如发现问题，欢迎您及时指正。

特别声明，本书非系统性学习资料，内容的准备上有点跳跃，学习某些内容时还需要具备一些相关的基础知识，Google和百度会是阅读过程中的常伴。

为便于教学，本书有教学视频、源代码、课件等配套资源，请扫描二维码获取。



感谢本书的另外两名作者胡锋、周晓然，他们为此书同样牺牲了很多个人时间；感谢我的同事，他们在本书的写作过程中提供了很多的帮助；最后，感谢我的父母和妻子，把家庭生活安排得井井有条，让我能无后顾之忧，安心地编写此书！

希望这本书能够为正在学习或想要学习Python的读者提供帮助。

翟 锴

2019年1月

目 录

[前言](#)

[第1章 Python开发环境搭建](#)

- [1.1 利器1: Notepad编辑器](#)
- [1.2 利器2: Anaconda](#)
- [1.3 利器3: Miniconda](#)
- [1.4 利器4: PyCharm IDE工具](#)
- [1.5 利器5: Spyder](#)
- [1.6 利器6: Jupyter Notebook](#)
- [1.7 小结](#)

[第2章 Python数据类型用法讲解](#)

- [2.1 变量](#)
- [2.2 字符串](#)
- [2.3 列表list](#)
 - [2.3.1 增 \(append、insert、extend\)](#)
 - [2.3.2 删 \(pop、remove、del\)](#)
 - [2.3.3 改、查](#)
 - [2.3.4 列表的循环遍历](#)
 - [2.3.5 排序 \(sort, reverse\)](#)
 - [2.3.6 列表的其他操作符](#)
- [2.4 集合set](#)
 - [2.4.1 创建集合](#)
 - [2.4.2 集合的增、删](#)
 - [2.4.3 集合的交、并、补等操作](#)
- [2.5 字典dictionary](#)
 - [2.5.1 字典的查找操作](#)
 - [2.5.2 字典的增、改操作](#)
 - [2.5.3 字典的删操作](#)

[2.5.4 字典的常用方法](#)

[2.5.5 有序字典](#)

[2.6 函数](#)

[2.7 小结](#)

[第3章 Python下的实际应用](#)

[3.1 Python连接MySQL数据库](#)

[3.2 Python连接MongoDB数据库](#)

[3.3 结巴分词和词云图](#)

[3.4 简单社交网络](#)

[3.5 JSON解析](#)

[3.6 OCR文字识别](#)

[3.7 pyecharts](#)

[3.8 stats简单统计分析](#)

[3.9 小结](#)

[第4章 异常样本识别](#)

[4.1 逻辑回归、交叉验证与欠采样](#)

[4.2 基于分布的异常样本识别](#)

[4.3 小结](#)

[第5章 自然语言处理案例——电商评论](#)

[5.1 数据加载与预处理](#)

[5.2 数据可视化](#)

[5.3 文本分析](#)

[5.4 情感分析](#)

[5.5 文本分类](#)

[5.6 小结](#)

[第6章 模型融合](#)

[6.1 分类模型的融合方法](#)

[6.2 回归模型的融合方法](#)

[6.3 小结](#)

[第7章 创建金融申请评分卡](#)

[7.1 变量选择](#)

[7.2 各变量按照 \$\ln\(\text{odds}\)\$ 进行分箱](#)

[7.3 计算WOE与IV值](#)

[7.4 逻辑回归建模](#)

[7.5 创建评分卡](#)

[7.6 申请评分卡的评价、使用与监控](#)

[7.7 小结](#)

[第8章 社交网络分析禹反欺诈](#)

[8.1 Neo4j的下载与安装](#)

[8.2 图形界面介绍](#)

[8.3 Cypher语言](#)

[8.4 Neo4j案例1——《天龙八部》的人物关系分析](#)

[8.5 Neo4j案例2——金融场景中的社交网络分析](#)

[8.6 Py2neo](#)

[8.7 小结](#)

[参考文献](#)

[附录A PyCharm安装步骤](#)

[附录B MyCQL安装步骤](#)

[附录C MongoDB安装步骤](#)

[附录D Neo4j安装步骤](#)

[附录E jdk安装步骤](#)

[附录F 第三方包安装步骤](#)

[返回总目录](#)

第1章 Python开发环境搭建

Python是一种面向对象的解释型计算机程序设计语言，于1989年由荷兰人Guido van Rossum发明。1991年，第一个公开的Python版正式发行。

近年来，随着机器学习、深度学习的快速发展，Python的受欢迎程度越来越高，具体的排名顺序可以查看TIOBE官网：<https://www.tiobe.com/tiobe-index/>。2018年7月Python的排名是第4名，如图1-1所示。

Apr 2018	Apr 2017	Change	Programming Language	Ratings	Change
1	1		Java	15.777%	+0.21%
2	2		C	13.589%	+6.62%
3	3		C++	7.218%	+2.66%
4	5	^	Python	5.803%	+2.35%
5	4	v	C#	5.265%	+1.69%
6	7	^	Visual Basic .NET	4.947%	+1.70%
7	6	v	PHP	4.218%	+0.84%

图1-1 TIOBE 2018年7月编程语言排行榜

2018年3月，Python作者在邮件列表上宣布将于2020年1月1日终止支持Python 2.7版本。同时，由于越来越多的第三方库都不再维护Python 2版本，Python 3是大势所趋。本书主要讲述Python 3的用法，有关Python 2的用法部分，会做特殊说明。

工欲善其事，必先利其器。在实际开发Python代码的过程中，我们常用的“利器”有哪些呢？

本章只是简单地对Windows系统上相关Python开发软件的安装过程进行讲解，而Mac OS系统上的安装过程与之类似。Linux系统上的安装，可以使用Google或者百度来查找相关的安装教程。在Linux系统上

安装相关软件时，可能会遇到提示的错误信息情况，可在网上搜出相应的解决办法。

1.1 利器1：Notepad编辑器

Notepad（记事本）是代码编辑器或Windows中的小程序，用于文本编辑，是一款开源、小巧、免费的纯文本编辑器，具有运行便携、体积小、资源占用小，以及支持众多程序语言等优点。Notepad支持的语言有C++、C#、Java等主流程程序语言，HTML、XML、Python、JavaScript等网页 / 脚本语言。Notepad内置支持多达27种语法高亮度显示，还可实现语法折叠、宏等常用功能。Notepad的官网链接地址为：<https://notepad-plus-plus.org/>，对应的界面如图1-2所示。



图1-2 Notepad官网首页界面图

安装好Notepad之后，其操作界面如图1-3所示。

Notepad的优点主要包括以下5个方面。

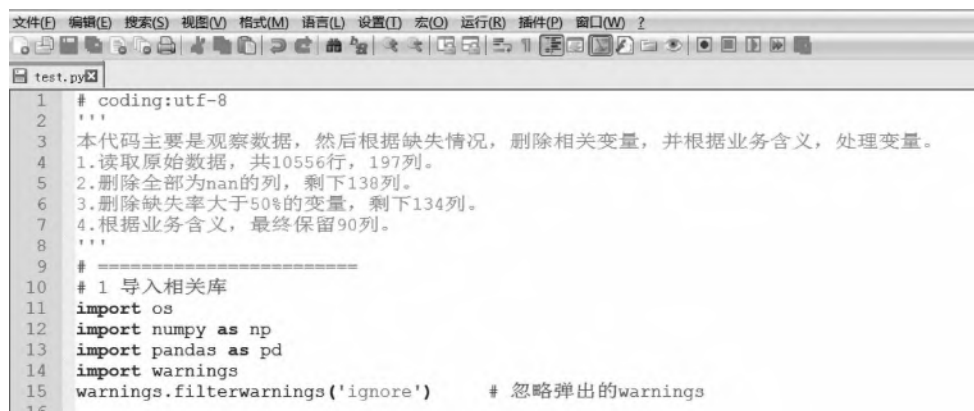


图1-3 Notepad界面图

(1) 语法高亮，具有字词自动完成功能，支持同时编辑多重文档；支持自定义语言。

(2) 自动检测文件类型，根据关键字显示节点，节点可自由折叠或打开，还可显示缩进引导线，使代码富有层次感。

(3) 在分窗口中又可打开多个子窗口，可以使用F11键快捷切换至全屏显示模式，支持鼠标滚轮改变文档显示比例。

(4) 可显示选中文本的字节数，并非普通编辑器所显示的字数；提供了一些实用工具，如邻行互换位置、宏功能等。

(5) 能够进行列块编辑，可快速地插入或者删除文本，能够成批替换文本。

1.2 利器2：Anaconda

Anaconda可以简单地理解为一个工具箱，其中包含了conda、flask、nltk、pandas、pip等180多个科学包及其依赖项，可以方便地实现包的安装、更新和卸载，比如机器学习包scikit-learn、词云包wordcloud等。Anaconda最大的好处在于它集成了Jupyter Notebook和

Spyder，这两个工具可以快速地让我们看到代码的运行结果，以便进行调试。

Anaconda官网链接地址为：<https://www.anaconda.com>，官网首页如图1-4所示。Anaconda下载链接地址为：<https://www.anaconda.com/download/>和<https://repo.continuum.io/archive/>，如图1-5和图1-6所示。



图1-4 Anaconda官网首页界面图



图1-5 Anaconda下载界面图（1）

Anaconda installer archive				
Filename	Size	Last Modified	MD5	
Anaconda2-5.2.0-Linux-ppc64le.sh	269.6M	2018-05-30 13:04:31	479633a95906ea6d41056ebe84a4c47b	
Anaconda2-5.2.0-Linux-x86.sh	488.7M	2018-05-30 13:05:30	758e172a824f467ea6b55d3d076c132f	
Anaconda2-5.2.0-Linux-x86_64.sh	603.4M	2018-05-30 13:04:33	5c034a4ab36ec9b6ae01fa13d8a04462	
Anaconda2-5.2.0-MacOSX-x86_64.pkg	616.8M	2018-05-30 13:05:32	2836c839d29be8d9569a715f4c631a3b	
Anaconda2-5.2.0-MacOSX-x86_64.sh	527.1M	2018-05-30 13:05:34	b1f3fef58955830b65613a4a8d75c3cf	
Anaconda2-5.2.0-Windows-x86.exe	443.4M	2018-05-30 13:04:17	4a3729b14c2d3fced3a050821679c702	
Anaconda2-5.2.0-Windows-x86_64.exe	564.0M	2018-05-30 13:04:16	595e427e4b625b6eab92623a28dc4e21	
Anaconda3-5.2.0-Linux-ppc64le.sh	288.3M	2018-05-30 13:05:40	cbd1d5435ead2b0b97dba5b3cf45d694	
Anaconda3-5.2.0-Linux-x86.sh	507.3M	2018-05-30 13:05:46	81d5a1648e3aca4843f88ca3769c0830	
Anaconda3-5.2.0-Linux-x86_64.sh	621.6M	2018-05-30 13:05:43	3e58f494ab9fba12db4460dc152377b5	
Anaconda3-5.2.0-MacOSX-x86_64.pkg	613.1M	2018-05-30 13:07:00	9c35bf27e9986701f7d80241616c665f	
Anaconda3-5.2.0-MacOSX-x86_64.sh	523.3M	2018-05-30 13:07:03	b5b789c01e1992de55ee911754c310d4	
Anaconda3-5.2.0-Windows-x86.exe	506.3M	2018-05-30 13:04:19	285387e7b6ea81edba98c011922e235a	
Anaconda3-5.2.0-Windows-x86_64.exe	631.3M	2018-05-30 13:04:18	62244c0382b8142743622fdc3526eda7	
Anaconda2-5.1.0-Linux-ppc64le.sh	267.3M	2018-02-15 09:08:49	e894dcc547a1c7d67deb04f6bba7223a	
Anaconda2-5.1.0-Linux-x86.sh	431.3M	2018-02-15 09:08:51	e26fb9d3e53049f6e32212270af6b987	
Anaconda2-5.1.0-Linux-x86_64.sh	533.0M	2018-02-15 09:08:50	5b1b5784cae93cf696e11e66983d8756	
Anaconda2-5.1.0-MacOSX-x86_64.pkg	588.0M	2018-02-15 09:08:52	4f9c197dfe6d3dc7e50a8611b4d3cfa2	
Anaconda2-5.1.0-MacOSX-x86_64.sh	505.9M	2018-02-15 09:08:53	e9845ccf67542523c5be09552311666e	
Anaconda2-5.1.0-Windows-x86.exe	419.8M	2018-02-15 09:08:55	a09347a53e04a15ee965300c2b95dfde	

图1-6 Anaconda下载界面图（2）

本节以Anaconda3 5.0.0为例，进行安装说明。

- （1）选中安装包，右击，再单击“以管理员身份运行”。
- （2）单击Next按钮，如图1-7所示。

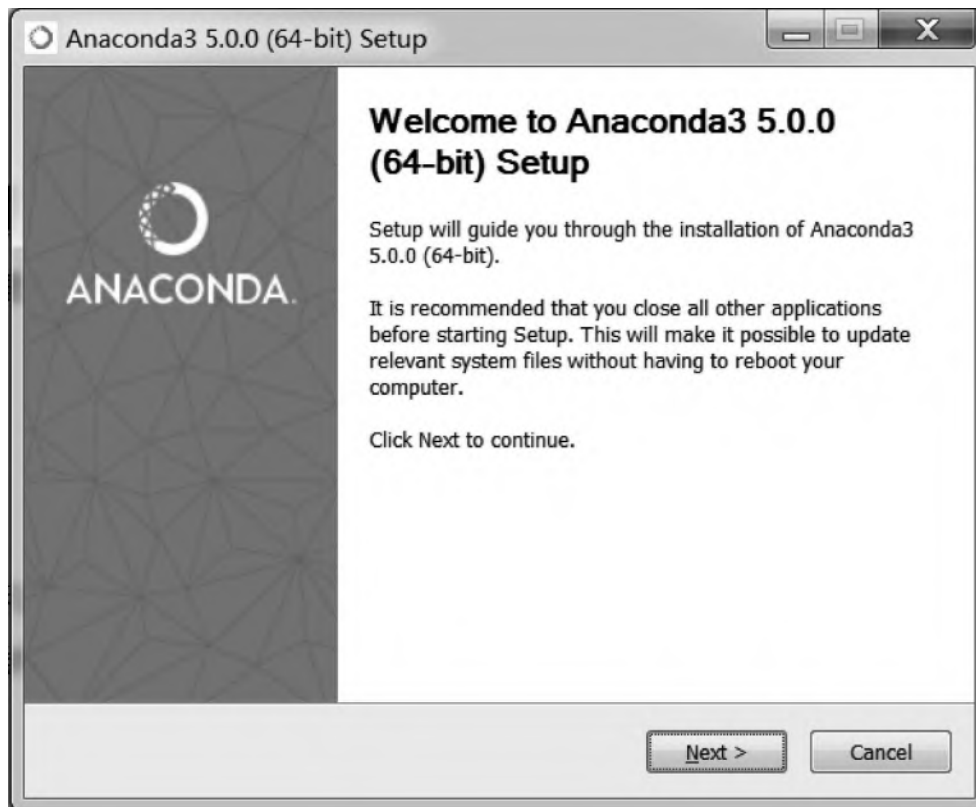


图1-7 Anaconda3 5.0.0安装界面图（1）

（3）单击I Agree按钮，如图1-8所示。

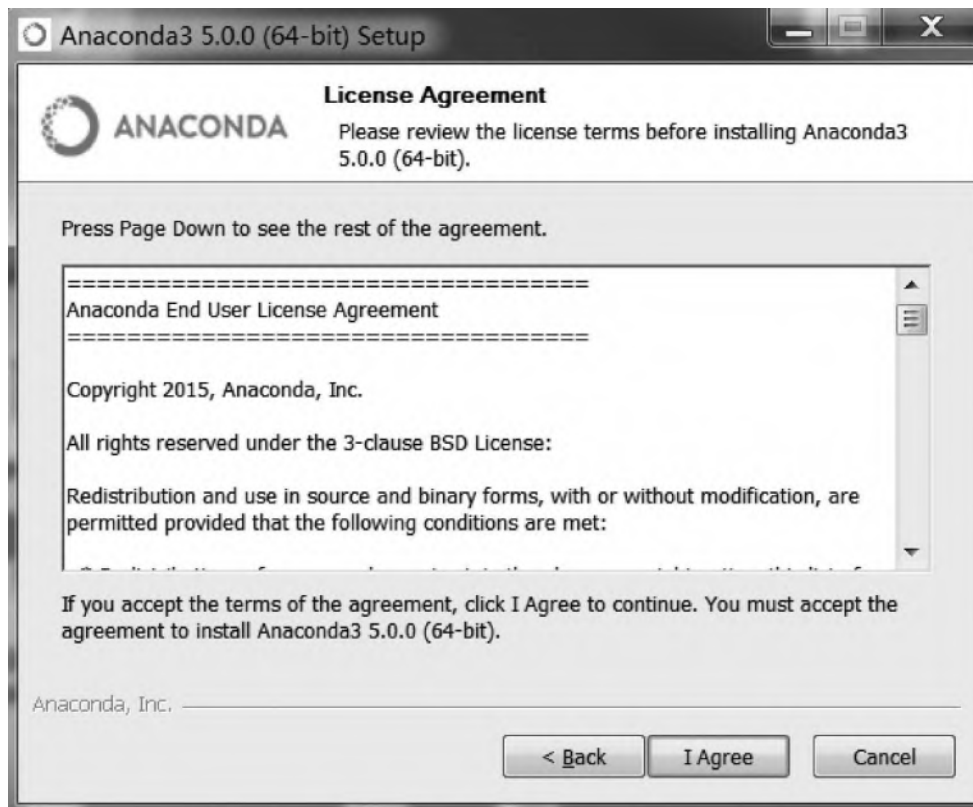


图1-8 Anaconda3 5.0.0安装界面图（2）

（4）勾选All Users（requires admin privileges），单击Next按钮，如图1-9所示。

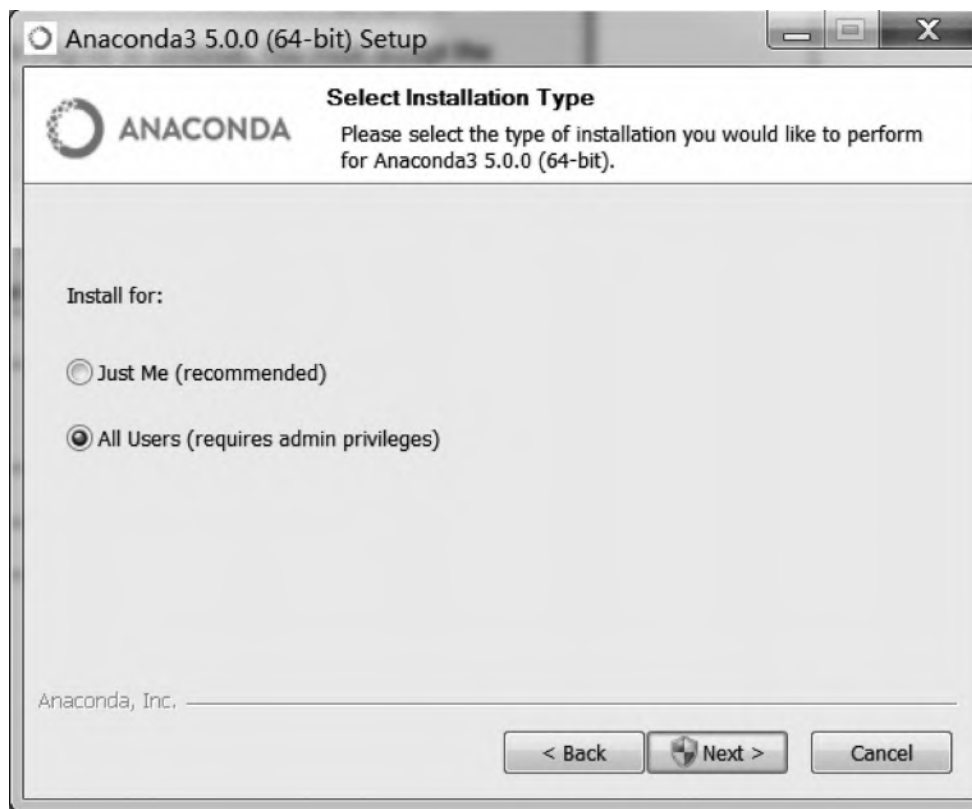


图1-9 Anaconda3 5.0.0安装界面图（3）

（5）程序默认安装位置为C:\ProgramData\Anaconda3，单击Browse可选择自定义安装目录，本安装教程的自定义安装路径为D:\DevTools\Anaconda3，单击Next按钮，如图1-10所示。

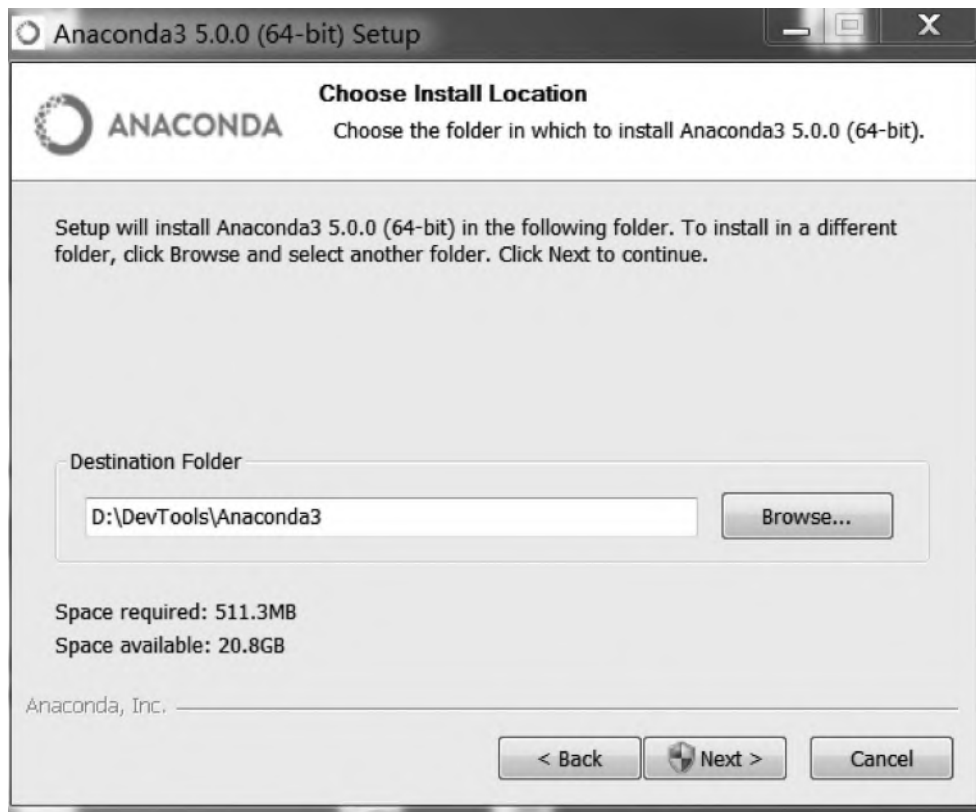


图1-10 Anaconda3 5.0.0安装界面图（4）

（6）同时勾选Add Anaconda to the system PATH environment variable和Register Anaconda as the system Python 3.6，单击Install按钮，如图1-11所示。

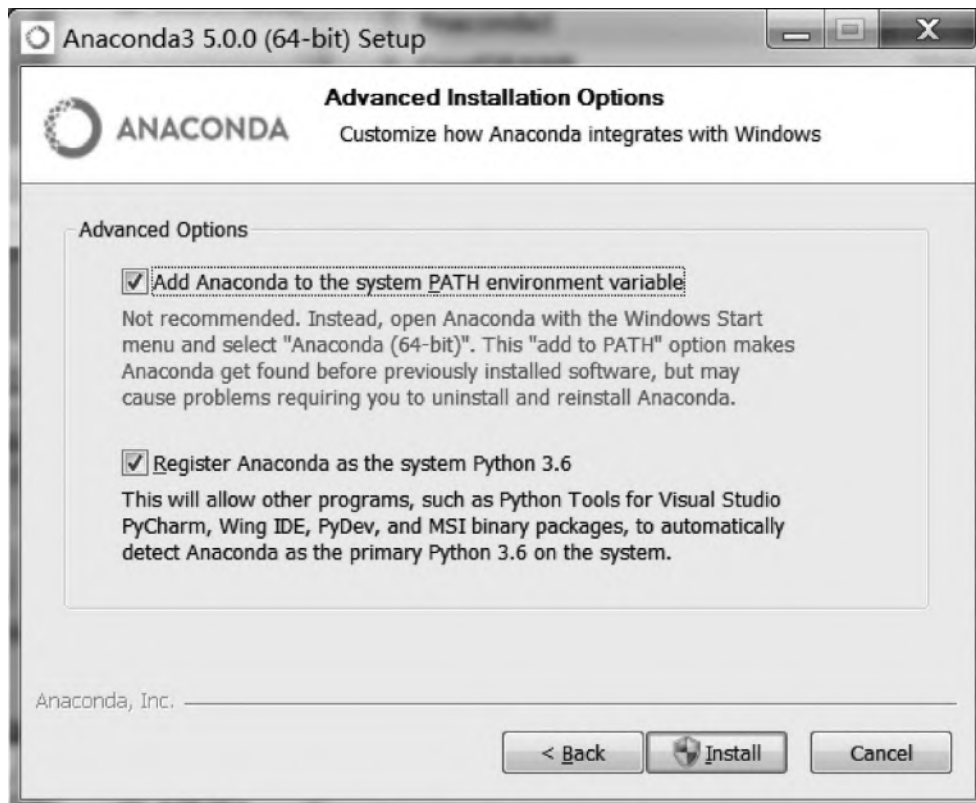


图1-11 Anaconda3 5.0.0安装界面图（5）

（7）安装过程需要4~5分钟，单击Next按钮，如图1-12所示。

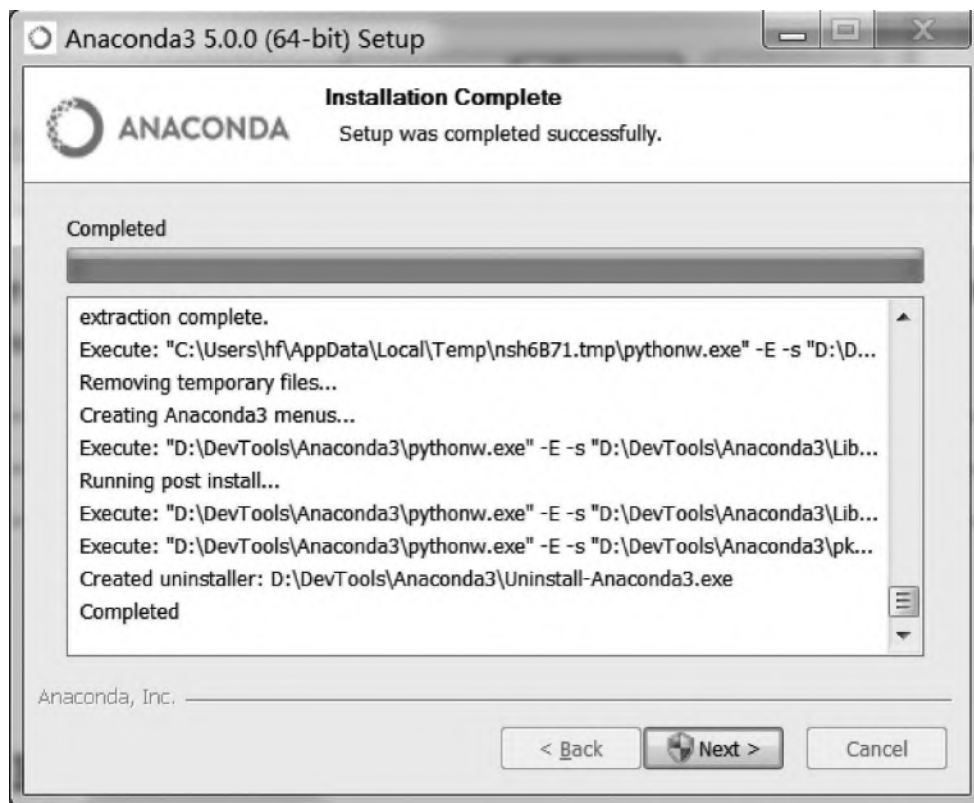


图1-12 Anaconda3 5.0.0安装界面图（6）

（8）取消勾选Learn more about Anaconda Cloud和Learn more about Anaconda Support，然后单击Finish按钮，至此完成了Anaconda的安装，如图1-13所示。

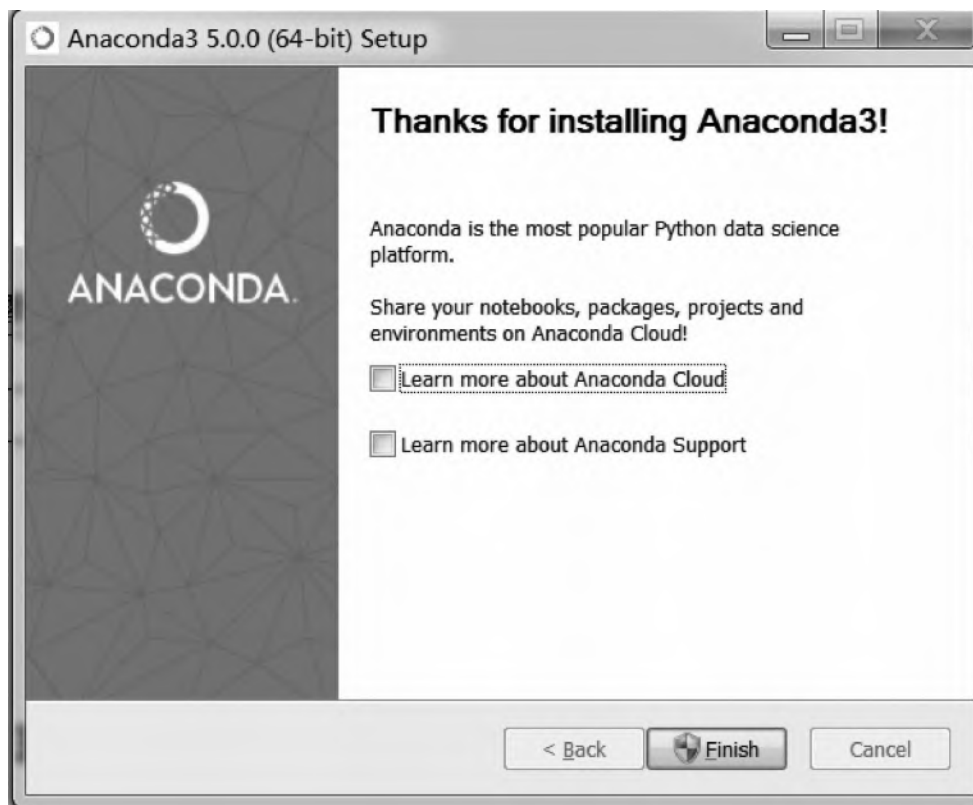


图1-13 Anaconda3 5.0.0安装界面图（7）

1.3 利器3：Miniconda

Miniconda相当于迷你版的Anaconda，其功能比Anaconda稍微少一些。如果计算机存储空间比较小的话，可以安装Miniconda，而不必安装Anaconda。

Miniconda 官网 链接 地址 为：
<https://conda.io/miniconda.html>，如图1-14所示。Miniconda下载
链接地址为：<https://repo.continuum.io/miniconda/>，如图1-15所
示。



图1-14 Miniconda官网首页界面图

Miniconda installer archive				
Filename	Size	Last Modified		MD5
Miniconda2-4.5.4-Linux-ppc64le.sh	36.9M	2018-06-06	23:07:18	3e26ee6447c8025609eae1e410f768417
Miniconda2-4.5.4-Linux-x86.sh	35.5M	2018-06-06	22:27:33	a638ae058a0ce15c5b289d151c488045
Miniconda2-4.5.4-Linux-x86_64.sh	38.1M	2018-06-06	22:24:38	8a1c02f6941d8778f8afad7328265cf5
Miniconda2-4.5.4-MacOSX-x86_64.pkg	34.5M	2018-06-06	23:12:27	6040ee82686b36f9bffa32d5f2a1341b
Miniconda2-4.5.4-MacOSX-x86_64.sh	29.8M	2018-06-06	23:12:26	35f4ca99d33ed56f68745eeaf1449274
Miniconda2-4.5.4-Windows-x86.exe	51.8M	2018-06-07	00:09:59	55502ce28ba3a16aa12954522d3624d2
Miniconda2-4.5.4-Windows-x86_64.exe	55.9M	2018-06-06	23:52:04	d285b7451deb4a33037033307c153684
Miniconda2-latest-Linux-ppc64le.sh	36.9M	2018-06-06	23:07:18	3e26ee6447c8025609eae1e410f768417
Miniconda2-latest-Linux-x86.sh	35.5M	2018-06-06	22:27:33	a638ae058a0ce15c5b289d151c488045
Miniconda2-latest-Linux-x86_64.sh	38.1M	2018-06-06	22:24:38	8a1c02f6941d8778f8afad7328265cf5
Miniconda2-latest-MacOSX-x86_64.pkg	34.5M	2018-06-06	23:12:27	6040ee82686b36f9bffa32d5f2a1341b
Miniconda2-latest-MacOSX-x86_64.sh	29.8M	2018-06-06	23:12:26	35f4ca99d33ed56f68745eeaf1449274
Miniconda2-latest-Windows-x86.exe	51.8M	2018-06-07	00:09:59	55502ce28ba3a16aa12954522d3624d2
Miniconda2-latest-Windows-x86_64.exe	55.9M	2018-06-06	23:52:04	d285b7451deb4a33037033307c153684
Miniconda3-4.5.4-Linux-ppc64le.sh	54.9M	2018-06-06	23:07:24	05c1e073f262105179cf57920dfc4d43
Miniconda3-4.5.4-Linux-x86.sh	53.7M	2018-06-06	22:27:35	0fcc79d640d82b7d36ea39654a82dd9d
Miniconda3-4.5.4-Linux-x86_64.sh	55.8M	2018-06-06	22:24:39	a946ea1d0c4a642dd0c3a26a18bb16d
Miniconda3-4.5.4-MacOSX-x86_64.pkg	40.2M	2018-06-06	23:12:28	242882072fda2ada8551239e9041f5e9
Miniconda3-4.5.4-MacOSX-x86_64.sh	34.9M	2018-06-06	23:12:26	164ec263c4070db642ce31bb45d68813
Miniconda3-4.5.4-Windows-x86.exe	51.1M	2018-06-07	00:10:06	bc2f687a9e92455a099242929df8471d
Miniconda3-4.5.4-Windows-x86_64.exe	54.8M	2018-06-06	23:52:12	1c73051ccd997770288275ee6474b423
Miniconda3-latest-Linux-ppc64le.sh	54.9M	2018-06-06	23:07:24	05c1e073f262105179cf57920dfc4d43
Miniconda3-latest-Linux-x86.sh	53.7M	2018-06-06	22:27:35	0fcc79d640d82b7d36ea39654a82dd9d

图1-15 Miniconda下载界面图

Miniconda的安装步骤与Anaconda的安装步骤类似，可参照1.2节中Anaconda的安装步骤，这里不再赘述。

1.4 利器4：PyCharm IDE工具

学过Java、Go、Web开发的，可能都知道JetBrains公司，该公司的主要产品有针对Java语言开发的IntelliJ IDEA，针对Go语言开发的GoLand，针对Web开发的WebStorm，以及针对Python语言开发的IDE工具——PyCharm。

PyCharm是一款Python IDE，可以显著提高Python的开发效率，比如可以提高调试、语法高亮、Project管理、代码跳转、智能提示、单元测试、版本控制等开发速率。在编写代码的过程中，PyCharm可快速实现错误高亮，智能检测以及一键式代码快速补全建议，使得编码更加优化。

PyCharm官网链接地址为：<http://www.jetbrains.com/pycharm>，如图 1-16 所示。下载链接地址为：<http://www.jetbrains.com/pycharm/download/previous.html>，如图 1-17所示。

其具体的安装步骤，详见附录A。



图1-16 PyCharm官网首页界面图

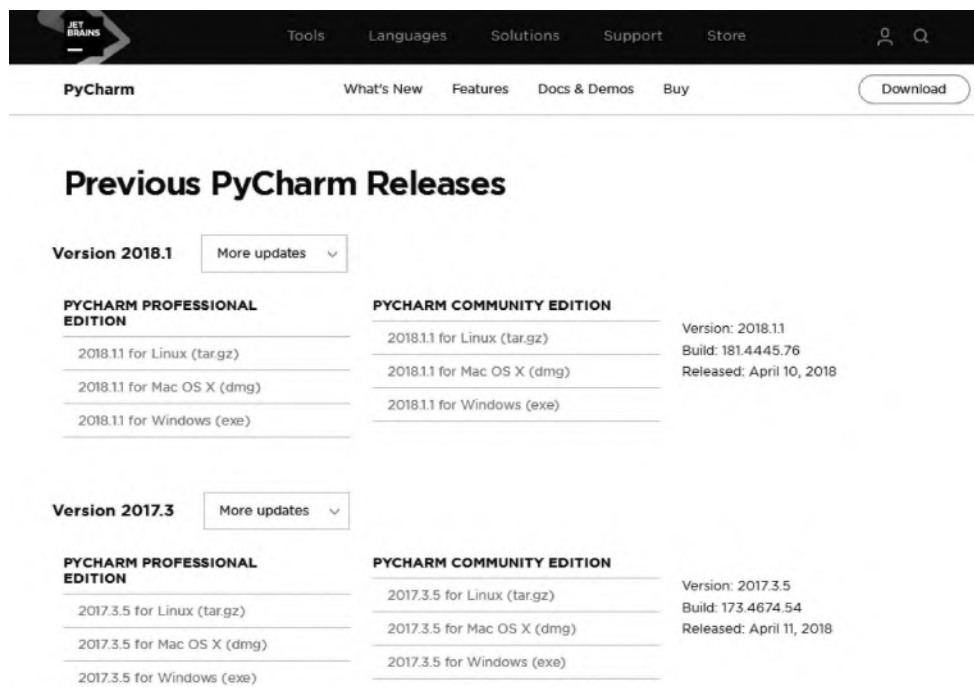


图1-17 PyCharm下载界面图

1.5 利器5: Spyder

Spyder是一个简单的集成开发环境，与PyCharm相比，具有更轻量级的特点，可以快速地查看代码运行的结果。Spyder的界面是由许多窗格构成，用户可以根据自己的喜好调整它们的位置和大小。当多个窗格出现在一个区域时，将以标签页的形式显示。

Anaconda安装包默认集成了Spyder，安装好Anaconda后，就可以在Anaconda文件夹下查看到Spyder。Spyder的界面如图1-18所示，最左边是编写的代码，在右上角可以查看定义的变量，在右下角可以进行代码输出，看看代码的具体运行结果。



图1-18 Spyder界面图

1.6 利器6: Jupyter Notebook

Jupyter Notebook（此前被称为IPython notebook）是一个交互式笔记本，支持运行40多种编程语言。其本质是一个Web应用程序，便于创建和共享程序文档，支持实时代码，数学方程，可视化和Markdown。

在编写Python代码的过程中，可以进行代码提示和结果显示。在机器学习中，Jupyter Notebook常用于数据清洗、统计建模等。本书的代码主要是在Jupyter Notebook中进行编写和展现。

Anaconda安装包默认集成了Jupyter Notebook，安装好Anaconda后，就可以查看到Jupyter Notebook。单击New按钮，选择Python 3，就可以创建一个后缀为.ipynb的Jupyter Notebook文件，具体界面如图1-19所示。



图1-19 创建Jupyter Notebook文件界面图

双击Untitled，可以修改文件名称，本例中修改为test，输入以下代码之后，按住Shift+Enter键，就可以快速地查看输出结果并跳转到下一行，如图1-20所示。

```
import numpy as np
import pandas as pd
1 + 2
3 * 5
```

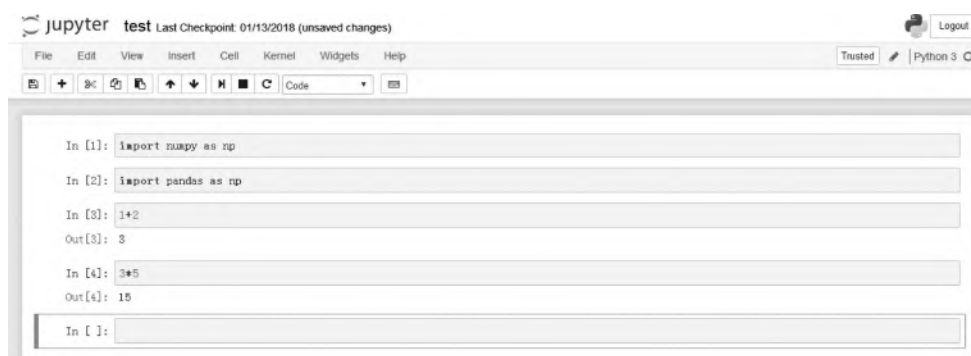


图1-20 Jupyter Notebook具体操作界面图

1.7 小结

本章主要介绍了在Python实际开发过程中，常用的编辑器、IDE开发工具的安装和简介。关于Linux和MacOS系统上相关软件的安装过程，读者可以上网搜索相关的教程。

第2章 Python数据类型用法讲解

2.1 变量

在数学中，我们都学过二元一次方程组，比如：假设 x ， y 满足方程 $x + y = 3$ ， $x - y = 1$ ，求 x 和 y 的值。这里求出的结果是 $x = 2$ ， $y = 1$ 。实际上，求出的 x 和 y 就是变量，只不过在Python中，很少用 x ， y 来表示一个数值型变量，大部分都使用 i ， j ， k 来表示数值型变量。

先以 $x = 2$ 为例， x 就是一个变量，这个变量存储了一个数值为2的值，这个2就是与变量相关的信息。通过等号“=”，式中的2被赋值给了变量 x ，即变量 x 保存了2在内存中的地址。同理，例如`msg = 'hello world!'`，这里的`msg`就是一个变量。通过等号“=”，`'hello world!'`被赋值给了`msg`，即`msg`保存了`'hello world!'`在内存中的地址。

在使用变量时，良好的命名规范能让代码更容易阅读和理解，方便与项目组的其他成员进行交流。

以下是需要注意的有关变量的规则。

(1) 变量名只能包含字母、数字和下划线。变量名可以字母或下划线开头，但不能以数字开头。例如，可将变量命名为`message_1`，但不能将其命名为`1_message`。

(2) 变量名不能包含空格，但可使用下划线来分隔其中的单词。例如，变量名`greeting_message`可行，但变量名`greeting message`会引发错误。

(3) 不要使用Python关键字和函数名用作变量名，Python内置的33个关键字如表2-1所示，要想查看Python的关键字，可以输入import keyword，然后再输入keyword.kwlist即可看到Python内置的33个关键字，如图2-1所示。

(4) 变量名应该简短，并且让程序员能够见名知意。例如，age比a好，stu_name比s_n好等。

(5) 慎用小写字母l和大写字母O，因为它们可能被人错看成数字1和0。

```
In [1]: import keyword
        keyword.kwlist

Out[1]: ['False',
         'None',
         'True',
         'and',
         'as',
         'assert',
         'break',
         'class',
         'continue',
         'def',
         'del',
         'elif',
         'else',
         'except',
         'finally',
         'for',
         'from',
         'global',
```

图2-1 Python内置的33个关键字列表

表2-1 Python内置的33个关键字

False	None	True	and	as	assert	break
class	continue	def	del	elif	else	except
finally	for	from	global	if	import	in
is	lambda	nonlocal	not	or	pass	raise
return	try	while	with	yield		

要创建良好的变量名，需要经过一定的实践，在阅读源码的时候，可以学学源码的变量是怎么命名的，从而提高自己的编码能力。

2.2 字符串

在Python中，可以使用单引号 ‘’ 或者双引号 “” 来创建字符串。在创建字符串时，只要为变量分配一个值即可。例如：

```
str1 = 'hello world!'  
str2 = "hello python!"
```

同时，单引号 ‘’ 和双引号 “” 也可以结合使用，即可以在字符串中包含引号和撇号。例如：

```
'I told my friend, "Python is my favorite language!"'  
"One of Python's strengths is its diverse and supportive community."
```

那么，了解字符串的定义之后，该如何对字符串进行操作呢？此时就需要先知道字符串的下标和切片。

所谓的下标，可以理解为编号，就好比超市存储柜的编号、学生名单表中的学号，通过这个编号或者学号就能找到相应的存储空间或者学生信息。需要注意的是，在Python中，下标是从0开始的，不是从1开始的。

字符串实际上就是字符的数组，所以支持下标索引。如：msg= "吃饭了吗？"，msg[0]= "吃"，msg[1]= "饭"，以此类推。相似地，name='hello'，则name[0]='h'，name[1]='e'，以此类推。

切片是指对操作的对象截取其中一部分的操作。字符串、列表、元组都支持切片操作。切片语法：[起始：结束：步长]。选取的区间属于左闭右开型，即从“起始”位开始，到“结束”位的前一位结束

（不包含结束位本身）。对应到数学中，就是前闭后开型区间，也可以理解为 $\text{start} \leq i < \text{end}$ 。

例如：name='hello'，则切片的常见操作如表2-2所示。

表2-2 切片的常见操作示例

字符串切片语法	含 义
name[0: 3] = 'hel'	取下标 0~2 的字符
name[0: 5] = 'hello'	取下标 0~4 的字符
name[3: 5] = 'lo'	取下标 3、4 的字符
name[2:] = 'llo'	取下标为 2 开始到最后的字符
name[1: -1] = 'ell'	取下标为 1 开始到倒数第 2 个之间的字符
name[: : 2] = 'hlo'	从下标 0 开始到最后字符,每隔 2 个位置,取出字符
name[5: 1: -2] = 'ol'	从下标 5 开始,到下标 2 逆序,每隔 2 个位置,取出字符

在实际操作过程中，如果不知道某些函数的具体用法，则可以使用help命令来进行提示和理解。比如，想知道字符串str的简介，那么可以在Jupyter Notebook中，输入help（str）命令，就可以看到与字符串有关的介绍；输入dir（str）命令，则会显示出字符串所有的私有方法和公有方法（在Python中，以两个下画线开头的方法就默认为私有方法）。对于字符串中的某一特定方法，比如find的用法，可以输入help（str.find）命令，就可以看到find方法的具体介绍和用法，包括函数的传入参数以及函数返回值，如图2-2、图2-3和图2-4所示。

```
In [2]: help(str)
```

Help on class str in module builtins:

```
class str(object)
| str(object='') -> str
| str(bytes_or_buffer[, encoding[, errors]]) -> str
|
| Create a new string object from the given object. If encoding or
| errors is specified, then the object must expose a data buffer
| that will be decoded using the given encoding and error handler.
| Otherwise, returns the result of object.__str__() (if defined)
| or repr(object).
| encoding defaults to sys.getdefaultencoding().
| errors defaults to 'strict'.
|
| Methods defined here:
|
| __add__(self, value, /)
|     Return self+value.
```

图2-2 help (str) 的用法

```
In [3]: dir(str)
```

```
Out[3]: ['__add__',
          '__class__',
          '__contains__',
          '__delattr__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getitem__',
          '__getnewargs__',
          '__gt__',
          '__hash__',
          '__init__',
          '__init_subclass__',
          '__iter__',
          '__le__',
          '__len__',
          '__lt__',
          '__mod__',
```

图2-3 dir (str) 的用法

下面介绍字符串的常用操作。

字符处理类函数及用法如表2-3所示。

```
In [4]: help(str.find)

Help on method_descriptor:

find(...)
    S.find(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.
```

图2-4 help (str.find) 的用法

表2-3 字符处理类函数及用法

函数名及含义	例 子
str.upper() 将所有小写字母变为大写(只对字母起作用)	>>> str1 = 'HELLO world 123' >>> str1.upper() 'HELLO WORLD 123'
str.lower() 将所有大写字母变为小写(只对字母起作用)	>>> str1 = 'HELLO world 123' >>> str1.lower() 'hello world 123'
str.swapcase() 所有大小写字母互换	>>> str1 = 'HELLO world 123' >>> str1.swapcase() 'hello WORLD 123'
str.capitalize() 首字母大写,其余字母小写(要保证字符串的第一个元素是字母)	>>> str1 = 'HELLO world 123' >>> str1.capitalize() 'Hello world 123' >>> str2 = '3HELLO world' >>> str2.capitalize() '3hello world'
str.title() 字符串的每个单词首字母大写	>>> str1 = 'HELLO world 123' >>> str1.title() 'Hello World 123' >>> str2 = '123hello world' >>> str2.title() '123Hello World'

字符串搜索相关类函数及用法如表2-4所示。

表2-4 字符串搜索相关类函数及用法

函数名及含义	例 子
<code>str.find(substr,start,end)</code> 检测字符串 <code>str</code> 是否包含子字符串 <code>substr</code> 。如果包含,返回子字符串 <code>substr</code> 开始的位置索引值,否则返回-1	<pre>>>> str1 = 'HELLO world 123' >>> str1.find('wo') 6 >>> str1.find('wo', 10) -1</pre>
<code>str.rfind(substr,start,end)</code> 类似于 <code>find()</code> 函数,不过是从右边开始查找 <code>substr</code>	<pre>>>> str1 = 'HELLO world 123' >>> str1.rfind('L') 3</pre>
<code>str.index(substr,start,end)</code> 与 <code>find()</code> 方法一样,如果 <code>substr</code> 不在字符串中则会报异常	<pre>>>> str1 = 'HELLO world 123' >>> str1.index('wo') 6 >>> str1.index('wo',10) ValueError: substring not found</pre>
<code>str.rindex(substr,start,end)</code> 类似于 <code>index()</code> ,不过是从右边开始查找 <code>substr</code>	<pre>>>> str1 = 'HELLO world 123' >>> str1.rindex('L') 3 >>> str1.rindex('h') ValueError: substring not found</pre>
<code>str.count(substr,start,end)</code> 计算子字符串 <code>substr</code> 在字符串 <code>str</code> 中出现的次数	<pre>>>> str1 = 'HELLO world 123' >>> str1.count('L') 2 >>> str1.count('z') 0</pre>

字符串判断类函数及用法如表2-5所示。

表2-5 字符串判断类函数及用法

函数名及含义	例 子
<code>str.startswith(substr)</code> 检查字符串是否是以 <code>substr</code> 开头, 是则返回 <code>True</code> , 否则返回 <code>False</code>	<pre>>>> str1 = 'HELLO world 123' >>> str1.startswith('HE') True >>> str1.startswith('he') False</pre>
<code>str.endswith(substr)</code> 检查字符串是否以 <code>substr</code> 结束, 如果是返回 <code>True</code> , 否则返回 <code>False</code>	<pre>>>> str1 = 'HELLO world 123' >>> str1.endswith('123') True >>> str1.endswith('456') False</pre>

续表

函数名及含义	例 子
str.isupper() 检查字符串里的字母是否都是大写字母（注意：只检查字母，不检查空格、数字等）	<pre>>>> str1 = 'HELLO world 123' >>> str1.isupper() False >>> str2 = 'HELLO WORLD 123' >>> str2.isupper() True</pre>
str.islower() 检查字符串里的字母是否都是小写字母（注意：只检查字母，不检查空格、数字等）	<pre>>>> str1 = 'hello world 123' >>> str1.islower() True</pre>
str.isalpha() 检查字符串是否全由字母组成，如果是返回 True，否则返回 False。（注意字符串中是否含有空格）	<pre>>>> str1 = 'HELLOworld' >>> str1.isalpha() True >>> str2 = 'HELLO world' >>> str2.isalpha() False</pre>
str.isdigit() 检查字符串是否全由数字组成，如果是返回 True，否则返回 False。	<pre>>>> str1 = '123' >>> str1.isdigit() True >>> str2 = '12 3' >>> str2.isdigit() False >>> str3 = '123!' >>> str3.isdigit() False</pre>
str.isalnum() 检查字符串是否全为字母或数字	<pre>>>> str1 = 'hello123' >>> str1.isalnum() True >>> str2 = 'hello 123' >>> str2.isalnum() False >>> str3 = 'hello123!' >>> str3.isalnum() False</pre>

续表

函数名及含义	例 子
<code>str.isspace()</code> 检查字符串中是否只包含空格,如果是则返回 True,否则返回 False	<pre>>>> str1 = '' >>> str1.isspace() False >>> str2 = ' ' >>> str2.isspace() True >>> str3 = ' ' >>> str3.isspace() True</pre>

字符串格式化类函数及用法如表2-6所示。

表2-6 字符串格式化类函数及用法

函数名及含义	例 子
<code>str.ljust(width)</code> 返回一个原字符串左对齐,并使用空格填充至长度为 <code>width</code> 的新字符串	<pre>>>> str1 = 'hello' >>> str1.ljust(10) 'hello '(此处有 5 个空格)</pre>
<code>str.rjust(width)</code> 返回一个原字符串右对齐,并使用空格填充至长度为 <code>width</code> 的新字符串	<pre>>>> str1 = 'hello123' >>> str1.rjust(10) ' hello123'(此处有两个空格)</pre>
<code>str.center(width)</code> 返回一个原字符串居中,并使用空格填充至长度为 <code>width</code> 的新字符串(当字符串不可能居中对齐时,左边的空格会比右边的少一个)	<pre>>>> str1 = 'hello' >>> str1.center(7) 'hello '(左右两边各一个空格) >>> str1.center(8) 'hello '(左边一个空格,右边两个空格)</pre>

字符串其他类函数及用法如表2-7所示。

表2-7 字符串其他类函数及用法

函数名及含义	例 子
<code>replace(str1, str2, count)</code> 用 <code>str2</code> 替换掉 <code>str1</code> ,如果 <code>count</code> 指定,则替换不超过 <code>count</code> 次	<pre>>>> name = 'hello world world' >>> name.replace('world', 'Python') 'hello Python Python ' >>> name.replace('world', 'Python', 1) 'hello Python world'</pre>

续表

函数名及含义	例 子
<code>split(str=' ', maxsplit)</code> 以 <code>str</code> 为分隔符切片字符串,如果 <code>maxsplit</code> 有指定值,则仅分隔 <code>maxsplit</code> 个子字符串	<pre>>>> name = 'hello world Python Go' >>> name.split(' ') ['hello', 'world', 'Python', 'Go'] >>> name.split(' ', 2) ['hello', 'world', 'Python Go']</pre>
<code>strip()</code> 删除字符串两端的空白字符	<pre>>>> mystr = '\n\t hello \t\n' >>> mystr.strip() 'hello'</pre>
<code>lstrip()</code> 删除字符串左边的空白字符	<pre>>>> mystr = ' hello' >>> mystr.lstrip() 'hello' >>> mystr = ' hello ' >>> mystr.lstrip() 'hello '</pre>
<code>rstrip()</code> 删除字符串末尾的空白字符	<pre>>>> mystr = ' hello ' >>> mystr.rstrip() ' hello'</pre>
<code>partition(str)</code> 把字符串以 <code>str</code> 分割成三部分, <code>str</code> 前、 <code>str</code> 和 <code>str</code> 后,并保留 <code>str</code> ,返回的是一个元组 <code>tuple</code>	<pre>>>> mystr = 'hello world Python and Go' >>> mystr.partition('Python') ('hello world ', 'Python', ' and Go')</pre>
<code>rpartition(str)</code> 类似于 <code>partition()</code> 函数,不过是从右边开始	<pre>>>> mystr = 'hello world Python and Go' >>> mystr.rpartition('and') ('hello world Python ', 'and', 'Go')</pre>
<code>splitlines()</code> 按照行分割,返回一个包含各行作为元素的列表	<pre>>>> mystr = 'hello\nworld' >>> print(mystr) hello world >>> mystr.splitlines() ['hello', 'world']</pre>
<code>join(str)</code> 字符串中每个字符后面插入 <code>str</code> ,构造出一个新的字符串	<pre>>>> str = ' ' >>> li = ['my', 'name', 'is', 'Python'] >>> str.join(li) 'my name is Python' >>> str = '_ ' >>> str.join(li) 'my_name_is_Python'</pre>

看完上述字符串常用操作后，可以试着理解一下这些函数的可能实现方法。比如：判断一个字符串是否是以字母“a”开头，或者是否以字母“c”结束。首先可能会想到使用startswith（或者endswith）函数，但对于一名初学者来说，如果只接触过基本的下标或者切片，能不能实现该函数的功能呢？

以“a”开头，取下标为0的字符，与字母“a”进行比较，如果相等，则返回True，否则返回False。同理，取出字符串的最后一个字符，与字母“c”进行比较，如果相等则返回True，如果不相等则返回False。

但是在实际开发过程中，可能要考虑更多的情况。假设传进来的参数不是一个字符串，该怎样处理？如果是字符串，会不会是空字符串呢？字符串的长度是否大于要比较的字符串的长度等情况，都需要考虑到。

下面简单写一个判断字符串是不是以某一特定字符串开头的函数（默认传进来的参数都是字符串类型）。如果是，则返回True；否则返回False。其代码运行结果如图2-5所示。

```
def startswith_prefix(mystr, prefix):
    if len(mystr) >= len(prefix) and int(mystr[:len(prefix)] == prefix):
        return True
    else:
        return False
```

```
In [5]: def startswith_prefix(mystr, prefix):
        if len(mystr) >= len(prefix) and int(mystr[:len(prefix)] == prefix):
            return True
        else:
            return False
```

```
In [6]: startswith_prefix('python', 'py')
```

```
Out[6]: True
```

```
In [7]: startswith_prefix('python', 'go')
```

```
Out[7]: False
```

图2-5 startswith_prefix函数运行结果

希望读者在学习字符串的常用操作时，多思考哪些基本的操作能够组合起来，从而实现一个内置函数的用法。也希望读者在面对一个项目问题时，能够快速想到一些内置函数，以提高自己的代码质量，从而减轻自己的测试负担。

2.3 列表list

列表list是一种有序的集合，可以随时添加和删除其中的元素，其长度是可变的。在Python中，使用中括号[]来表示一个列表，列表中的元素可以是int型、string型，也可以是int和string的混合型。比如，列出班级里所有学生的名字，就可以用一个list表示：

```
classmates = ["张三", "李四", "王五"]
```

变量classmates就是一个列表，可以使用下标访问列表元素的值，也可以修改列表的值。其中，classmates[0]的输出结果就是"张三"。

使用len()函数可以获得list元素的个数：len(classmates)的输出结果为3。注意，列表中的元素是可以重复的，并且列表list的索引是从0开始的。当访问的索引值超出列表长度时，就会报错，提示IndexError: list index out of range。所以要确保索引不要越界，并且列表的最后一个元素的索引是len(classmates)-1。

下面介绍列表的常见操作。

2.3.1 增 (append、insert、extend)

列表list是一个长度可变的有序表，可往list中动态地添加元素。有两种追加方法，一种是在列表的末尾添加元素，另一种是在列表的指定位置添加元素。

例如：假设编程语言列表language=["python", "go", "java"], 现在想把C语言和PHP添加到编程语言列表的末尾，使用append或者extend。

当通过append方法向列表中添加新元素时，新元素的数据格式不会发生改变。但当使用extend方法向列表中添加新元素时，如果新元素也是一个列表，则会循环遍历该列表中的元素，将其添加到原始列表中。具体用法和不同之处，如图2-6所示。

insert方法是在指定的索引位置处，直接插入新元素，并且保持新元素的格式不变。具体用法如图2-7所示。

```
In [8]: language = ["python", "go", "java"]
        language.append("c")
        language.append("php")
        language
Out[8]: ['python', 'go', 'java', 'c', 'php']

In [9]: language = ["python", "go", "java"]
        language.append(["c", "php"])
        language
Out[9]: ['python', 'go', 'java', ['c', 'php']]

In [10]: language = ["python", "go", "java"]
         language.extend(["c", "php"])
         language
Out[10]: ['python', 'go', 'java', 'c', 'php']
```

图2-6 append和extend用法示例

```
In [11]: language = ["python", "go", "java"]
         language.insert(1, "c")
         language

Out[11]: ['python', 'c', 'go', 'java']

In [12]: language = ["python", "go", "java"]
         language.insert(1, ["c", "php"])
         language

Out[12]: ['python', ['c', 'php'], 'go', 'java']
```

图2-7 insert用法示例

2.3.2 删（pop、remove、del）

列表中常用的删除方法有3种：第一种是从列表末尾删除元素，可使用pop（）方法；第二种是删除指定索引位置的元素，可使用pop（），也可使用del；第三种是删除列表中特定元素的值，可使用remove（）方法，如果列表中该元素值有多个，则删除第一个匹配的值，其他的不会删除。具体的用法如图2-8所示。

2.3.3 改、查

对于列表中的元素改写，先要找出元素的索引位置，然后再使用等号“=”进行赋值，就可以改写列表中的元素值。具体示例如图2-9所示。

所谓的查找，实际上就是判断某一特定的元素是否存在于列表中。常用的查找方法有：in、not in、index和count。具体用法如表2-8所示。

```
In [13]: language = ["python", "go", "java", "c"]  
language.pop()  
language
```

```
Out[13]: ['python', 'go', 'java']
```

```
In [14]: language = ["python", "go", "java", "c"]  
language.pop(1)  
language
```

```
Out[14]: ['python', 'java', 'c']
```

```
In [15]: language = ["python", "go", "java", "c"]  
del language[1]  
language
```

```
Out[15]: ['python', 'java', 'c']
```

```
In [16]: language = ["python", "go", "java", "c", "go"]  
language.remove("go")  
language
```

```
Out[16]: ['python', 'java', 'c', 'go']
```

```
In [17]: language = ["python", "go", "java", "c", "go"]  
language.remove("java")  
language
```

```
Out[17]: ['python', 'go', 'c', 'go']
```

图2-8 pop ()、remove ()、del用法示例

```
In [18]: language = ["python", "go", "java", "c"]  
language[1] = "php"  
language
```

```
Out[18]: ['python', 'php', 'java', 'c']
```

图2-9 列表的改操作示例

表2-8 列表的查操作用法示例

方法名及含义	例 子
<code>in</code> 判断元素是否存在于列表中,存在则返回 True,不存在则返回 False	<pre>>>> language=["python", "go", "java", "c"] >>> "c" in language True >>> "php" in language False</pre>
<code>not in</code> 判断元素是否不存在于列表中,不存在则返回 True,存在则返回 False	<pre>>>> language=["python", "go", "java", "c"] >>> "c" not in language False >>> "php" not in language True</pre>
<code>index</code> 与字符串的 <code>index</code> 用法相同,判断元素是否存在于列表中,存在则返回该元素在列表中的索引位置,不存在则直接报错	<pre>>>> language=["python", "go", "java", "c"] >>> language.index("c") 3 >>> language.index("c", 1, 3) ValueError: 'c' is not in list</pre>
<code>count</code> 判断元素在列表中出现的次数,存在则返回次数,不存在则返回 0	<pre>>>> language=["python", "go", "java", "c", "go"] >>> language.count("go") 2 >>> language.count("php") 0</pre>

列表中的查询，直接使用索引即可访问该元素。要确保索引不要越界，并且列表的最后一个元素的索引是列表长度减去1。

1. 使用索引查询元素

如果要取最后一个元素，除了计算索引位置外，还可以用-1做索引，直接获取最后一个元素，以此类推，可以获取倒数第2个、倒数第3个。具体操作如图2-10所示。

```
In [19]: language = ["python", "go", "java", "c"]
         language[1]

Out[19]: 'go'

In [20]: language = ["python", "go", "java", "c"]
         language[-1]

Out[20]: 'c'

In [21]: language = ["python", "go", "java", "c"]
         language[-2]

Out[21]: 'java'

In [22]: language = ["python", "go", "java", "c"]
         language[-3]

Out[22]: 'go'
```

图2-10 使用索引查询列表元素

注意，列表list中的元素，可以是一个列表，这就是所谓的列表嵌套。在学习了字典之后，列表的元素也可以是字典。

```
>>> language = ["python", "go", ["php", "scala"], "java", "c"]
>>> len(language)
5
```

要注意language只有5个元素，其中language[2]又是一个list，如果拆开写就更容易理解了。因此，上述代码可改写为：

```
p = ["php", "scala"]
language = ["python", "go", p, "java", "c"]
```

要想取得 " php "，可以写 p[0] 或者 language[2][0]，因此 language 可以看成是一个二维数组，类似的还有三维、四维等数组，但在实际开发过程中，很少会用到三维及以上的数组。

2. 使用切片访问列表

这一点与字符串的访问相类似。其切片语法为：[起始：结束：步长]。选取的区间属于左闭右开型区间，即从“起始”位开始，到“结束”位的前一位结束（不包含结束位本身）。对应到数学中，就是前闭后开型区间，也可以理解为start <=i <end。

例如：language=[" python " , " go " , " php " , " scala " , " java " , " c "]，则切片访问列表的示例如表2-9所示。

表2-9 切片访问列表的示例

列表切片语法	含 义
language[0: 3]='python', 'go', 'php'	取下标 0~2 的元素
language[0: 5]='python', 'go', 'php', 'scala', 'java'	取下标 0~4 的元素
language[3: 5]='scala', 'java'	取下标 3、4 的元素
language[2:]='php', 'scala', 'java', 'c'	取下标为 2 开始到最后的元素
language[1: -1]='go', 'php', 'scala', 'java'	取下标为 1 开始到倒数第 2 个之间的元素
language[: : 2]='python', 'php', 'java'	从下标 0 开始到最后字符,每隔 2 个位置取出元素
language[5: 1: -2]='c', 'scala'	从下标 5 开始,到下标 2 逆序,每隔 2 个位置取出元素

2.3.4 列表的循环遍历

如果要查看列表的所有元素，则需要对列表进行循环遍历。以下是两种常见的循环语句写法。

(1) 使用for循环

```
languages = ["python", "go", "php", "scala", "java", "c"]
for language in languages:
    print(language)
```

(2) 使用while循环

```
languages = ["python", "go", "php", "scala", "java", "c"]
length = len(languages)
i = 0
while i < length:
    print(languages[i])
    i += 1
```

2.3.5 排序（sort，reverse）

sort方法是将list按特定顺序重新排列，排序方式默认为由小到大，参数reverse=True，可改为倒序，由大到小排序。

reverse方法可以直接将list中所有的元素进行逆置。

```
>>>a = [1, 4, 2, 3]
>>>a.reverse( )
>>>a
[3, 2, 4, 1]
>>>a.sort( )
>>>a
[1, 2, 3, 4]
>>>a.sort(reverse = True)
>>>a
[4, 3, 2, 1]
```

2.3.6 列表的其他操作符

列表对+和×的操作符与字符串相似，也可以进行成员检查。+号用于组合列表，×号用于重复列表。其具体用法如表2-10所示。

表2-10 +、×操作示例

表 达 式	输 出 结 果	描 述
[1,2,3]+[4,5,6]	[1,2,3,4,5,6]	将列表拼接组合起来
["python"]× 3	["python", "python", "python"]	重复
1 in [1,2,3]	True	判断元素是否存在于列表中
for i in [1,2,3]: print(i)	1 2 3	迭代

2.4 集合set

在学数学的时候，都会学习到集合的概念。在数学中，集合实际上就是“确定的一堆东西”。集合里的“东西”，叫作元素。由一个或多个确定的元素所构成的整体叫作集合。若 x 是集合 A 的元素，则记作 $x \in A$ 。

集合中的元素有3个特征。

(1) 确定性（集合中的元素必须是确定的）。

(2) 互异性（集合中的元素互不相同）。例如：集合 $A = \{1, a\}$ ，则 a 不能等于1。

(3) 无序性（集合中的元素没有先后之分），如集合 $\{3, 4, 5\}$ 和 $\{3, 5, 4\}$ 算作同一个集合。

在Python中，集合也是类似的，用`set()`表示，也可以使用大括号`{}`来表示集合，其中的元素也是无顺序的。由于在实际编写代码的过程中，集合的用处比较少，下面简单介绍集合的常见用法。

【注意】

由于集合具有互异性（集合中的元素互不相同），因此可以对一个有重复元素的列表，添加上`set()`之后，就会将重复的元素隐藏。

2.4.1 创建集合

创建集合时，使用`set()`即可。如`var=set([1, 3, 5, 6, 4, 2])`，如图2-11所示。

```
In [23]: var = set([1, 3, 5, 6, 4, 2])  
var
```

```
Out[23]: {1, 2, 3, 4, 5, 6}
```

```
In [24]: for i in var:  
         print(i)
```

```
1  
2  
3  
4  
5  
6
```

图2-11 创建集合实例

由图2-11可以看出，集合内部实际上已经对元素进行了排序。那么，集合中能不能包含不同类型的元素呢？具体如图2-12、图2-13和图2-14所示。

```
In [25]: var = {1, "d", 3, "c", 5, "a", 6, "f", 4, "b", 2}  
var
```

```
Out[25]: {1, 2, 3, 'f', 5, 6, 4, 'a', 'b', 'c', 'd'}
```

```
In [26]: for i in var:  
         print(i)
```

```
1  
2  
3  
f  
5  
6  
4  
a  
b  
c  
d
```

图2-12 集合中包含整型和字符型示例

```
In [27]: var = {1, 3, 5, 6, 4, 2, {12}}
var

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-27-f5912640a9c1> in <module>()
----> 1 var = {1, 3, 5, 6, 4, 2, {12}}
      2 var

TypeError: unhashable type: 'set'
```

图2-13 集合中包含整型和字典示例

由此可以看出，如果放入的元素类型是集合或者列表，则会报TypeError错误。如果集合里都是整型和字符型的元素，则循环遍历集合的输出结果就不会进行排序。那么是不是说，如果元素类型相同，就一定会排序呢？具体如图2-15所示。

```
In [28]: var = {1, 3, 5, 6, 4, 2, [12, 34]}
var

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-28-5b774cfbc444> in <module>()
----> 1 var = {1, 3, 5, 6, 4, 2, [12, 34]}
      2 var

TypeError: unhashable type: 'list'
```

图2-14 集合中包含整型和列表示例

```
In [29]: var = {"d", "a", "f", "c", "e", "d"}
var
```

```
Out[29]: {'a', 'c', 'd', 'e', 'f'}
```

```
In [30]: for i in var:
          print(i)
```

```
e
f
a
c
d
```

图2-15 集合中字符型排序示例

可见，如果集合的元素全是字符型，那么实际上输出的结果并没有进行排序。

总之，集合中可以放不同类型的元素，但是存放的元素只能为数值型和字符型，不能是列表类型和集合类型。如果元素都是数值型，则循环变量的输出结果为从小到大排序后的结果；如果元素都是字符型，循环变量的输出结果并不会进行排序；如果元素既有数值型又有字符型，则循环变量的输出结果也不会进行排序。

2.4.2 集合的增、删

如果向集合中添加元素，则使用add（）函数。集合的add（）示例如图2-16所示。

```
In [31]: names = {"Jack", "David", "Rose"}
names.add("Jone")
print(names)

{'David', 'Jone', 'Rose', 'Jack'}
```

图2-16 集合的add（）示例

如果要随机删除集合中的一个元素，可以使用pop（）方法；如果要删除指定的元素，可以使用remove（）和discard（）方法。但当remove（）方法找不到指定的元素时，会报错；而当discard（）方法找不到指定的元素时，并不会报错。如图2-17、图2-18和图2-19所示。

```
In [32]: names = {"David", "Jack", "Jone", "Rose"}
result = names.pop()
print(result)
```

David

图2-17 集合的pop（）示例

```

In [33]: names = {"David", "Jack", "Jone", "Rose"}
names.remove("Jack")
names

Out[33]: {'David', 'Jone', 'Rose'}

In [34]: names = {"David", "Jack", "Jone", "Rose"}
names.remove("Jackl")
names

-----
KeyError                                Traceback (most recent call last)
<ipython-input-34-16d1bfbled44> in <module>()
      1 names = {"David", "Jack", "Jone", "Rose"}
----> 2 names.remove("Jackl")
      3 names

KeyError: 'Jackl'

```

图2-18 集合的remove（）示例

```

In [35]: names = {"David", "Jack", "Jone", "Rose"}
names.discard("Jack")
names

Out[35]: {'David', 'Jone', 'Rose'}

In [36]: names = {"David", "Jack", "Jone", "Rose"}
names.discard("Jackl")
names

Out[36]: {'David', 'Jack', 'Jone', 'Rose'}

```

图2-19 集合的discard（）示例

2.4.3 集合的交、并、补等操作

在数学中，集合有交集、并集、补集等概念，那么Python中是否支持交集、并集、补集等操作呢？集合的交、并、补等用法如表2-11所示。集合的交、并、补操作示例如图2-20所示。

表2-11 集合的交并补等用法

数学符号	Python 符号和内置方法	含 义
—或\	—或 difference()	差集
$\cap$	& 或 intersection()	交集
$\cup$	或 union()	并集、合集
$\neq$	$!=$	不等于
$=$	$==$	等于
$\in$	in	是成员关系
$\notin$	not in	不是成员关系

```
In [37]: names1 = {"David", "Jack", "Jone", "Rose"}
names2 = {"Tom", "Lily", "Rose"}
names1.difference(names2)

Out[37]: {'David', 'Jack', 'Jone'}
```

```
In [38]: names1 = {"David", "Jack", "Jone", "Rose"}
names2 = {"Tom", "Lily", "Rose"}
names1.union(names2)

Out[38]: {'David', 'Jack', 'Jone', 'Lily', 'Rose', 'Tom'}
```

```
In [39]: names1 = {"David", "Jack", "Jone", "Rose"}
names2 = {"Tom", "Lily", "Rose"}
names1.intersection(names2)

Out[39]: {'Rose'}
```

图2-20 集合的交并补操作示例

2.5 字典dictionary

在Python中，字典用大括号{}表示，但字典与集合不同，字典的{}里面必须有键和对应的值，而集合的{}里只需要有值即可。字典的每个元素是由两部分组成，即键和值，俗称键值对（key-value）。每个键值对key=>value用冒号“:”分割，每个键值对之间用逗号“,”分割。其语法格式如下：

```
d = {key1 : value1, key2 : value2 ,.....}
```

【注意】

(1) 字典的键一般是唯一的，如果字典的键发生重复，则最后一个键值对就会替换掉前面的键值对。字典含有重复键的输出示例如图2-21所示。

```
In [40]: dict = {"a": 1, "b": 2, "b": '3'}
          dict["b"]

Out[40]: '3'

In [41]: dict

Out[41]: {'a': 1, 'b': '3'}
```

图2-21 字典含有重复键的输出示例

(2) 字典的值可以取任何数据类型，但是键必须是不可变类型，如数字、字符串和元组，但是不能使用列表作为字典的键。字典键是列表时的示例如图2-22所示。

```
In [42]: dict = [{"name": "Tom", "b": 2, "b": '3'}]
          dict

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-42-afc6b08f35b6> in <module>()
----> 1 dict = [{"name": "Tom", "b": 2, "b": '3'}]
      2 dict

TypeError: unhashable type: 'list'
```

图2-22 字典键是列表时的示例

2.5.1 字典的查找操作

在字典中，如果想获取一个元素，那么只需要根据字典的键，就可以访问到对应的值。例如：在学生信息系统中，可以查询学生姓名，也可以查询学生的学号。字典的查找操作示例如图2-23所示。

```
In [43]: stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
          print(stu_info["name"])
          print(stu_info["id"])

Tom
100
```

图2-23 字典的查找操作示例

在这个例子中，name、id都是字典的键，它们存储了对应的姓名和学号。当然，也可以查询性别、地址等信息。

如果查询的键不存在，比如，想查询stu_info中的年龄age和对应的数值，得到的结果是什么呢？字典中不存在对应键的操作示例如图2-24所示。

在查询年龄age时，stu_info中并没有存储age和对应的值，导致结果报错。错误信息为：KeyError: 'age'，即提示stu_info中并没有age这个键。当不确定某一个字典中是否存在某个键，但是又想获取对应的值时，可以使用get（）方法，同时还可以设置获取值的默认值。字典的get（）方法示例如图2-25所示。

```
In [44]: print(stu_info["age"])

-----
KeyError                                Traceback (most recent call last)
<ipython-input-44-3c4e1ca6eb> in <module>()
----> 1 print(stu_info["age"])

KeyError: 'age'
```

图2-24 字典中不存在对应键的操作示例

```
In [45]: print(stu_info.get("age"))
None

In [46]: print(stu_info.get("age", 15))
15
```

图2-25 字典的get（）方法示例

如果stu_info中不存在age这个键，则get（）方法返回None。给get（）方法传入一个默认值15。当不存在这个键时，则返回默认值15。

2.5.2 字典的增、改操作

在查找出元素的基础上，就可以对元素进行添加或修改。当查找的键存在时，使用变量名['key']=value的方式，实际上是对元素进行修改操作。当查找的键不存在时，使用变量名['key']=value的方式，可以将该key和对应的value添加到字典中。字典的增操作示例如图2-26所示。

```
In [47]: stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
print(stu_info)
stu_info["age"] = 18
print(stu_info)

{'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'China'}
{'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'China', 'age': 18}
```

图2-26 字典的增操作示例

字典的每个元素中的数据是可以修改的，只要通过key找到，就可以修改。字典的改操作示例如图2-27所示。

```
In [48]: stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
print(stu_info)
stu_info["address"] = "England"
print(stu_info)

{'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'China'}
{'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'England'}
```

图2-27 字典的改操作示例

2.5.3 字典的删操作

如果想要删除字典中的元素，可以使用del、pop（）或clear（）。del用于删除指定的元素，也可用于删除整个字典，pop（）用于删除指定元素，clear（）用于删除整个字典。字典的del操作示例如图2-28所示。

```
In [49]: stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
print(stu_info)
del stu_info["sex"]
print(stu_info)

{'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'China'}
{'name': 'Tom', 'id': 100, 'address': 'China'}
```

图2-28 字典的del操作示例

del除了可以删除指定的元素之外，还可以删除整个字典。del删除整个字典操作示例如图2-29所示。

```
In [50]: stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
print(stu_info)
del stu_info
print(stu_info)

{'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'China'}

-----
NameError                                Traceback (most recent call last)
<ipython-input-50-68e2b1254419> in <module>()
      2 print(stu_info)
      3 del stu_info
----> 4 print(stu_info)

NameError: name 'stu_info' is not defined
```

图2-29 del删除整个字典操作示例

pop（）用于删除指定元素，如果键不存在，则报错KeyError。字典的pop（）操作示例如图2-30所示。

clear（）用于删除整个字典，与del不同，clear（）只是删除字典里的键值对，得到的结果是一个空字典，del则是删除了整个字典结构。可以类比于一本书，clear（）是把书里的所有内容删除，虽然变成了空白页，但是书的结构还在；del则是完全销毁了这本书，即书的结构也就不存在了。字典的del和clear（）对比示例如图2-31所示。

```

In [51]: stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
          print(stu_info)
          stu_info.pop("sex")
          print(stu_info)

{'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'China'}
{'name': 'Tom', 'id': 100, 'address': 'China'}

In [52]: stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
          print(stu_info)
          stu_info.pop("age")
          print(stu_info)

{'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'China'}

-----
KeyError                                Traceback (most recent call last)
<ipython-input-52-fb853e565135> in <module>()
      1 stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
      2 print(stu_info)
----> 3 stu_info.pop("age")
      4 print(stu_info)

KeyError: 'age'

```

图2-30 字典的pop（）操作示例

```

In [53]: stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
          print(stu_info)
          stu_info.clear()
          print(stu_info)

{'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'China'}
{}

```

图2-31 字典的del和clear（）对比示例

2.5.4 字典的常用方法

除了上述介绍的字典的增删改查等操作，字典还内嵌了一些其他的方法，字典的内嵌方法示例具体如表2-12所示。

表2-12 字典的内嵌方法示例

方法名及含义	例 子
len() 返回字典中键值对的个数	<pre>>>> stu_info = {'name': 'Tom', 'id': 100} >>> len(stu_info) 2</pre>
keys() 返回字典中所有 key 的列表	<pre>>>> stu_info = {'name': 'Tom', 'id': 100} >>> stu_info.keys() ['name', 'id']</pre>
values() 返回字典所有 value 的列表	<pre>>>> stu_info = {'name': 'Tom', 'id': 100} >>> stu_info.values() ['Tom', 100]</pre>
items() 返回一个列表,列表里的元素是(键,值)元组	<pre>>>> stu_info = {'name': 'Tom', 'id': 100} >>> stu_info.items() [('name', 'Tom'), ('id', 100)]</pre>
has_key(key) 如果字典的键中含有指定的 key,返回 True,否则返回 False。(此方法在 Python3 中已被删除)	<pre>>>> stu_info = {'name': 'Tom', 'id': 100} >>> stu_info.has_key('name') True >>> stu_info.has_key('age') False</pre>
in 如果字典的键中含有指定的 key,返回 True,否则返回 False。(Python2,3 都可以使用)	<pre>>>> stu_info = {'name': 'Tom', 'id': 100} >>> 'name' in stu_info True >>> 'age' in stu_info False</pre>

2.5.5 有序字典

如果希望对字典做迭代,按照元素初始添加的顺序进行结果输出,那么就可以使用collections包下的OrderedDict方法。

对字典做迭代时,它会严格按照元素初始添加的顺序进行。有序字典OrderedDict的遍历示例如图2-32所示。

```
In [54]: from collections import OrderedDict
         od = OrderedDict()
         od["first"] = 1
         od["second"] = 2
         od["third"] = 3
         for key,value in od.items():
             print(key, value)

first 1
second 2
third 3
```

图2-32 有序字典OrderedDict的遍历示例

通过查看源码，如图2-33所示，OrderedDict内部维护了一个双向链表，它会根据元素加入的顺序来排列键的位置。第一个新加入的元素被放置在链表的末尾，再对已存在的键做重新赋值，不会改变键的顺序。

```
71 class OrderedDict(dict):
72     'Dictionary that remembers insertion order'
73     # An inherited dict maps keys to values.
74     # The inherited dict provides __getitem__, __len__, __contains__, and get.
75     # The remaining methods are order-aware.
76     # Big-O running times for all methods are the same as regular dictionaries.
77
78     # The internal self._map dict maps keys to links in a doubly linked list.
79     # The circular doubly linked list starts and ends with a sentinel element.
80     # The sentinel element never gets deleted (this simplifies the algorithm).
81     # The sentinel is in self._hardroot with a weakref proxy in self._root.
82     # The prev links are weakref proxies (to prevent circular references).
83     # Individual links are kept alive by the hard reference in self._map.
84     # Those hard references disappear when a key is deleted from an OrderedDict.
85
```

图2-33 OrderedDict的简介说明

由于采用了双向链表，OrderedDict的大小是普通字典的两倍多，因此，在想要构建一个涉及大量OrderedDict实例的数据结构时，首先需要判断增加的内存开销是否划算，然后再决定是否使用OrderedDict。

在实际开发过程中只知道常用的操作还是不够的，接到一个需求之后，最主要的任务就是编写出实现具体功能的函数。

2.6 函数

在数学中，给出长方形的长度 a 和宽度 b ，求出长方形的面积 $S=ab$ 。在Python中，输出的结果实际上就相当于return语句，长度和宽度实际上就相当于函数的输入参数。

在Python中，定义函数的语法如下所示：

```
def 函数名(输入参数 1,输入参数 2,...):  
    函数体  
    return 返回结果
```

比如这个计算长方形面积的函数，就可以写成：

```
def cal_area(length, width):  
    s = length * width  
    return s
```

在定义函数时，小括号中的参数，用于接收参数，称之为“形参”。在调用函数时，括号中的参数，传递给函数的参数，称之为“实参”。

在实际开发过程中，常常需要考虑，该函数需不需要参数？如果需要，需要几个参数？该函数需不需要返回值？需要返回几个返回值？

根据这些，可以把函数分为4种类型，如表2-13所示。

表2-13 函数的4种类型

函数类型	举 例
① 无参数,无返回值的函数 此类函数,不能接收参数,也没有返回值。一般情况下,打印提示类的功能,可使用这类函数	<pre>def printHello(): print('欢迎光临本店') print('祝您用餐愉快')</pre>
② 无参数,有返回值的函数 此类函数,不能接收参数,但是可以返回某个数据。一般情况下,如采集数据,可使用这类函数	<pre>def getTemperature(): return 24</pre>
③ 有参数,无返回值的函数 此类函数,能接收参数,但不返回数据。一般情况下,该函数主要用于对变量进行赋值,如最常见的 set()函数	<pre>def setData(num): self.num = num</pre>
④ 有参数,有返回值的函数 此类函数,不仅能接收参数,还可返回某个数据。一般情况下,对于数据处理并需要结果的应用,可用这类函数	<pre>def calculateNum(num): result = 0 i = 1 while i <= num: result = result + i i += 1 return result</pre>

在实际开发过程中,第一类函数主要用于打印代码调试、警告、错误信息等,方便开发人员进行错误跟踪,及时对代码进行修改和优化;第二类函数主要用于获取数据,最常见的函数就是get方法;第三类函数主要用于设置数据,最常见的函数就是set方法;第四类函数是重点,在处理数据的过程中,需要知道传进来的参数是什么,对其进行处理后,要将处理后的结果再返回出去,以便后续的操作。

2.7 小结

本章主要介绍了常见的Python数据类型的用法以及函数的4种类型。在实际开发过程中,可以通过查看源码,来帮助理解底层的实现方式。遇到问题的时候,多思考,多动手尝试。在此基础上,明确代

码的处理逻辑和顺序，但不重复发明轮子，使用已经开发好的包或者第三方库，可以达到事半功倍的目的，从而提高工作效率。

第3章 Python下的实际应用

Python是一种代表简单主义思想的语言，有“胶水语言”之称。目前，Python的应用层面也十分广泛，主要包括数据库的连接、系统编程、网络爬虫、人工智能、Web开发、系统运维、大数据、云计算、图形界面等。在当前环境下，Python重点应用于机器学习和人工智能方面。

3.1 Python连接MySQL数据库

MySQL数据库是企业最常用的数据库之一，而MySQL数据库最常用的功能是select查询。数据分析人员希望可以用Python直接连接MySQL数据库，然后读取数据到pandas框架中。

首先安装好MySQL数据库，安装步骤详见附录B。然后安装pymysql，安装语句为：pip install pymysql，安装好这个包以后，在Jupyter Notebook中使用以下代码：

```
import pandas as pd
import pymysql
conn = pymysql.connect(
    host = '主机 IP',

    user = '用户名',
    password = '密码',
    db = '数据库名',
    port = 端口,
    charset = 'utf8')
table = "select...from..."
```

上面的table就是连接mysql后查询到的表。然后用下面代码，就可以直接把数据读取到pandas框架中了。数据读取到pandas框架后，

就可以使用pandas的常见功能对数据进行处理，比如填补缺失值、删除数据等。

```
data = pd.read_sql(table, conn)
```

3.2 Python连接MongoDB数据库

MongoDB数据库也是企业常用的数据库之一。MongoDB数据库是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库中功能最丰富、最像关系数据库的一种数据库。

相对于MySQL数据库，MongoDB数据库具有独特的优势。同样地，Python也可以直接连接MongoDB数据库，并把数据读取到pandas框架中。安装步骤详见附录C。

首先安装pymongo，安装语句为：pip install pymongo。安装步骤详见附录F。安装好pymongo后，使用方法如下：

```
import pandas as pd
from pymongo import MongoClient
client = MongoClient('主机 IP', port = 端口)
db = client.数据库名
db.authenticate('用户名', '密码')
table = db.表名
```

与MySQL数据库一样，上面的table就是连接MongoDB后查询到的表，然后使用下面代码，就可以直接把数据读取到pandas框架了，其中，find括号里可以加入MongoDB数据库查询语句。

```
data = pd.DataFrame(list(table.find()))
```

3.3 结巴分词和词云图

Python可以处理文本数据，常见的有NLTK自然语言处理工具包、LSTM长短期记忆网络，但这些工具包基本上都用于处理英文。这里介绍下简单且常用的中文语言处理工具包jieba。

下面将以Mac系统为例讲解jieba和wordcloud的安装方法，安装语句为：`pip install jieba`，`pip install wordcloud`。步骤参考附录F第三方包安装步骤。

本节以网络下载的小说《天龙八部》为例，讲解jieba和wordcloud的用法。

```

import pandas as pd
import os
os.chdir('../data')
character = open('天龙八部人物表.txt', encoding = 'utf-8').read()
'''添加词性'''
import re
data = re.split(r'\s+|:|,', character)
data = pd.DataFrame(data, columns = ['姓名'])
data['词性'] = 'nr'
data.to_excel('天龙八部人物分词.xlsx', index = False, header = None)
'''添加自定义词库'''
import jieba
jieba.enable_parallel(4)
jieba.load_userdict('天龙八部人物词典.txt')
'''加载停用词库'''
stopwords = [line.rstrip() for line in open('停用词表.txt', encoding = 'utf-8')]
'''jieba分词'''
def seg_sentence(sentence):
    sentence_segged = jieba.cut(sentence.strip())
    outstr = ''
    for word in sentence_segged:
        if word not in stopwords:
            if word != '\t':
                outstr += word
                outstr += ' '
    return outstr
inputs = open('天龙八部.txt', 'r', encoding = 'GB18030')
outputs = open('天龙八部分词.txt', 'w', encoding = 'utf-8')

for line in inputs:
    line_seg = seg_sentence(line)
    outputs.write(line_seg + '\n')
outputs.close()
inputs.close()

```

分词过后，就可以进行文本分析了。先加载分词后的txt文本：

```

text = open('天龙八部分词.txt', encoding = 'utf-8').read()

```

然后指定词性，提取关键词，并打印出由TF-IDF方法计算出的权重最大的前10人物，效果如图3-1所示。

```

'''指定词性, 提取关键词: TF-IDF'''
import jieba.analyse

n=100          #指定关键词数量

result=jieba.analyse.extract_tags(text, topK=n, withWeight=True, allowPOS=('nr',))

result[:10]    #打印前10个最重要的人物

[('段誉', 0.5891293630142107),
 ('萧峰', 0.4638848540025745),
 ('乔峰', 0.3369411093846939),
 ('慕容复', 0.2771830168385107),
 ('王语嫣', 0.26712617741403344),
 ('武功', 0.23669144784504287),
 ('段正淳', 0.22647654172059356),
 ('阿朱', 0.21979838728524273),
 ('木婉清', 0.21805626004123815),
 ('鸠摩智', 0.1742127244004566)]

```

图3-1 TF-IDF计算出的权重最大的前10人

```

'''词云关键词'''
keywords = dict()
for i in result:
    keywords[i[0]] = i[1]

'''设置词云属性'''
from PIL import Image
import numpy as np
from wordcloud import WordCloud, ImageColorGenerator
import matplotlib.pyplot as plt
image = Image.open('图片.jpeg')
graph = np.array(image)

wc = WordCloud(font_path='/Library/Fonts/Songti.ttc', # 设置字体
               background_color = 'White',          # 设置背景颜色
               max_words = n,                        # 设置词云显示的最大词数
               mask = graph)                         # 设置背景样式

```

最后效果如图3-2所示。

```
In [41]: '''生成词云'''
wc.generate_from_frequencies(keywords)
plt.imshow(wc)
image_color = ImageColorGenerator(graph)
plt.imshow(wc.recolor(color_func=image_color))
plt.axis("off")
plt.show()
wc.to_file('词云.jpg')
```



图3-2 词云生成图

3.4 简单社交网络

目前，社交网络是大数据时代的热门课题之一，正式的社交网络图谱一般用Neo4j来做，Neo4j的入门与应用将在本书第8章中进行详细介绍。本章将介绍Python语言中的简单社交网络。

```
import pandas as pd

data1 = pd.read_excel('牛魔王关系图.xlsx')
data2 = pd.read_excel('孙悟空关系图.xlsx')
data3 = pd.read_excel('唐僧关系图.xlsx')
data4 = pd.read_excel('猪八戒关系图.xlsx')
data = pd.concat([data1, data2, data3, data4])
```

简单社交网络数据展示图，如图3-3所示。

Source是源节点，Target是目标节点，Weight是关系权重。

在社交网络中有4个重要的定义。

(1) 度中心性：一个节点在网络中的连接数。

```
In [5]: data.head()
```

```
Out[5]:
```

	Source	Target	Weight
0	牛魔王	铁扇公主	7
1	牛魔王	红孩儿	7
2	牛魔王	孙悟空	5
3	牛魔王	玉面狐狸	6
4	铁扇公主	红孩儿	8

图3-3 简单社交网络数据展示图

(2) 接近中心性：识别集群内部被高度连接的节点。

(3) 中介中心性：识别连接不同集群的节点。

(4) 网页排名：衡量节点活跃度，节点越活跃，pangrank值越大。

```
import networkx as nx  
graph = nx.from_pandas_dataframe(data, 'Source', 'Target', edge_attr=['Weight'])
```

度中心性、接近中心性、中介中心性、网页排名分别如图3-4、图3-5、图3-6和图3-7所示。

```
'''度中心性：一个节点在网络中的连接数'''
degree_df = pd.DataFrame(columns=['度中心性'])
degree_df['度中心性'] = pd.Series(nx.degree_centrality(graph))
degree_df.sort_values('度中心性', ascending=False)[:10].plot(kind='barh')
plt.show()
```

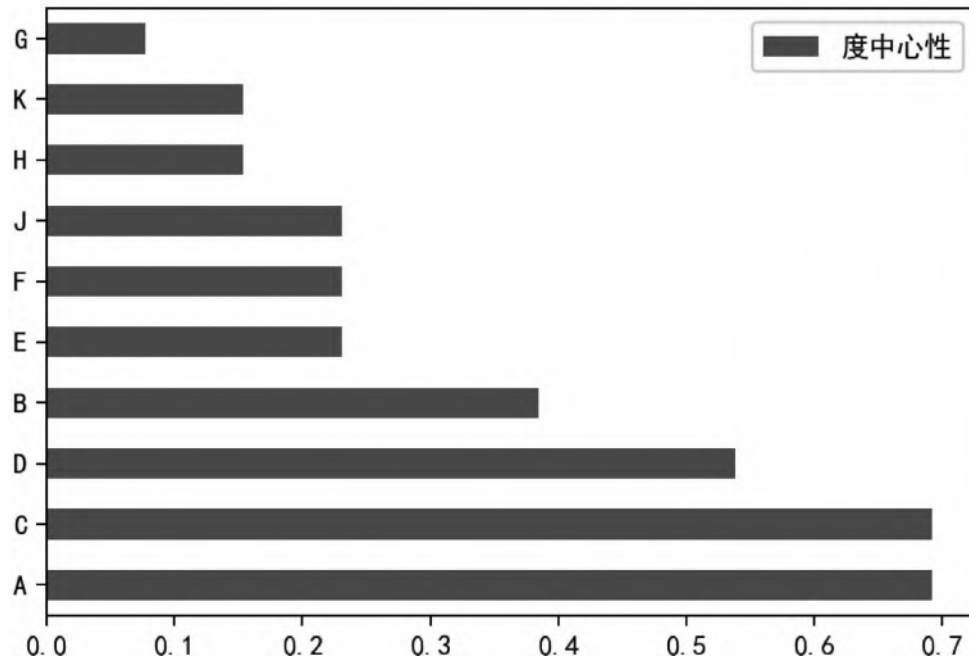


图3-4 度中心性展示图

```
'''接近中心性: 具有高接近中心性的节点通常在集群之间被高度连接'''
closeness_df = pd.DataFrame(columns=['接近中心性'])
closeness_df['接近中心性'] = pd.Series(nx.closeness centrality(graph))
closeness_df.sort_values('接近中心性', ascending=False)[:10].plot(kind='barh')
plt.show()
```

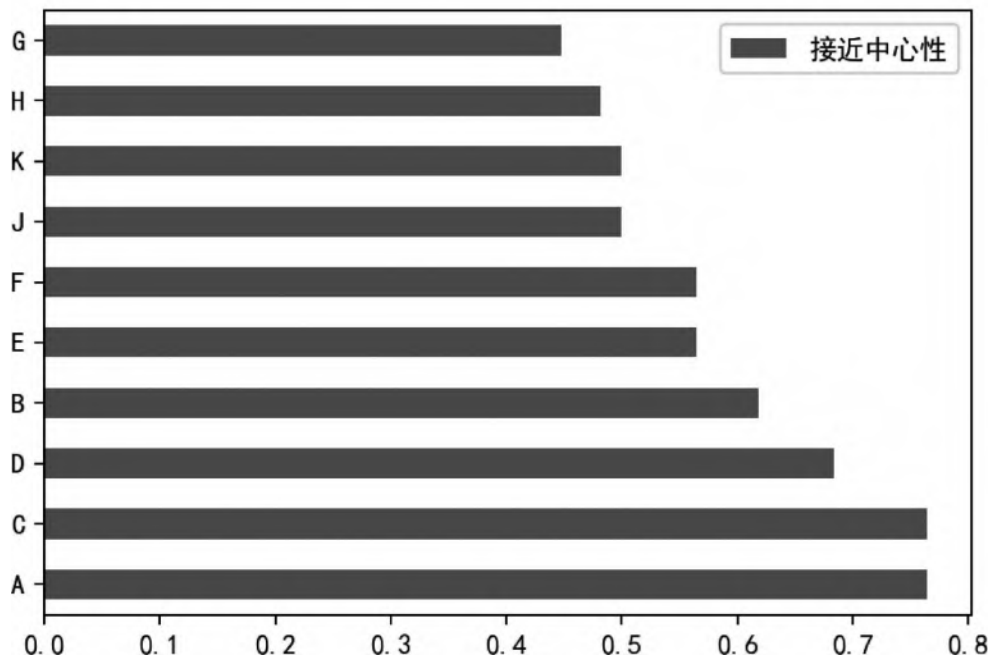


图3-5 接近中心性展示图

```
'''中介中心性: 识别连接不同集群的节点'''
betweenness_df = pd.DataFrame(columns=['中介中心性'])
betweenness_df['中介中心性'] = pd.Series(nx.betweenness centrality(graph))
betweenness_df.fillna(0, inplace=True)
betweenness_df.sort_values('中介中心性', ascending=False)[:10].plot(kind='barh')
plt.show()
```

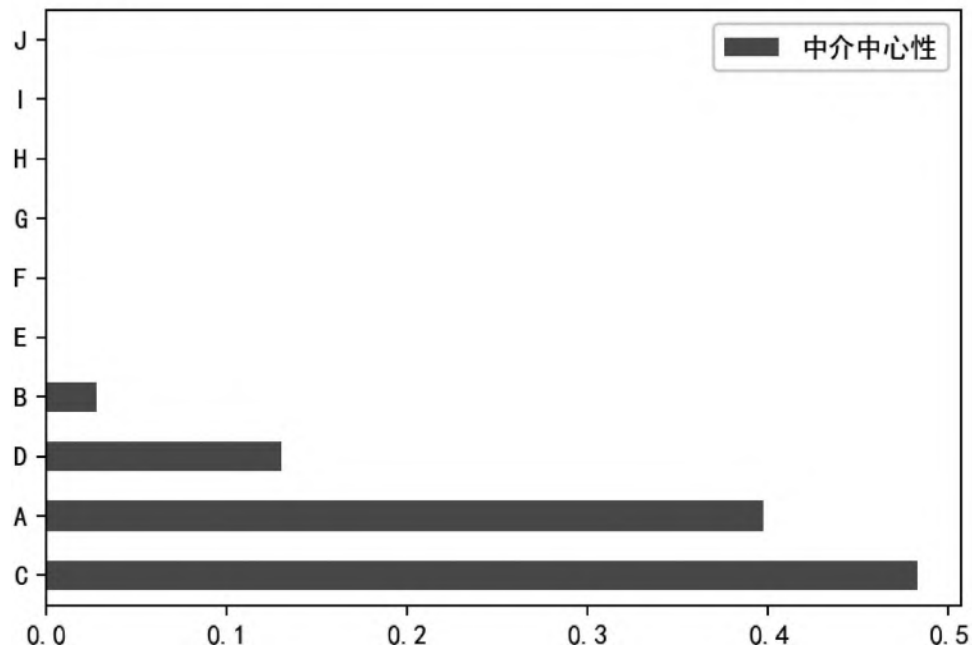


图3-6 中介中心性展示图

```
'''网页排名：找出活跃用户'''
pagerank_df = pd.DataFrame(columns=['网页排名'])
pagerank_df['网页排名'] = pd.Series(nx.pagerank(graph))
pagerank_df.sort_values('网页排名', ascending=False)[:10].plot(kind='barh')
plt.show()
```

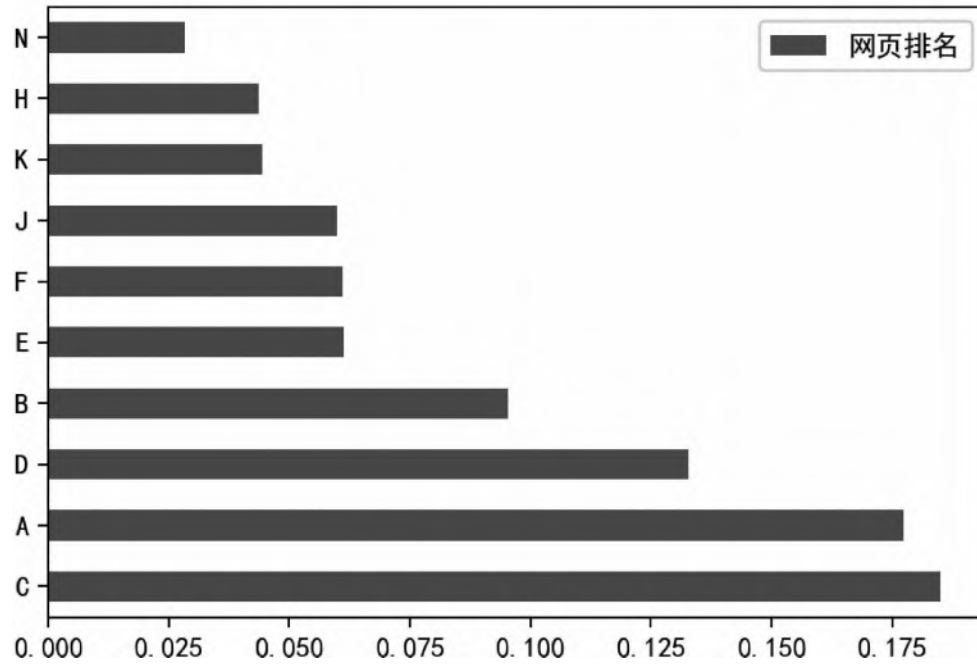


图3-7 网页排名展示图

社交网络可视化代码如下：

```

plt.figure(figsize = (20, 10))
node_color = [graph.degree(v) for v in graph]
# 节点颜色由节点的度决定
node_size = [5000 * nx.degree_centrality(graph)[v] for v in graph]
# 节点的大小由 degree centrality 决定
edge_width = [0.2 * graph[u][v]['Weight'] for u, v in graph.edges()]
# 边的宽度由权重决定
pos = nx.spring_layout(graph)
nx.draw_networkx(graph, pos, node_size = node_size, node_color = node_color, alpha = 0.7,
with_labels=False, width= edge_width)
top = degree_df.sort_values('度中心性', ascending = False)[: 10]
# 最重要的 top10 人物
top = top.index.values.tolist()
labels = {role: role for role in top}
# 创建 label, 不同的度量方法. 标签不一样
nx.draw_networkx_labels(graph, pos, labels = labels, font_size = 20)
# 添加 label
plt.show()
nx.write_gexf(graph, '度中心性.gexf')

```

社交网络可视化结果，如图3-8所示。

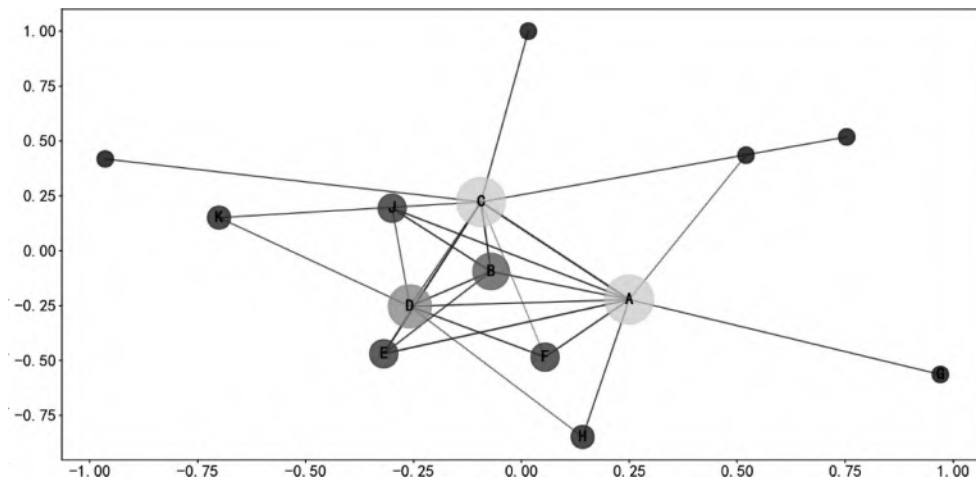


图3-8 社交网络可视化结果展示图

最小社区探测代码如下：

```

k = 3
klist = list(nx.k_clique_communities(graph,k))
print ("生成的社区数: %d" % len(klist))
plt.figure(figsize=(20, 10))
nx.draw_networkx(graph, pos = pos, nodelist = klist[0], node_size=node_size, node_color
= 'r', alpha=0.7, with_labels=False, width=edge_width)
nx.draw_networkx(graph, pos = pos, nodelist = klist[1], node_size=node_size, node_color
= 'y', alpha=0.7, with_labels=False, width=edge_width)
nx.draw_networkx_labels(graph, pos, labels=labels, font_size=20)
plt.show()

```

最小社区探测可视化结果，如图3-9所示。

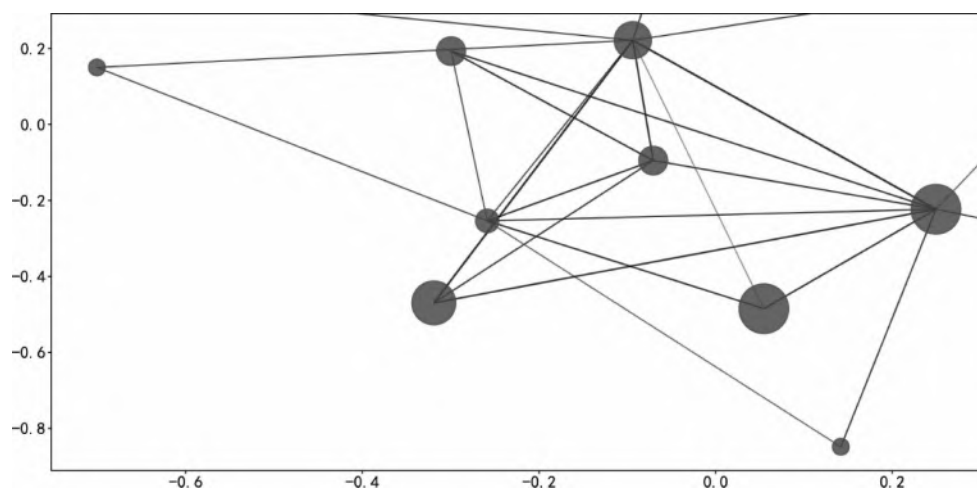


图3-9 最小社区探测可视化展示图

上述代码指定 $k=3$ ，即一个社区至少有3个节点两两相连，而这里生成了两个社区。在金融领域，社区探测法常用于寻找欺诈团体。

3.5 JSON解析

在进行数据分析时，从数据库中导出的数据，可能是JSON格式的。那么JSON是什么呢？JSON是JavaScript Object Notation的缩写，是一种轻量级的数据交换格式，或者说是一个像字典一样的字符串。下面以读取1个JSON文件为例，向大家进行展示。

```
import json
with open('food.json', 'r') as f:
    data = json.load(f)
```

预览结果如图3-10所示。

```
In [34]: data[0]
Out[34]: {'description': 'Cheese, caraway',
          'group': 'Dairy and Egg Products',
          'id': 1008,
          'manufacturer': '',
          'nutrients': [{'description': 'Protein',
                               'group': 'Composition',
                               'units': 'g',
                               'value': 25.18},
                        {'description': 'Total lipid (fat)',
                               'group': 'Composition',
                               'units': 'g',
                               'value': 29.2},
                        {'description': 'Carbohydrate, by differenc',
                               'group': 'Composition',
                               'units': 'g',
                               'value': 3.06},
                        {'description': 'Ash', 'group': 'Other', 'u',
                        {'description': 'Energy',
                               'group': 'Energy',
```

图3-10 food.json数据预览展示图

以上就是文件food.json里面的内容。通常，理想的格式是像pandas那样标准的格式，该怎么处理呢？

仔细观察上面的JSON文件，可以发现5个主键：description、group、id、manufacturer和nutrients。其中前4个键的取值是字符串，但nutrients的取值是一个列表，列表中包含的是另一个JSON格式的数据。

基于上述规律，先处理单独的4个主键（description、group、id和manufacturer），再处理主键nutrients的内容。

```

keys = ['id', 'description', 'group']
keys_df = pd.DataFrame(data, columns = keys)
nutrients = []
for i in data:
    temp = pd.DataFrame(i['nutrients'])
    temp['id'] = i['id']
    nutrients.append(temp)
nutrients = pd.concat(nutrients, ignore_index = True)

```

最终结果如图3-11所示。

```

In [36]: nutrients.head()

```

	description	group	units	value	id
0	Protein	Composition	g	25.18	1008
1	Total lipid (fat)	Composition	g	29.20	1008
2	Carbohydrate, by difference	Composition	g	3.06	1008
3	Ash	Other	g	3.28	1008
4	Energy	Energy	kcal	376.00	1008

图3-11 解析food.json中的nutrients数据

下面处理略微复杂一些的JSON。

```

with open('report.json', 'r') as f:
    data = json.load(f)

```

【注意】

这个JSON文件涉及敏感信息，所以本书仅作代码示例，不提供文件示例。

预览结果如图3-12所示。

```

In [40]: data[0]

```

```

Out[40]: {'create_time': 1504088427000,
          'id': 1233,
          'modify_time': None,
          'mongo_id': None,
          'report_data': '{"trip_analysis":[{"called_cnt":"620","talk_s
led_seconds":"58635","receive_cnt":"0","date_distribution":["2
-03"],"detail":[{"called_cnt":"151","talk_seconds":"29980","ta
1","receive_cnt":"0","month":"2017-08","call_cnt":"204","unkn
{"called_cnt":90,"talk_seconds":14568,"talk_cnt":186,"msg_cnt'
7-07","call cnt":96,"unknown cnt":0,"send cnt":0,"call seconds

```


图3-12 report.json数据预览展示图

下面展示解析report_data里的trip_analysis字段的代码示例。

```
trip_analysis = []
for i in data:
    str1 = i['report_data']
    dict1 = json.loads(str1)
    temp = pd.DataFrame(dict1['trip_analysis'])
    temp['user_id'] = i['user_id']
    trip_analysis.append(temp)
trip_analysis = pd.concat(trip_analysis, ignore_index = True)
```

最终结果如图3-13所示。

In [43]: `trip_analysis.head(1)`

Out[43]:

	call_cnt	call_seconds	called_cnt	called_seconds	date_distribution
0	847	66569	620	58635	[2017-08, 2017-07, 2017-06, 2017-05, 2017-04, ...]

图3-13 解析report.json中的trip_analysis数据

3.6 OCR文字识别

先给大家看一张带有文字内容的图片，如图3-14所示。

将图片识别成文字，如图3-15所示。

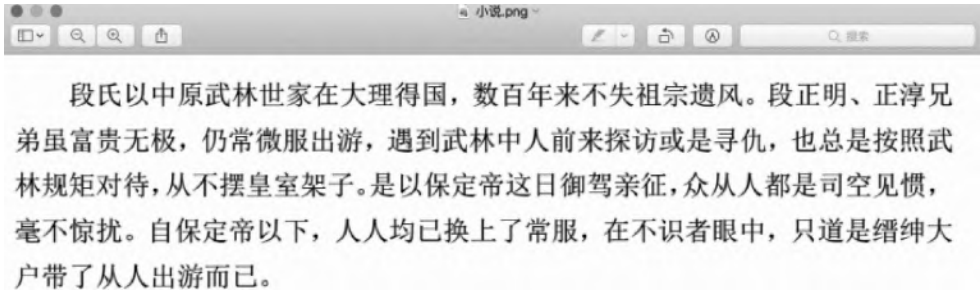


图3-14 小说.png图片展示图

```
zhaikundeMac:~ zhaikun$ python ocr.py
请输入文件名：小说.png
已收到，正在处理，请稍后...
处理完成
zhaikundeMac:~ zhaikun$
```

图3-15 图片识别程序运行展示图

下面是转化后的效果，如图3-16所示。

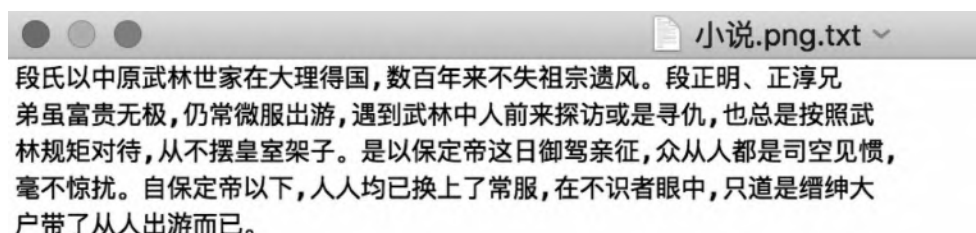


图3-16 图片识别为文字之后的txt文档展示图

是不是很酷！这是怎么做的呢？其实操作很简单，写一段简单的代码，调用百度API即可。

先打开网址：<https://login.bce.baidu.com/>，登录百度账号，如图3-17所示。



图3-17 登录百度云页面展示图

然后单击“创建应用”按钮，就能获取AppID、API Key、Secret Key，如图3-18和图3-19所示。



图3-18 创建文字识别应用

应用名称	AppID	API Key	Secret Key
------	-------	---------	------------

图3-19 应用名称及对应的AppID、API Key、Secret Key展示图

然后安装baidu-aip，安装语句为：`pip install baidu-aip`，安装教程详见附录F。安装好后，代码如下。

```

from aip import AipOcr
import codecs
import os
os.chdir('../data')
def ocr(path):
    with open(path, 'rb') as f:
        return f.read()
def main():
    file_name = str(input('请输入文件名:'))
    print ('已收到,正在处理,请稍后...')
    app_id = '你自己的 id'
    api_key = '你自己的 api_key '
    secret_key = '你自己的 secret_key '
    client = AipOcr(app_id, api_key, secret_key)
    image = ocr(file_name)
    dict1 = client.general(image)
    with codecs.open(file_name + ".txt", "w", "utf-8") as f:
        for i in dict1['words_result']:
            f.write(str(i['words']) + "\r\n")
    print ('处理完成')
if __name__ == '__main__':
    main()

```

通过运行以上代码就可以成功调用百度的API，从而进行OCR识别了。除了普通的文字识别，还能进行以下特殊文件识别，如图3-20所示。

- 网络图片文字识别
- 身份证识别
- 银行卡识别
- 驾驶证识别
- 行驶证识别
- 车牌识别
- 营业执照识别
- 表格文字识别
- 通用票据识别

图3-20 百度OCR的其他文件识别展示图

具体介绍的链接地址为：
<https://cloud.baidu.com/doc/OCR/OCR-API.html>。

3.7 pyecharts

ECharts是一个使用JavaScript实现的开源可视化库，可以流畅地运行在PC端和移动设备上，提供直观的、交互丰富的、可高度个性化定制的数据可视化图表，官网链接地址为：<http://echarts.baidu.com/>。用ECharts生成的图的可视化效果非常棒，为了与Python进行对接，同时方便在Python中直接使用数据生成图，国内开发人员开发出了第三方包：pyecharts。

首先安装pyecharts，安装语句为：`pip install pyecharts`，安装教程详见附录F。安装好后，打开Jupyter Notebook。pyecharts有交互效果，所以需要Jupyter Notebook运行以下代码。

先读取数据：

```
import pandas as pd
import os
os.chdir('../data')
data = pd.read_excel('房价与工资.xlsx')
```

数据预览如图3-21所示。

```
In [3]: data.head(3)
```

```
Out[3]:
```

	城市	省份	房屋均价	人均工资
0	北京市	北京	63239	9240
1	重庆市	重庆	10351	5962
2	广州市	广东	30894	7409

图3-21 房价与工资.xlsx数据预览展示图

下面指定 x 轴和 y 轴：

```
x = list(data['城市'])
y1 = list(data['房屋均价'])
y2 = list(data['人均工资'])
```

先看如何绘制垂直条形图，代码如下，结果如图3-22所示。

```
from pyecharts import Bar
bar = Bar()

bar.add("房屋均价(元)", x, y1, mark_point=["max", "min"], mark_line=["average"], is_label_show=True)
bar.add("人均工资(元)", x, y2)
bar
```

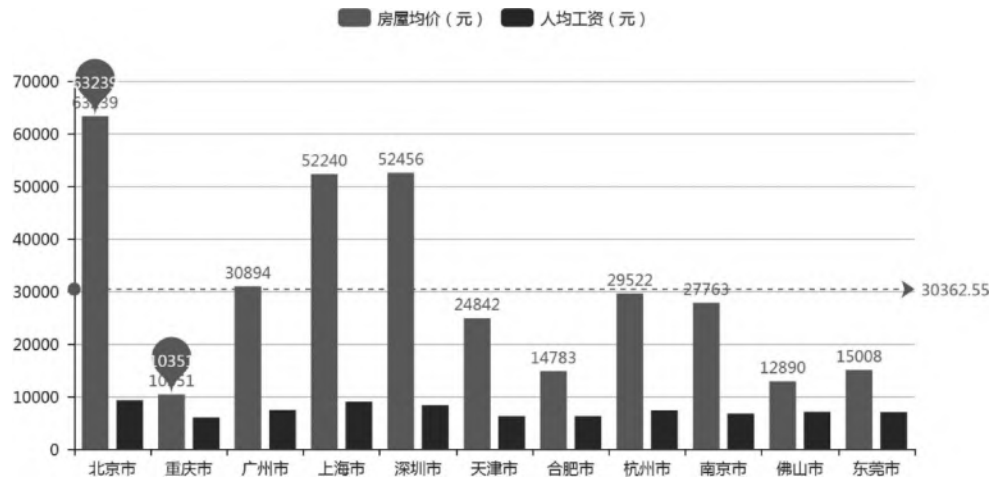


图3-22 房价与工资.xlsx数据垂直条形图

图例是可以交互的，单击图3-22正上方的“房屋均价（元）”，可得如图3-23所示的交互图。



图3-23 房价与工资.xlsx数据交互图

再看如何绘制水平条形图，代码如下，结果如图3-24所示。

```
from pyecharts import Bar
bar = Bar()
bar.add("房屋均价(元)", x, y1)
bar.add("人均工资(元)", x, y2, is_convert = True)
bar
```

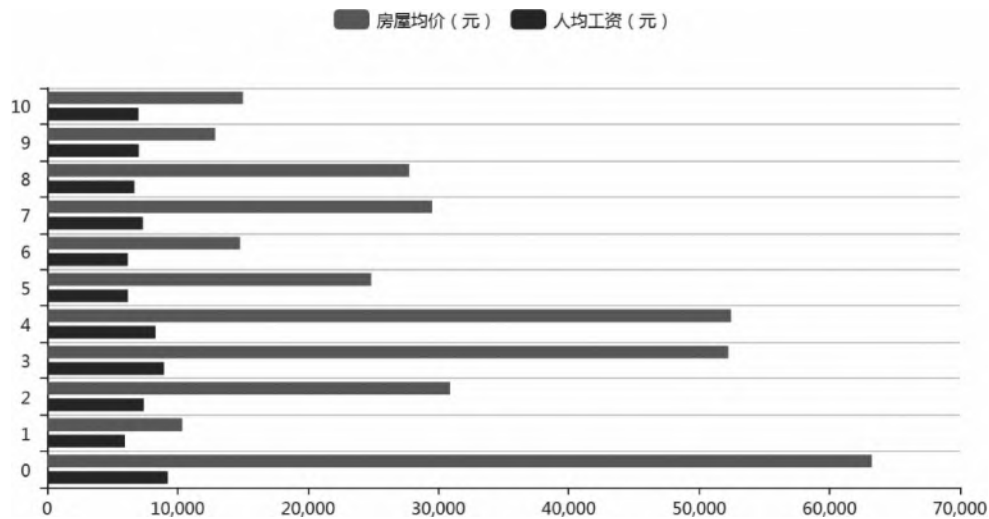


图3-24 房价与工资.xlsx数据水平条形图

再看如何绘制直方图，代码如下，结果如图3-25所示。

```

from pyecharts import Bar
bar = Bar()
bar.add('房屋均价(元)', x, y1, bar_category_gap=10, is_label_show=True)
bar

```

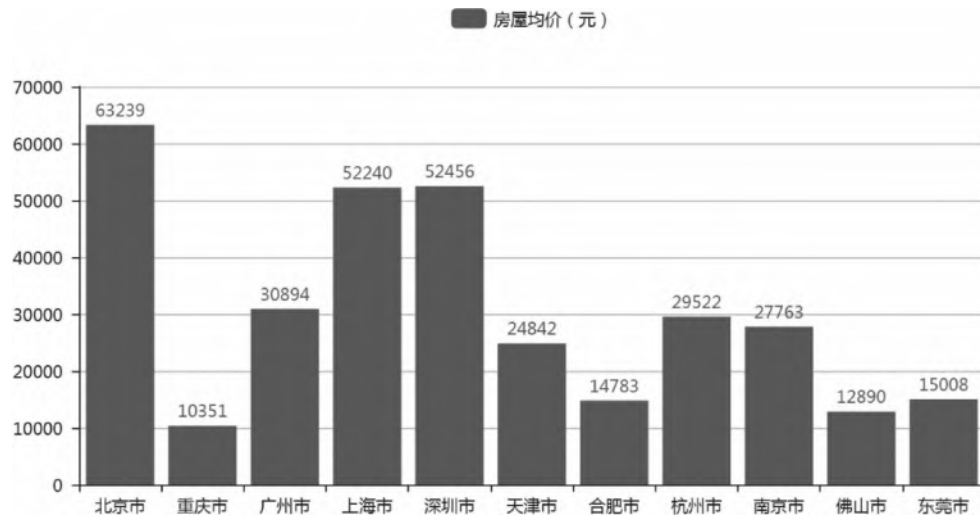


图3-25 房价与工资.xlsx的房屋均价直方图

再看如何绘制折线图，代码如下，结果如图3-26所示。

```

from pyecharts import Line
line = Line()

line.add('房屋均价(元)', x, y1, is_label_show=True)
line.add('人均收入(元)', x, y2)
line

```

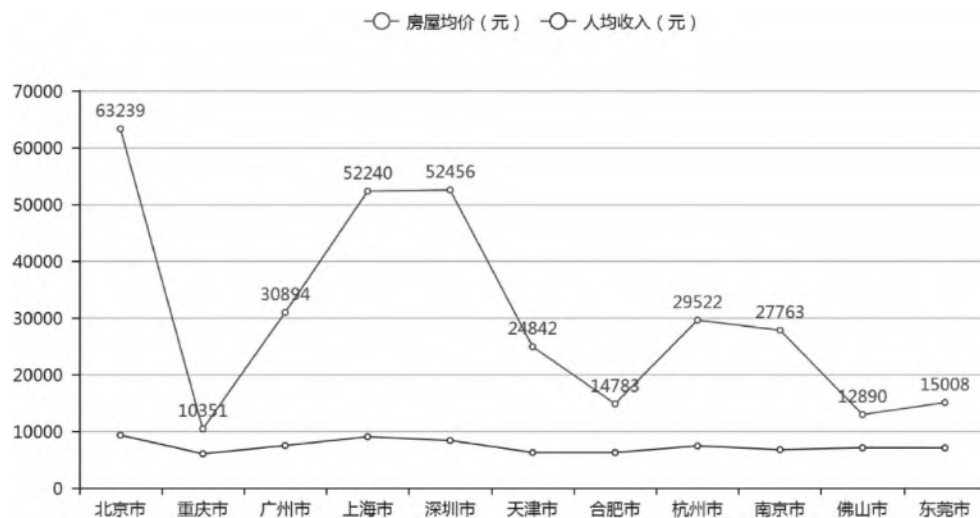


图3-26 房价与工资.xlsx的数据折线图

再看如何绘制饼图，代码如下，结果如图3-27所示。

```
from pyecharts import Pie
pie = Pie()
pie.add('房屋均价(元)', x, y1, is_label_show=True, legend_orient='vertical', legend_pos='left')
pie
```

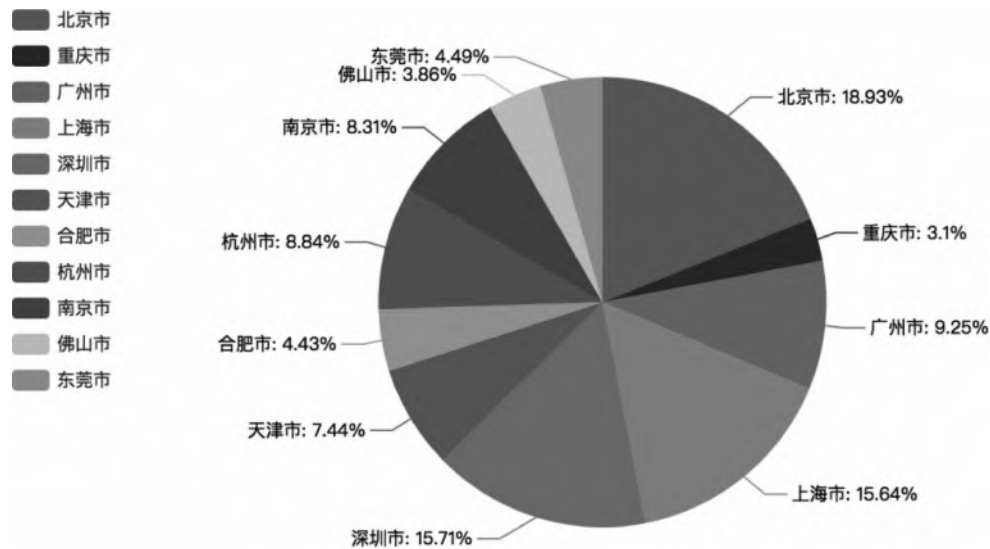


图3-27 房价与工资.xlsx的房屋饼图

也可将条形图和折线图绘制在同一个图形里，代码如下，结果如图3-28所示。

```
from pyecharts import Bar, Line, Grid
bar = Bar()
bar.add("房屋均价(元)", x, y1, is_label_show=True)
bar.add("人均工资(元)", x, y2)
line = Line()
line.add('房屋均价(元)', x, y1, is_label_show=True)
line.add('人均工资(元)', x, y2)
grid = Grid()
grid.add(bar, grid_bottom="60%")
grid.add(line, grid_top="60%")
grid
```

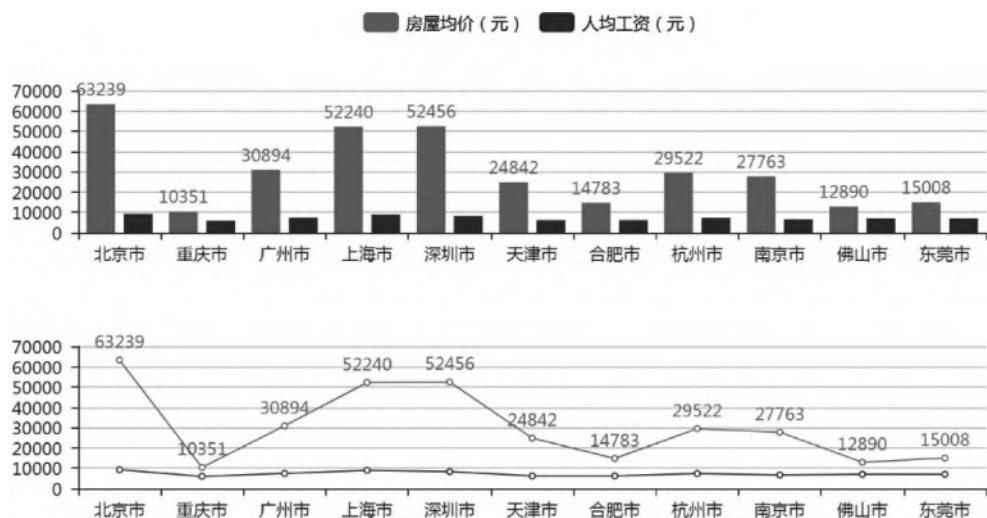


图3-28 房价与工资.xlsx的数据条形图和折线图

在条形图上叠加折线图，代码如下，结果如图3-29所示。

```
from pyecharts import Overlap, Bar, Line, Grid
bar = Bar()
bar.add('房屋均价(元)', x, y1)
bar.add('人均工资(元)', x, y2)
line = Line()
line.add('房屋均价(元)', x, y1, is_label_show=True)
line.add('人均工资(元)', x, y2, is_label_show=True)
overlap = Overlap()
overlap.add(bar)
overlap.add(line)
overlap
```

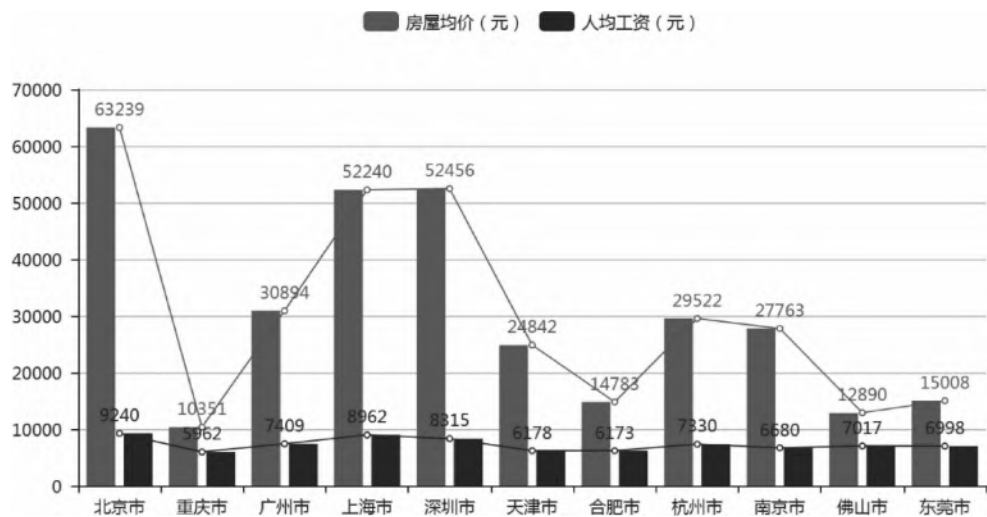


图3-29 房价与工资.xlsx的数据条形图叠加折线图

使用pyecharts也可创建仪表盘图，代码如下，结果如图3-30所示。

```
from pyecharts import Gauge
gauge = Gauge()
gauge.add('', '', '80', scale_range=[0, 100], angle_range=[180, 0])
gauge
```

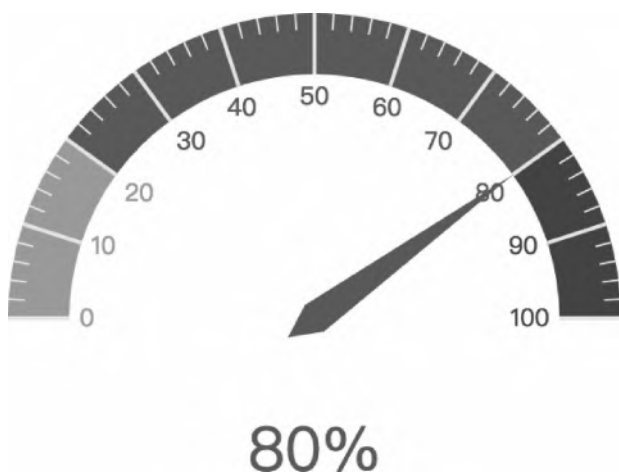


图3-30 pyecharts创建的仪表盘图

使用pyecharts也可创建城市散点图，感兴趣的读者可以尝试运行，看看运行结果。

```
import numpy as np
temp = np.array(data[['城市', '房屋均价']]).tolist()
a = []
for i in temp:
    z = tuple(i)
    a.append(z)
from pyecharts import Geo

geo = Geo("全国主要城市房价散点图", title_color="#fff", title_pos="center", width=800, height=600, background_color='#404a59')
city, value = geo.cast(a)
geo.add("", city, value, visual_range=[10000, 50000], visual_text_color="#fff", symbol_size=15, is_visualmap=True)
geo
```

使用pyecharts也可创建城市散点涟漪图，代码如下：

```
from pyecharts import Geo
geo = Geo("全国主要城市房价散点涟漪图", title_color = "#fff", title_pos = "center",
width=800, height=600, background_color = '#404a59')
city, value = geo.cast(a)
geo.add("", city, value, type = "effectScatter", is_random = True, effect_scale = 5)
geo
```

使用pyecharts也可创建热力图，代码如下：

```
from pyecharts import Geo
geo = Geo("全国主要城市房价热力图", title_color = "#fff", title_pos = "center", width =
800, height=600, background_color = '#404a59')
city, value = geo.cast(a)
geo.add("", city, value, type = "heatmap", is_visualmap = True, visual_range = [10000,
20000], visual_text_color = '#fff')
geo
```

使用pyecharts也可创建全国省份地图，代码如下：

```
from pyecharts import Map
city = list(data['省份'])
value = list(data['房屋均价'])
map = Map( width=800, height=600)
map.add("", city, value, matype = 'china', is_label_show = True, is_visualmap = True,
visual_range = [5000, 60000], visual_text_color = '#000')
map
```

使用pyecharts也可创建指定省份城市图，代码如下：

```
from pyecharts import Map
city = np.array(data['城市']).tolist()
value = np.array(data['房屋均价']).tolist()
map = Map( width=800, height=600)
map.add("",city,value,matype = '广东', is_label_show = True, is_visualmap = True, visual_
range = [10000, 20000],visual_text_color = '#000')
map
```

从上面的各种图形中可以看出，使用pyecharts作的图确实很漂亮，以后做数据分析展示图的时候，可以尝试使用pyecharts进行作

图，然后将这些图片放到PPT报告里，从而提高报告的展现力。

3.8 stats简单统计分析

Python虽然不像R语言那样，专为统计而生，但是它也可以进行统计分析与检验。scipy库中的stats包就能实现大多数的统计功能。

学生的考试成绩会受到多种因素的影响，如：是否分场、试卷难易程度和临场发挥情况。本次仅考虑“是否分场”这一因素。下面以北京市日坛实验中学的初三生物考试成绩为例，分析在考场合并前后，学生的成绩有无显著差异。

```
import pandas as pd
import matplotlib.pyplot as plt
from pylab import mpl
mpl.rcParams['font.sans-serif'] = ['SimHei'] # 指定默认字体
mpl.rcParams['axes.unicode_minus'] = False # 解决保存图像是负号'-'显示为方块的问题
import os
os.chdir('/Users/zhaikun/Desktop')
data = pd.read_excel('成绩.xls')
```

下面预览一下数据，如图3-31所示。

In [4]: data.head()

Out[4]:

	分场1	分场2	分场3	分场4	分场_月考	合场1	合场2	合场3	合场_期中
0	27	26	29	38	30	24	29	37	32
1	30	38	38	40	45	38	38	36	44
2	43	49	47	46	43	48	48	42	50
3	30	34	29	41	32	36	29	38	15
4	39	45	40	44	48	50	40	40	48

图3-31 初三生物考试成绩数据预览图

首先计算出每个分场的考试成绩平均值，再计算合并考场之后的成绩平均值，代码如下所示。

```
data['分场平均值'] = data.ix[:,0:5].mean(1)
data['合场平均值'] = data.ix[:,5:9].mean(1)
temp = data[['分场平均值', '合场平均值']]
temp.plot()
plt.show()
```

计算考试成绩平均值之后，可以通过可视化，查看每个考生的成绩差异，结果如图3-32所示。

```
In [10]: temp.plot()
plt.show()
```

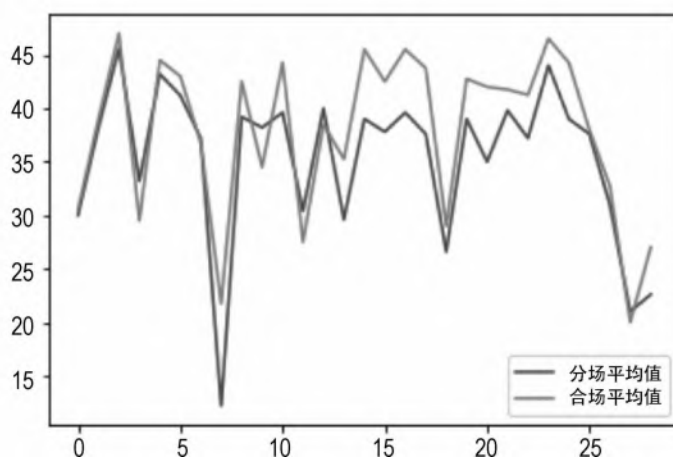


图3-32 生物考试成绩差异图

由图3-32可以发现，合并考场之后，考试平均成绩明显比分场考试平均成绩高。

下面进行“相关样本的t检验”，原假设是样本均值相等，即考生在考场合并前后的考试成绩应该是相等的。代码如下所示，得到的结果如图3-33所示。

```
from scipy import stats
stats.ttest_rel(temp['分场平均值'], temp['合场平均值'])
```

```
In [11]: from scipy import stats
stats.ttest_rel(temp['分场平均值'], temp['合场平均值'])
```

```
Out[11]: Ttest_relResult(statistic=-4.0781602275865065, pvalue=0.0003406411787764341)
```

图3-33 t检验结果图

从计算结果中，可以发现 $P < 0.05$ ，即原假设不成立，就是考场合并前后，考生的考试成绩均值差异显著。

stats包除了可以进行t检验以外，还可以进行卡方检验、方差分析等统计上常用的检验手段，这里不进行详细介绍，感兴趣的读者可自行查阅相关资料。

3.9 小结

本章主要介绍了与Python语言相关的实际应用，这些应用均使用了第三方包，主要包括数据的连接、词云图、社交网络、JSON解析、OCR文字识别以及pyecharts绘图等。通过这些实际应用，展示了Python作为“胶水语言”的魅力，感兴趣的读者可以查找相关资料，进行进一步的学习与探索。

第4章 异常样本识别

什么是不平衡数据？比如，10万个普通人中找出10个恐怖分子，10万个正常还款的客户中找出10个欺诈客户，这些都属于不平衡数据。由于极少数异常样本可能带来很大的损失，在工作中，主要任务就是要在这些不平衡数据中找到那些占比极少的异常样本。

4.1 逻辑回归、交叉验证与欠采样

首先，导入包和加载数据：

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import os
os.chdir('../data')
data = pd.read_csv('data.csv')
```

在Jupyter Notebook中运行代码，其数据预览结果如图4-1所示。

图4-1 data.csv数据预览展示图

这个数据集是284807行，31列的数据。


```
data['Class'].value_counts()
0      284315
1         492
Name: Class, dtype: int64
```

目标变量'Class' 非常不平衡，异常客户数据只有492个，异常占比为 $492 \div 284807 = 1.73\%$ 。

先对变量列'Amount'，'Time' 进行标准化处理。

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
data[['Amount', 'Time']] = sc.fit_transform(data[['Amount', 'Time']])
```

如果对数据不进行任何处理，而直接用逻辑回归建模，来看看模型效果。

```
allFeatures = list(data.columns)
allFeatures.remove('Class')
X = data[allFeatures]
y = data['Class']
from sklearn.cross_validation import train_test_split as sp
X_train, X_test, y_train, y_test = sp(X, y, test_size=0.3, random_state=1)
from sklearn.linear_model import LogisticRegression as LR
lr = LR()
lr.fit(X_train, y_train)
from sklearn import metrics
y_test_label = lr.predict(X_test)
y_test_value = lr.predict_proba(X_test)[: , 1]
print("测试集准确率是: {:.2 %}".format(metrics.accuracy_score(y_test, y_test_label)))
print("测试集 AUC 是: {:.4}".format(metrics.roc_auc_score(y_test, y_test_value)))
from sklearn.metrics import classification_report
print(classification_report(y_test, y_test_label))
```

运行结果如下：

```
测试集准确率是: 99.92 %
测试集 AUC 是: 0.9657
```

	precision	recall	f1 - score	support
0	1.00	1.00	1.00	85308
1	0.84	0.58	0.68	135
avg / total	1.00	1.00	1.00	85443

看起来效果还不错，但是能进行优化吗？

这里，首先要明白目标是什么？衡量分类器的主要指标是什么？在这个不平衡的异常样本检测案例中，衡量的指标主要是recall，以此来衡量，模型表现显然不是很好。

那么关于不平衡样本，可以试试欠采样的方法。这里要使用到第三方包imblearn，安装语句为：pip install imblearn。

```
allFeatures = list(data.columns)
allFeatures.remove('Class')
X = data[allFeatures]
y = data['Class']
n_pos_sample = y[y == 0].shape[0]
n_neg_sample = y[y == 1].shape[0]
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(ratio = {0:(4 * n_neg_sample),1:(n_neg_sample)},random_state = 1)
X, y = rus.fit_sample(X, y)
data_X = pd.DataFrame(X, columns = [allFeatures])
data_y = pd.DataFrame(y, columns = ['target'])
data = pd.concat([data_X,data_y], axis = 1)
```

在这个重新构成的新样本中，目标变量target的水平分布是：

```
data['target'].value_counts()
0    1968
1     492
Name: target, dtype: int64
```

下面重新建模：

```

X = data[allFeatures]
y = data['target']
from sklearn.cross_validation import train_test_split as sp
X_train, X_test, y_train, y_test = sp(X, y, test_size=0.3, random_state=1)
from sklearn.linear_model import LogisticRegression as LR

lr = LR()
lr.fit(X_train, y_train)
from sklearn import metrics
y_test_label = lr.predict(X_test)
y_test_value = lr.predict_proba(X_test)[:, 1]
print("测试集准确率是: {:.2%}".format(metrics.accuracy_score(y_test, y_test_label)))
print("测试集 AUC 是: {:.4}".format(metrics.roc_auc_score(y_test, y_test_value)))
from sklearn.metrics import classification_report
print(classification_report(y_test, y_test_label))

```

运行结果如下：

```

测试集准确率是:97.15 %
测试集 AUC 是:0.9805

```

	precision	recall	f1 - score	support
0	0.97	1.00	0.98	588
1	0.98	0.87	0.93	150
avg / total	0.97	0.97	0.97	738

召回率recall是0.87，模型表现有了很大的提升。

下面再试试5折交叉验证：

```

from sklearn.cross_validation import KFold
from sklearn.metrics import recall_score
fold = KFold(len(y_train), 5, shuffle=False)
c_param_range = [0.01, 0.1, 1, 10, 100]
results_table = pd.DataFrame(columns = ['C 值', '平均召回率得分'])
results_table['C 值'] = c_param_range
j = 0
for c_param in c_param_range:
    print('C 值: ', c_param)
    recall_accs = []
    for iteration, indices in enumerate(fold, start=1):
        lr = LR(C = c_param, penalty = 'l1')
        X_train = X_train.reset_index(drop=True)
        y_train = y_train.reset_index(drop=True)
        lr.fit(X_train.iloc[indices[0],:], y_train.iloc[indices[0]].values.ravel())
        y_pred = lr.predict(X_train.iloc[indices[1],:].values)
        recall_acc = recall_score(y_train.iloc[indices[1]].values, y_pred)
        recall_accs.append(recall_acc)
    print('迭代次数', iteration, ':召回率得分 = ', recall_acc)
    results_table.ix[j, '平均召回率得分'] = np.mean(recall_accs)
    j += 1
    print('平均召回率得分: ', np.mean(recall_accs))
best_c = results_table.loc[results_table['平均召回率得分'].idxmax()]['C 值']
print('交叉验证最好的 C 值是', best_c)

```

运行结果如下：

```
C 值:0.01
迭代次数 1:召回率得分 = 0.7972972972972973
迭代次数 2:召回率得分 = 0.8769230769230769
迭代次数 3:召回率得分 = 0.875
迭代次数 4:召回率得分 = 0.7878787878787878
迭代次数 5:召回率得分 = 0.8461538461538461
平均召回率得分:0.8366506016506016
C 值:0.1
迭代次数 1:召回率得分 = 0.8243243243243243
迭代次数 2:召回率得分 = 0.9076923076923077
迭代次数 3:召回率得分 = 0.875
迭代次数 4:召回率得分 = 0.8484848484848485
迭代次数 5:召回率得分 = 0.8615384615384616
平均召回率得分:0.8634079884079885
C 值:1
迭代次数 1:召回率得分 = 0.8243243243243243
迭代次数 2:召回率得分 = 0.9230769230769231
迭代次数 3:召回率得分 = 0.875
迭代次数 4:召回率得分 = 0.8484848484848485
迭代次数 5:召回率得分 = 0.8923076923076924
平均召回率得分:0.8726387576387576
C 值:10
迭代次数 1:召回率得分 = 0.8378378378378378
迭代次数 2:召回率得分 = 0.9230769230769231
迭代次数 3:召回率得分 = 0.8888888888888888
迭代次数 4:召回率得分 = 0.8787878787878788
迭代次数 5:召回率得分 = 0.8923076923076924
平均召回率得分:0.8841798441798442
C 值:100
迭代次数 1:召回率得分 = 0.8378378378378378
迭代次数 2:召回率得分 = 0.9230769230769231
迭代次数 3:召回率得分 = 0.8888888888888888
迭代次数 4:召回率得分 = 0.8787878787878788
迭代次数 5:召回率得分 = 0.8923076923076924
平均召回率得分:0.8841798441798442
交叉验证最好的 C 值是 10.0.
```

下面把用交叉验证找到的C值用于模型中，先用欠采样后的数据训练模型，再把训练好的模型用于原始数据。

```

lr = LR(C = best_c, penalty = 'l1')

lr.fit(X_train, y_train)
data = pd.read_csv('data.csv')
allFeatures = list(data.columns)
allFeatures.remove('Class')
X = data[allFeatures]
y = data['Class']
y_label = lr.predict(X)
print(classification_report(y, y_label))

```

运行结果如下：

	precision	recall	f1 - score	support
0	1.00	0.99	1.00	284315
1	0.17	0.89	0.29	492
avg / total	1.00	0.99	1.00	284807

召回率recall有很大的提升，达到了0.89。

```

y_value = lr.predict_proba(X)[: , 1]
print("AUC 是: {:.4}".format(metrics.roc_auc_score(y, y_value)))

```

运行结果如下：

```
AUC 是:0.9855
```

可以看出模型的表现很好。

在这个不平衡样本案例中，可以学到以下两点。

(1) 在不平衡的数据集中，准确率不再是衡量的指标。根据场景的不同，衡量指标可能是recall，也可能是precision，还可能是f1-score，不要仅局限于auc和ks。

(2) 对不平衡数据集，可以采用过采样或欠采样的方法，训练出在平衡样本（采样之后的样本）下的模型，然后把模型用于原始的不平衡数据中。

4.2 基于分布的异常样本识别

除了欠采样的异常样本识别，还可以根据变量取值的分布情况，进行异常样本识别，代码如下：

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import os
os.chdir('../data')
dataset = pd.read_csv('data.csv')
'''连续单变量密度分布图'''
col = ['Amount', 'Time']
for i in col:
    plt.figure(figsize = (5,5))
    sns.distplot(dataset[i].dropna())
    plt.show()
```

运行结果如图4-2所示。

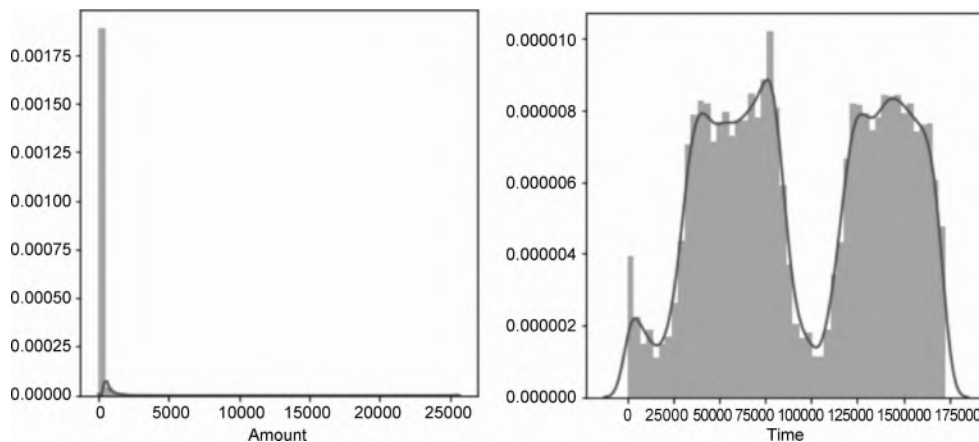


图4-2 Amount和Time变量的密度分布图

从图4-2中可以看出数据分布明显是偏斜的，那么作变量进行对数转换。

```
dataset['Amount'] = np.log(dataset['Amount'] + 1)
dataset['Time'] = np.log(dataset['Time'] + 1)
```

对数转换后的分布图，如图4-3所示。

先来回顾一下目标变量水平分布。

```
dataset['Class'].value_counts()
0    284315
1      492
Name: Class, dtype: int64
```

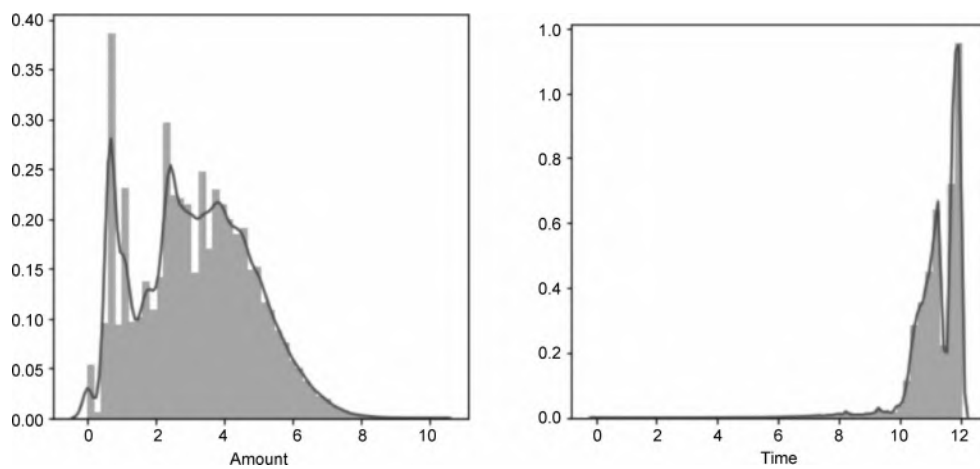


图4-3 Amount和Time变量的对数密度分布图

由目标变量水平分布可以看出，Class等于1的样本数占比为 $492 \div 284807 \approx 0.1727\%$ ，是严重不平衡数据。

下面进行数据集切分：

```
normal = dataset[dataset['Class'] == 0]
anomaly = dataset[dataset['Class'] == 1]
from sklearn.model_selection import train_test_split as sp
train, test = sp(normal, test_size=0.2)
normal_valid, normal_test = sp(test, test_size=0.5)
anomaly_valid, anomaly_test = sp(anomaly, test_size=0.5)
train = train.reset_index(drop=True)
valid = pd.concat([normal_valid, anomaly_valid])
valid = valid.reset_index(drop=True)
test = pd.concat([normal_test, anomaly_test])
test = test.reset_index(drop=True)
```


看一下各数据集形状：

```
train.shape      # (227452, 31)
valid.shape      # (28677, 31)
test.shape       # (28678, 31)
#####
# Step 1:基于高斯分布的无监督异常值检测
#####
'''计算协方差矩阵和均值'''
from scipy.stats import multivariate_normal
mu = train.drop('Class', axis=1).mean(axis=0).values
sigma = train.drop('Class', axis=1).cov().values

# 多元正态分布,指定协方差、均值,允许奇异协方差矩阵
model = multivariate_normal(cov=sigma, mean=mu, allow_singular=True)
# logpdf:概率密度函数对数
print(np.median(model.logpdf(valid[valid['Class'] == 0].drop('Class', axis=1).values)))
print(np.median(model.logpdf(valid[valid['Class'] == 1].drop('Class', axis=1).values)))
```

运行结果如下：

```
- 33.91079260533387
- 716.0503164497757
```

这里解释一下上面代码的含义。

- `multivariate_normal`是多元正态分布，指定协方差、均值，同时允许奇异协方差矩阵。`mu`是训练集的均值，`sigma`是训练集的协方差。
- `logpdf`是计算概率密度函数的对数，这里打印出中位数，方便指定下面代码中的阈值范围。

```
'''计算不同阈值下的 recall_score、precision_score 和 fbeta_score'''
from sklearn.metrics import recall_score, precision_score, fbeta_score
thresholds = np.linspace(-1000, -10, 150)
scores = []
for threshold in thresholds:
    y_valid_lable = (model.logpdf(valid.drop('Class', axis=1)) < threshold).astype(int)
    scores.append([recall_score(valid['Class'], y_valid_lable), precision_score(valid
['Class'],
                                y_valid_lable),fbeta_score(valid['Class'], y_valid_lable, beta=2)])
scores = np.array(scores)
print(scores[:, 2].max(), scores[:, 2].argmax())
```

这里解释一下各分数，recall_score是计算召回率分数，precision_score是计算命中率分数，fbeta_score是精度和召回率的加权调和平均值。

运行结果如下：

```
0.7723250201126307 110
```

可视化不同阈值下的 recall_score、precision_score 和 fbeta_score，代码如下。

```
'''可视化不同阈值下的 recall_score、precision_score 和 fbeta_score'''
plt.plot(thresholds, scores[:, 0], label = '$ Recall $ ')
plt.plot(thresholds, scores[:, 1], label = '$ Precision $ ')
plt.plot(thresholds, scores[:, 2], label = '$ F_2 $ ')

plt.ylabel('Score')
plt.xlabel('Threshold')
plt.legend(loc = 'best')
plt.show()
```

可视化结果如图4-4所示。

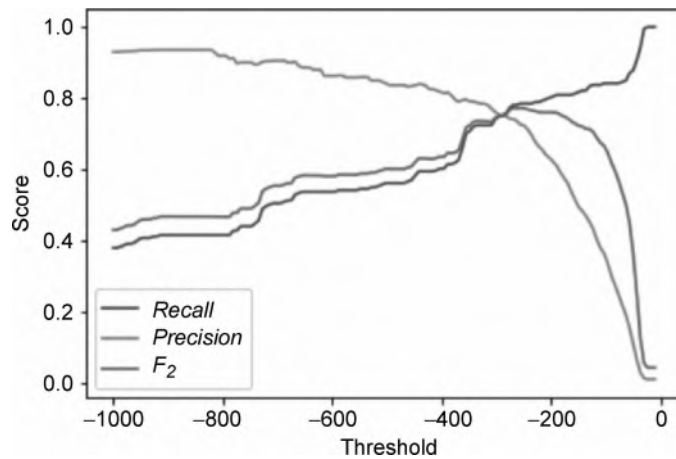


图4-4 不同阈值下的recall、precision、f2可视化图

```
'''计算测试集评价标准'''
final_tresh = thresholds[scores[:, 2].argmax()]
y_test_label = (model.logpdf(test.drop('Class', axis = 1).values) < final_tresh).astype(
    int)
print('最终阈值是:%d' % final_tresh)
print('测试集召回率分数是:%.3f' % recall_score(test['Class'], y_test_label))
print('测试集命中率分数是:%.3f' % precision_score(test['Class'], y_test_label))
print('测试集 F2 分数是:%.3f' % fbeta_score(test['Class'], y_test_label, beta = 2))
```

运行结果如下：

```
最终阈值是：-269
测试集召回率分数是：0.772
测试集命中率分数是：0.704
测试集 F2 分数是：0.758
```

混淆矩阵可视化，代码如下。

```
'''混淆矩阵可视化'''
import itertools
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, classes, title = '混淆矩阵', cmap = plt.cm.Blues):
    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
    plt.colorbar()

    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 0)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 3.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j], horizontalalignment = "center", color = "white" if cm[i, j] >
            thresh else "black")

    plt.tight_layout()
    plt.ylabel('实际')
    plt.xlabel('预测')

cnf_matrix = confusion_matrix(test['Class'], y_test_label)
class_names = [0,1]
plot_confusion_matrix(cnf_matrix, classes = class_names)
```

运行结果如图4-5所示。

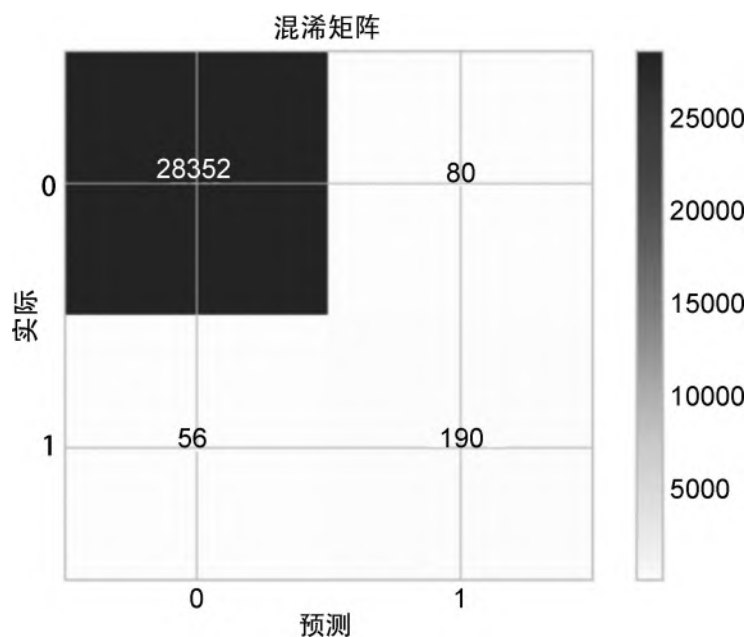


图4-5 混淆矩阵图

整体来说，样本的召回率效果还是不错的，在这个极度不平衡的数据集中，找到了190个异常样本。

```
#####
# Step 2: 基于混合高斯模型的无监督异常值检测
#####
'''混合高斯模型'''
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3, n_init=4, random_state=1)
gmm.fit(train.drop('Class', axis=1).values)
print(gmm.score(valid[valid['Class'] == 0].drop('Class', axis=1).values))
print(gmm.score(valid[valid['Class'] == 1].drop('Class', axis=1).values))
```

运行结果如下：

```
6.8138687337843695
-132514.22047425093
```

下面的方法、原理与高斯模型一致。

```
'''计算不同阈值下的 recall_score, precision_score 和 fbeta_score'''
thresholds = np.linspace(-140000, 20, 1000)
scores = []
y_pred_score = gmm.score_samples(valid.drop('Class', axis=1).values)
for treshhold in thresholds:
    y_valid_label = (y_pred_score < treshhold).astype(int)
    scores.append([recall_score(valid['Class'], y_valid_label), precision_score(valid
['Class'],
                                y_valid_label), fbeta_score(valid['Class'], y_valid_label, beta=2)])
scores = np.array(scores)
print(scores[:, 2].max(), scores[:, 2].argmax()) # fbeta 最大为 0.81, 索引是 68
'''可视化不同阈值下的 recall_score, precision_score 和 fbeta_score'''
plt.plot(thresholds, scores[:, 0], label=' $ Recall $ ')
plt.plot(thresholds, scores[:, 1], label=' $ Precision $ ')
plt.plot(thresholds, scores[:, 2], label=' $ F_2 $ ')
plt.ylabel('Score')
plt.xlabel('Threshold')
plt.legend(loc='best')
plt.show()
'''计算测试集评价标准'''
final_tresh = thresholds[scores[:, 2].argmax()]
y_test_label = (gmm.score_samples(test.drop('Class', axis=1).values) < final_tresh).
astype(int)
print('最终阈值是: %d' % final_tresh)
print('测试集召回率分数是: %.3f' % recall_score(test['Class'], y_test_label))
print('测试集命中率分数是: %.3f' % precision_score(test['Class'], y_test_label))
print('测试集 F2 分数是: %.3f' % fbeta_score(test['Class'], y_test_label, beta=2))
cnf_matrix = confusion_matrix(test['Class'].values, y_test_label)
plot_confusion_matrix(cnf_matrix, classes=class_names)
```

最终运行结果如图4-6所示。

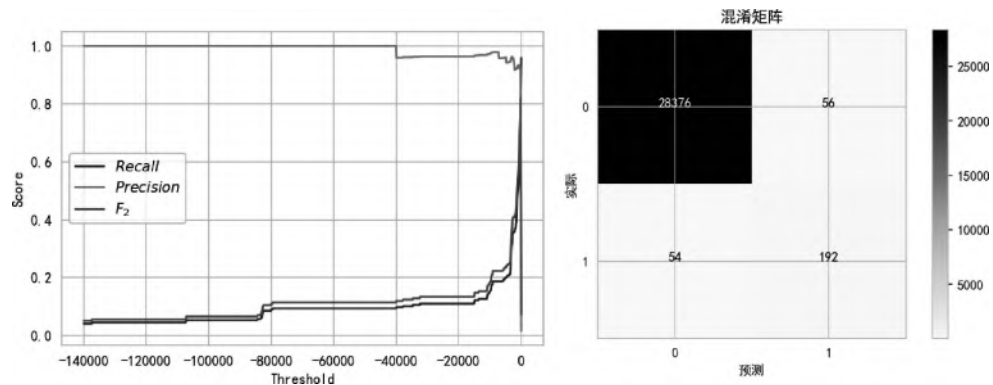


图4-6 混合高斯模型下的相关结果展示图

下面再介绍孤立森林和直方图的异常值检测方法，代码如下：

```
#####
# Step 3: 基于孤立森林的无监督异常值检测
#####
'''孤立森林模型'''
from sklearn.ensemble import IsolationForest
model = IsolationForest(n_estimators = 50, bootstrap = True, max_samples = train.shape[0],
                        n_jobs = 4, random_state = 1)
model.fit(train.drop('Class', axis = 1).values)
print(model.decision_function(valid[valid['Class'] == 0].drop('Class', axis = 1).values).
      mean())
print(model.decision_function(valid[valid['Class'] == 1].drop('Class', axis = 1).values).
      mean())
'''计算不同阈值下的 recall_score, precision_score 和 fbeta_score'''
thresholds = np.linspace(-0.2, 0.2, 200)
scores = []
y_pred_score = model.decision_function(valid.drop('Class', axis = 1).values)
for threshold in thresholds:
    y_valid_label = (y_pred_score < threshold).astype(int)
    scores.append([recall_score(valid['Class'], y_valid_label), precision_score(valid['Class'],
                                     y_valid_label), fbeta_score(valid['Class'], y_valid_label, beta = 2)])
scores = np.array(scores)
print(scores[:, 2].max(), scores[:, 2].argmax())
'''可视化不同阈值下的 recall_score, precision_score 和 fbeta_score'''
plt.plot(thresholds, scores[:, 0], label = '$ Recall $ ')
plt.plot(thresholds, scores[:, 1], label = '$ Precision $ ')
plt.plot(thresholds, scores[:, 2], label = '$ F_2 $ ')
plt.ylabel('Score')
plt.xlabel('Threshold')
plt.legend(loc = 'best')
plt.show()
'''计算测试集评价标准'''
final_tresh = thresholds[scores[:, 2].argmax()]
y_test_label = (model.decision_function(test.drop('Class', axis = 1).values) < final_tresh).
               astype(int)
print('最终阈值是: %d' % final_tresh)
print('测试集召回率分数是: %.3f' % recall_score(test['Class'], y_test_label))
print('测试集命中率率分数是: %.3f' % precision_score(test['Class'], y_test_label))
print('测试集 F2 分数是: %.3f' % fbeta_score(test['Class'], y_test_label, beta = 2))
cnf_matrix = confusion_matrix(test['Class'].values, y_test_label)
plot_confusion_matrix(cnf_matrix, classes = class_names)
```

运行结果如图4-7所示。

```
#####
# Step 4: 基于直方图的异常值检测
#####
```

```

'''直方图探索:摆脱了高斯分布的限制,但是没有捕捉到变量之间的关系'''
class hist_model(object):
    def __init__(self, bins = 50):
        self.bins = bins
    def fit(self, X):
        bin_hight, bin_edge = [], []
        for var in X.T:
            bh, bedge = np.histogram(var, bins = self.bins)
            bin_hight.append(bh)
            bin_edge.append(bedge)
        self.bin_hight = np.array(bin_hight)
        self.bin_edge = np.array(bin_edge)
    def predict(self, X):
        scores = []
        for obs in X:
            obs_score = []
            for i, var in enumerate(obs):
                bin_num = (var > self.bin_edge[i]).argmin() - 1
                obs_score.append(self.bin_hight[i, bin_num])
            scores.append(np.mean(obs_score))
        return np.array(scores)

model = hist_model()
# 为每个特征建立直方图,将 bin 的高度合并为分数,如果分数很低,说明异常
model.fit(train.drop('Class', axis = 1).values)
print(np.median(model.predict(valid[valid['Class'] == 0].drop('Class', axis = 1).values)))
print(np.median(model.predict(valid[valid['Class'] == 1].drop('Class', axis = 1).values)))

'''计算不同阈值下的 recall_score、precision_score 和 fbeta_score'''
thresholds = np.linspace(10000, 80000, 100)
scores = []
y_pred_score = model.predict(valid.drop('Class', axis = 1).values)
for threshold in thresholds:
    y_valid_lable = (y_pred_score < threshold).astype(int)
    scores.append([recall_score(valid['Class'], y_valid_lable), precision_score(valid['Class'],
                                                                                y_valid_lable), fbeta_score(valid['Class'], y_valid_lable, beta = 2)])

scores = np.array(scores)
print(scores[:, 2].max(), scores[:, 2].argmax()) # fbeta 最大为 0.42,索引是 42
'''可视化不同阈值下的 recall_score、precision_score 和 fbeta_score'''
plt.plot(thresholds, scores[:, 0], label = '$ Recall $')
plt.plot(thresholds, scores[:, 1], label = '$ Precision $')
plt.plot(thresholds, scores[:, 2], label = '$ F_2 $')
plt.ylabel('Score')
plt.xlabel('Threshold')
plt.legend(loc = 'best')
plt.show()

'''计算测试集评价标准'''

```



```

final_tresh = thresholds[scores[:, 2].argmax()]
y_test_label = (model.predict(test.drop('Class', axis=1).values) < final_tresh).astype(int)
print('最终阈值是:%d' % final_tresh)
print('测试集召回率分数是:%.3f' % recall_score(test['Class'], y_test_label))
print('测试集命中率分数是:%.3f' % precision_score(test['Class'], y_test_label))
print('测试集 F2 分数是:%.3f' % fbeta_score(test['Class'], y_test_label, beta=2))
cnf_matrix = confusion_matrix(test['Class'].values, y_test_label)
plot_confusion_matrix(cnf_matrix, classes=class_names)

```

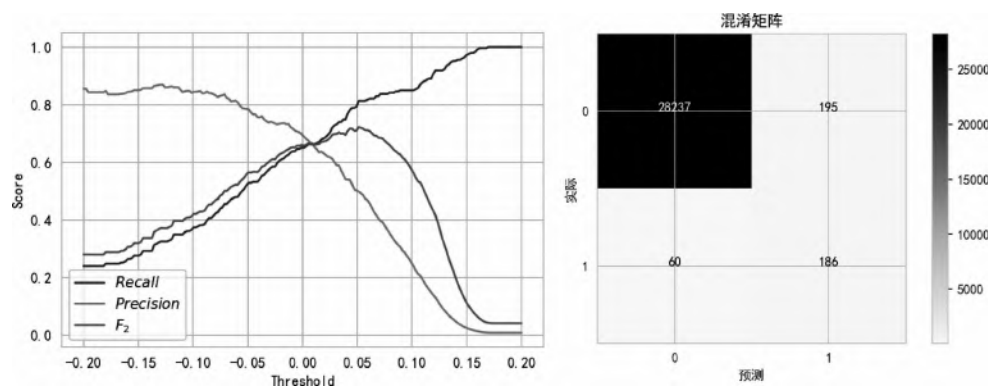


图4-7 孤立森林模型下的相关结果展示图

运行结果如图4-8所示。

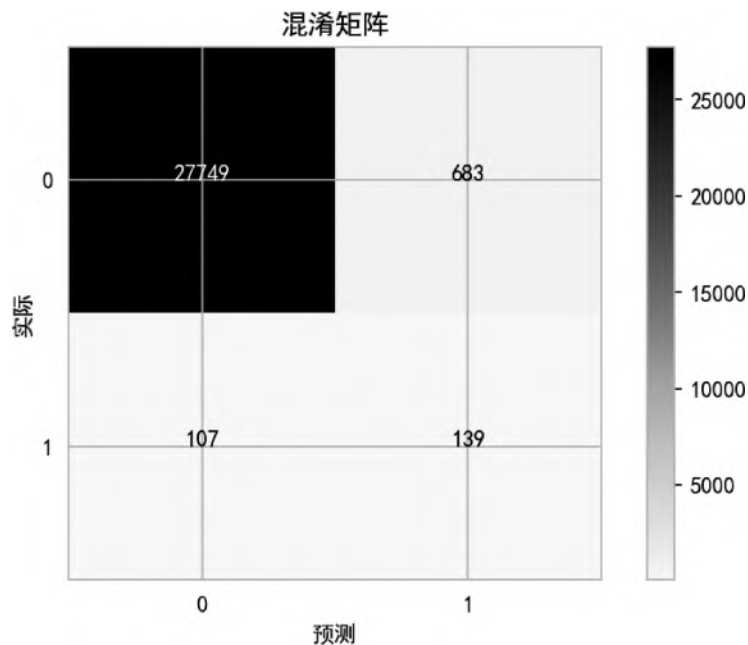


图4-8 直方图异常检测模型下的相关结果展示图

最后介绍基于SVM的异常值检测。

```
#####
# Step 5:基于 SVM 的异常值检测
#####
'''一类 svm,用于异常值检测'''
from sklearn.svm import OneClassSVM
model = OneClassSVM()
model.fit(train.drop('Class', axis = 1).values)
print(model.decision_function(valid[valid['Class'] == 0].drop('Class', axis = 1).values).
mean())
print(model.decision_function(valid[valid['Class'] == 1].drop('Class', axis = 1).values).
mean())
'''计算不同阈值下的 recall_score,precision_score 和 fbeta_score'''
thresholds = np.linspace(- 50000, - 400, 500)
scores = []
y_pred_score = model.decision_function(valid.drop('Class', axis = 1).values)
for threshold in thresholds:
    y_valid_lable = (y_pred_score < threshold).astype(int)
    scores.append([recall_score(valid['Class'], y_valid_lable), precision_score(valid['Class'],
                                     y_valid_lable), fbeta_score(valid['Class'], y_valid_lable, beta = 2)])
scores = np.array(scores)
print(scores[:, 2].max(), scores[:, 2].argmax()) # fbeta 最大为 0.81,索引是 68
'''可视化不同阈值下的 recall_score,precision_score 和 fbeta_score'''
plt.plot(thresholds, scores[:, 0], label = '$ Recall $ ')
plt.plot(thresholds, scores[:, 1], label = '$ Precision $ ')
plt.plot(thresholds, scores[:, 2], label = '$ F_2 $ ')
plt.ylabel('Score')
plt.xlabel('Threshold')
plt.legend(loc = 'best')
plt.show()
'''计算测试集评价标准'''
final_tresh = thresholds[scores[:, 2].argmax()]
y_test_label = (model.decision_function(test.drop('Class',axis = 1).values)< final_tresh).
                astype(int)
print('最终阈值是:%d' % final_tresh)
print('测试集召回率分数是:%.3f' % recall_score(test['Class'], y_test_label))
print('测试集命中率分数是:%.3f' % precision_score(test['Class'], y_test_label))
print('测试集 F2 分数是:%.3f' % fbeta_score(test['Class'], y_test_label, beta = 2))
cnf_matrix = confusion_matrix(test['Class'].values, y_test_label)
plot_confusion_matrix(cnf_matrix, classes = class_names)
```

SVM最大的缺陷是在大数据集上计算的过程较为缓慢，所以建议慎用。

4.3 小结

在日常工作中，会经常碰到不平衡样本。一般来说，建立模型所需要的坏样本比例最好在20%以上。如果坏样本比例1%以上，可以考虑过采样 / 欠采样；如果坏样本比例1%以下，可以考虑以上基于分布的异常值检测方法。但要注意以下两点。

（1）数据质量决定模型效果。数据质量不好的话，建模效果也不会太理想。

（2）不平衡样本哪怕再怎么用技巧，有时候模型效果也会很不理想。模型不是万能的，也不会有某个算法是万能的，最应该做的是将模型与具体业务进行结合，使模型真正发挥实际作用。

第5章 自然语言处理案例——电商评论

自然语言处理通常用于分词、搜索、模糊匹配、正

则表达式、词云、情感分析；深一些来说，自然语言处理一般与语音识别结合在一起，用深度学习神经网络训练，结合语料库，进行语义分析、人机问答等。

本章主要通过电商评论案例来讲解自然语言的实际应用。

5.1 数据加载与预处理

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import os
os.chdir('../data/电商评论')
df = pd.read_csv("comment.csv")
df.head()
```

数据预览结果如图5-1所示。

In [3]: df.head()

Out[3]:

	Unnamed: 0	Clothing ID	Age	Title	Review Text	Rating	Recommended IND	Positive Feedback Count	Division Name	Department Name	Class Name
0	0	767	33	NaN	Absolutely wonderful - silky and sexy and comf...	4	1	0	Intimates	Intimate	Intimates
1	1	1080	34	NaN	Love this dress! it's sooo pretty. i happene...	5	1	4	General	Dresses	Dresses
2	2	1077	60	Some major design flaws	I had such high hopes for this dress and reali...	3	0	0	General	Dresses	Dresses
3	3	1049	50	My favorite buy!	I love, love, love this jumpsuit. it's fun, fl...	5	1	0	General Petite	Bottoms	Pants
4	4	847	47	Flattering shirt	This shirt is very flattering to all due to th...	5	1	6	General	Tops	Blouses

图5-1 电商评论数据预览展示图

由图5-1可以发现第1列也是序号列，与数据自动创建的索引重复，所以删除第1列。

```
df.drop(df.columns[0], inplace = True, axis = 1) # 删除第1列
```

该数据集各特征列是英文，在这里使用rename（）函数将列名重命名为中文名。

```
new_col = {'Clothing ID': '服装 id', 'Age': '评论者年龄', 'Title': '评论标题',  
           'Review Text': '评论内容', 'Rating': '服装评分', 'Recommended IND': '是否推荐',  
           'Positive Feedback Count': '赞同该评论的人数', 'Division Name': '产品高级分类',  
           'Department Name': '产品大类', 'Class Name': '产品二级分类'}  
df = df.rename(columns = new_col) # 重命名  
df.head()
```

再次预览数据，如图5-2所示。

In [2]: df.head()

Out[2]:

	服装 id	评论者年龄	评论标题	评论内容	服装评分	是否推荐	赞同该评论的人数	产品高级分类	产品大类	产品二级分类
0	767	33	NaN	Absolutely wonderful - silky and sexy and comf...	4	1	0	Intimates	Intimate	Intimates
1	1080	34	NaN	Love this dress! it's sooo pretty. i happene...	5	1	4	General	Dresses	Dresses
2	1077	60	Some major design flaws	I had such high hopes for this dress and reall...	3	0	0	General	Dresses	Dresses
3	1049	50	My favorite buy!	I love, love, love this jumpsuit. it's fun, fl...	5	1	0	General Petite	Bottoms	Pants
4	847	47	Flattering shirt	This shirt is very flattering to all due to th...	5	1	6	General	Tops	Blouses

图5-2 电商评论书局重命名后的预览展示图

使用missingno包预览缺失值。

```
import missingno  
missingno.matrix(df, fontsize = 25) # 缺失值可视化
```

缺失值可视化结果如图5-3所示。

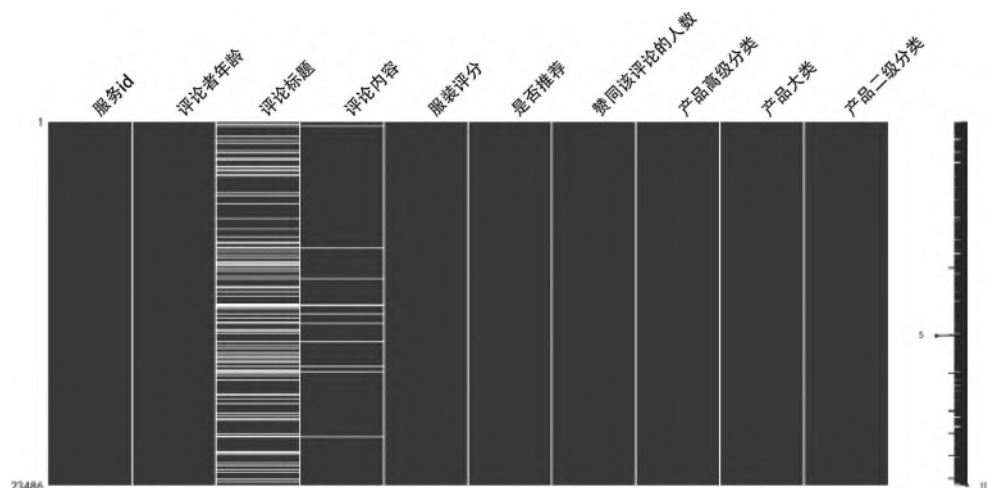


图5-3 数据缺失值可视化图

由图5-3可以发现评论内容有缺失值，那么删除缺失值。

```
col = ["评论内容"]
df = df.dropna(subset = [col])          # 删除缺失值
```

可以做一些特征衍生的工作，在这里创建了新特征：评论字数。

```
df['评论内容'] = df['评论内容'].astype(str)
df['评论字数'] = df['评论内容'].apply(len)      # 计算字符串总长度
```

5.2 数据可视化

为了查看不同变量与评分之间的关系，可以使用可视化进行展示。

```
'''评分与字数关系的可视化 1'''
g = sns.FacetGrid(data = df, col = '服装评分')
g.map(plt.hist, '评论字数', bins = 50)
plt.show()
```

结果如图5-4所示。

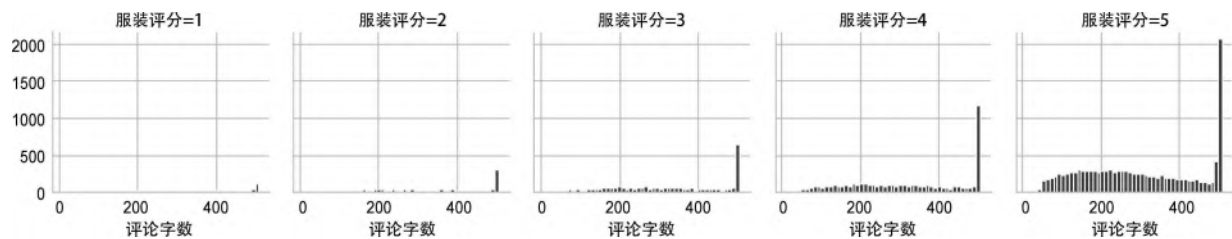


图5-4 服装评分与评论字数的可视化图（1）

```
'''评分与字数关系的可视化 2'''
sns.pointplot(x='服装评分', y='评论字数', data=df)
plt.show()
```

结果如图5-5所示。

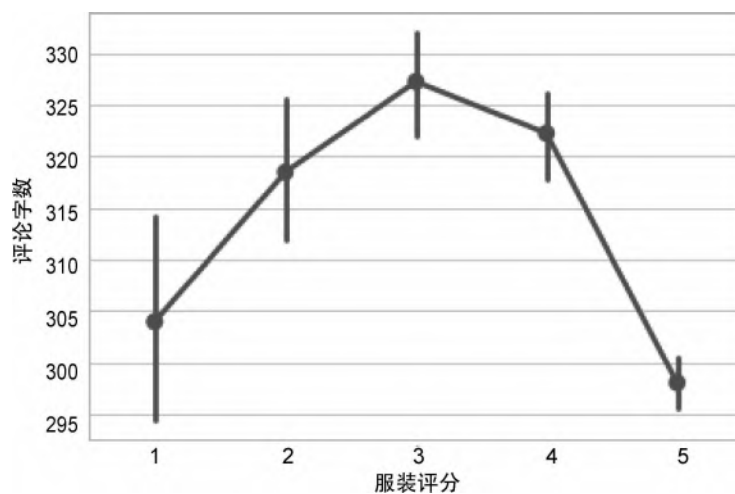


图5-5 服装评分与评论字数的可视化图（2）

由图5-4可以发现，评分越高，写评论的人就越多。由图5-5可以发现，就平均字数而言，评分为3的人最喜欢写长评论。

```
'''找出与服装评分相关性最强的 10 个变量'''
k = 10
corr = round(df.corr(), 2)
cols = corr.nlargest(k, '服装评分')['服装评分'].index
cm = round(df[cols].corr(), 2)
mask = np.zeros_like(cm, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
plt.figure(figsize=(8, 8))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
sns.heatmap(cm, mask=mask, cmap=cmap, center=0, annot=True, cbar_kws={"shrink": .5})
```

运行结果如图5-6所示。

由图5-6可以发现，“是否推荐”这个变量与“服装评分”相关性最强。

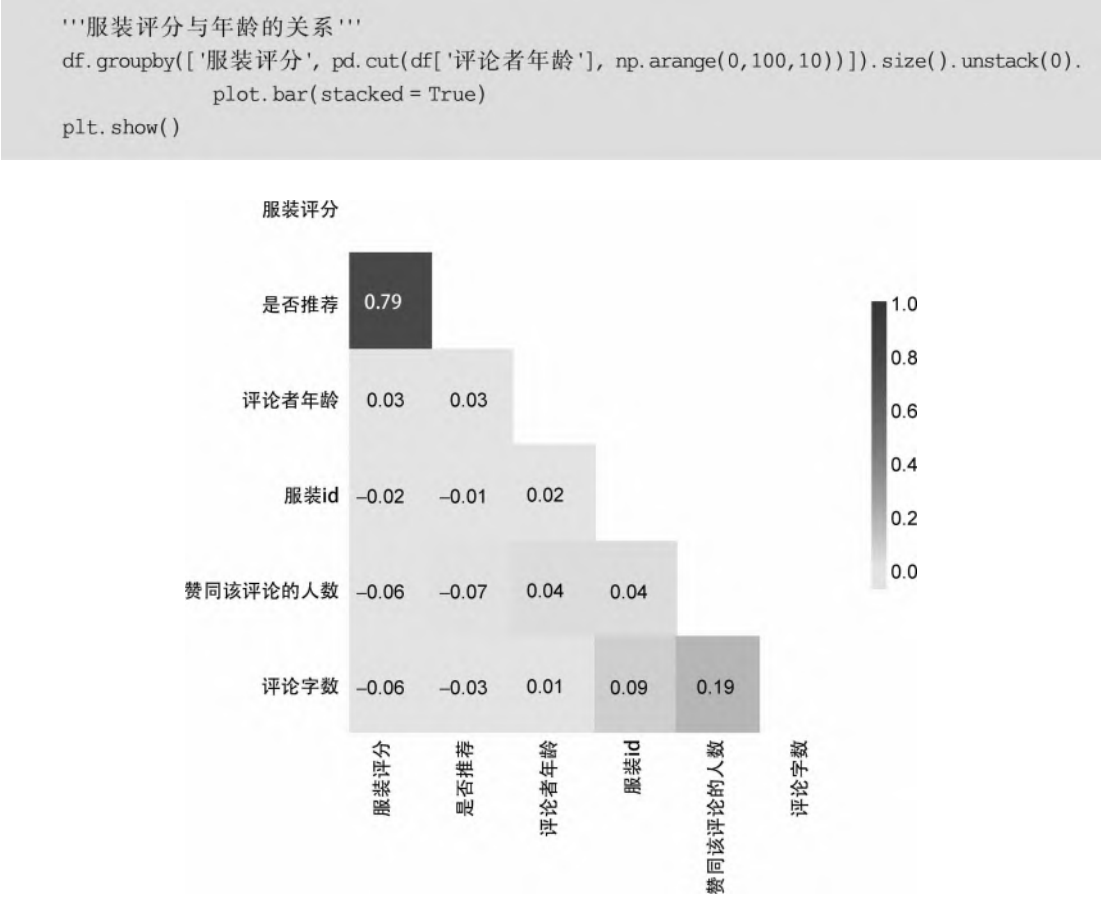


图5-6 与服装评分相关性最强的10个变量展示图

运行结果如图5-7所示。

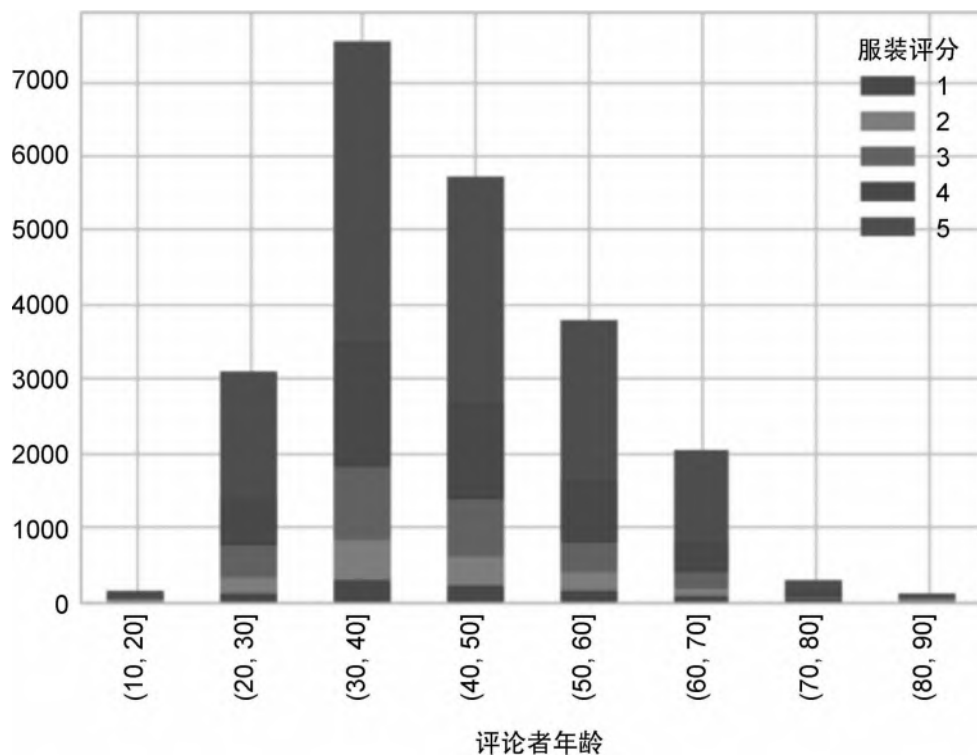


图5-7 服装评分与评论者年龄的关系展示图

由图5-7可以发现，“30~40”与“40~50”这两个年龄段的人在评论时比较喜欢给高分。

```
'''产品类别与年龄的关系'''  
df.groupby(['产品大类', pd.cut(df['评论者年龄'], np.arange(0,100,10))]).size().unstack(0).  
plot.bar(stacked=True)  
plt.show()
```

运行结果如图5-8所示。

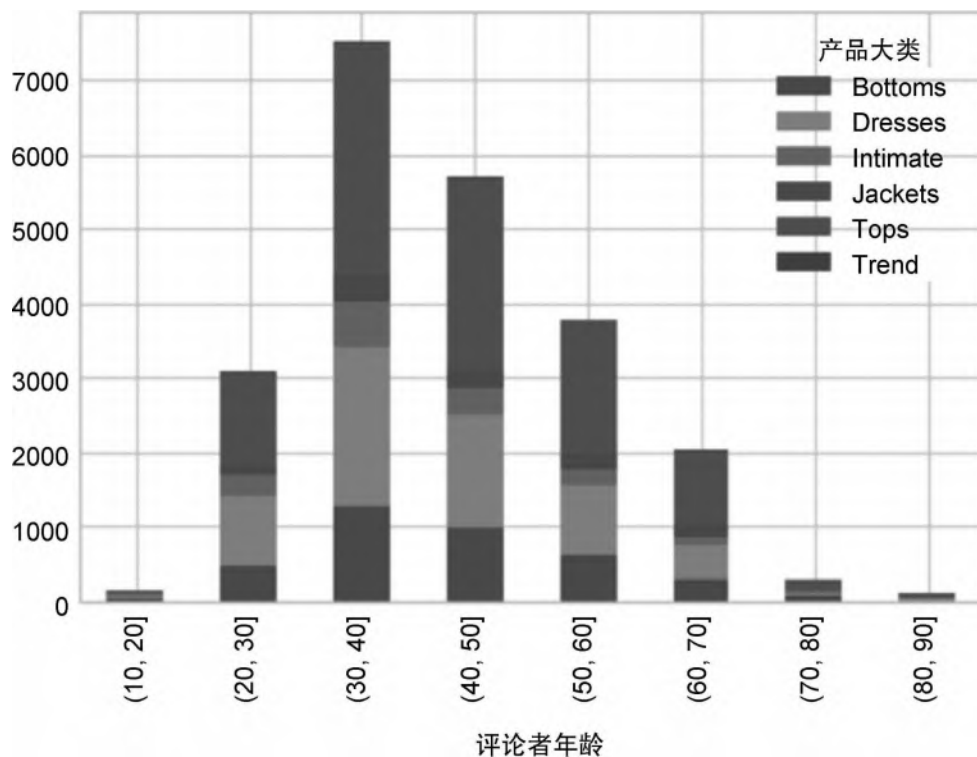


图5-8 产品大类与评论者年龄的关系展示图

由图5-8可以发现，“30~40”和“40~50”是购物的主要年龄段；在各产品类别中，“Tops”在各年龄段的销量都不错，“Dresses”和“Bottoms”在“30~40”销量最高。

5.3 文本分析

对于电商评论内容，除了衍生出评论字数这一变量外，还可对其进行文本分析，找出评论内容中最常见的词语有哪些，然后再对常用词进行情感分析。

```

'''字符串清洗'''
import re
a = df['评论内容'].str.lower().str.cat(sep = '')
b = re.sub('[^A-Za-z]+', '', a)
'''加载停用词库,从字符串文本中移除停用词'''
from stop_words import get_stop_words
stop_words = list(get_stop_words('english'))          # 选择英语停用词
import nltk
from nltk.corpus import stopwords
nltk_words = list(stopwords.words('english'))

stop_words.extend(nltk_words)
from nltk import word_tokenize
word_tokens = word_tokenize(b)          # 分词
filtered_sentence = [i for i in word_tokens if i not in stop_words]

```

在这里说明一下，不同国家的停用词库是不一样的。如果文本内容是中文，可以在网上下载中文停用词库；如果文本内容是英语，Python有专门打包好的停用词库。

这里使用了两个英语停用词库，一个是 `stop_words.get_stop_words`，可以打开网址：<https://pypi.org/project/stopwords/>，查看支持的国家语言；一个是有名的自然语言处理包 `nltk`，在用这个包之前，需要在 `pycharm` 中输入 `nltk.download()` 语句，下载停用词包。

停用词库准备好后，就是分词和去除停用词了。如果文本内容是中文，一般用 `jieba` 分词，如果文本内容是英文，一般用 `nltk` 里的 `word_tokenize`。

分词过后，一般需要删除过短的评论。

```

'''选择长度大于2的字符'''
without_single_chr = [i for i in filtered_sentence if len(i) > 2]
'''计算词频'''
top_N = 100
word_dist = nltk.FreqDist(without_single_chr)
rslt = pd.DataFrame(word_dist.most_common(top_N), columns = ['单词', '频率'])
sns.barplot(x = '单词', y = '频率', data = rslt.head(7))
plt.show()

```

运行结果如图5-9所示。

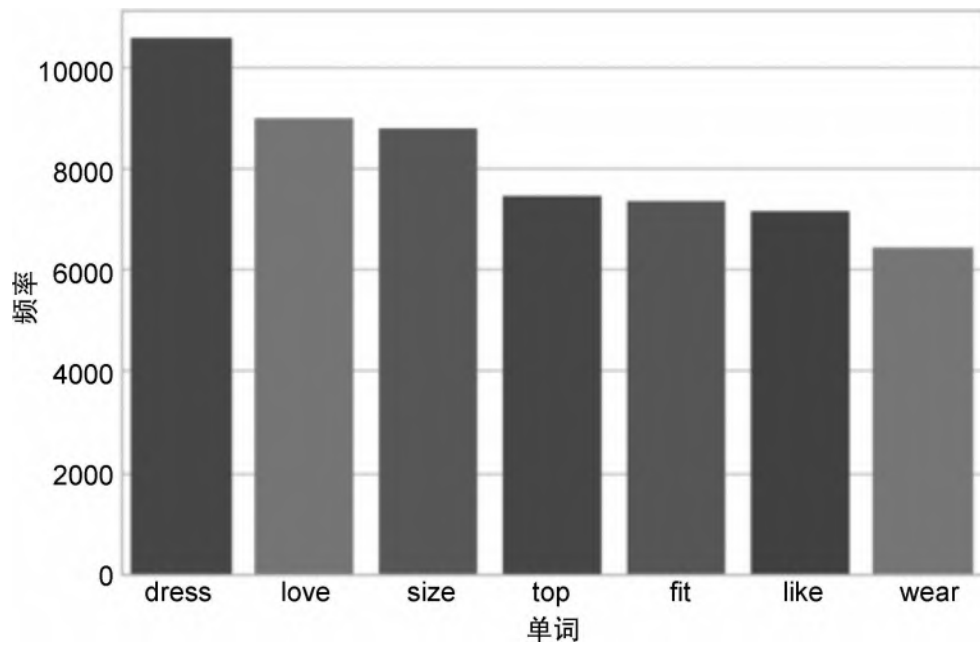


图5-9 不同单词的词频统计展示图

```
'''词云图'''
from wordcloud import WordCloud
def wc(data, bgcolor, title):
    plt.figure(figsize = (10,10))
    plt.title(title)
    wc = WordCloud(background_color = bgcolor, max_words = 1000, max_font_size = 50)
    wc.generate(' '.join(data))
    plt.imshow(wc)
    plt.axis('off')
    plt.show()
wc(without_single_chr, 'black', '最常用的单词')
```

运行结果如图5-10所示。

最常用的单词



图5-10 最常用的单词词云图

5.4 情感分析

情感分析是对带有情感色彩的主观性文本进行分析、处理、归纳和推理的过程。在电商的评论内容中，商家可以根据购买者评论的内容，分析出购买者对商品的喜欢程度，从而对商品或服务进行优化与改进。

```
from textblob import TextBlob                                # 情感分析
bloblist_desc = []
df_review_str = df['评论内容'].astype(str)
for i in df_review_str:
    blob = TextBlob(i)
    # polarity 正数表示积极,负数表示消极,subjectivity 0 表示客观,1 表示主观
    bloblist_desc.append((i, blob.sentiment.polarity, blob.sentiment.subjectivity))
df_polarity_desc = pd.DataFrame(bloblist_desc, columns = ['评论', '情绪', '极性'])
def f(x):
    if x['情绪'] > 0:
        val = '积极评论'
    elif x['情绪'] == 0:
        val = '中性评论'
    else:
        val = '负面评论'
    return val
df_polarity_desc['情绪类型'] = df_polarity_desc.apply(f, axis=1)
sns.countplot(x='情绪类型', data=df_polarity_desc)
plt.show()
```

运行结果如图5-11所示。

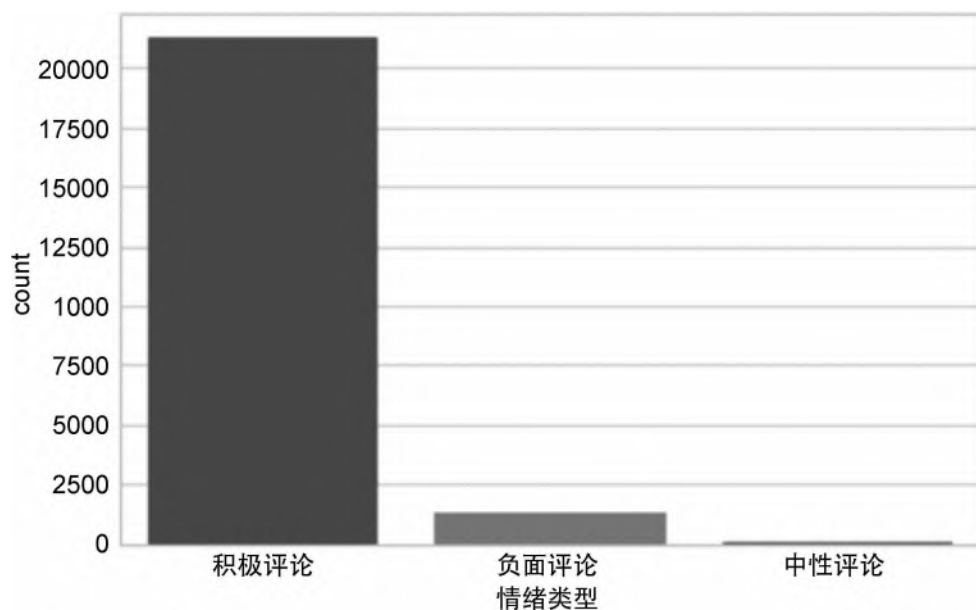


图5-11 情感分析词频图

由图5-11可以发现，绝大部分评论还是积极的。

```
'''正面评论词云图'''
positive_reviews = df_polarity_desc[df_polarity_desc['情绪类型'] == '积极评论']
wc(positive_reviews['评论'], 'black', '正面评论')
```

运行结果如图5-12所示。



图5-12 正面评论词云图

```
negative_reviews = df_polarity_desc[df_polarity_desc['情绪类型'] == '负面评论']
wc(negative_reviews['评论'], 'black', '负面评论')
```

运行结果如图5-13所示。

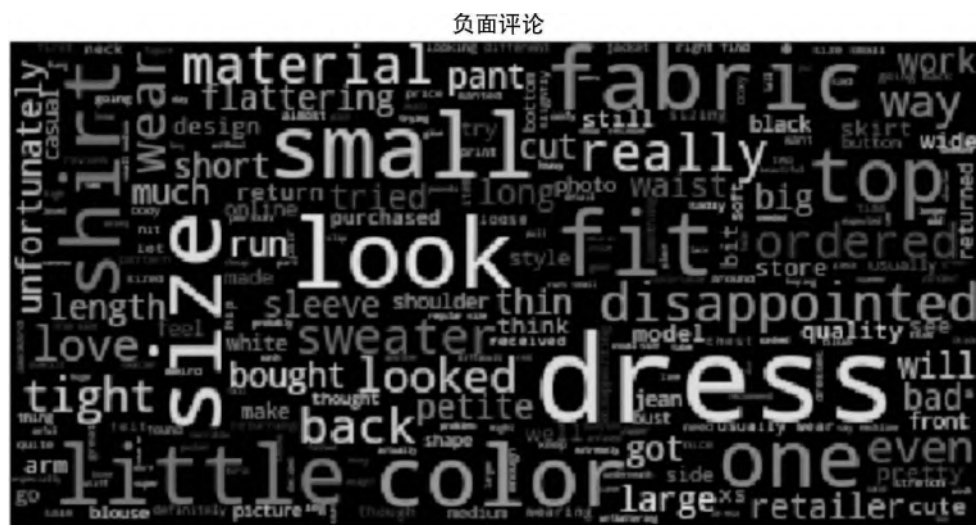


图5-13 负面评论词云图

5.5 文本分类

在对电商评论内容的常用词语进行情感分析之后，可根据顾客对商品的喜爱程度，推荐其他类似的商品，从而提高商品销量和商家的收益。

```

'''停用词处理'''
import string
def text_process(x):
    nopunc = [i for i in x if i not in string.punctuation]
    nopunc = ''.join(nopunc)
    return [i for i in nopunc.split() if i.lower() not in stopwords.words('english')]
df['评论内容'] = df['评论内容'].apply(text_process)
'''朴素贝叶斯模型进行文本分类'''
rating_class = df[(df['服装评分'] == 1) | (df['服装评分'] == 5)]
X_review = rating_class['评论内容']
y = rating_class['服装评分']
from sklearn.feature_extraction.text import CountVectorizer # 文本特征提取
bow_transformer = CountVectorizer(analyzer = text_process).fit(X_review)
X_review = bow_transformer.transform(X_review)
from sklearn.model_selection import train_test_split as sp
X_train,X_test,y_train,y_test = sp(X_review, y, test_size=0.3, random_state=101)
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()
nb.fit(X_train, y_train)
predict = nb.predict(X_test)
'''预测推荐产品'''
X_predict_recommend = df['评论内容']
y_recommend = df['是否推荐']
bow_transformer = CountVectorizer(analyzer = text_process).fit(X_predict_recommend)
X_predict_recommend = bow_transformer.transform(X_predict_recommend)
X_train, X_test, y_train, y_test = sp(X_predict_recommend, y_recommend,
                                     test_size=0.3, random_state=101)
nb = MultinomialNB()
nb.fit(X_train, y_train)
predict_recommendation = nb.predict(X_test)

```

5.6 小结

本章主要通过电商评论案例，简单地介绍了自然语言处理的流程，其中包括可视化、文本分析、情感分析以及文本分类等。现阶段，自然语言处理也是人工智能的重要应用方向之一，感兴趣的读者可以先确定好自己的研究方向，再进行深入研究。

第6章 模型融合

模型融合是综合考虑不同模型的情况，然后再将它们的输出结果融合到一起。模型融合主要有以下5种方法：bagging、boosting、stacking、blending和输出结果加权融合。

为什么要做模型融合？因为它通常可以使模型效果获得提升，同时，模型融合是机器学习竞赛中常用的方法。

本章主要介绍stacking和blending方法。

1. 两层 N 个模型的 K 折stacking原理

(1) 将模型列表中每个模型的输出结果作为新的训练集特征，用于第2层模型。

(2) 单个模型 K 折交叉训练，将每折训练出的模型放在测试集上预测，输出 K 次结果，均值作为新的测试集特征，用于第2层模型。

(3) 把New Features和label作为新的分类器的输入进行训练，然后通过测试集的新Features获得最终的预测结果。

2. 两层 N 个模型的blending原理

(1) 训练集划分为两部分（train1，train2），测试集为test。

(2) 模型列表中，每个模型在训练集train2部分的预测结果作为 X ，与train2部分的 y 一起，组成新的训练集，用于第2层模型。

(3) 模型列表中，每个模型在测试集上的预测结果作为 X ，组成新的测试集，用于第2层模型。

【注意】

基模型之间的差异性要尽可能大，最好是融合不同类型的模型；基模型之间的性能表现不能差距太大，如果有性能特别不好的模型，建议从模型列表中剔除。

6.1 分类模型的融合方法

用于风控领域的机器学习建模主要包括两大类：分类模型与回归模型。下面介绍分类模型的融合方法。

首先是导入包和加载数据，代码如下：

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import os
os.chdir('../data')
data = pd.read_csv('data1.csv')
data.head()
```

预览数据，如图6-1所示。

In [2]: data.head()

Out[2]:

	贷款金额	贷款期限	利率	每月还款金额	贷款等级	工作年限	房屋所有权	年收入	月负债比	过去两年借款人逾期30天以上的数字	...	贷款目的_MOVING	贷款目的_OTHER	贷款目的_RENEWABLE_ENERGY	贷款目的_SMALL_BUSINESS
0	0.417669	1.0	-1.142293	0.465600	1.0	2.0	2.0	-0.268487	1.032490	-0.381014	...	0.0	0.0	0.0	0.0
1	-0.124930	1.0	-1.680677	-0.150988	1.0	10.0	2.0	0.272756	0.011931	-0.381014	...	0.0	0.0	0.0	0.0
2	-0.993088	1.0	-0.638188	-0.969661	2.0	10.0	3.0	-0.074261	-0.668442	-0.381014	...	0.0	1.0	0.0	0.0
3	-0.331118	1.0	-1.680677	-0.355540	1.0	3.0	3.0	-0.268487	0.011931	-0.381014	...	0.0	0.0	0.0	0.0
4	1.177307	2.0	3.031694	1.269438	6.0	10.0	3.0	0.039781	-0.482416	-0.381014	...	0.0	0.0	0.0	0.0

5 rows x 72 columns

图6-1 数据预览展示图

看一下目标变量的水平分布。

```
data['target'].value_counts()
0    7963
1    2037
Name: target, dtype: int64
```

然后对数据集进行切分，代码如下：

```
allFeatures = list(data.columns)
allFeatures.remove('target')
X = data[allFeatures]
y = data['target']
from sklearn.cross_validation import train_test_split as sp
X_train, X_test, y_train, y_test = sp(X, y, test_size=0.3, random_state=1)
```

用默认参数在训练集上训练模型。

```
from sklearn.linear_model import LogisticRegression as LR
lr = LR(random_state=1)
lr.fit(X_train, y_train)
from sklearn import metrics
y_test_label = lr.predict(X_test)
y_test_value = lr.predict_proba(X_test)[:, 1]
print("测试集准确率是: {:.2 %}".format(metrics.accuracy_score(y_test, y_test_label)))
print("测试集 AUC 是: {:.4}".format(metrics.roc_auc_score(y_test, y_test_value)))
```

输出结果如下：

```
测试集准确率是: 80.50 %
测试集 AUC 是: 0.6815
```

下面使用5折交叉训练试试，代码如下：

```

from sklearn.cross_validation import KFold
kf = KFold(len(y_train), 5, random_state=1)
scores = []
for iteration, indices in enumerate(kf, start=1):
    X_train = X_train.reset_index(drop=True)
    y_train = y_train.reset_index(drop=True)
    lr = LR(random_state=1)
    lr.fit(X_train.iloc[indices[0],:], y_train.iloc[indices[0]].values.ravel())
    y_pred = lr.predict_proba(X_train.iloc[indices[1],:].values)[: , 1]
    score = metrics.roc_auc_score(y_train.iloc[indices[1]].values, y_pred)
    scores.append(score)

    print('迭代次数: ', iteration, '得分 = ', score)
print('平均得分 ', np.mean(scores))
y_test_value = lr.predict_proba(X_test)[: , 1]
print('测试集 AUC 是: {:.4}'.format(metrics.roc_auc_score(y_test, y_test_value)))

```

输出结果如下：

```

迭代次数 1: 得分 = 0.691516182342
迭代次数 2: 得分 = 0.666559048428
迭代次数 3: 得分 = 0.666094545455
迭代次数 4: 得分 = 0.670818425474
迭代次数 5: 得分 = 0.667348864361
平均得分 0.672467413212
测试集 AUC 是: 0.6797

```

1. blending

下面介绍模型融合之blending方法。本例中，第1层模型选择的是支持向量机、朴素贝叶斯、 K 最近邻、随机森林，第2层模型选择的是逻辑回归。

第1层：

```

from sklearn.svm import SVC                                # 支持向量机
from sklearn.naive_bayes import GaussianNB as GNB         # 朴素贝叶斯
from sklearn.neighbors import KNeighborsClassifier as KNN   # K 最近邻
from sklearn.ensemble import RandomForestClassifier as RFC  # 随机森林
from sklearn.linear_model import LogisticRegression as LR  # 逻辑回归

svc = SVC(probability=True)
gnb = GNB()
knn = KNN()
rfc = RFC(random_state=1)
model = [svc, gnb, knn, rfc]
X_train_d1, X_train_d2, y_train_d1, y_train_d2 = sp(X_train, y_train, test_size=0.5,
                                                    random_state=1)
X_train_d2_blending = np.zeros((X_train_d2.shape[0], len(model)))
X_test_blending = np.zeros((X_test.shape[0], len(model)))
for j, clf in enumerate(model):
    clf.fit(X_train_d1, y_train_d1)
    y_test_value = clf.predict_proba(X_train_d2)[: , 1]
    # 模型列表中每个模型的预测结果,组成新的训练集,用于第 2 层模型
    X_train_d2_blending[:, j] = y_test_value
    # 模型列表中每个模型的预测结果,组成新的测试集,用于第 2 层模型
    X_test_blending[:, j] = clf.predict_proba(X_test)[: , 1]
    print('测试集 AUC 是: {:.4}'.format(metrics.roc_auc_score(y_test, X_test_blending[:,
j])))

```

输出结果如下:

```

测试集 AUC 是: 0.6301
测试集 AUC 是: 0.6521
测试集 AUC 是: 0.6105
测试集 AUC 是: 0.7953

```

第2层:

```

from sklearn.linear_model import LogisticRegression as LR
lr = LR()
lr.fit(X_train_d2_blending, y_train_d2)
y_test_value = lr.predict_proba(X_test_blending)[: , 1]
y_test_value = (y_test_value - y_test_value.min()) / (y_test_value.max() - y_test_value.min())
print('融合后,测试集 AUC 是: {:.4}'.format(metrics.roc_auc_score(y_test, y_test_value)))

```

输出结果如下:

```

融合后,测试集 AUC 是: 0.7943

```

由此可见，默认参数的逻辑回归算出测试集上的AUC是0.6815，融合模型算出来的AUC是0.7943，此次模型融合预测结果提升了16.55个百分点。

2. stacking

下面介绍stacking方法，本例中的模型列表与blending方法一样，采用的是支持向量机、朴素贝叶斯、 K 最近邻和随机森林。

第1层：

```
X_train = np.array(X_train)
y_train = np.array(y_train)
X_train_stacking = np.zeros((X_train.shape[0], len(model)))
X_test_stacking = np.zeros((X_test.shape[0], len(model)))
from sklearn.cross_validation import StratifiedKFold as SKF
skf = list(SKF(y_train, 5))
for j, clf in enumerate(model):
    X_test_stacking_j = np.zeros((X_test.shape[0], len(skf)))
    for i, (fold1, fold2) in enumerate(skf):
        X_train_fold, y_train_fold, X_test_fold, y_test_fold = X_train[fold1], y_train[fold1],
            X_train[fold2], y_train[fold2]
        clf.fit(X_train_fold, y_train_fold)
        y_test_value = clf.predict_proba(X_test_fold)[:, 1]
        X_train_stacking[fold2, j] = y_test_value

    X_test_stacking_j[:, i] = clf.predict_proba(X_test)[:, 1]
X_test_stacking[:, j] = X_test_stacking_j.mean(axis=1)
print('测试集 AUC 是: {:.4}'.format(metrics.roc_auc_score(y_test, X_test_stacking[:, j])))
```

预测结果如下：

```
测试集 AUC 是: 0.6578
测试集 AUC 是: 0.6635
测试集 AUC 是: 0.6346
测试集 AUC 是: 0.8557
```

第2层：

```

from sklearn.linear_model import LogisticRegression as LR
lr = LR()
lr.fit(X_train_stacking, y_train)
y_test_value = lr.predict_proba(X_test_stacking)[: , 1]
# 标准化
y_test_value = (y_test_value - y_test_value.min()) / (y_test_value.max() - y_test_value.min())
print('融合后,测试集 AUC 是: {:.4}'.format(metrics.roc_auc_score(y_test, y_test_value)))

```

预测结果如下：

```
融合后,测试集 AUC 是: 0.8561
```

此次逻辑回归结果相对于blending又有了较大的提升。

关于stacking方法，有个封装好的第三方包mlxtend，首先安装mlxtend，安装代码为：pip install mlxtend，然后导入包，进行分析，代码如下：

```

from mlxtend.classifier import StackingClassifier
from sklearn.model_selection import cross_val_score
sclf = StackingClassifier(classifiers=model, meta_classifier=lr, use_probas=True)
for clf, label in zip([svc, gnb, knn, rfc, sclf], ['svc', 'gnb', 'knn', 'rfc', 'sclf']):
    scores = cross_val_score(clf, X_train, y_train, cv=5, scoring='roc_auc')
    print('roc_auc {}, {}'.format(scores.mean(), label))

```

运行结果如下：

```

roc_auc 0.6413919395505555, svc
roc_auc 0.6606336495667565, gnb
roc_auc 0.6203925329163198, knn
roc_auc 0.8019715220818838, rfc
roc_auc 0.8028352114075513, sclf

```

不难发现，模型效果还是不错的，融合后的sclf模型指标高于其他模型。

6.2 回归模型的融合方法

上面介绍了分类模型的模型融合方法，现在介绍一下回归模型的融合方法。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import os
os.chdir('../data')
data = pd.read_csv('data2.csv')
allFeatures = list(data.columns)
allFeatures.remove('target')
allFeatures.remove('id')
X = data[allFeatures]
y = data['target']
from sklearn.cross_validation import train_test_split as sp
X_train, X_test, y_train, y_test = sp(X, y, test_size=0.3, random_state=1)
'''定义评价指标'''
from sklearn.model_selection import cross_val_score
def rmse_cv(model):
    rmse = np.sqrt(-cross_val_score(model, X_train, y_train, scoring =
                                     "neg_mean_squared_error", cv = 5))
    return rmse
```

这里的评价指标是交叉验证后的均方误差。

下面开始建模，先看看单个模型的验证效果，代码如下：


```

from sklearn.pipeline import make_pipeline
from sklearn.linear_model import Lasso                # 对异常值敏感
from sklearn.preprocessing import RobustScaler        # 取四分位差
lasso = make_pipeline(RobustScaler(), Lasso(random_state=1))
score = rmse_cv(lasso)
print('lasso 分数是{:.2f}'.format(score.mean()))
from sklearn.ensemble import RandomForestRegressor as RFR
rfr = RFR()
score = rmse_cv(rfr)
print('rfr 分数是{:.2f}'.format(score.mean()))
from sklearn.ensemble import GradientBoostingRegressor as GBR

gbr = GBR()
score = rmse_cv(gbr)
print('gbr 分数是{:.2f}'.format(score.mean()))
import xgboost as xgb
xgbr = xgb.XGBRegressor()
score = rmse_cv(xgbr)
print('分数是{:.2f}'.format(score.mean()))
import lightgbm as lgb
lgbr = lgb.LGBMRegressor()
score = rmse_cv(lgbr)
print('分数是{:.2f}'.format(score.mean()))

```

运行结果是：

```

lasso 分数是 2725.84
rfr 分数是 2489.27
gbr 分数是 2502.07
xgb 分数是 2502.23
lgb 分数是 2321.25

```

下面开始使用Stacking方法进行模型融合。

Stacking1: 这里写了一个类，原理是多个基学习器对数据进行训练和预测，然后根据学习器的预测结果，求出平均值，将其作为最终的预测结果。

```
from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, clone
class AveragingModels(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self, models):
        self.models = models
    def fit(self, X, y):
        self.models_ = [clone(x) for x in self.models]
        for model in self.models_:
            model.fit(X, y)
        return self
    def predict(self, X):
        # 纵向合并 1 维数组
        predictions = np.column_stack([model.predict(X) for model in self.models_])
        return np.mean(predictions, axis=1)
averaged_models = AveragingModels(models = (lasso, gbr, xgbr, lgbr))
score = rmse_cv(averaged_models)
print('分数是{:.2f}'.format(score.mean()))
```

运行结果是：

分数是 2468.91

下面看第二个Stacking方法，使用5折交叉验证，各模型预测结果取平均值，并由元模型预测。

```

from sklearn.model_selection import KFold
class StackingAveragedModels(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self, base_models, meta_model, n_folds=5):
        self.base_models = base_models
        self.meta_model = meta_model
        self.n_folds = n_folds
    def fit(self, X, y):
        self.base_models_ = [list() for x in self.base_models]
        self.meta_model_ = clone(self.meta_model)
        kfold = KFold(n_splits=self.n_folds, random_state=1)
        out_of_fold_predictions = np.zeros((X.shape[0], len(self.base_models)))
        for i, model in enumerate(self.base_models):
            for train_index, holdout_index in kfold.split(X, y):
                instance = clone(model)
                self.base_models_[i].append(instance)
                instance.fit(X[train_index], y[train_index])
                y_pred = instance.predict(X[holdout_index])
                out_of_fold_predictions[holdout_index, i] = y_pred
        self.meta_model_.fit(out_of_fold_predictions, y)
        return self
    def predict(self, X):
        meta_features = np.column_stack([np.column_stack([model.predict(X) for model
            in base_models]).mean(axis=1) for base_models in self.base_models_ ])
        return self.meta_model_.predict(meta_features)
stacked_averaged_models = StackingAveragedModels(base_models = (gbr, xgbr, lgbr),
                                                    meta_model = lasso)

score = rmse_cv(averaged_models)
print('分数是{:.2f}'.format(score.mean()))

```

预测结果是：

分数是 2468.76

6.3 小结

以上的Stacking方法是可以直接使用的，具有普适性。以上代码也许不是最好的，在特定场景下也许不是最优的。那么，做到在已有代码的基础上，理解、提炼、创新出更好的代码与算法，才是优秀风控建模师能力的体现。

第7章 创建金融申请评分卡

业内某知名人士曾说过这么一段话，“以信用评分模型的方式进行风险管理，这是国际上比较成功的实践经验。由于消费信贷业务具有笔数多、单笔金额小、数据维度丰富的特征，决定了需要对其实行智能化、概率化的管理模式。信用评分模型运用现代的数理统计技术，通过对消费者信用历史记录和业务活动记录的深度挖掘、分析和提炼，从而发现蕴藏在纷繁复杂数据中的消费者风险特征，并通过评分的方式总结出来，进而作为管理决策的科学依据。这种管理方法，已成为国际上普遍运用的风险管理最佳操作典范。除了风险管理这一核心领域，这种智能化、概率化的管理模式还被贯穿到消费信贷和信用卡的收益管理、客户忠诚度管理、客户关系管理等方面，涵盖了信贷生命周期管理的各个阶段，包括从产品设计到市场营销、从信贷审批到账户管理、从坏账催收到反欺诈等。在这些方面，国际上都有比较成功的经验值得我们借鉴。”

说到金融领域的评分卡，就不得不提到美国的FICO。自美国三大征信局形成体系之后，FICO总结了征信数据的所有重要规律，以一个简单扼要的评分方式对2亿多美国人进行一个准确的风险评估。这就是如今一个普通美国人能够快速地获得普惠金融服务的最根本条件。如果用户要去银行申请贷款只需要银行查一下用户的信用评分是多少，就会知道你是否符合审批标准，能批多少额度，而且利率也会非常贴近用户真实的风险水平。这是美国金融市场的一个重大里程碑。

如今，FICO是全球最大的个人信用评分机构，每年向市场提供上百亿个FICO评分，管理着全球65%的信用卡，其采用的很多消费信贷和信用卡业务的技术手段都成为了行业标准。

评分卡按照应用场景分类，可以分风险评分卡、营销响应评分卡、客户流失评分卡、员工离职倾向评分卡等；按照用途和时间的不同，风险评分卡又可以分为贷前的申请评分卡、申请欺诈评分卡，贷中的行为评分卡以及贷后的催收评分卡。

本章主要介绍风险评分卡里的申请评分卡，这是最常用也是最重要的评分卡之一。

首先必须要明白两个概念：观察期和表现期。

观察期主要用于收集信用历史和行为特征等信息，从而提炼出能够预测未来信用表现的预测变量。一般来说，根据产品的不同，可以将放款前的6~12个月作为观察期，用来获取特征变量。

表现期主要用于收集信用表现的信息，如是否拖欠、是否逾期、是否流失等，从而提炼表现变量，确定目标客户。

FICO中国区总裁陈建先生曾经说过，申请风险评分模型的表现变量的定义一般根据表现期终的信用表现界定如下：

(1) 3期拖欠以上、呆账、破产的账户定义为“坏”。

(2) 未拖欠或1期拖欠的账户定义为“好”。

(3) 2期拖欠的账户定义为“不确定”，被排除于模型之外。

(4) 如果银行业务历史较短，“坏”的样本量不足（一般一个模型需要800~1500个“坏”账户为样本），则可以把2期拖欠的账户也定义为“坏”。

(5) 如果按“表现期终”的界定方法“坏”样本量不足，也可以把“表现期内”最大拖欠达3期以上的统统定义为“坏”（即使其在表现期终恢复到相对“好”的地位）。

以上这段话更多地倾向于长期循环贷或银行信用卡产品。从实际应用层面上来说，根据各信贷产品的应用场景不同，目标变量是不能

一概而论的。目标客户的定义可以根据业务规则强制来定，也可以根据坏客户迁徙矩阵来定；表现期的长短除了根据自身的历史数据定义以外，还可以把逾期率趋于稳定的时间窗口定义为表现期。

再说说观察期与特征变量。申请评分卡时，各特征必须是贷前就能获取的特征，一般围绕着客户的还款能力、还款意愿、稳定性和多头借贷这4点展开。选择数据供应商时，除了上述4个核心点，还需要兼顾自己的产品客群定位、用户体验、数据成本。比如产品是农户贷，那么社保、公积金、网银工资流水就是不需要获取的特征。如果产品目标是电商小商户，则电商数据就是非常有用的特征。

那么，明确表现期和观察期的概念以后，该如何创建申请评分卡呢？

创建申请评分卡有两种方法：一种是逻辑回归模型；另一种是机器学习模型。

逻辑回归模型是个非常好的模型，效果非常稳定，也容易从业务角度进行解释，只要通过简单的变换，就能得到入模各变量的评分，但是它对数据的质量要求较高，还要服从前提假设，并且精度不是很高。

再说说机器学习模型，申请评分本质上是违约概率的映射，违约概率完全可以用机器学习模型计算出来，而且准确度也较高。可是机器学习模型未能完全代替逻辑回归模型，其主要原因是模型很难从业务角度进行解释，并且模型部署难度较高。因此，目前申请评分卡主要还是使用逻辑回归模型。那么本章就详细介绍一下如何用Python创建逻辑回归模型下的金融申请评分卡。

逻辑回归模型创建金融申请评分卡的核心步骤，如下所述。

(1) 变量分箱。常用的分箱方法是等宽、等频、人工指定分箱切割点、卡方分箱、C4.5决策树分箱这5种。前面3种方法是无监督分箱

法，需要人工主动调试，更多地依赖于建模人员的经验和对业务的理解；后两种分箱法又称为最优分箱法。

(2) WOE编码 (Weight of Evidence)，又称为证据权重。计算公式是每箱好样本比例与坏样本比例的比值的自然对数。

(3) IV值 (Information Value)，又称为信息浓度。计算公式是每箱好样本比例与坏样本比例的差值，再乘以对应的WOE值。一般选择 $IV \geq 0.02$ 的变量。

(4) 共线性、相关性、显著性检验。

(5) 计算每个变量对应切分点的分数。

7.1 变量选择

在构建逻辑回归模型时，从观察期收集的指标可能有几百个，那么就需要从收集到的所有指标中筛选出对违约状态影响最大的指标，作为入模指标，来开发模型，这就是变量选择。

```
#####  
# Step 1: LightGBM 重要性选择变量  
#####  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
import warnings  
warnings.filterwarnings('ignore')  
import os  
os.chdir('../data')  
data = pd.read_csv('data_all_values.csv')
```

预览数据，如图7-1所示。

In [3]:

data.shape

Out[3]:

(103434, 19)

In [4]:

data.head()

Out[4]:

	贷款金额	贷款期限	利率	每月还款金额	贷款等级	工作年限	房屋所有权	年收入	收入是否由LC验证	target	贷款目的	月负债比	过去两年借款人逾期30天以上的数字	过去6个月内被查询次数	摧毁公共记录的数量	额度循环使用率	总贷款笔数	拖欠的逾期款项	重置税款数量
0	18000.0	1	7.99	563.98	1	2.0	2	60000.0	3	0	debt_consolidation	30.76	0.0	0	0.0	17.8	22.0	0.0	0.0
1	13000.0	1	5.32	391.50	1	10.0	2	101800.0	3	0	credit_card	18.91	0.0	0	3.0	57.5	21.0	0.0	0.0
2	5000.0	1	10.49	162.49	2	10.0	3	75000.0	3	0	other	11.01	0.0	1	1.0	47.3	9.0	0.0	0.0
3	11100.0	1	5.32	334.28	1	3.0	3	60000.0	3	0	credit_card	18.91	0.0	0	0.0	34.4	23.0	0.0	0.0
4	25000.0	2	28.69	788.84	6	10.0	3	83807.4	1	0	debt_consolidation	13.17	0.0	0	0.0	38.6	25.0	0.0	0.0

图7-1 评分卡数据预览展示图

这是个有103434行、19列的数据。

目标变量的水平分布如下：

```
data['target'].value_counts()
0      95087
1       8347
Name: target, dtype: int64
```

先把变量分成连续变量与分类变量，分类依据是连续变量的水平种类在10种以上，分类变量的水平种类在10种以下。

```
data_copy = data.copy()
allFeatures = list(data.columns)
allFeatures.remove('target')
for i in allFeatures:
    print('变量: {}的不同水平值有{}个'.format(i, len(data[i].unique())))
```

运行结果如下：

变量: 贷款金额的不同水平值有 1481 个
变量: 贷款期限的不同水平值有 2 个
变量: 利率的不同水平值有 57 个
变量: 每月还款金额的不同水平值有 15228 个
变量: 贷款等级的不同水平值有 7 个
变量: 工作年限的不同水平值有 10 个
变量: 房屋所有权的不同水平值有 3 个
变量: 年收入的不同水平值有 8454 个
变量: 收入是否由 LC 验证的不同水平值有 3 个
变量: 贷款目的的不同水平值有 13 个
变量: 月负债比的不同水平值有 4196 个
变量: 过去两年借款人逾期 30 天以上的数字的不同水平值有 22 个
变量: 过去 6 个月内被查询次数的不同水平值有 6 个
变量: 摧毁公共记录的数量不同水平值有 19 个
变量: 额度循环使用率的不同水平值有 1085 个
变量: 总贷款笔数的不同水平值有 108 个
变量: 拖欠的逾期款项的不同水平值有 420 个
变量: 留置税数量的不同水平值有 18 个

所以，分类变量和连续变量分别如下：

```
categorical_var = ['贷款期限', '贷款等级', '工作年限', '房屋所有权', '收入是否由 LC 验证', '贷款目的', '过去 6 个月内被查询次数', '留置税数量']  
continuous_var = ['贷款金额', '利率', '每月还款金额', '年收入', '月负债比', '过去两年借款人逾期 30 天以上的数字', '摧毁公共记录的数量', '额度循环使用率', '总贷款笔数', '拖欠的逾期款项']
```

首先进行变量筛选。这里使用LightGBM算法先进行重要性排序，再对变量进行粗筛。

```
'''连续变量标准化'''
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
data[continuous_var] = sc.fit_transform(data[continuous_var])
'''数值分类变量转整型'''
string_var = list(data.select_dtypes(include=["object"]).columns)
col = list(set(categorical_var) - set(string_var))
data[col] = data[col].astype(int)
'''字符分类变量按照坏样本率进行编码'''
def Encoder(df, col, target):
    encoder = {}
    for v in set(df[col]):
        if v == v:
            subDf = df[df[col] == v]
        else:
            xList = list(df[col])
            nanInd = [i for i in range(len(xList)) if xList[i] != xList[i]]
            subDf = df.loc[nanInd]
        encoder[v] = sum(subDf[target]) * 1.0/subDf.shape[0]
    newCol = [encoder[i] for i in df[col]]
    return newCol
string_var = list(data.select_dtypes(include=["object"]).columns)
col = list(set(categorical_var) & set(string_var))
for i in col:
    data[i] = Encoder(data, i, 'target')
```

现在再预览一下数据，如图7-2所示。

In [9]: data.head()

Out[9]:

	贷款金额	贷款期限	利率	每月还款金额	贷款等级	工作年限	房屋所有权	年收入	收入是否由LC验证	target	贷款目的	月负债比	过去两年借款人逾期30天以上的数字	过去6个月内被查询次数	摧毁公共记录的数量	额度循环使用率	总贷款笔数	拖欠的逾期款项	留置税数量
0	0.417400	1	-1.142543	0.465334	1	2	2	-0.268366	3	0	0.084803	1.032461	-0.381038	0	-0.401308	-1.349822	-0.166258	-0.022457	0
1	-0.125116	1	-1.680914	-0.151171	1	10	2	0.272899	3	0	0.066410	0.012113	-0.381038	0	4.053237	0.295178	-0.250177	-0.022457	0
2	-0.993140	1	-0.638450	-0.969736	2	10	3	-0.074132	3	0	0.080146	-0.668120	-0.381038	1	1.083540	-0.127467	-1.257209	-0.022457	0
3	-0.331272	1	-1.680914	-0.355696	1	3	3	-0.268366	3	0	0.066410	0.012113	-0.381038	0	-0.401308	-0.661988	-0.082339	-0.022457	0
4	1.176921	2	3.031349	1.269065	6	10	3	0.039914	1	0	0.084803	-0.482132	-0.381038	0	-0.401308	-0.487958	0.085500	-0.022457	0

图7-2 评分卡数据标准化之后的预览展示图

LightGBM模型有个非常好的功能，就是能指定分类变量，省去了one-hot编码的步骤。

```

'''指定整型分类变量作为 LightGBM 的分类特征'''
col = list(set(categorical_var) - set(string_var))
'''保存变量和文件'''
import pickle
f = open('lgb_col.pkl', 'wb')
pickle.dump(col, f)
f.close()
'''建模'''
allFeatures = list(data.columns)
allFeatures.remove('target')
X = data[allFeatures]
y = data['target']
from sklearn.cross_validation import train_test_split as sp
X_train, X_test, y_train, y_test = sp(X, y, test_size=0.3, random_state=1)
'''加载分类变量'''
f = open('lgb_col.pkl', 'rb')
col = pickle.load(f)
f.close()

'''LightGBM 建模'''
import lightgbm as LGB
params = {
    'objective': 'binary',
    'boosting': 'gbdt',
    'num_leaves': 4,
    'min_data_in_leaf': 20,
    'subsample': 0.9,
    'colsample_bytree': 0.8,
    'learning_rate': 0.09,
    'tree_learner': 'voting',
    'metric': 'auc'
}
dtrain = LGB.Dataset(X_train, y_train, categorical_feature=col)
dtest = LGB.Dataset(X_test, y_test, reference=dtrain, categorical_feature=col)
lgb = LGB.train(params, dtrain, valid_sets=[dtrain, dtest], num_boost_round=3000,
                early_stopping_rounds=100, verbose_eval=10)

```

运行结果如下：

```
Training until validation scores don't improve for 100 rounds.  
[10] training's auc: 0.670416 valid_1's auc: 0.668585  
[20] training's auc: 0.67935 valid_1's auc: 0.675068  
[30] training's auc: 0.68141 valid_1's auc: 0.67786  
[40] training's auc: 0.684492 valid_1's auc: 0.679672  
[50] training's auc: 0.687648 valid_1's auc: 0.681506  
[60] training's auc: 0.689867 valid_1's auc: 0.682638  
[70] training's auc: 0.691886 valid_1's auc: 0.683422  
[80] training's auc: 0.693716 valid_1's auc: 0.683969  
[90] training's auc: 0.695131 valid_1's auc: 0.684526  
[100] training's auc: 0.69629 valid_1's auc: 0.68495  
[110] training's auc: 0.697477 valid_1's auc: 0.685131  
[120] training's auc: 0.69868 valid_1's auc: 0.685219  
[130] training's auc: 0.699542 valid_1's auc: 0.685314  
[140] training's auc: 0.700352 valid_1's auc: 0.685386  
[150] training's auc: 0.701254 valid_1's auc: 0.685586  
[160] training's auc: 0.702065 valid_1's auc: 0.685466  
[170] training's auc: 0.702856 valid_1's auc: 0.685529  
[180] training's auc: 0.70363 valid_1's auc: 0.685604  
[190] training's auc: 0.704406 valid_1's auc: 0.685665  
[200] training's auc: 0.704822 valid_1's auc: 0.68562  
[210] training's auc: 0.705378 valid_1's auc: 0.685495  
[220] training's auc: 0.70624 valid_1's auc: 0.685289  
[230] training's auc: 0.706735 valid_1's auc: 0.685263  
[240] training's auc: 0.707306 valid_1's auc: 0.685158  
[250] training's auc: 0.7078 valid_1's auc: 0.68513  
[260] training's auc: 0.708471 valid_1's auc: 0.685116  
  
[270] training's auc: 0.709135 valid_1's auc: 0.684981  
[280] training's auc: 0.709565 valid_1's auc: 0.685044  
[290] training's auc: 0.710201 valid_1's auc: 0.684811  
Early stopping, best iteration is:  
[191] training's auc: 0.704434 valid_1's auc: 0.685696
```

说明最好的迭代轮次是第191轮，最好的结果是训练集auc是0.70，最好的测试集auc是0.68。

下面计算变量重要性。

```
'''LightGBM 重要性选择变量'''
importance = list(lgb.feature_importance())
allFeatures = list(lgb.feature_name())
featureImportance = zip(allFeatures, importance)
featureImportanceSorted = sorted(featureImportance, key = lambda k: k[1], reverse = True)
plt.figure(figsize = (5, 10))
sns.barplot(x = [k[1] for k in featureImportanceSorted], y = [k[0] for k in
                    featureImportanceSorted])
plt.xticks(rotation = 'vertical')
plt.show()
```

运行结果如图7-3所示。

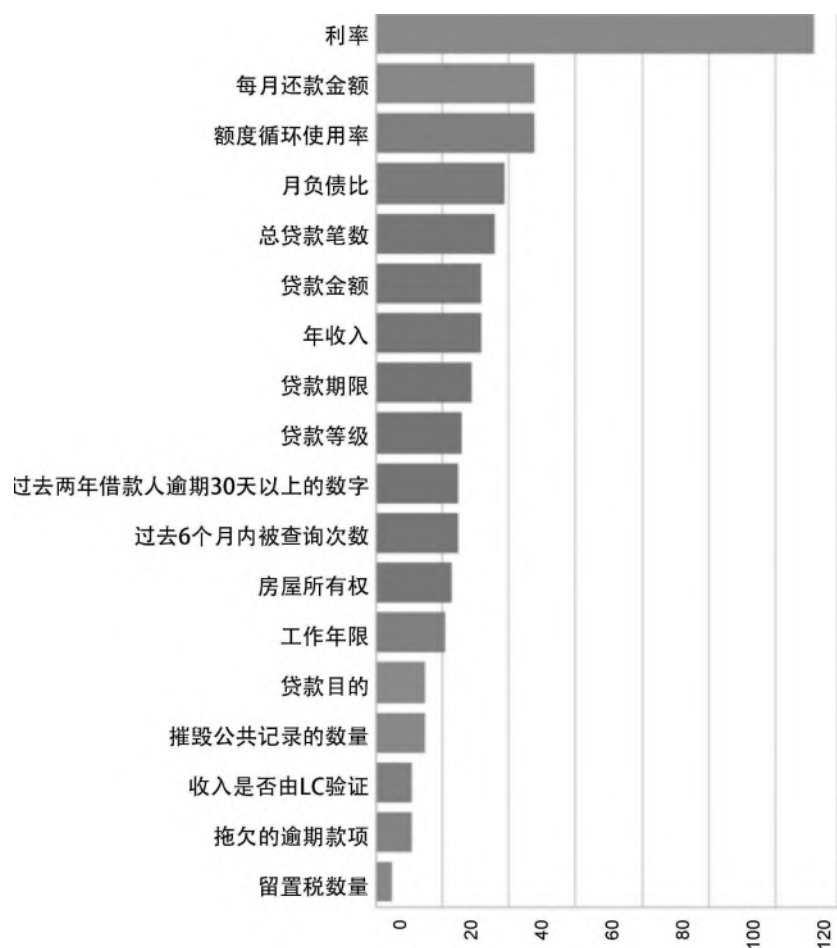


图7-3 LightGBM重要性排序展示图

这里选择前13个最重要的变量。

```
feature_selection_lgb = [k[0] for k in featureImportanceSorted[:13]]
```

7.2 各变量按照 $\ln(\text{odds})$ 进行分箱

odds是好坏比之意，其计算公式为： $\text{odds} = \text{好客户比例} / \text{坏客户比例}$ 。如果该箱好客户多，则 $\ln(\text{odds}) > 0$ ；反之， $\ln(\text{odds}) < 0$ 。

```
#####
# Step 2: 分类变量分箱
#####
data = data_copy.copy()
data = data[feature_selection_lgb + ['target']]
data_copy = data.copy()
'''分类变量按照 ln(odds)进行编码'''
def Ln_odds(df, col, target):
    total = df.groupby([col])[target].count()
    total = pd.DataFrame({'total': total})
    bad = df.groupby([col])[target].sum()
    bad = pd.DataFrame({'bad': bad})
    regroup = total.merge(bad, left_index=True, right_index=True, how='left')
    regroup.reset_index(level=0, inplace=True)
    N = sum(regroup['total'])
    B = sum(regroup['bad'])
    regroup['good'] = regroup['total'] - regroup['bad']
    G = N - B
    regroup['bad_pcnt'] = regroup['bad'].map(lambda x: x * 1.0 / B)
    regroup['good_pcnt'] = regroup['good'].map(lambda x: x * 1.0 / G)
    regroup[col + '_WOE'] = regroup.apply(lambda x: np.log(x.good_pcnt * 1.0 / x.bad_pcnt),
                                          axis=1)

    df = pd.merge(df, regroup[[col, col + '_WOE']], on=col, how='left')
    return df
'''分类变量分箱'''
categorical_var = list(set(categorical_var) & set(feature_selection_lgb))
categorical_var = ['贷款期限', '房屋所有权', '贷款等级', '过去6个月内被查询次数', '工作年限']
for i in categorical_var:
    data = Ln_odds(data, i, 'target')
    sns.pointplot(x=i, y=i + '_WOE', data=data)
    plt.xticks(rotation=0)
    plt.show()
```

运行结果如图7-4所示。

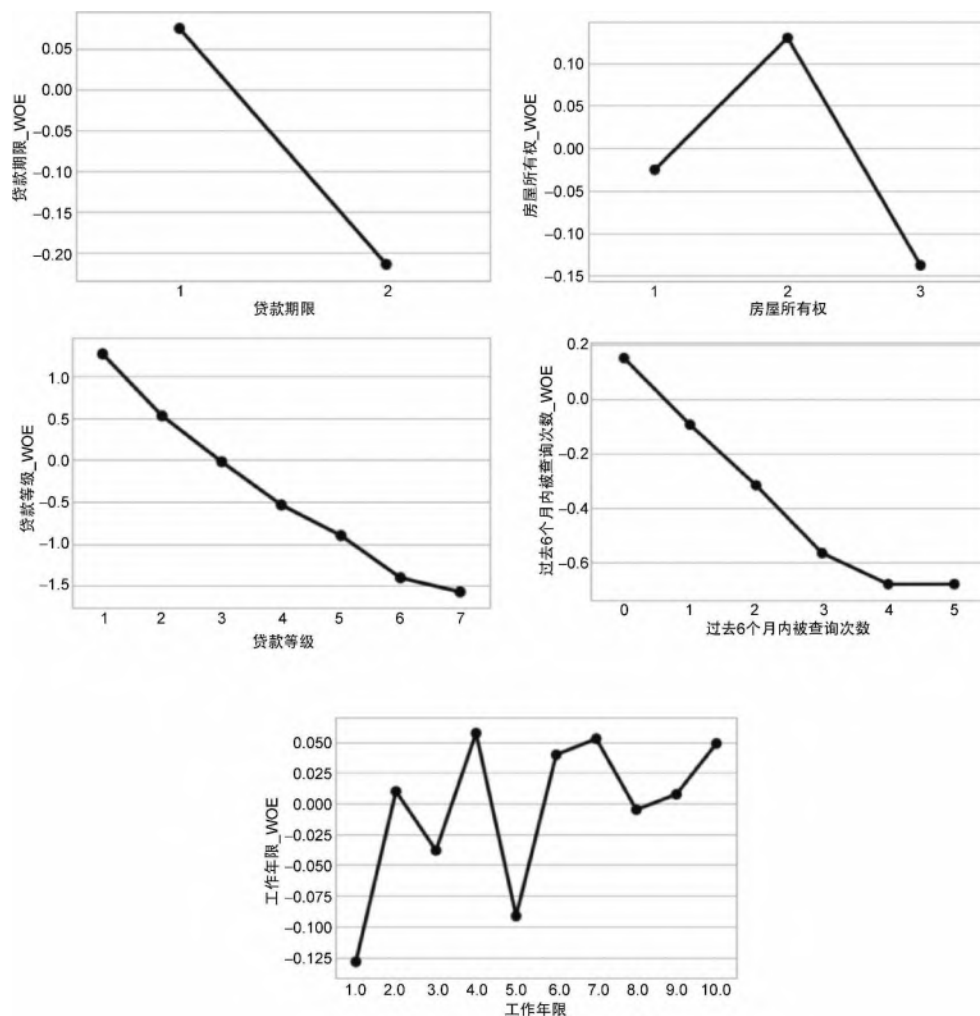


图7-4 分类变量的WOE编码分箱展示图

在变量进行分箱时，一般最多分成5箱，且分箱结果必须单调，否则将进行水平合并或重新分箱。下面逐个进行调试。

1. 变量“贷款等级”分箱

```
'''逐个调试'''
data = data_copy.copy()
i = '贷款等级'
data[i] = data[i].apply(lambda x: 2 if 2 <= x < 5 else x)

data[i] = data[i].apply(lambda x: 3 if x >= 5 else x)
data = Ln_odds(data, i, 'target')
sns.pointplot(x=i, y=i + '_WOE', data=data)
plt.xticks(rotation=0)
plt.show()
```

变量“贷款等级”分箱结果如图7-5所示。

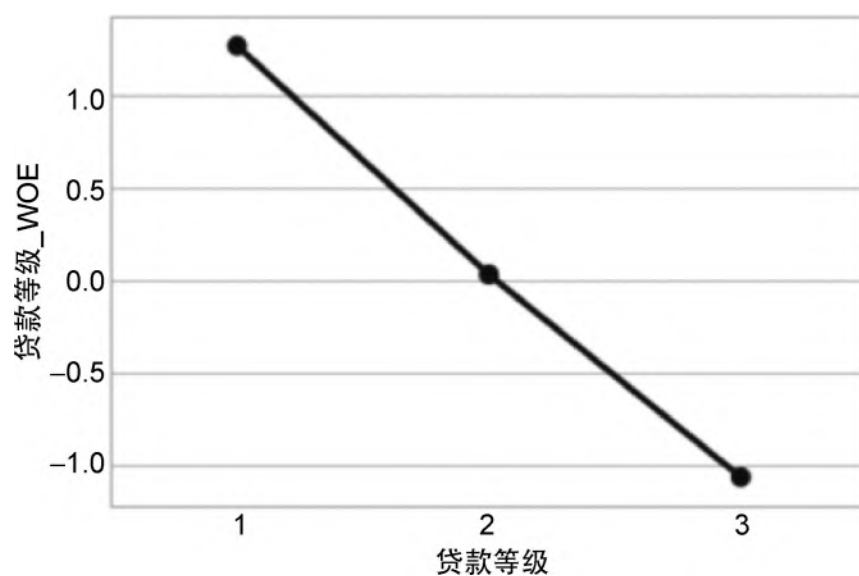


图7-5 “贷款等级”的WOE编码分箱展示图

2. 变量“过去6个月内被查询次数”分箱

```
i = '过去6个月内被查询次数'
data[i] = data[i].apply(lambda x: 4 if x > 3 else x)
data = Ln_odds(data, i, 'target')
sns.pointplot(x=i, y=i + '_WOE', data=data)
plt.xticks(rotation=0)
plt.show()
```

变量“过去6个月内被查询次数”分箱结果如图7-6所示。

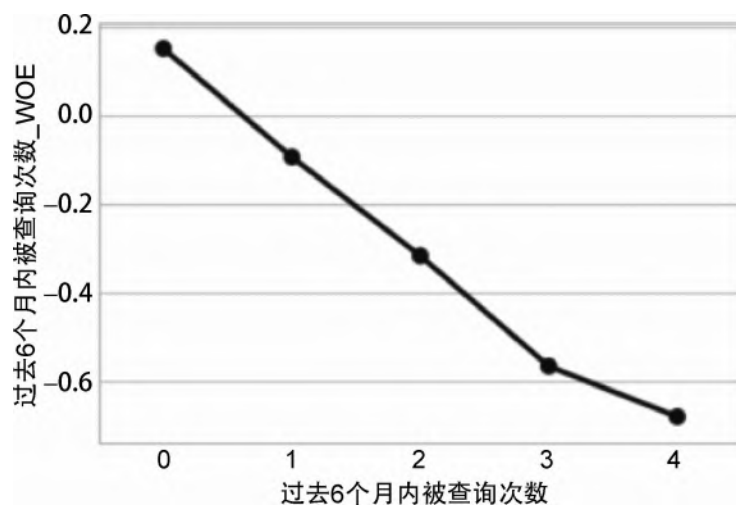


图7-6 “过去6个月内被查询次数”的WOE编码分箱展示图

3. 变量“工作年限”分箱

```
i = '工作年限'
data[i] = data[i].apply(lambda x: 1 if 1 <= x < 5 else x)
data[i] = data[i].apply(lambda x: 2 if x >= 5 else x)
data = Ln_odds(data, i, 'target')
sns.pointplot(x=i, y=i + '_WOE', data=data)
plt.xticks(rotation=0)
plt.show()
```

变量“工作年限”分箱结果如图7-7所示。

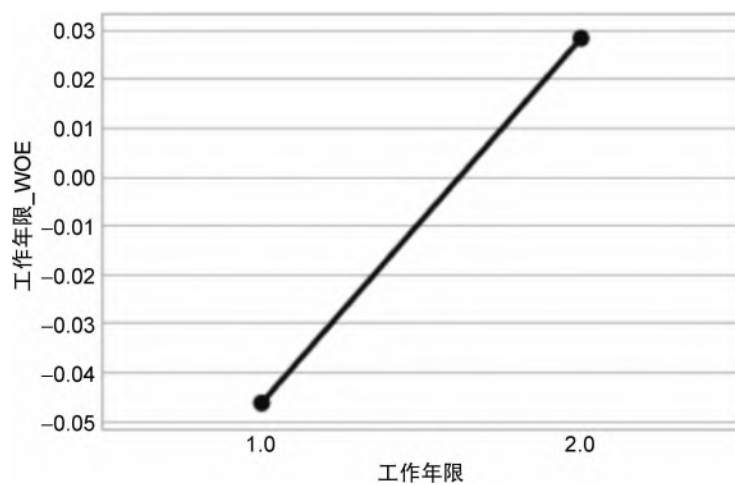


图7-7 “工作年限”的WOE编码分箱展示图

4. 连续变量分箱

(1) 变量“总贷款笔数”分箱

```
#####  
# Step 3: 连续变量分箱  
#####  
data = pd.read_csv('分类变量分箱.csv')  
continuous_var = list(set(continuous_var) & set(feature_selection_lgb))  
continuous_var = ['总贷款笔数', '每月还款金额', '过去两年借款人逾期 30 天以上的数字',  
                  '贷款金额', '年收入', '利率', '月负债比', '额度循环使用率']  
describe = data[continuous_var].describe().T[['max', 'min']]  
'''连续变量分箱'''  
i = '总贷款笔数'  
sns.distplot(data[i][data['target'] == 0].dropna(), color = 'blue')  
sns.distplot(data[i][data['target'] == 1].dropna(), color = 'red')  
plt.show()  
bins = [-1, 20, 30, 200] # 左开右闭, 如果可能出现 0, 那么需要以 -1 开始  
  
cats = pd.cut(list(data[i]), bins, precision = 0)  
cats.value_counts()  
data[i + '组别'] = pd.Series(cats)  
data = Ln_odds(data, i + '组别', 'target')  
sns.pointplot(x = i + '组别', y = i + '组别_WOE', data = data.sort_values(i + '组别_WOE',  
                                ascending = False))  
plt.show()
```

变量“总贷款笔数”分箱结果如图7-8所示。

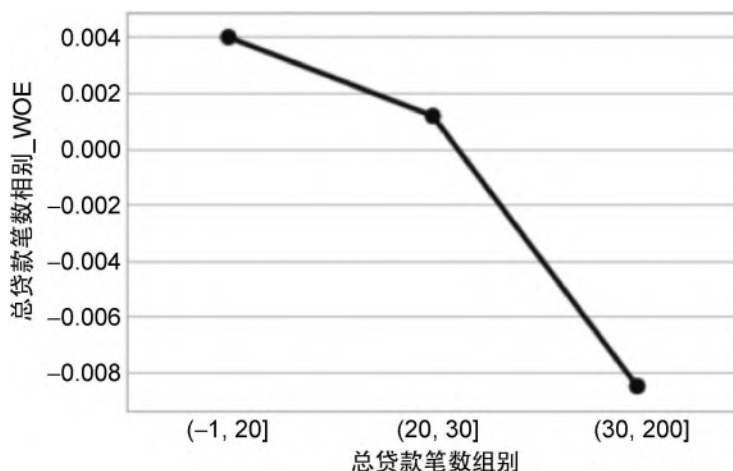


图7-8 “总贷款笔数”组别的WOE编码分箱展示图

(2) 变量“每月还款金额”分箱

```

i = '每月还款金额'
sns.distplot(data[i][data['target'] == 0].dropna(),color = 'blue')
sns.distplot(data[i][data['target'] == 1].dropna(),color = 'red')
plt.show()
bins = [-1, 300, 750, 2000]
cats = pd.cut(list(data[i]), bins, precision=0)
cats.value_counts()
data[i+'组别'] = pd.Series(cats)
data = Ln_odds(data, i+'组别', 'target')
sns.pointplot(x=i+'组别', y=i+'组别_WOE', data=data.sort_values(i+'组别_WOE',
                                                                    ascending=False))
plt.show()

```

变量“每月还款金额”分箱结果如图7-9所示。

(3) 变量“过去两年借款人逾期30天以上的数字”分箱

```

i = '过去两年借款人逾期 30 天以上的数字'
sns.distplot(data[i][data['target'] == 0].dropna(),color = 'blue')
sns.distplot(data[i][data['target'] == 1].dropna(),color = 'red')
plt.show()
bins = [-1, 1, 50]

cats = pd.cut(list(data[i]), bins, precision=0)
cats.value_counts()
data[i+'组别'] = pd.Series(cats)
data = Ln_odds(data, i+'组别', 'target')
sns.pointplot(x=i+'组别', y=i+'组别_WOE', data=data.sort_values(i+'组别_WOE',
                                                                    ascending=False))
plt.show()

```

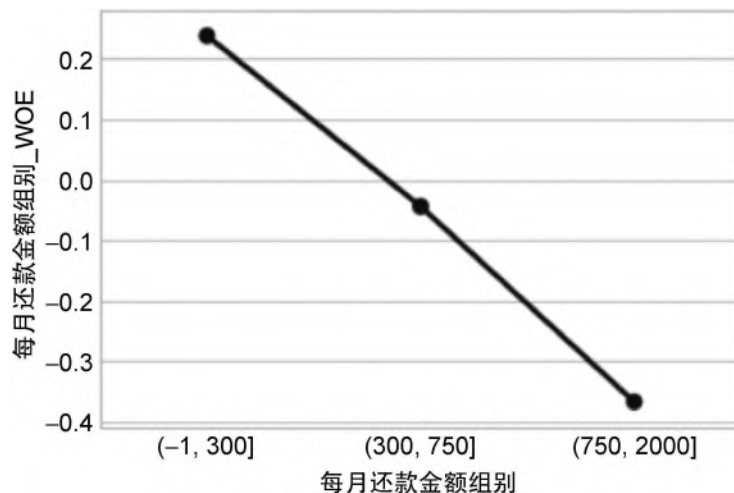


图7-9 “每月还款金额”组别的WOE编码分箱展示图

变量“过去两年借款人逾期30天以上的数字”分箱结果如图7-10所示。

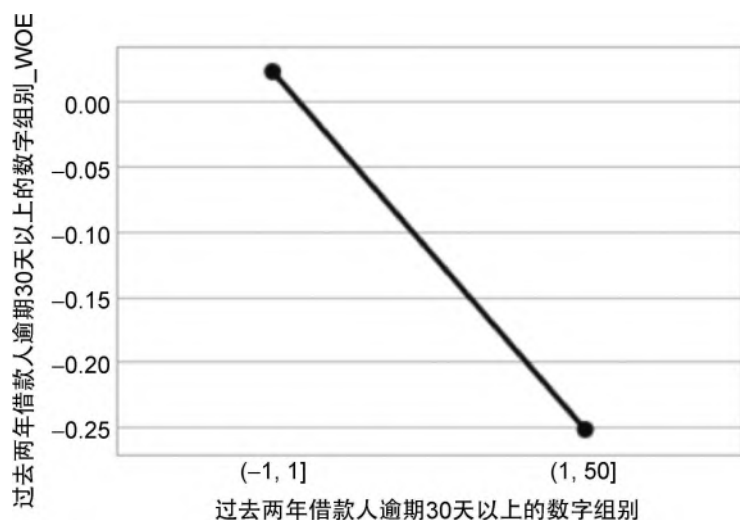


图7-10 “过去两年借款人逾期30天以上的数字”组别的WOE编码分箱展示图

(4) 变量“贷款金额”分箱

```
i = '贷款金额'
sns.distplot(data[i][data['target'] == 0].dropna(), color = 'blue')
sns.distplot(data[i][data['target'] == 1].dropna(), color = 'red')
plt.show()
bins = [-1, 10000, 20000, 50000]
cats = pd.cut(list(data[i]), bins, precision=0)
cats.value_counts()
data[i + '组别'] = pd.Series(cats)

data = Ln_odds(data, i + '组别', 'target')
sns.pointplot(x=i + '组别', y=i + '组别_WOE', data=data.sort_values(i + '组别_WOE',
    ascending=False))
plt.show()
```

变量“贷款金额”分箱结果如图7-11所示。

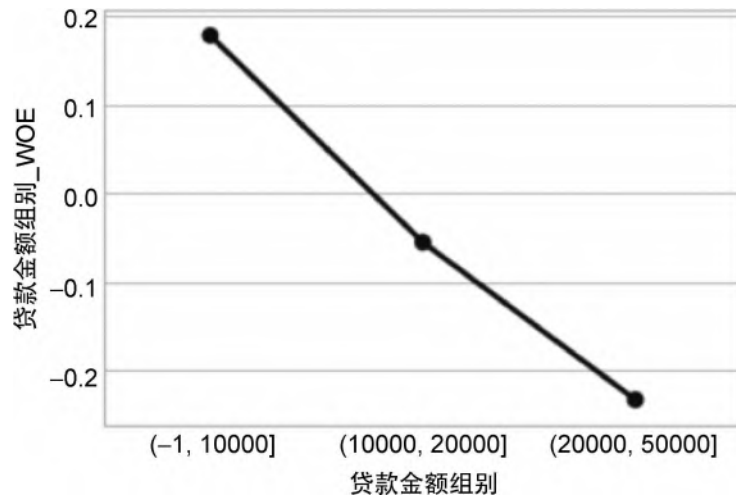


图7-11 “贷款金额”组别的WOE编码分箱展示图

(5) 变量“年收入”分箱

```
i = '年收入'
sns.distplot(data[i][data['target'] == 0].dropna(),color = 'blue')
sns.distplot(data[i][data['target'] == 1].dropna(),color = 'red')
plt.xlim([0,1000000])
plt.show()
bins = [-1, 150000, 300000, 1000000]
cats = pd.cut(list(data[i]), bins, precision=0)
cats.value_counts()
data[i+'组别'] = pd.Series(cats)
data = Ln_odds(data, i+'组别', 'target')
sns.pointplot(x=i+'组别', y=i+'组别_WOE', data=data.sort_values(i+'组别_WOE',
    ascending=False))
plt.show()
```

变量“年收入”分箱结果如图7-12所示。

(6) 变量“利率”分箱

```

i = '利率'
sns.distplot(data[i][data['target'] == 0].dropna(),color = 'blue')
sns.distplot(data[i][data['target'] == 1].dropna(),color = 'red')
plt.show()
bins = [0, 8, 13, 50]
cats = pd.cut(list(data[i]), bins, precision=0)
cats.value_counts()

data[i+'组别'] = pd.Series(cats)
data = Ln_odds(data, i+'组别', 'target')
sns.pointplot(x=i+'组别', y=i+'组别_WOE', data=data.sort_values(i+'组别_WOE',
    ascending=False))
plt.show()

```

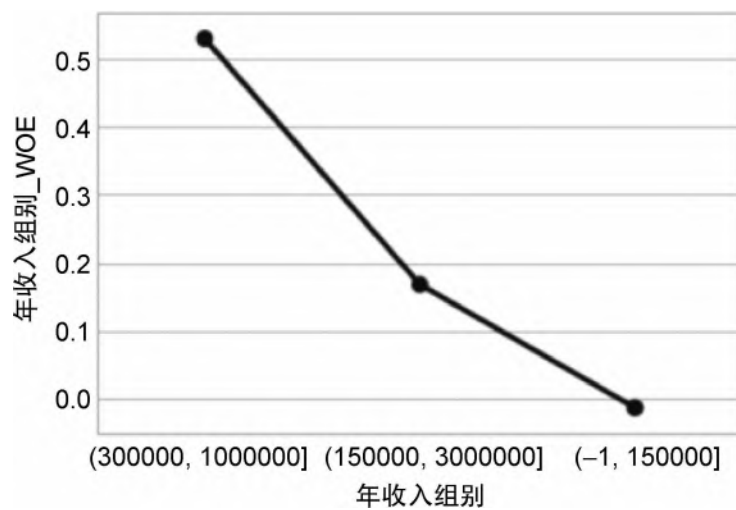


图7-12 “年收入”组别的WOE编码分箱展示图

变量“利率”分箱结果如图7-13所示。

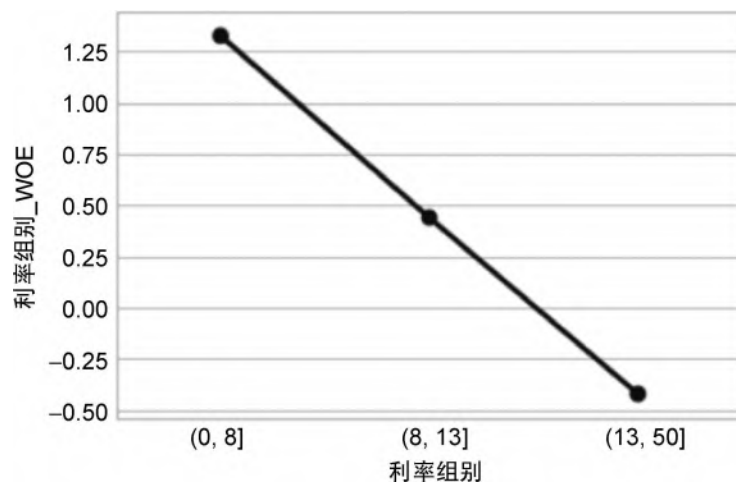


图7-13 “利率”组别的WOE编码分箱展示图

(7) 变量“月负债比”分箱

```
i = '月负债比'
sns.distplot(data[i][data['target'] == 0].dropna(),color = 'blue')
sns.distplot(data[i][data['target'] == 1].dropna(),color = 'red')
plt.xlim([0,200])
plt.show()
bins = [-1, 20, 1000]
cats = pd.cut(list(data[i]), bins, precision=0)
cats.value_counts()
data[i+'组别'] = pd.Series(cats)
data = Ln_odds(data, i+'组别', 'target')
sns.pointplot(x=i+'组别', y=i+'组别_WOE', data=data.sort_values(i+'组别_WOE',
                                                                    ascending=False))
plt.show()
```

变量“月负债比”分箱结果如图7-14所示。

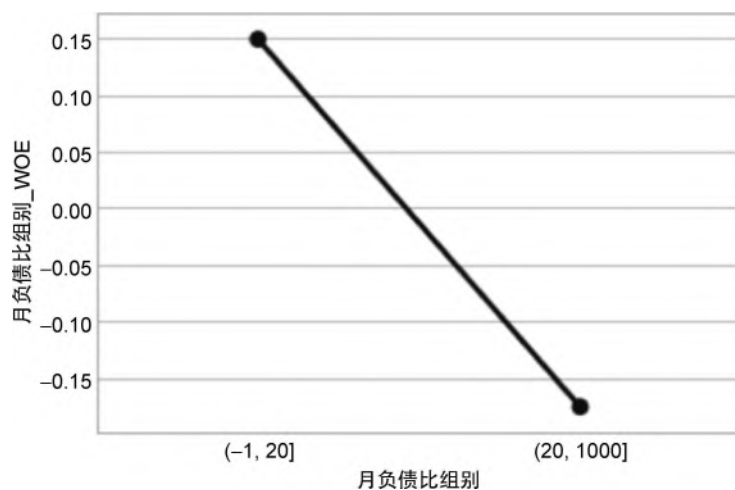


图7-14 “月负债比”组别的WOE编码分箱展示图

(8) 变量“额度循环使用率”分箱

```

i = '额度循环使用率'
sns.distplot(data[i][data['target'] == 0].dropna(),color = 'blue')
sns.distplot(data[i][data['target'] == 1].dropna(),color = 'red')
plt.show()
bins = [-1, 50, 200]
cats = pd.cut(list(data[i]), bins, precision=0)
cats.value_counts()
data[i+'组别'] = pd.Series(cats)
data = Ln_odds(data, i+'组别', 'target')
sns.pointplot(x=i+'组别', y=i+'组别_WOE', data=data.sort_values(i+'组别_WOE',
                                                                ascending=False))
plt.show()

```

变量“额度循环使用率”分箱结果如图7-15所示。

```

'''保存文件'''
col = []
for i in list(data.columns):
    if i.find('_WOE')<0:
        col.append(i)
data = data[col]
data.to_csv('分箱完成.csv', index=False, encoding='utf-8')

```

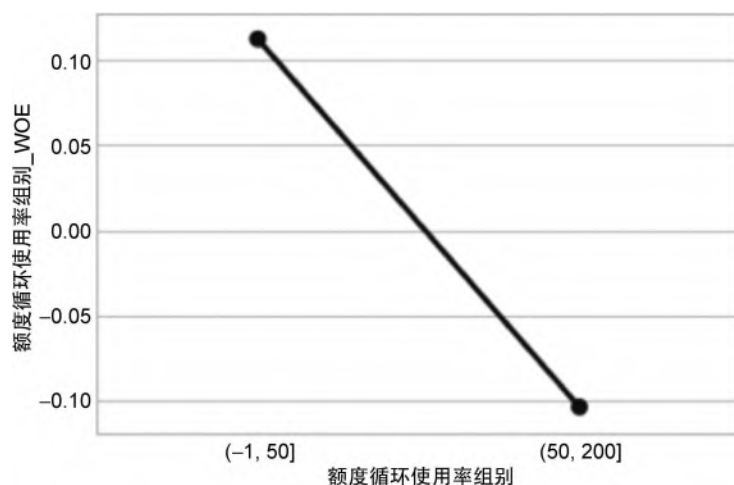


图7-15 “额度循环使用率”组别的WOE编码分箱展示图

7.3 计算WOE与IV值

证据权重（weight of evidence, WOE）用于计算每箱的好坏比，WOE的数值越大，表明每箱中的好样本越多。IV值就是WOE值对应的信

息浓度。

```
#####  
# Step 4: 计算 WOE 和 IV  
#####  
def CalcWOE(df, col, target):  
    total = df.groupby([col])[target].count()  
    total = pd.DataFrame({'total': total})  
    bad = df.groupby([col])[target].sum()  
    bad = pd.DataFrame({'bad': bad})  
    regroup = total.merge(bad, left_index=True, right_index=True, how='left')  
    regroup.reset_index(level=0, inplace=True)  
    N = sum(regroup['total'])      # 计算总样本数  
    B = sum(regroup['bad'])        # 计算坏样本数  
    regroup['good'] = regroup['total'] - regroup['bad']  
    G = N - B  
    regroup['bad_pcnt'] = regroup['bad'].map(lambda x: x * 1.0/B)  
    regroup['good_pcnt'] = regroup['good'].map(lambda x: x * 1.0 / G)  
    regroup['WOE'] = regroup.apply(lambda x: np.log(x.good_pcnt * 1.0/x.bad_pcnt), axis=1)  
    WOE_dict = regroup[[col, 'WOE']].set_index(col).to_dict(orient='index')  
    for k, v in WOE_dict.items():  
        WOE_dict[k] = v['WOE']  
        IV = regroup.apply(lambda x: (x.good_pcnt - x.bad_pcnt) *  
                                np.log(x.good_pcnt * 1.0/x.bad_pcnt), axis=1)  
        IV = sum(IV)  
    return {"WOE": WOE_dict, 'IV': IV}  
  
all_var = []  
for i in list(data.columns):  
    if i.find('组别') > 0:  
        all_var.append(i)  
all_var = all_var + categorical_var  
WOE_dict = {}  
IV_dict = {}  
for var in all_var:  
    woe_iv = CalcWOE(data, var, 'target')  
    WOE_dict[var] = woe_iv['WOE']  
    IV_dict[var] = woe_iv['IV']
```

现在预览一下数据，如图7-16所示。

In [36]: data.head()

Out[36]:

	利率	每月还款金额	额度循环使用率	月负债比	总贷款笔数	贷款金额	年收入	贷款期限	贷款等级	过去两年借款人逾期30天以上的数字	...	工作年限	target	总贷款笔数组别	每月还款金额组别	过去两年借款人逾期30天以上的数字组别	贷款金额组别	年收入组别	利率组别	月负债比组别	额度循环使用率组别
0	7.99	563.98	17.8	30.76	22.0	18000.0	60000.0	1	1	0.0	...	1.0	0	(20, 30]	(300, 750]	(-1, 1]	(10000, 20000]	(-1, 150000]	(0, 8]	(20, 1000]	(-1, 50]
1	5.32	391.50	57.5	18.91	21.0	13000.0	101800.0	1	1	0.0	...	2.0	0	(20, 30]	(300, 750]	(-1, 1]	(10000, 20000]	(-1, 150000]	(0, 8]	(-1, 1000]	(50, 200]
2	10.49	162.49	47.3	11.01	9.0	5000.0	75000.0	1	2	0.0	...	2.0	0	(-1, 20]	(-1, 300]	(-1, 1]	(-1, 10000]	(-1, 150000]	(8, 13]	(-1, 20]	(-1, 50]
3	5.32	334.28	34.4	18.91	23.0	11100.0	60000.0	1	1	0.0	...	1.0	0	(20, 30]	(300, 750]	(-1, 1]	(10000, 20000]	(-1, 150000]	(0, 8]	(-1, 20]	(-1, 50]
4	28.69	788.84	38.6	13.17	25.0	25000.0	83807.4	2	3	0.0	...	2.0	0	(20, 30]	(750, 2000]	(-1, 1]	(20000, 50000]	(-1, 150000]	(13, 50]	(-1, 20]	(-1, 50]

5 rows x 22 columns

图7-16 评分卡数据预览展示图

7.4 逻辑回归建模

逻辑回归（Logistic Regression），虽然名字上带有“回归”的字样，但其实质却是一种常用的分类模型，主要用于二分类问题。逻辑回归可以通过logistics函数，将特征空间映射成某些事件发生的概率。计算出变量WOE值和IV值之后，就可以对WOE变量进行logistics映射，并将其转换为逾期概率。

```
#####
# Step 5: 逻辑回归建模
#####
'''选取 IV>= 0.02 的变量'''
IV_dict_sorted = sorted(IV_dict.items(), key = lambda x: x[1], reverse = True)
IV_name = [i[0] for i in IV_dict_sorted]
IV_values = [i[1] for i in IV_dict_sorted]

high_IV = {k:v for k, v in IV_dict.items() if v >= 0.02}
high_IV_sorted = sorted(high_IV.items(), key = lambda x:x[1], reverse = True)
print ('总共有',len(high_IV_sorted),'个变量的 IV >= 0.02')
```

运行结果如下：

```

总共有 6 个变量的 IV >= 0.02
'''创建 WOE 变量'''
short_list = high_IV.keys()
short_list_2 = []
for var in short_list:
    newVar = var + '_WOE'
    data[newVar] = data[var].map(WOE_dict[var])
    short_list_2.append(newVar)
'''计算相关系数矩阵,删除与重要业务变量相关性较强的自变量'''
dataWOE = data[short_list_2]
corr = round(dataWOE.corr(),2)
mask = np.zeros_like(corr, dtype = np.bool)
mask[np.triu_indices_from(mask)] = True
plt.figure(figsize = (5, 5))
cmap = sns.diverging_palette(220, 10, as_cmap = True)
sns.heatmap(corr,mask = mask, cmap = cmap, center = 0, annot = True, cbar_kws = {"shrink": .5})
plt.show()

```

运行结果如图7-17所示。

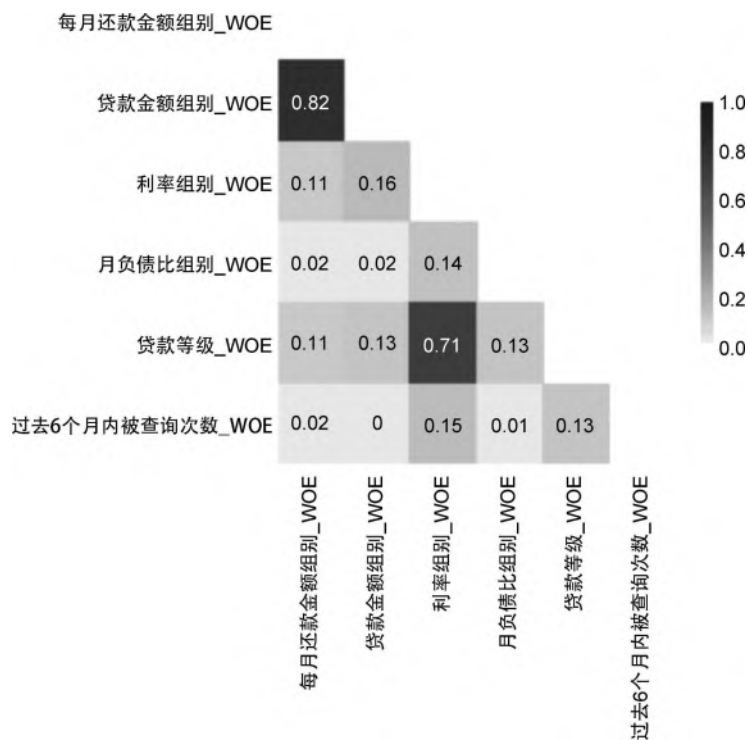


图7-17 6个变量的WOE相关系数图

计算相关系数之后，还需要考虑多重共线性问题。评价多重共线性的指标主要是方差膨胀因子。方差膨胀因子（Variance Inflation

Factor, VIF) 是指解释变量之间存在多重共线性时的方差与不存在多重共线性时的方差之比。VIF值越大, 代表共线性越严重。经验判断方法表明: 当 $0 < VIF < 10$, 不存在多重共线性; 当 $10 \leq VIF < 100$, 存在较强的多重共线性; 当 $VIF \geq 100$, 存在严重多重共线性。

```
"""选择方差共线性<10 的变量"""
col = np.array(data[short_list_2])
from statsmodels.stats.outliers_influence import variance_inflation_factor as vif
for i in range(len(short_list_2)):
    print ('{} VIF 是{}'.format(short_list_2[i], vif(col, i)))
```

运行结果如下:

```
每月还款金额组别_WOE VIF 是 3.1504765201473144
贷款金额组别_WOE VIF 是 3.1868414919401897
利率组别_WOE VIF 是 2.1775374815158592
月负债比组别_WOE VIF 是 1.0255937275175637
贷款等级_WOE VIF 是 2.1297714474712253
过去 6 个月内被查询次数_WOE VIF 是 1.0298508719860746
```

在确定变量之间不存在多重共线性之后, 就要对变量进行显著性分析, 删除P值不显著的变量。

```
"""判断显著性"""
X = data[short_list_2]
X['intercept'] = [1] * X.shape[0]
y = data['target']
import statsmodels.api as sm
lr_sm = sm.Logit(y, X).fit()
lr_sm.summary()
```

运行结果如图7-18所示。

最后建模, 代码如下:

```

X = data[short_list_2]
y = data['target']
from sklearn.cross_validation import train_test_split as sp
X_train, X_test, y_train, y_test = sp(X, y, test_size=0.3, random_state=1)
from sklearn.linear_model import LogisticRegression as LR

```

Out[51]: Logit Regression Results

Dep. Variable:	target	No. Observations:	103434
Model:	Logit	Df Residuals:	103427
Method:	MLE	Df Model:	6
Date:	Sun, 18 Mar 2018	Pseudo R-squ.:	0.04845
Time:	16:33:07	Log-Likelihood:	-27605.
converged:	True	LL-Null:	-29010.
		LLR p-value:	0.000

	coef	std err	z	P> z	[0.025	0.975]
每月还款金额组别_WOE	-1.0481	0.098	-10.661	0.000	-1.241	-0.855
贷款金额组别_WOE	0.6825	0.124	5.497	0.000	0.439	0.926
利率组别_WOE	-0.6757	0.029	-23.313	0.000	-0.733	-0.619
月负债比组别_WOE	-0.5186	0.072	-7.213	0.000	-0.660	-0.378
贷款等级_WOE	-0.5519	0.029	-18.903	0.000	-0.609	-0.495
过去6个月内被查询次数_WOE	-0.6137	0.053	-11.500	0.000	-0.718	-0.509
intercept	-2.4398	0.012	-202.610	0.000	-2.463	-2.416

图7-18 6个变量的显著性结果展示图

```

lr = LR(random_state=1)
lr.fit(X_train, y_train)
from sklearn import metrics
y_test_label = lr.predict(X_test)
y_test_value = lr.predict_proba(X_test)[:, 1]
print('测试集准确率是: {:.2 %}'.format(metrics.accuracy_score(y_test, y_test_label)))
print('测试集 AUC 是: {:.4}'.format(metrics.roc_auc_score(y_test, y_test_value)))

```

运行结果如下：

```

测试集准确率是: 91.82 %
测试集 AUC 是: 0.6683

```

7.5 创建评分卡

在确定模型的准确度和AUC的可接受范围后，可以对每个入模变量进行单变量得分计算，从而求出变量不同取值下的得分。

```
#####  
# Step 6: 创建评分卡  
#####  
b = lr.intercept_          # 截距
```

```

coe = lr.coef_                                # 系数
a0 = coe[0][0]                                # 第一个变量的系数
a1 = coe[0][1]
a2 = coe[0][2]
a3 = coe[0][3]
a4 = coe[0][4]
a5 = coe[0][5]
A = 500
PDO = 20                                     # 每增加 20 分, odds 增加 1 倍
B = PDO/np.log(2)
'''创建 每月还款金额 单变量得分'''
WOE_dict['每月还款金额组别']                # 获取字典 key, 即变量水平值
woe1 = WOE_dict['每月还款金额组别']['(-1, 300]']
score1 = -(B * a0 * woe1) + (A - B * b)/dataWOE.shape[0]
woe2 = WOE_dict['每月还款金额组别']['(300, 750]']
score2 = -(B * a0 * woe2) + (A - B * b)/dataWOE.shape[0]
woe3 = WOE_dict['每月还款金额组别']['(750, 2000]']
score3 = -(B * a0 * woe3) + (A - B * b)/dataWOE.shape[0]
'''对应的 SQL 语句
case
    when 0 <= '每月还款金额' <= 300 then 7
    when 300 < '每月还款金额' <= 750 then -1
    when '每月还款金额' > 750 then -11
else 0                                     # 以业务逻辑或补缺规则来定
'''

'''创建 贷款金额 单变量得分'''
WOE_dict['贷款金额组别']
woe1 = WOE_dict['贷款金额组别']['(-1, 10000]']
score1 = -(B * a1 * woe1) + (A - B * b)/dataWOE.shape[0]
woe2 = WOE_dict['贷款金额组别']['(10000, 20000]']
score2 = -(B * a1 * woe2) + (A - B * b)/dataWOE.shape[0]
woe3 = WOE_dict['贷款金额组别']['(20000, 50000]']
score3 = -(B * a1 * woe3) + (A - B * b)/dataWOE.shape[0]
'''对应的 SQL 语句
case
    when 0 <= '贷款金额' <= 10000 then -3
    when 10000 < '贷款金额' <= 20000 then 1
    when '贷款金额' > 20000 then 4
else 0
'''

'''创建 利率 单变量得分'''
WOE_dict['利率组别']
woe1 = WOE_dict['利率组别']['(0, 8]']
score1 = -(B * a2 * woe1) + (A - B * b)/dataWOE.shape[0]
woe2 = WOE_dict['利率组别']['(8, 13]']
score2 = -(B * a2 * woe2) + (A - B * b)/dataWOE.shape[0]
woe3 = WOE_dict['利率组别']['(13, 50]']
score3 = -(B * a2 * woe3) + (A - B * b)/dataWOE.shape[0]
'''对应的 SQL 语句
case
    when 0 < '利率' <= 8 then 25

```

```

        when 8 < '利率' <= 13 then 8
        when '利率' > 13 then -8
    else 0
    '''
'''创建 月负债比 单变量得分'''
WOE_dict['月负债比组别']
woe1 = WOE_dict['月负债比组别']['(-1, 20]']
score1 = -(B * a3 * woe1) + (A - B * b)/dataWOE.shape[0]
woe2 = WOE_dict['月负债比组别']['(20, 1000]']
score2 = -(B * a3 * woe2) + (A - B * b)/dataWOE.shape[0]
'''对应的 SQL 语句
case
    when 0 < '月负债比' <= 20 then 2
    when '月负债比' > 20 then -3
else 0
'''
'''创建 贷款等级 单变量得分'''
WOE_dict['贷款等级']
woe1 = WOE_dict['贷款等级'][1]
score1 = -(B * a4 * woe1) + (A - B * b)/dataWOE.shape[0]
woe2 = WOE_dict['贷款等级'][2]
score2 = -(B * a4 * woe2) + (A - B * b)/dataWOE.shape[0]
woe3 = WOE_dict['贷款等级'][3]
score3 = -(B * a4 * woe3) + (A - B * b)/dataWOE.shape[0]
'''对应的 SQL 语句
case
    when '贷款等级' = 1 then 20
    when '贷款等级' = 2 then 0
    when '贷款等级' = 3 then -17
else 0
'''
'''创建 过去 6 个月内被查询次数 单变量得分'''
WOE_dict['过去 6 个月内被查询次数']
woe1 = WOE_dict['过去 6 个月内被查询次数'][0]
score1 = -(B * a5 * woe1) + (A - B * b)/dataWOE.shape[0]
woe2 = WOE_dict['过去 6 个月内被查询次数'][1]
score2 = -(B * a5 * woe2) + (A - B * b)/dataWOE.shape[0]
woe3 = WOE_dict['过去 6 个月内被查询次数'][2]
score3 = -(B * a5 * woe3) + (A - B * b)/dataWOE.shape[0]
woe4 = WOE_dict['过去 6 个月内被查询次数'][3]
score4 = -(B * a5 * woe4) + (A - B * b)/dataWOE.shape[0]
woe5 = WOE_dict['过去 6 个月内被查询次数'][4]
score5 = -(B * a5 * woe5) + (A - B * b)/dataWOE.shape[0]
'''对应的 SQL 语句
case
    when '过去 6 个月内被查询次数' = 1 then 3
    when '过去 6 个月内被查询次数' = 2 then -2
    when '过去 6 个月内被查询次数' = 3 then -5
    when '过去 6 个月内被查询次数' = 4 then -10
    when '过去 6 个月内被查询次数' = 5 then -12
else 0
'''

```


如此，可根据case when算出每个区间的得分，然后将得分相加，即可得到总分，该总分即为金融申请评分卡的最终分数。

```
SELECT LOAN_NO, (每月还款金额 + 贷款金额 + 利率 + 月负债比 + 贷款等级 + 过去 6 个月内被查询
次数) AS risk_score
FROM
(SELECT
    LOAN_NO,
    CASE
        WHEN 0 <= 每月还款金额 <= 300 THEN 7
        WHEN 300 < 每月还款金额 <= 750 THEN -1
        WHEN 每月还款金额 > 750 THEN -11
        ELSE 0
        # 以业务逻辑或缺规则来定
    END 每月还款金额,
    CASE
        WHEN 0 <= 贷款金额 <= 10000 THEN -3
        WHEN 10000 < 贷款金额 <= 20000 THEN 1
        WHEN 贷款金额 > 20000 THEN 4
        ELSE 0
    END 贷款金额,
    CASE
        WHEN 0 < 利率 <= 8 THEN 25
        WHEN 8 < 利率 <= 13 THEN 8
        WHEN 利率 > 13 THEN -8
        ELSE 0
    END 利率,
    CASE
        WHEN 0 < 月负债比 <= 20 THEN 2
        WHEN 月负债比 > 20 THEN -3
        ELSE 0
    END 月负债比,
    CASE
        WHEN 贷款等级 = 1 THEN 20
        WHEN 贷款等级 = 2 THEN 0
        WHEN 贷款等级 = 3 THEN -17
        ELSE 0
    END 贷款等级,
    CASE
        WHEN 过去 6 个月内被查询次数 = 1 THEN 3
        WHEN 过去 6 个月内被查询次数 = 2 THEN -2
        WHEN 过去 6 个月内被查询次数 = 3 THEN -5
        WHEN 过去 6 个月内被查询次数 = 4 THEN -10
        WHEN 过去 6 个月内被查询次数 = 5 THEN -12
        ELSE 0
    END 过去 6 个月内被查询次数,
    FROM 数据库中对应的表) GROUP BY LOAN_NO
```

7.6 申请评分卡的评价、使用与监控

评分卡创建成功后，该如何评价、使用和监控呢？

评价方法除了二分类模型通用的auc、ks指标以外，还有区分度曲线、拟合度曲线。使用的主要方法是风险评分分布表。

评分卡监控主要从两方面来衡量，一个是稳定性指标，一个是有效性指标。

模型的稳定性指标主要是指PSI值，一般来说，如果PSI值在0.25以下，则说明模型是稳定的。除此以外，还可以做出变量稳定性分析报告和评分分布报告。

模型的有效性指标主要是指kendalls' tau值，值越接近1，说明逾期率在分数上的单调下降性越明显，分数越有效。除此以外，还可以做出模型拟合度曲线、逾期拖欠分布报告与模型区分度检验报告。

7.7 小结

本章主要介绍了申请评分卡创建的整个过程，从好坏样本的定义到最终各变量得分都进行了详细讲解。在建模过程中，一定要细心地处理数据，搞清楚业务逻辑，这样得出的评分对于业务才具有实际意义。

第8章 社交网络分析与反欺诈

在实际应用中，创建社交网络需要专门的图数据库，其中Neo4j就是最常用的图数据库之一。Neo4j最大的好处就是可以快速地、直观地匹配节点与节点之间的关系。Neo4j节点间的关系图如图8-1所示，中间的节点连接了很多其他的节点。

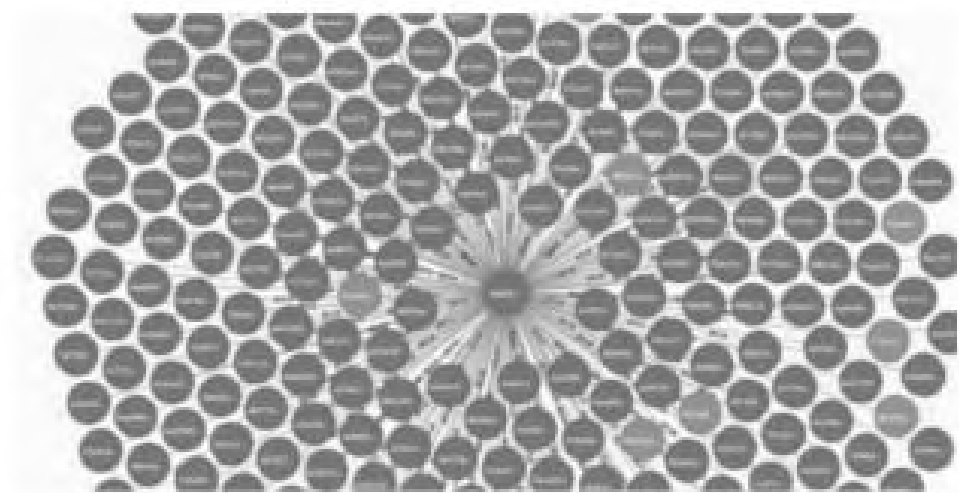


图8-1 Neo4j节点间的关系图

在金融领域，可以利用社交网络查找失联用户、发现欺诈社区、找到活跃中介。在其他领域，可以用Neo4j构造实时推荐引擎，创建六度人脉社交网络，创建工商企业控制人图谱等。本书介绍的Neo4j，一方面是各大公司越来越看重的社交网络图谱，另一方面是可以使用Python与Neo4j数据库进行连接，能够很方便地与数据库进行交互，只需要下载一个第三方包Py2neo即可，安装语句为：`pip install py2neo`。

8.1 Neo4j的下载与安装

Neo4j有很多版本，有企业服务器版，有社区版，还有个人桌面版。一般来说，如果仅是个人单机使用，使用个人桌面版是最方便的；如果需要运维人员部署到服务器中，则需要下载安装社区版；如果数据量特别大，那么可能就要付费购买专用企业服务器版了。

Neo4j的官方链接地址为：<https://neo4j.com/>，如图8-2所示，单击右上角的Download按钮，即可进入下载页面，如图8-3所示，单击左下角的DOWNLOAD NEO4J SERVER按钮，可以看到更多的Neo4j版本，如图8-4所示。安装步骤详见附录D。

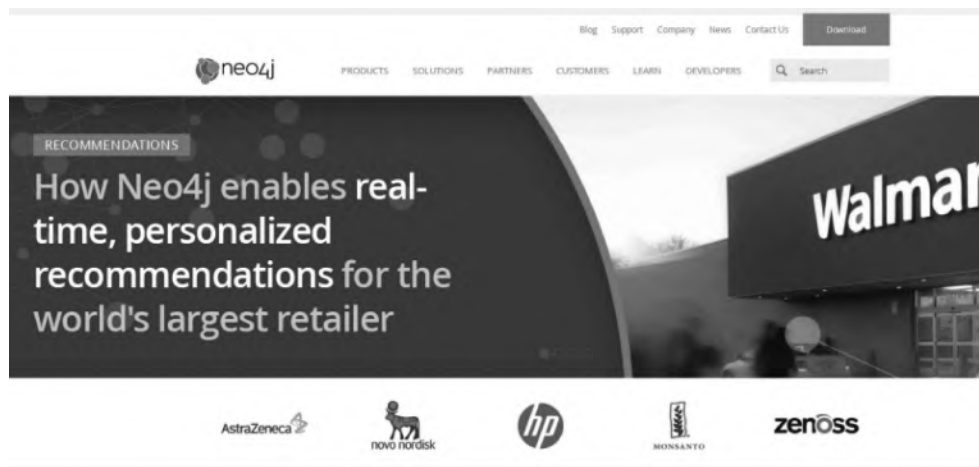


图8-2 Neo4j官方首页展示图

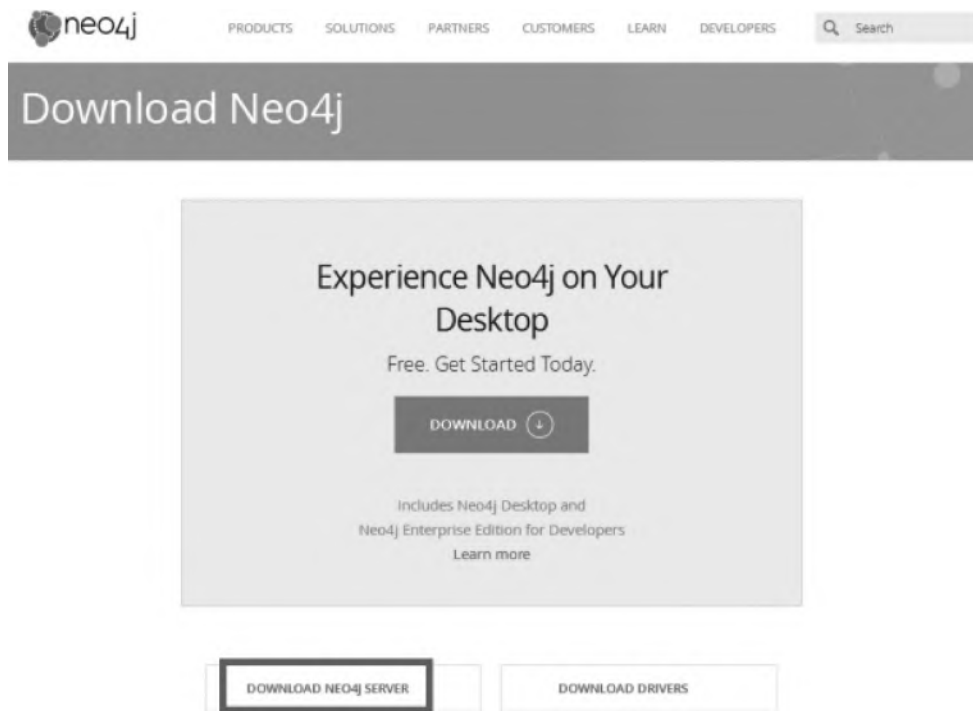


图8-3 Neo4j官方下载链接页面展示图

Neo4j Pre-releases

Get an Early Peek at Neo4j 3.5

Neo4j 3.5.0 alpha01 – For Development Only Release Notes

By downloading and installing this pre-release software, you agree to the terms of the Pre-Release Agreement for Neo4j Software and will be automatically enrolled in our feedback program. Check out our User Experience Improvement Program for details.

Community Edition	Enterprise Edition
Neo4j 3.5.0 alpha01 (Windows, zip) SHA-256	Neo4j 3.5.0 alpha01 (Windows, zip) SHA-256
Neo4j 3.5.0 alpha01 (Linux/Mac, tar) SHA-256	Neo4j 3.5.0 alpha01 (Linux/Mac, tar) SHA-256

Neo4j Desktop

Mac	Neo4j Desktop (dmg)
Linux	Neo4j Desktop (AppImage)
Windows	Neo4j Desktop (exe)

图8-4 Neo4j不同版本页面展示图

Community Edition是社区版，Enterprise Edition是企业版，Neo4j Desktop是桌面版，Neo4j Pre-releases是预发行的最新版。一般来说，下载稳定版本即可，切勿盲目追求最新版。本例下载的是Mac版的Neo4j-community-3.3.2-unix.tar。Neo4j需要jdk 8以上才能运行，所以需要安装jdk 8。jdk 8的具体安装方法详见附录E。

```
zhaikundeMac:bin zhaikun$ ./neo4j start
Active database: graph.db
Directories in use:
  home:      /Users/zhaikun/neo4j
  config:    /Users/zhaikun/neo4j/conf
  logs:      /Users/zhaikun/neo4j/logs
  plugins:   /Users/zhaikun/neo4j/plugins
  import:    /Users/zhaikun/neo4j/import
  data:      /Users/zhaikun/neo4j/data
  certificates: /Users/zhaikun/neo4j/certificates
  run:       /Users/zhaikun/neo4j/run
Starting Neo4j.
Started neo4j (pid 1280). It is available at http://localhost:7474/
There may be a short delay until the server is ready.
See /Users/zhaikun/neo4j/logs/neo4j.log for current status.
zhaikundeMac:bin zhaikun$
```

图8-5 Mac版Neo4j启动界面图

配置好环境以后，在Mac下打开Neo4j的命令是./neo4j start，如图8-5所示。启动成功以后，在浏览器中打开链接地址：<http://localhost:7474/>，如图8-6所示。

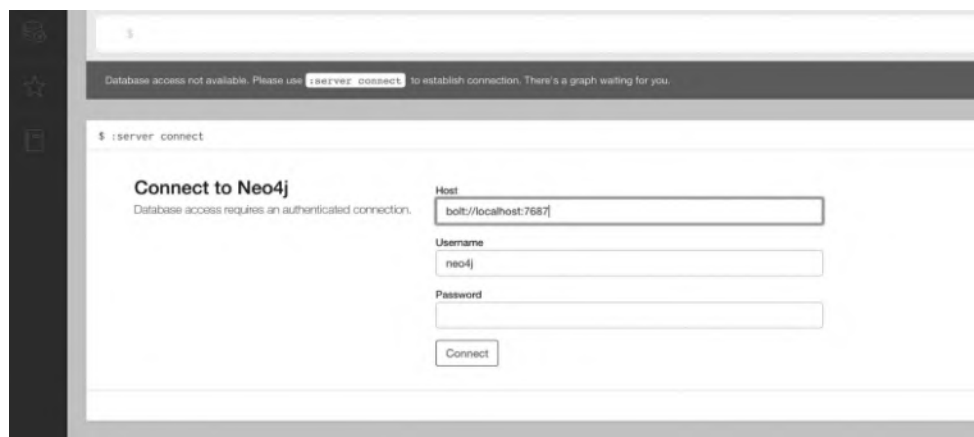


图8-6 Neo4j配置连接的主界面图

初始用户名和密码都是Neo4j，首次打开需要修改密码，如图8-7所示。

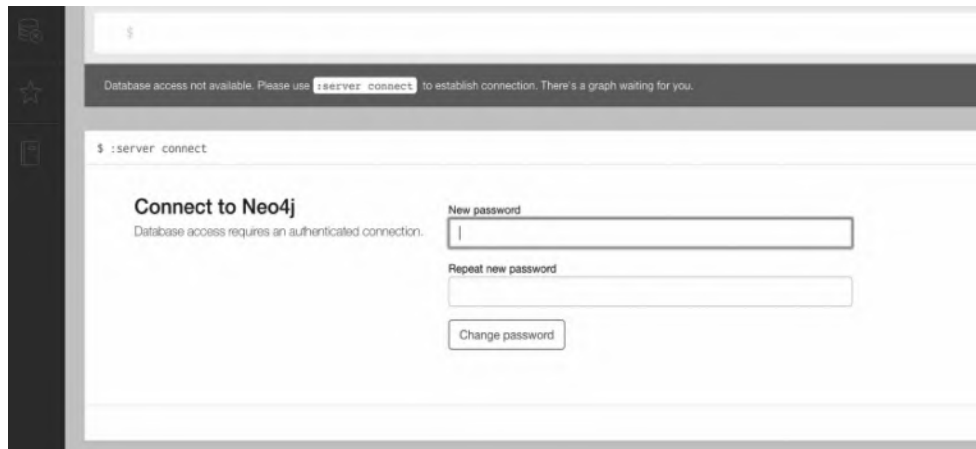


图8-7 Neo4j重置密码界面图

重置密码后，就进入了如下界面，如图8-8所示。

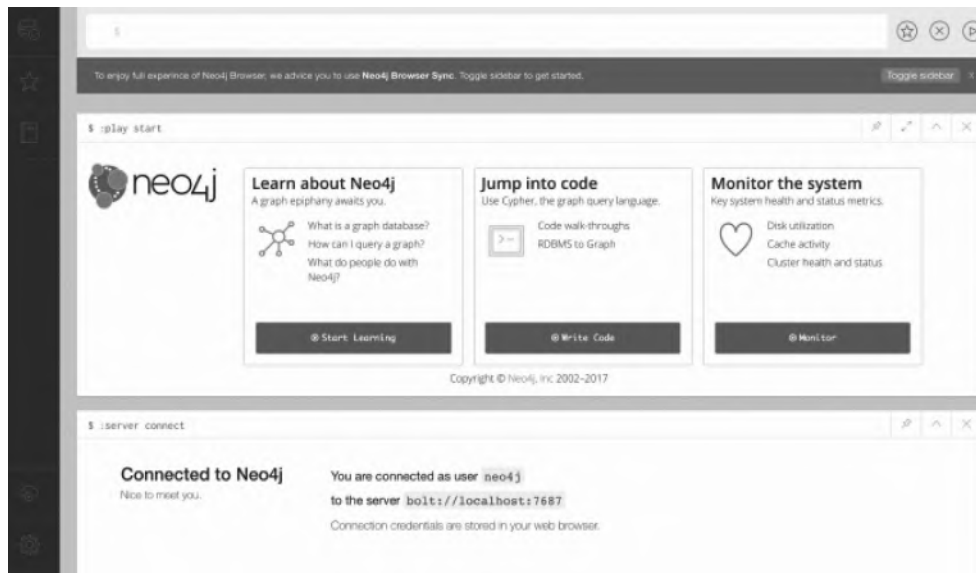


图8-8 Neo4j配置连接完成界面图

想要使用Neo4j中文版的读者，可以访问<http://www.we-yun.com/>，下载中文版的进行练习。

8.2 图形界面介绍

相信大家一定迫不及待地想看看图数据库是个什么样的效果。这边先举个简单的例子：单击中间框Write Code按钮，然后单击create a graph按钮，会出现movie graph案例介绍，往右翻页就是相关代码，运行代码后的结果如图8-9所示。



图8-9 Neo4j自带图例界面图

由图8-9可以看出，每个圆圈代表一个节点，每条边代表着一个关系。左上角蓝色图标“person”表示用蓝色代表“人物”，绿色图标“movie”表示用绿色代表“电影”。

单击蓝色人物图标，如图8-10所示。

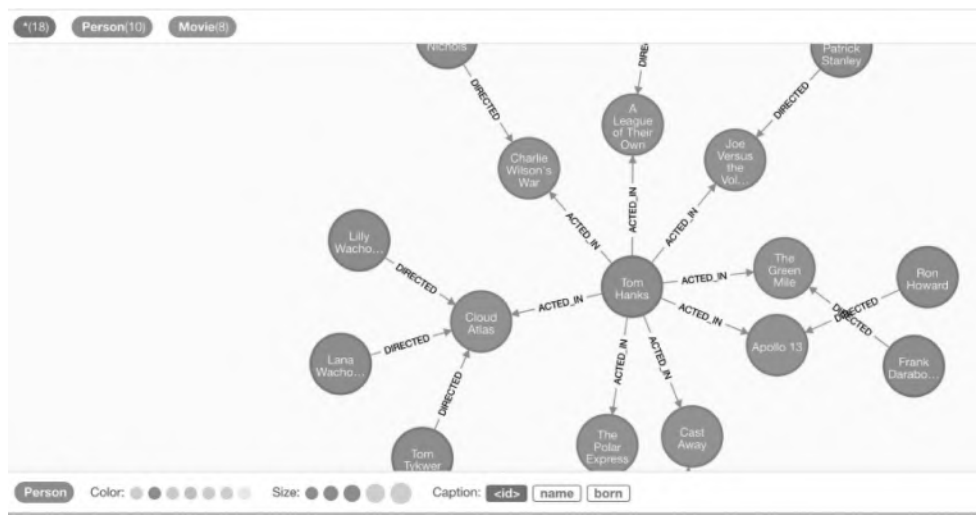


图8-10 Neo4j自带图例单击人物图标后的界面图

最下方出现了一行功能栏，表示可以选择颜色、大小和显示的内容。调整图像大小和颜色之后，所得结果如图8-11所示。

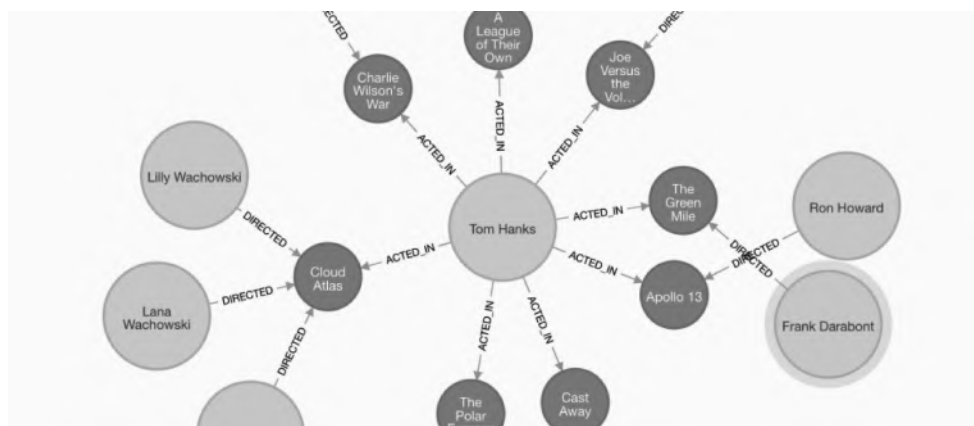


图8-11 Neo4j自带图例调整后的界面图

大家会发现人物图标变大了，颜色也变了。这样的功能可以很方便地标识出不同的节点，更加直观易懂。

单击节点，如图8-12所示。

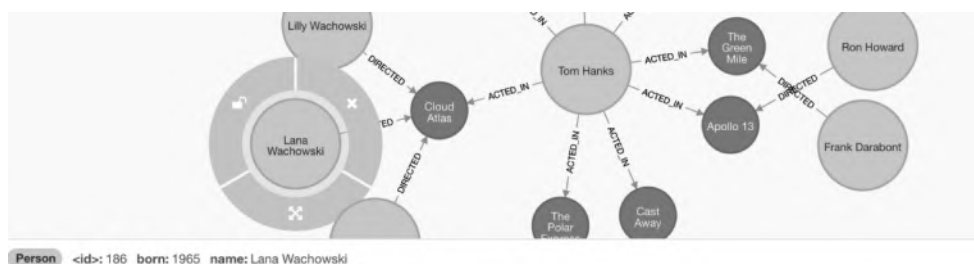


图8-12 Neo4j自带图例单击节点后的界面图

这时，下面的功能区会显示节点的属性。

同理，单击关系，下面的功能区也会显示关系的属性。如图8-13所示。

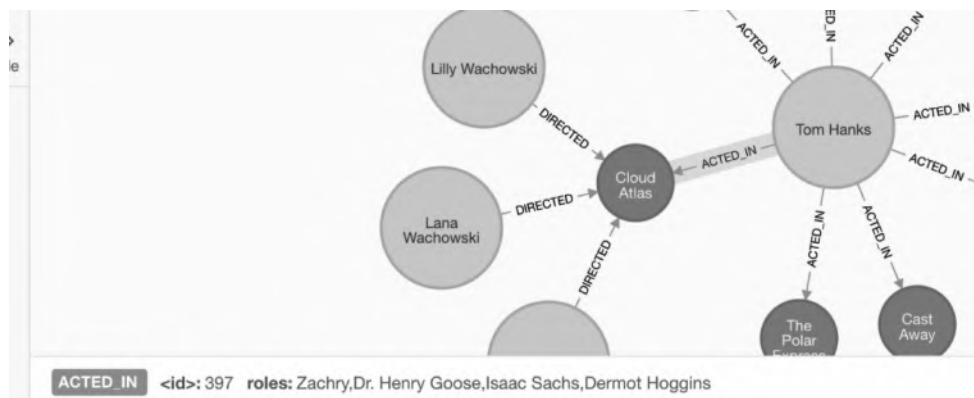


图8-13 Neo4j自带图例单击关系后的界面图

8.3 Cypher语言

Neo4j是一个NoSQL数据库，用的查询语言是Cypher语言，与SQL语句非常相似。本节将介绍Cypher的基本语法。

Cypher是一种声明式图查询语言，具有语法简单、功能强大等优点。通常，学习过SQL语句的读者可以很轻易地学会Cypher语言的使用。

1. Cypher语言常用关键词语法

Cypher语言常用的关键词有4个，其语法的详细教程可在Neo4j客户端查看。当Neo4j连接配置完成后，在界面最上方输入“: help cypher”，如图8-14所示。Cypher语言各关键词对应语法如图8-15所示。

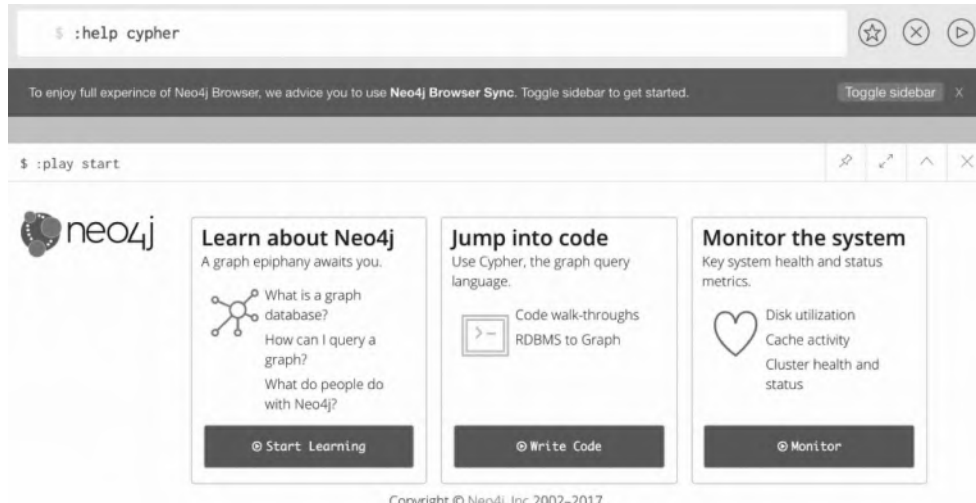


图8-14 Neo4j中输入“: help cypher”的界面图

2. Cypher语言常用关键词的作用与用法

(1) CREATE的作用——创建

创建节点：

```
CREATE (n {name: $ value})
```

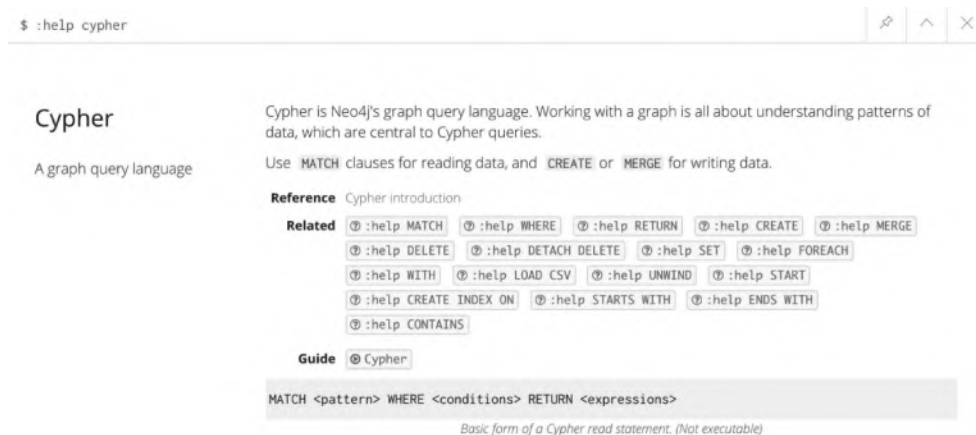


图8-15 Cypher语言各关键词对应语法信息界面图

创建关系：

```
CREATE (n) - [r:KNOWS] ->(m)
CREATE (n) - [:LOVES {since: $ value}] ->(m)
```

(2) MATCH的作用——查询匹配

查询节点：

```
MATCH (n:Person) - [:KNOWS] ->(m:Person)
WHERE n.name = 'Alice'
```

查询路径：

```
MATCH p = (n) -->(m)
```

查询具有属性的节点：

```
MATCH (n {name: 'Alice'}) -->(m)
```

(3) WHERE的作用——条件选择

```
WHERE n.property = 100
```

(4) LOAD CSV的作用——导入文件

```
LOAD CSV WITH HEADERS FROM
'file:///test.csv' as line
CREATE (:Artist {name: line.Name, year:toInteger(line.Year)})
```

8.4 Neo4j案例1——《天龙八部》的人物关系分析

本节将以《天龙八部》这本小说为例，讲解一下Neo4j的具体使用方法。

```

'''导入文件'''
using periodic commit 1000
load csv with headers from 'file:///tianlongbabu.csv' as line
merge (n1:人物 {姓名:line.Source})           //创建源节点
merge (n2:人物 {姓名:line.Target})           //创建目标节点
merge (n1) - [r:相识] -> (n2)                 //创建关系
on create set r.weight = toInt(line.Weight)   //添加关系属性 weight

```

将天龙八部数据的文件放在Neo4j文件夹下的import文件夹里，如图8-16和图8-17所示。

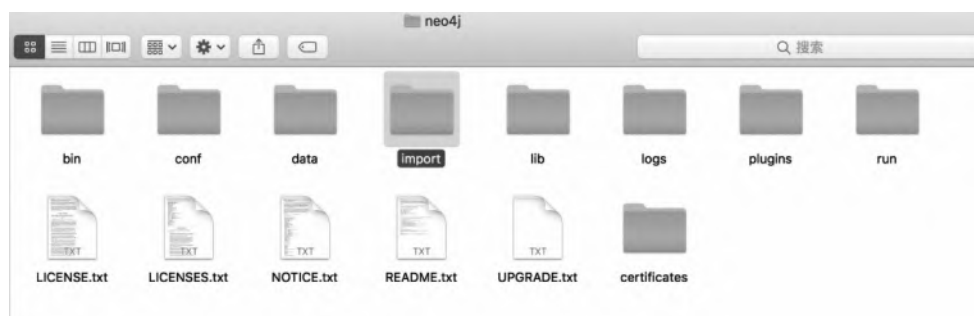


图8-16 Neo4j的文件结构展示图



图8-17 天龙八部数据放在import文件夹下的展示图

在浏览器中打开链接地址：<http://localhost:7474/>，单击中间框Write Code按钮，输入如图8-18所示的代码，导入文件。

导入结果为导入了1164个标签、1164个节点，设置了9934个属性，创建了8770个边。

```
1 using periodic commit 1000
2 load csv with headers from 'file:///tianlongbabu.csv' as line
3 merge (n1:人物 {姓名:line.Source}) //创建源节点
4 merge (n2:人物 {姓名:line.Target}) //创建目标节点
5 merge (n1)-[r:相识]->(n2) //创建关系
6 on create set r.weight = toInt(line.Weight) //添加关系属性weight
7
```

Added 1164 labels, created 1164 nodes, set 9934 properties, created 8770 relationships, completed after 2378 ms.

图8-18 导入数据之后的结果展示图

下面预览一部分数据，让大家有个直观的印象，如图8-19所示。

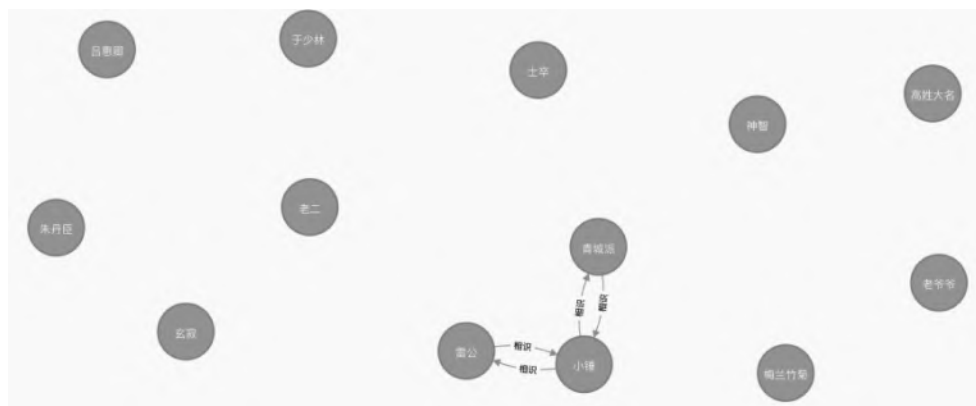


图8-19 导入数据之后的节点预览展示图

图数据库最重要的应用就是进行社交网络分析，应用如下。

一度人脉：

```
match p = (n1{姓名:'木婉清'}) -[*..1] - (n2{姓名:'阿朱'}) return p limit 10
```

运行结果如图8-20所示。说明木婉清与阿朱没有直接认识。

(no changes, no records)

图8-20 木婉清与阿朱的一度人脉结果图

二度人脉。

```
match p = (n1{姓名:'木婉清'}) -[*..2] - (n2{姓名:'阿朱'}) return p limit 10
```

运行结果如图8-21所示。

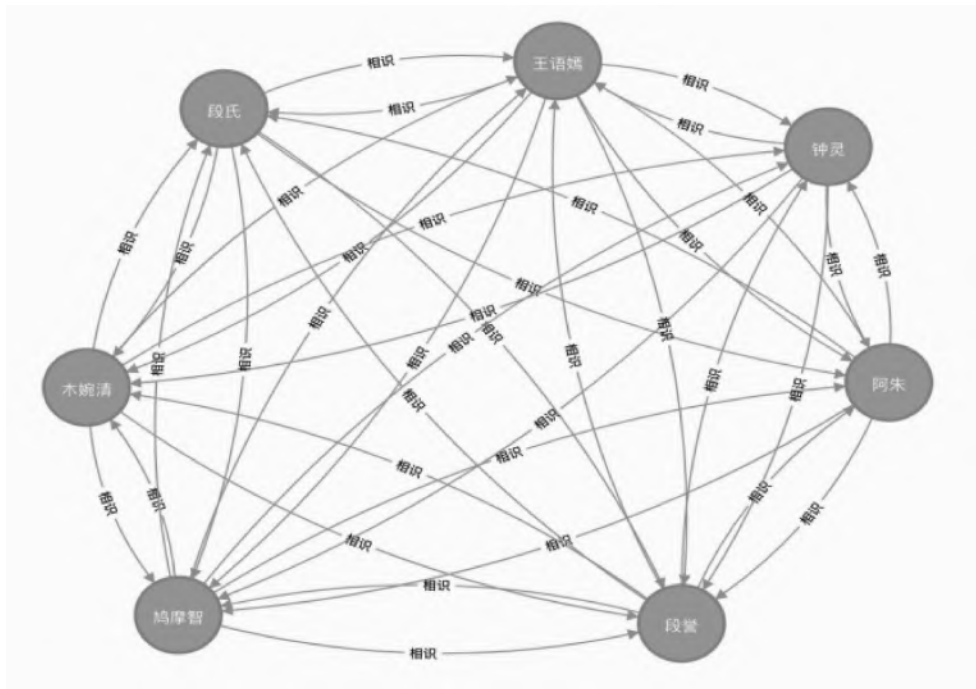


图8-21 木婉清与阿朱的二度人脉结果图

说明木婉清想认识阿朱可以通过“段氏”“王语嫣”“钟灵”等人。金融上常用的是一度和二度人脉，再扩展下去，其实际意义不是很大，感兴趣的读者可以继续扩展。

最短路径：

```
match p = shortestpath((n1{姓名:'木婉清'}) -[*..6] - (n2{姓名:'阿朱'})) return p
```

运行结果如图8-22所示。

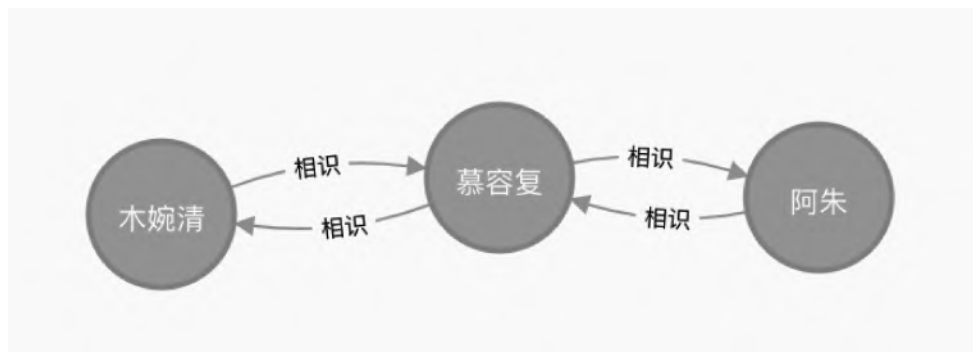


图8-22 木婉清与阿朱认识的最短路径结果图

这是六度人脉范围内，木婉清希望认识阿朱的任意一条最短路径。

度中心性：就是节点的连接边数。

```

match (n:人物)
return n.姓名 as 姓名, size((n)-[]-()) as degree
order by degree desc
limit 10
  
```

运行结果如图8-23所示。

姓名	degree
"段誉"	600
"萧峰"	428
"乔峰"	310
"慕容复"	266
"阿朱"	246
"王语嫣"	236
"段正淳"	236
"木婉清"	228
"鸠摩智"	222
"少林寺"	200

图8-23 天龙八部中的度中心性结果排序展示图

结果很有意思，段誉的关系数最多。那么谁与段誉的关系最亲密呢？代码如下：


```

match (n1:人物{姓名:'段誉'})-[:r]->(n2)
return n1.姓名 as 源, n2.姓名 as 目标, r.weight as 关系权重
order by 关系权重 desc
limit 10

```

运行结果如图8-24所示。

源	目标	关系权重
"段誉"	"王语嫣"	568
"段誉"	"木婉清"	469
"段誉"	"慕容复"	259
"段誉"	"钟灵"	220
"段誉"	"鸠摩智"	198
"段誉"	"王姑娘"	130
"段誉"	"段公子"	121
"段誉"	"阿朱"	119
"段誉"	"段正淳"	113
"段誉"	"萧峰"	110

图8-24 天龙八部中与段誉关系最亲密排序展示图

结果不出所料，段誉还是很有妹子缘的，并且王语嫣遥遥领先。

8.5 Neo4j案例2——金融场景中的社交网络分析

本节将用工业上常用的Neo4j数据库进行社交网络与反欺诈分析。

先用admin import语句导入节点csv文件和关系csv文件。因为此文件涉及敏感信息，所以不会上传。那么大家在尝试的时候，可以根据文件格式，编造类似数据，进行文件导入。

先看看坏客户与其相关联的亲友界面图，如图8-25所示。

语句为：

```

MATCH p = (n1:坏客户) -->(n2:坏客户亲友) RETURN p LIMIT 25

```



图8-25 坏客户与其相关联的亲友图谱

好客户与其相关联的亲友图谱，如图8-26所示。

语句为：

```
MATCH p = (n1:好客户) -->(n2:好客户亲友) RETURN p LIMIT 25
```

善于观察的读者可以发现，图8-25中正中间的圆圈代表的是坏客户，周边的圆圈代表的是坏客户的联系人；图8-26中正中间的圆圈代表的是好客户，周边的圆圈代表的是好客户的联系人。

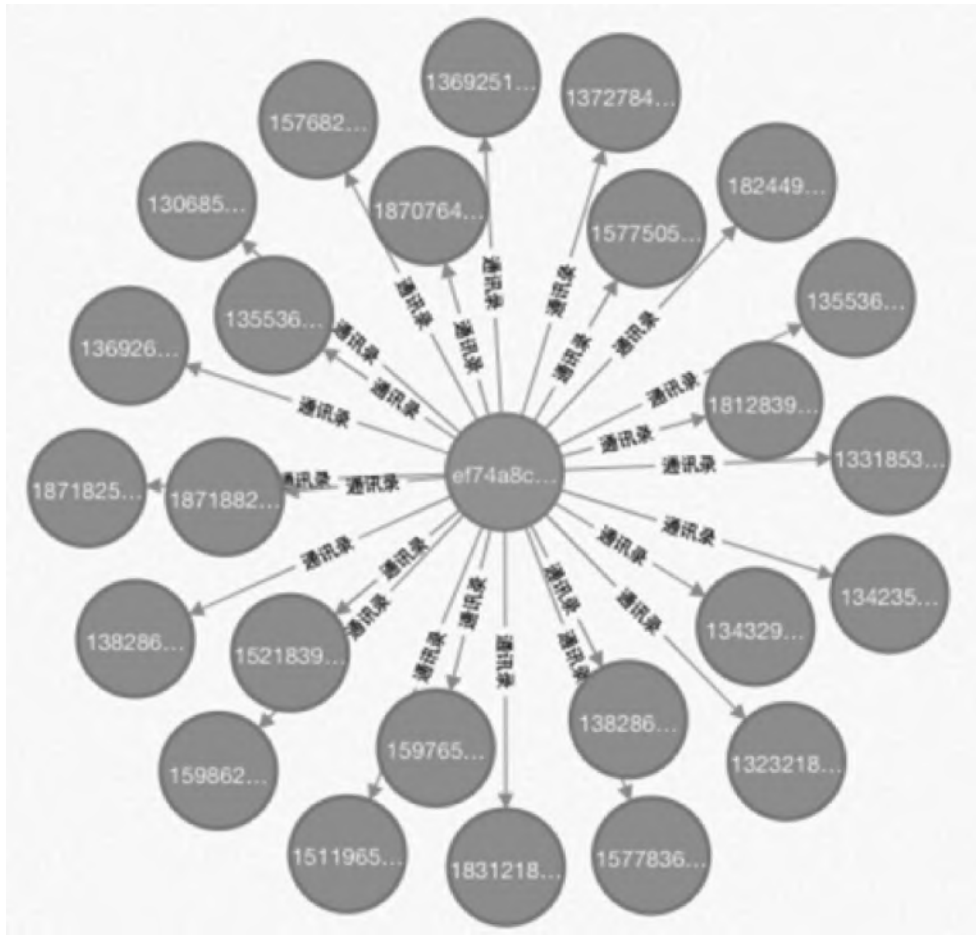


图8-26 好客户与其相关联的亲友图谱

接下来看看二度人脉。

先看看坏客户与坏客户之间有没有连接呢？坏客户与坏客户之间的图谱如图8-27所示。

```
match p = (n1:坏客户) -[*..2] - (n2:坏客户) return p limit 25
```

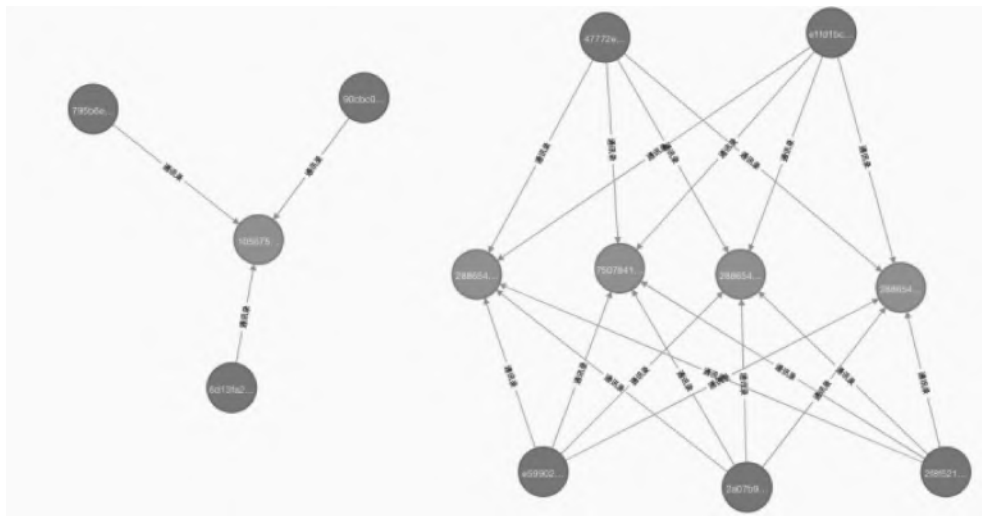


图8-27 坏客户与坏客户之间的图谱

从图8-27中，可以发现坏客户与坏客户之间是有连接的。仔细一看，相关联的电话号码还很类似，如果这些号码是黑中介的号码，那么一般情况下，这些客户就是欺诈客户，可以拒绝他们的贷款申请。

再来看看好客户与好客户之间有没有连接吧。好客户与好客户之间的图谱如图8-28所示。



图8-28 好客户与好客户之间的图谱

六度人脉内都没有连接，看来没有这么一个人同时连接了多个正常客户。

接下来，再分析3个指标吧。

(1) 度中心性

```
match (n:好客户)
return n.user_id as id, size((n)-[:通讯录]-()) as 度中心性, labels(n) as 标签
order by 度中心性 desc
```

运行结果如图8-29所示。

id	度中心性	标签
"f88a5539983543e5a88f57a8668720e8"	772	["好客户"]
"f48dc6a67b1d4f66b6511d63e2deadf1"	723	["好客户"]
"fb28ad2735b44191a2e9fd4e51f94001"	204	["好客户"]
"4b8af0e28ea84323b4923ebb68545486"	172	["好客户"]
"a45c2e3738e348a59612f40eb8480d0c"	137	["好客户"]
"879c9769a27542c391036cf47b2a693c"	123	["好客户"]
"87dcde927e9d460599488926cbad38b2"	121	["好客户"]
"c27d6537ebce49d9b0777bb24045f956"	118	["好客户"]
"11018701134d41b7beb351e3cabb912c"	117	["好客户"]
"c68bafdfb1be4a0f921f62c70ad5e98e"	117	["好客户"]
"5487e8751773485cb0c149e455c062ae"	87	["好客户"]
"d155a1e0cade4f34ba16d1948b88fdd9"	74	["好客户"]
"72a6e71233454e9c96accfe2c0cdb6e6"	70	["好客户"]
"cb9457e3a9084b329ef5a67c16d4840f"	51	["好客户"]
"ef74a8cb9aa640ca97d0b71e4cf9e159"	47	["好客户"]
"2576bbd4427d442fa3293f2be769a760"	24	["好客户"]

图8-29 好客户的度中心性排序图

(2) 接近中心性

```
match (n1)
with collect(n1) as nodes
call apoc.algo.closeness([ '通讯录'], nodes, 'both') yield node, score
return node.user_id as id, score as 接近中心性
order by 接近中心性 desc limit 100
```

运行结果如图8-30所示。

id	接近中心性
"10e072d66a5a4f92a7d385aeeb9cceb2"	0.1
"8cdf08b21bba46d4a411b9dc4b1e933a"	0.1
"f615f73c8e7f4d26b0283a0c4d3b693d"	0.07692307692307693
"0a9ee82e15e8473ca4fc432dbc822d42"	0.07142857142857142
null	0.05263157894736842
null	0.05263157894736842
null	0.05263157894736842
null	0.05263157894736842
null	0.05263157894736842
null	0.05263157894736842
null	0.05263157894736842

图8-30 客户的接近中心性排序图

(3) 中介中心性

```
match (n1)
with collect(n1) as nodes
call apoc.algo.betweenness(['通讯录'], nodes, 'both') yield node, score
set node.betweenness = score
return node.user_id as id, score as 中介中心性
order by 中介中心性 desc limit 100
```

运行结果如图8-31所示。

下面把以上指标最高的节点挑出来，看看他们的连接状态。

为了让大家有更直观的印象，下面选度中心性为24的节点，看看它的连接情况。

运行代码如下：

```
match p = (n1{user_id:'2576bbd4427d442fa3293f2be769a760'}) - [] ->(n2) return p
```

运行结果如图8-32所示。

由图8-32可知，这个节点的连接边数就是24。

id	中介中心性
"e59902bf0bd94f378913451c2ea650bc"	833759.2
"e1fd1bc45f6e4b908f327b94d281edb9"	667349.2
"0b9643dcdb4543738069f91a38b0ffc3"	506521
"f88a5539983543e5a88f57a8668720e8"	297606
"f48dc6a67b1d4f66b6511d63e2deadf1"	261003
null	194094
null	194094
null	194094
null	194094
"47772e023725426e9a1fe2c4920940a2"	192833.2
"6d13fa2ece9a4cc2a73905233835c3b5"	155991

图8-31 客户的中介中心性排序图

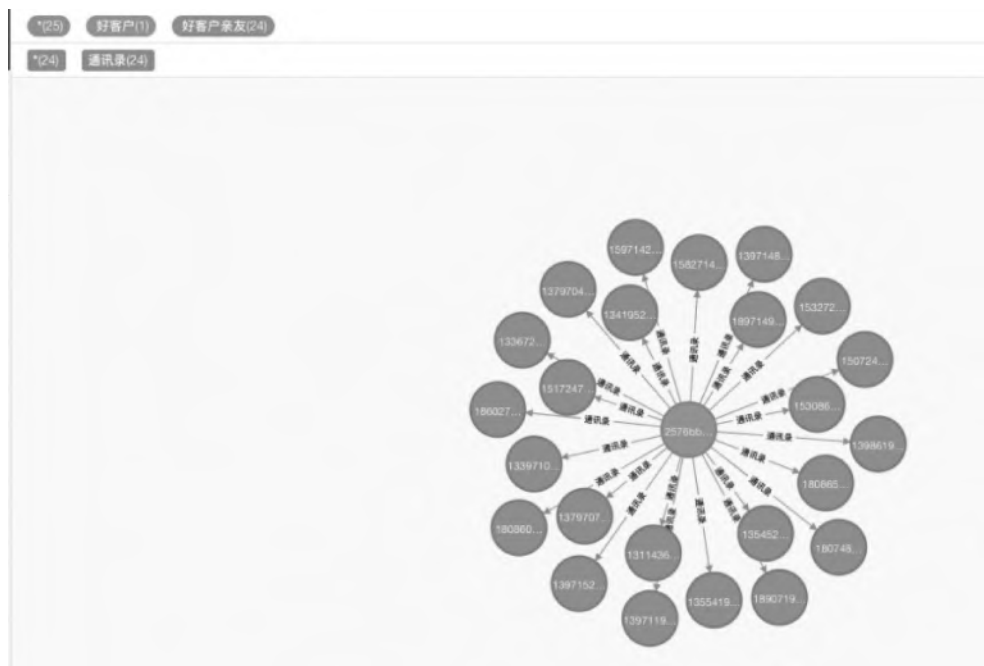


图8-32 度中心性为24的节点关系图谱

Neo4j的语法与应用场景非常多，还能用于坏客户社区探索，交易欺诈判定、供应链金融等。

8.6 Py2neo

Neo4j是一个非常好用的图关系数据库，它可以支持Python驱动，一般用Py2neo，安装语句为：`pip install py2neo`，具体使用方法官网上有非常详细的解释，感兴趣的读者可以参考官网地址：<http://py2neo.org/v3/index.html>，如图8-33所示。安装步骤详见附录F。

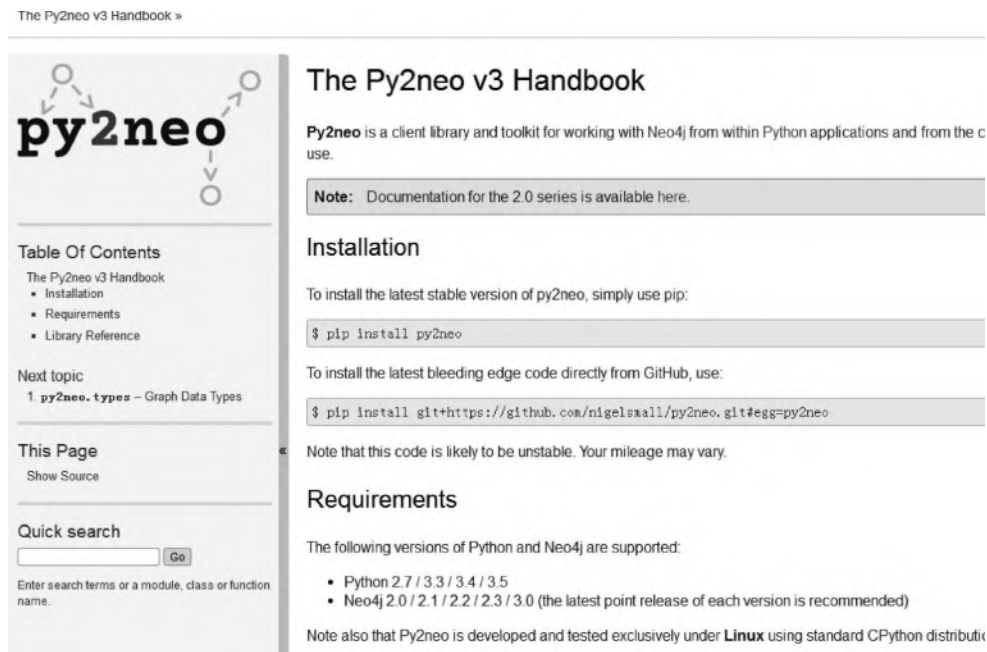


图8-33 Py2neo官网指导手册展示图

在完成安装之后，在Python中调用py2neo即可，常用的两个数据结构就是节点和关系，即Node和Relationship。

```
from py2neo import Graph, Node, Relationship
```

连接Neo4j的方法很简单，代码如下：

```
test_graph = Graph(
    "http://localhost:7474",
    username = "neo4j",
    password = "neo4j"
)
```


test_graph就是已建立好的Neo4j的连接。在连接Neo4j数据库之后，就可以创建节点了。节点的建立要用到py2neo.Node，建立节点的时候要定义它的节点类型（label）以及一个基本属性（property，包括property_key和property_value）。以下代码为建立了两个测试节点。

```
test_node_1 = Node(label="Person", name="test_node_1")
test_node_2 = Node(label="Person", name="test_node_2")
test_graph.create(test_node_1)
test_graph.create(test_node_2)
```

这两个节点的类型（label）都是Person，而且都具有属性（property_key）为name，属性值（property_value）分别为 "test_node_1"，"test_node_2"。

创建好节点之后，就可以创建节点间的关系（Relationship）了。由于节点关系是有向的，所以在建立关系时，必须定义一个起始节点和一个结束节点。值得注意的是，起始节点可以和结束节点是同一个点，这时候该节点就存在一个指向自己的关系。

```
node_1_call_node_2 = Relationship(test_node_1, 'CALL', test_node_2)
node_1_call_node_2['count'] = 1
node_2_call_node_1 = Relationship(test_node_2, 'CALL', test_node_1)
node_2_call_node_1['count'] = 2
test_graph.create(node_1_call_node_2)
test_graph.create(node_2_call_node_1)
```

如以上代码，分别建立了test_node_1指向test_node_2和test_node_2指向test_node_1两条关系，关系的类型为"CALL"，两条关系都有属性count，且值为1。

在这里有必要提一下，当建立关系时，如果起始节点或者结束节点不存在，则在建立关系的同时建立这个节点。

感兴趣的读者可以参考官方文档进行属性、关系的更新等操作，这里就不再赘述。

8.7 小结

本章主要介绍了Neo4j数据库、Cypher语言，针对Neo4j数据库的操作，以两个案例进行讲解，最后介绍了Py2neo对Neo4j进行相关操作。由于Neo4j数据库应用场景非常多，那么在后续的学习中，考验读者的还是查找资料、理解消化能力。

参考文献

- [1] David Beazley, Brian K Jones. Python Cookbook (第3版) 中文版[M]. 陈舸, 译. 北京: 人民邮电出版社, 2015.
- [2] Mamdouh Refaat. 信用风险评分卡研究: 基于SAS的开发与实施[M]. 王松奇, 林治乾, 译. 北京: 社会科学文献出版社, 2013.
- [3] Raymond Anderson. 信用评分工具: 自动化信用管理的理论与实践[M]. 李志勇, 译. 北京: 中国金融出版社, 2017.
- [4] Wes McKinney. 利用Python进行数据分析[M]. 唐学韬, 等译. 北京: 机械工业出版社, 2014.
- [5] 陈建. 信用评分模型技术与应用[M]. 北京: 中国财政经济出版社, 2005.

附录A PyCharm安装步骤

PyCharm官网链接地址为：<http://www.jetbrains.com/pycharm>，
下载链接地址为：
<http://www.jetbrains.com/pycharm/download/previous.html>。

下面以PyCharm Community为例，进行安装说明。

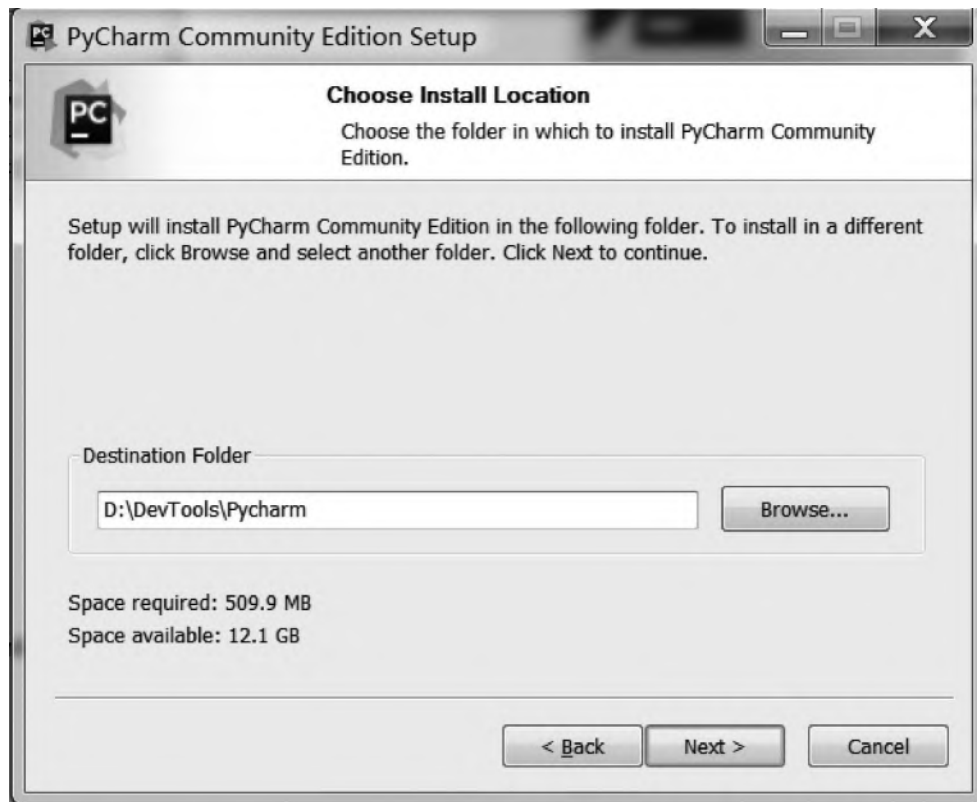
- (1) 右击“安装包”，单击“以管理员身份运行”。
- (2) 单击Next按钮，如图A-1所示。



图A-1 PyCharm Community安装界面图（1）

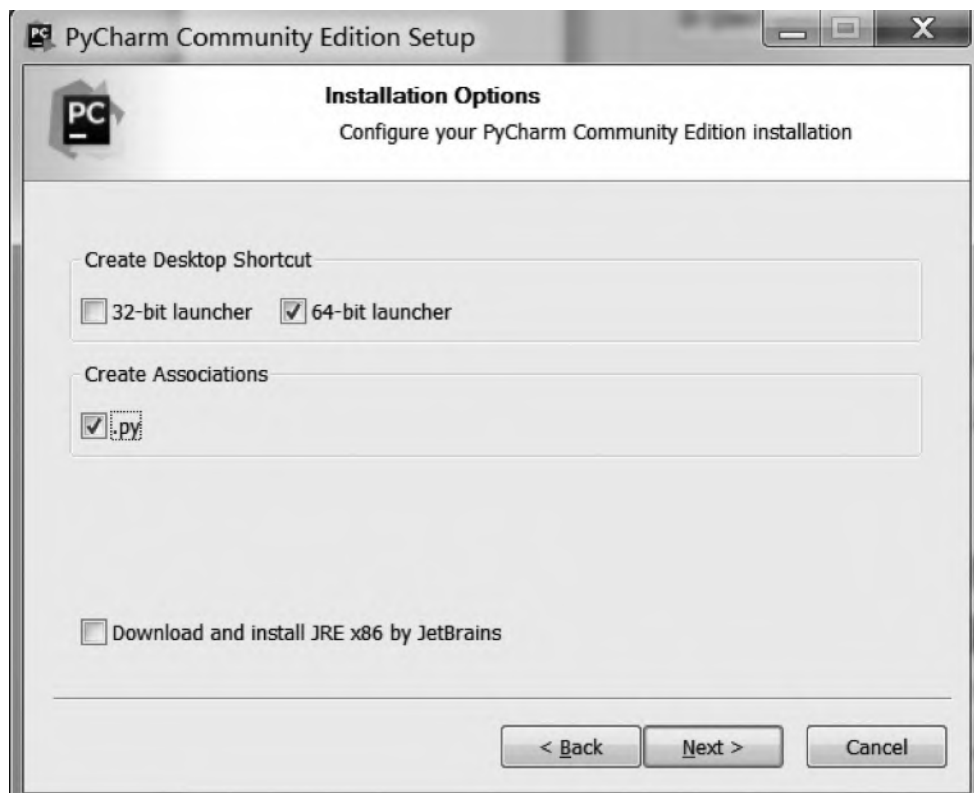
(3) 程序默认安装位置为C:\Program Files\JetBrains\PyCharm Community Edition 2018.2.5，单击Browse可选择自定义安装目录，

本安装教程的自定义安装路径为D:\DevTools\Pycharm，单击Next按钮，如图A-2所示。



图A-2 PyCharm Community安装界面图（2）

（4）勾选64-bit launcher和.py（如果是32位操作系统，则勾选32-bit launcher），单击Next按钮，如图A-3所示。



图A-3 PyCharm Community安装界面图（3）

（5）单击Install按钮，如图A-4所示。



图A-4 PyCharm Community安装界面图（4）

（6）安装大概需要3~4分钟，单击Finish按钮，至此完成了PyCharm Community的安装，如图A-5所示。



图A-5 PyCharm Community安装界面图（5）

附录B MySQL安装步骤

MySQL官网链接地址为：<https://www.mysql.com/>，下载链接地址为：<https://www.mysql.com/downloads/>。

本节以MySQL 5.7.13为例，进行安装说明。

(1) 右击“安装包”，单击“安装”按钮。

(2) 勾选I accept the license terms，单击Next按钮，如图B-1所示。

(3) 默认勾选Developer Default，本安装教程选择勾选Custom，进行自定义安装，单击Next按钮，如图B-2所示。

(4) 本安装教程只选择安装MySQL Server 5.7.13，单击Next按钮，如图B-3所示。

(5) 单击Execute按钮，开始安装MySQL Server 5.7.13，如图B-4所示。

(6) 单击Next按钮，如图B-5所示。

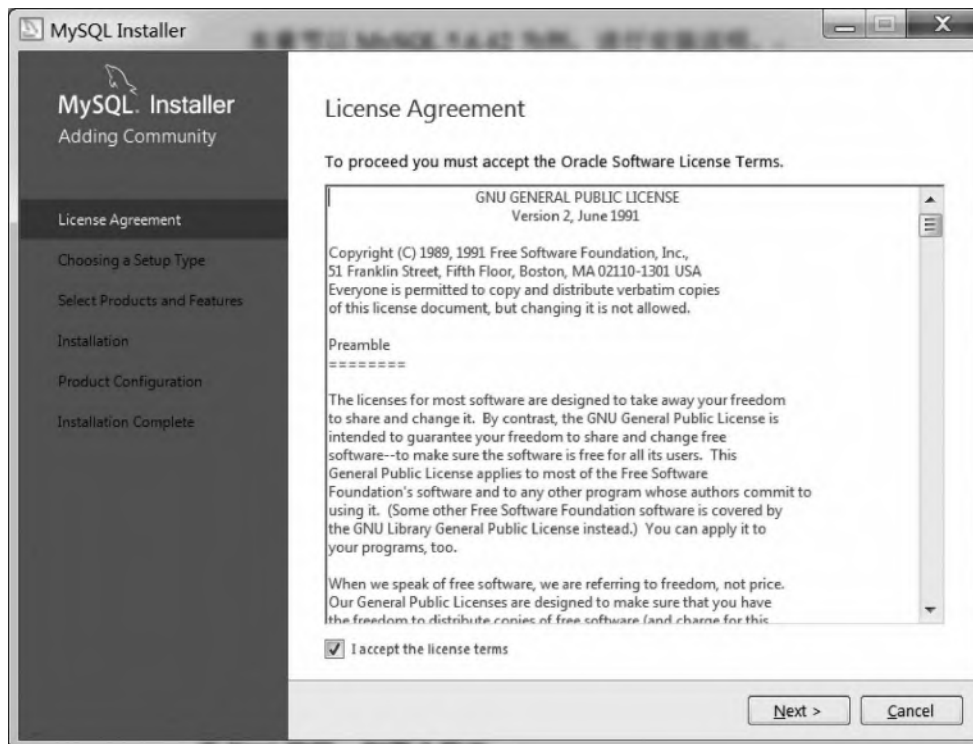
(7) 单击Next按钮，如图B-6所示。

(8) 单击Next按钮，如图B-7所示。

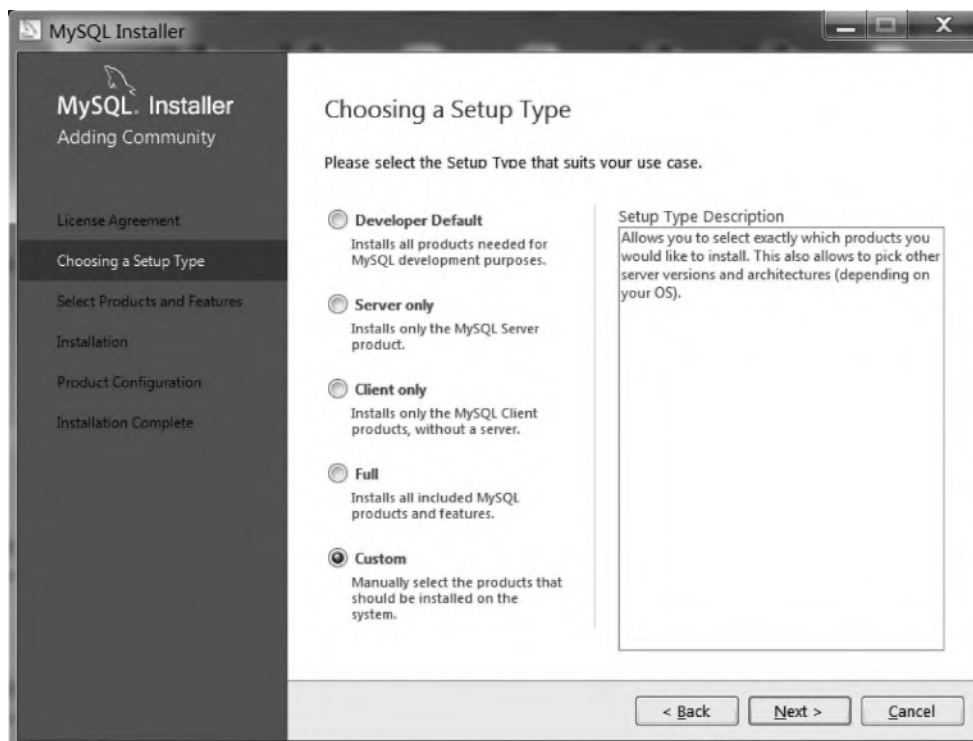
(9) 输入两次密码，然后单击Next按钮，如图B-8所示。

(10) 单击Next按钮，如图B-9所示。

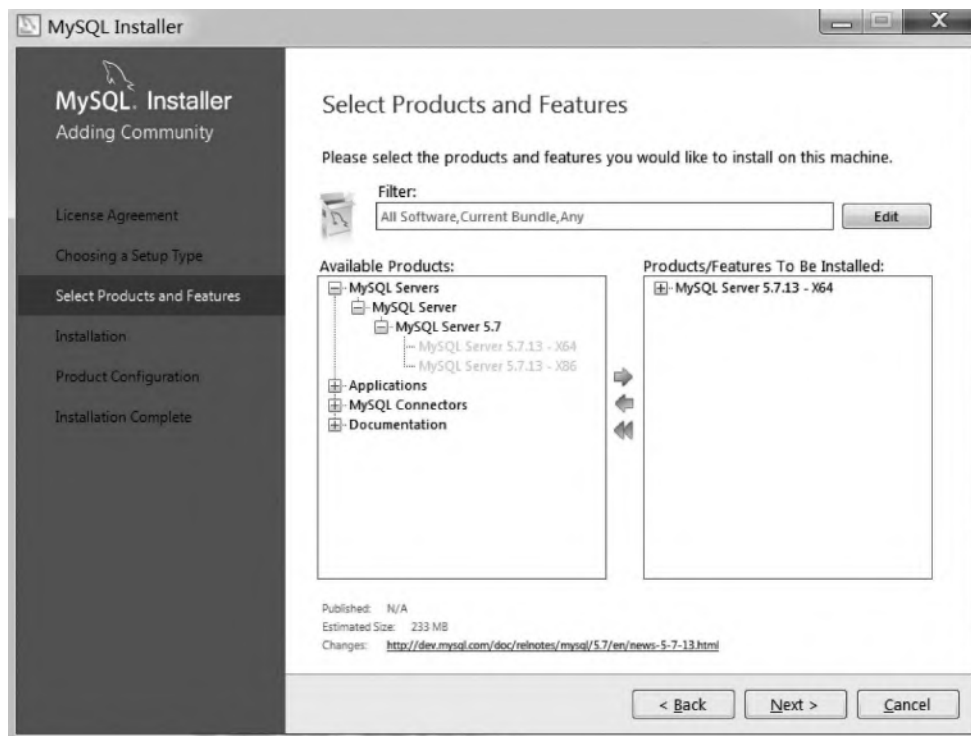
(11) 单击Next按钮，如图B-10所示。



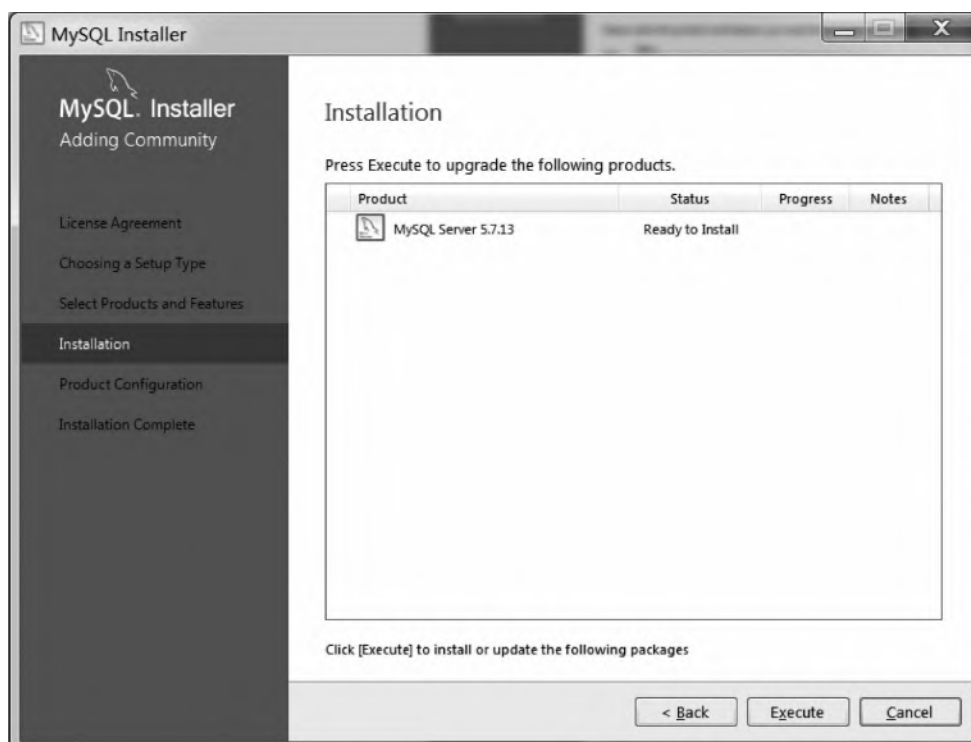
图B-1 MySQL 5.7.13安装界面图（1）



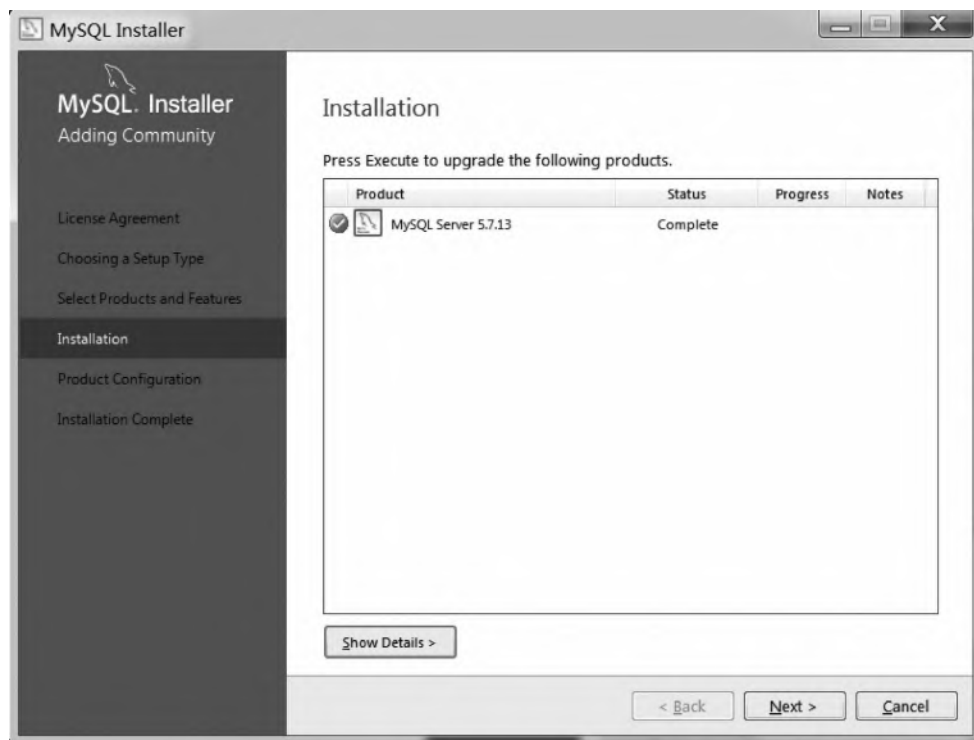
图B-2 MySQL 5.7.13安装界面图（2）



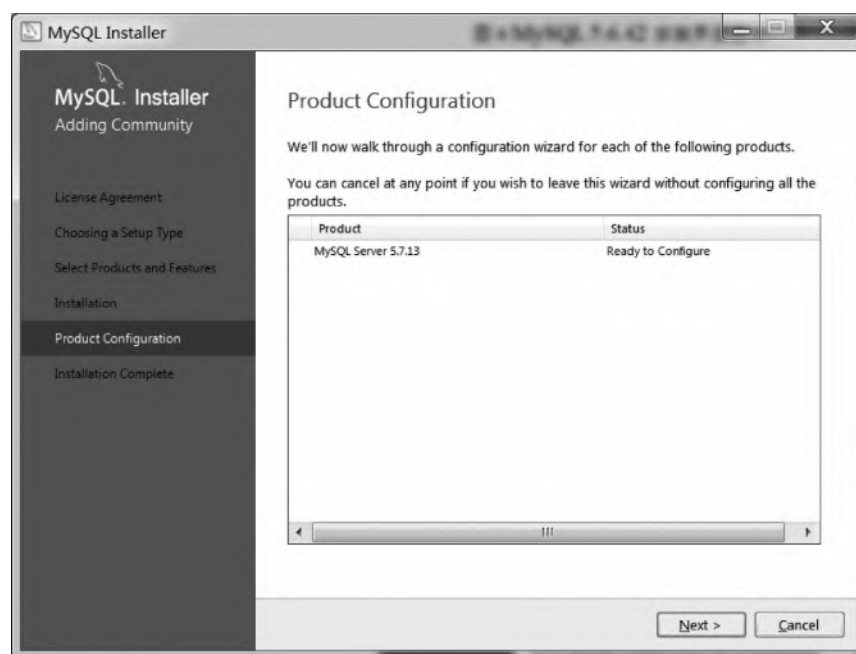
图B-3 MySQL 5.7.13安装界面图（3）



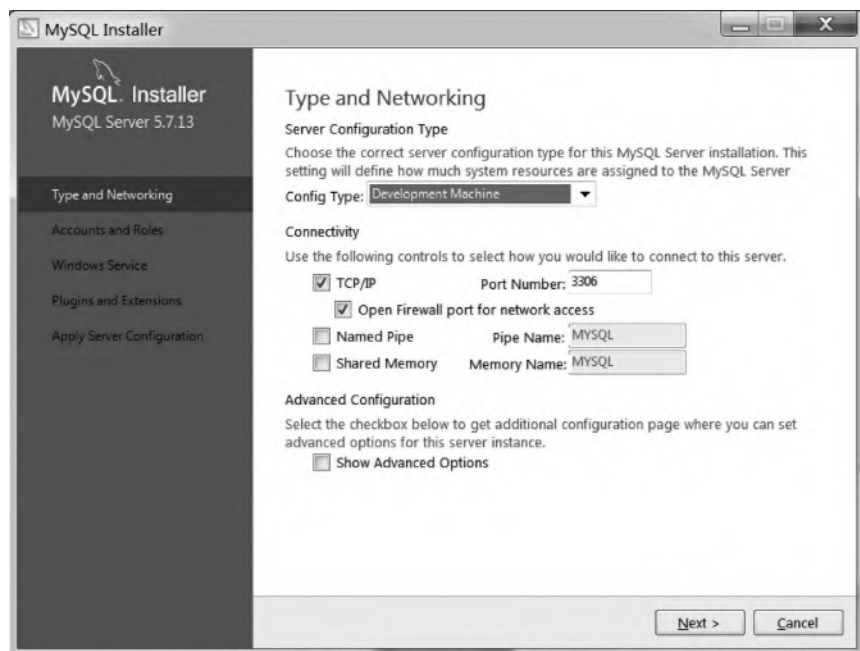
图B-4 MySQL 5.7.13安装界面图（4）



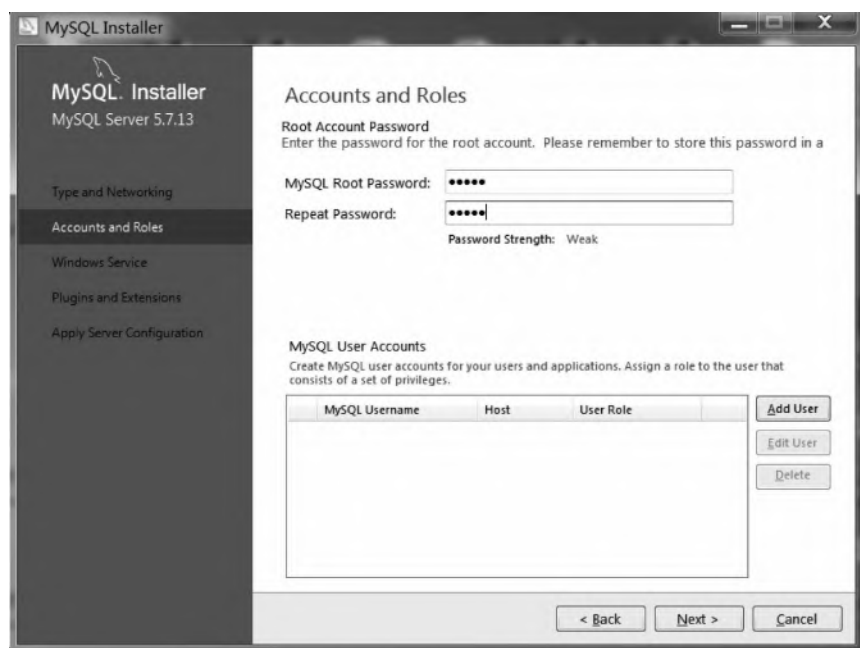
图B-5 MySQL 5.7.13安装界面图（5）



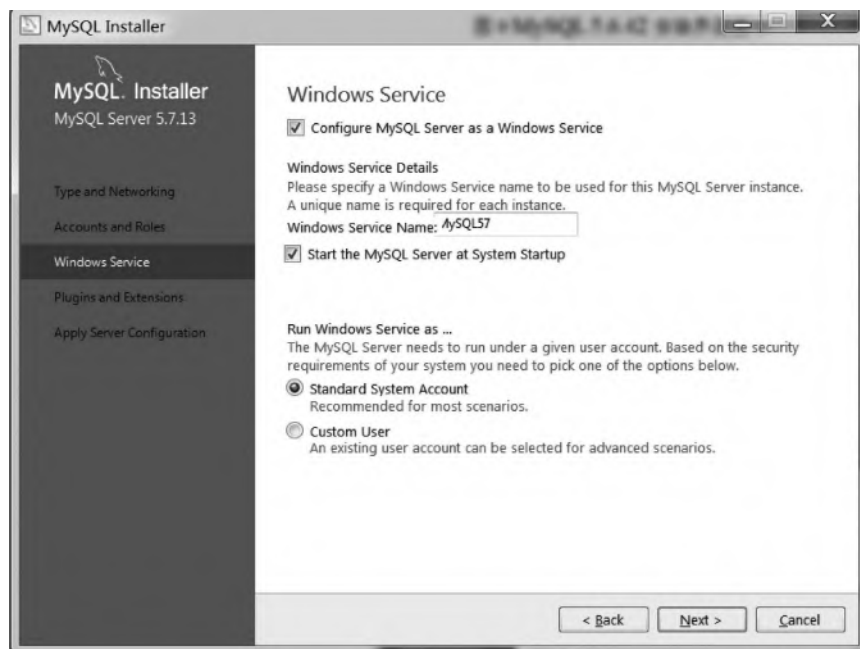
图B-6 MySQL 5.7.13安装界面图（6）



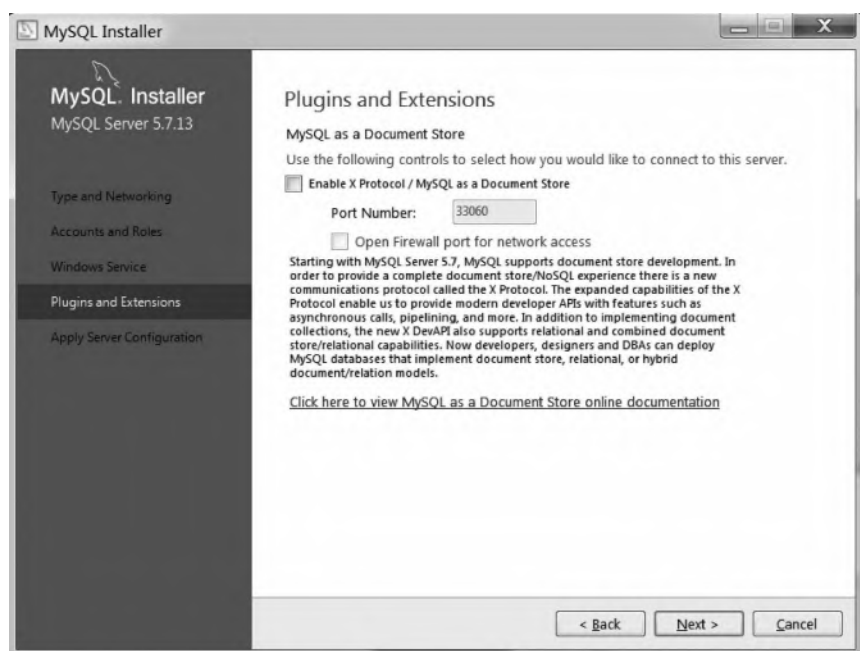
图B-7 MySQL 5.7.13安装界面图（7）



图B-8 MySQL 5.7.13安装界面图（8）

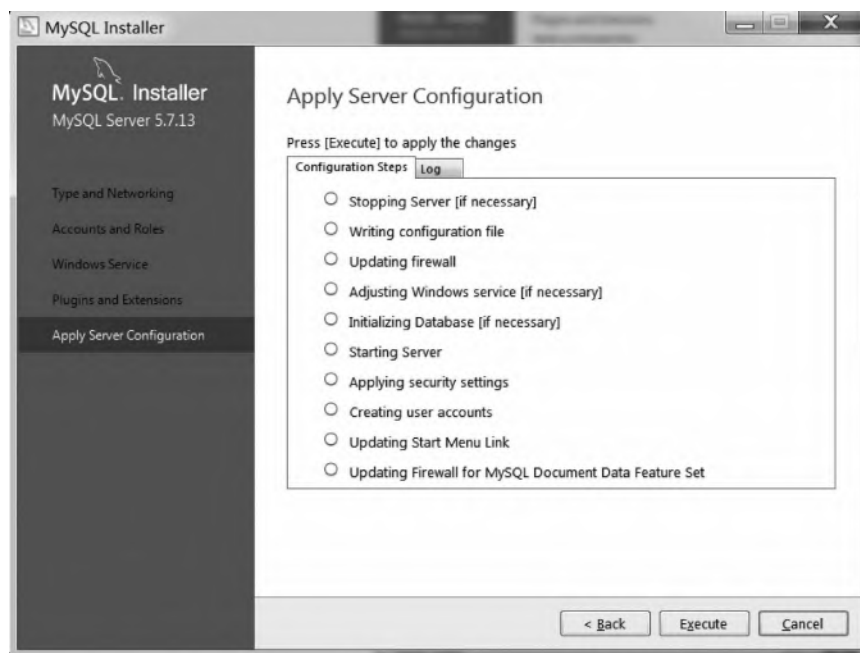


图B-9 MySQL 5.7.13安装界面图（9）



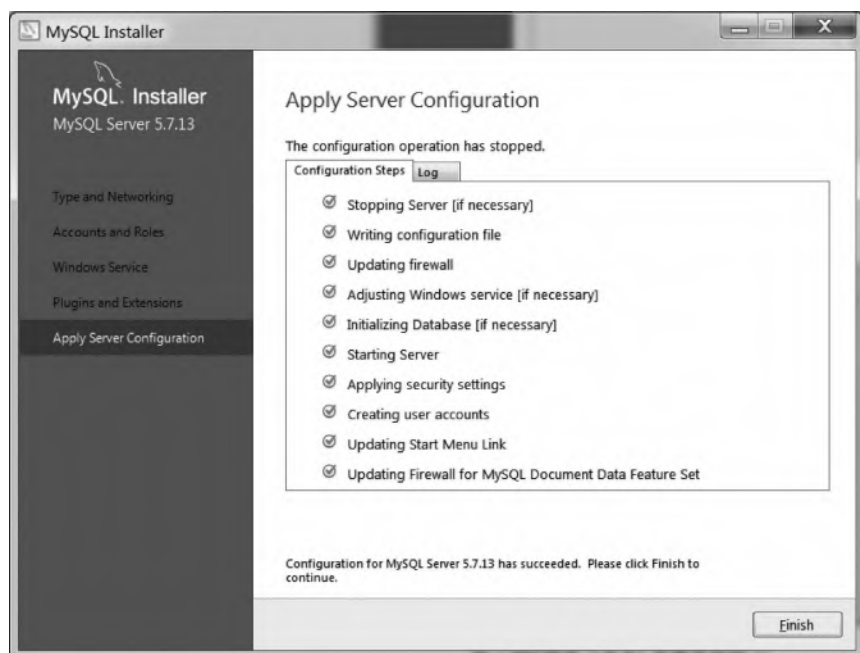
图B-10 MySQL 5.7.13安装界面图（10）

（12）单击Execute按钮，如图B-11所示。



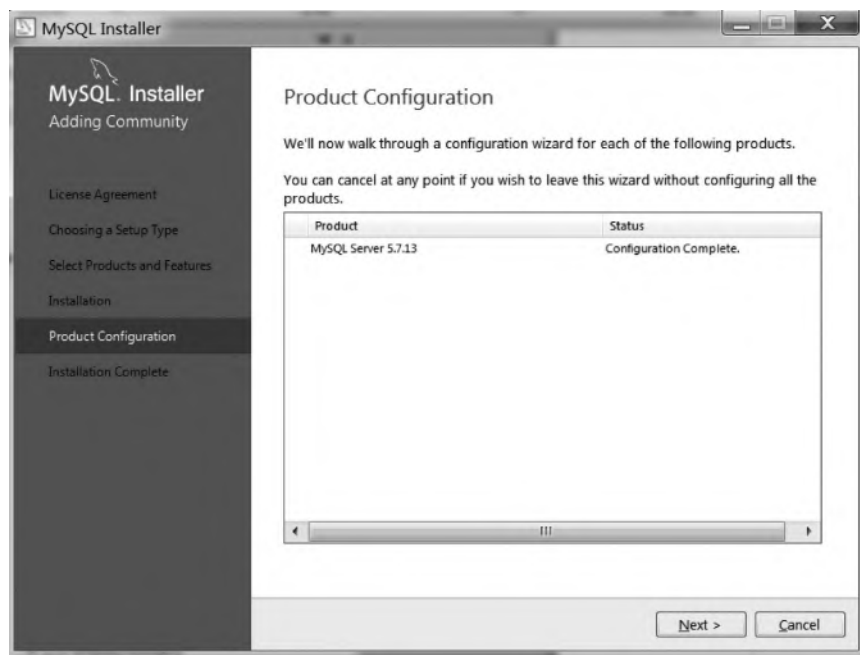
图B-11 MySQL 5.7.13安装界面图（11）

（13）单击Finish按钮，完成Product Configuration的安装，如图B-12所示。



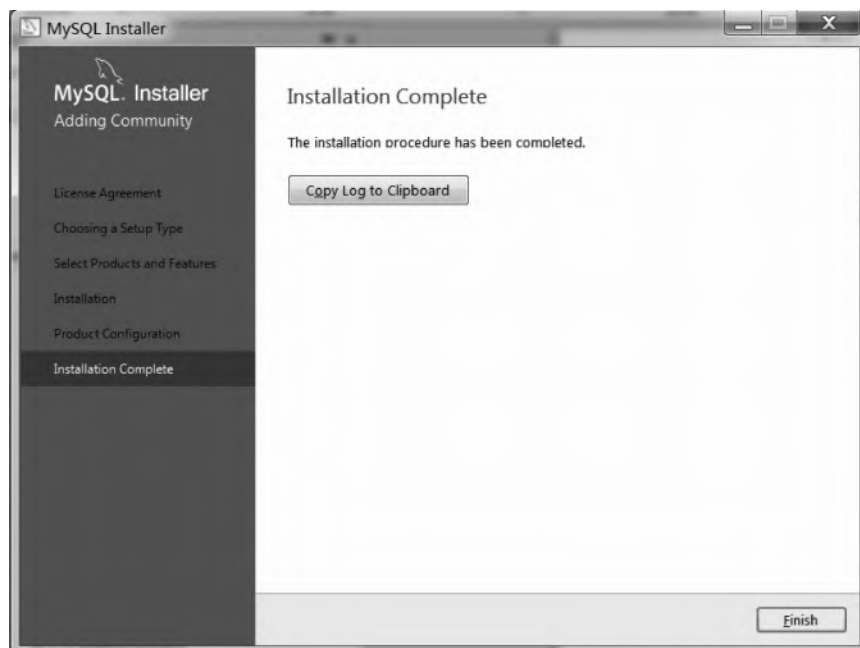
图B-12 MySQL 5.7.13安装界面图（12）

（14）单击Next按钮，如图B-13所示。



图B-13 MySQL 5.7.13安装界面图（13）

（15）单击Finish按钮，至此完成了MySQL 5.7.13的安装，如图B-14所示。



图B-14 MySQL 5.7.13安装界面图（14）

附录C MongoDB安装步骤

MongoDB官网链接地址为：<https://www.mongodb.com/>，下载链接地址为：<https://www.mongodb.com/download-center/community>。

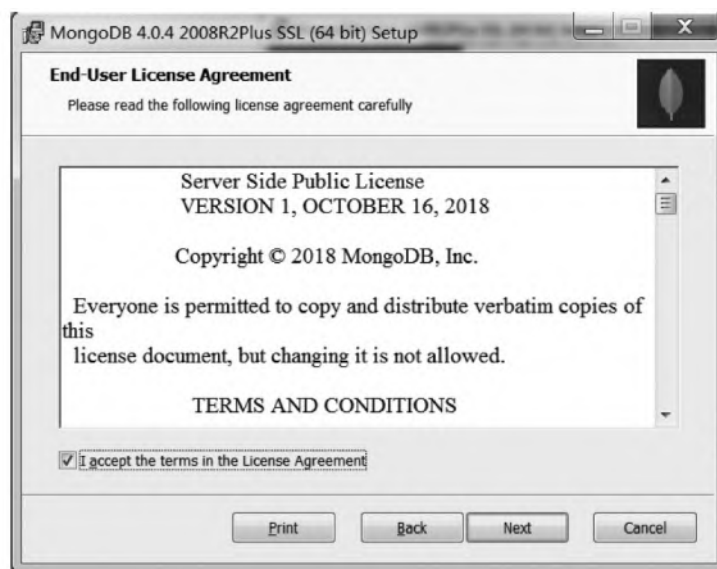
本节以MongoDB 4.0.4社区版为例，进行安装说明。

- (1) 右击“安装包”，单击“安装”按钮。
- (2) 单击Next按钮，如图C-1所示。



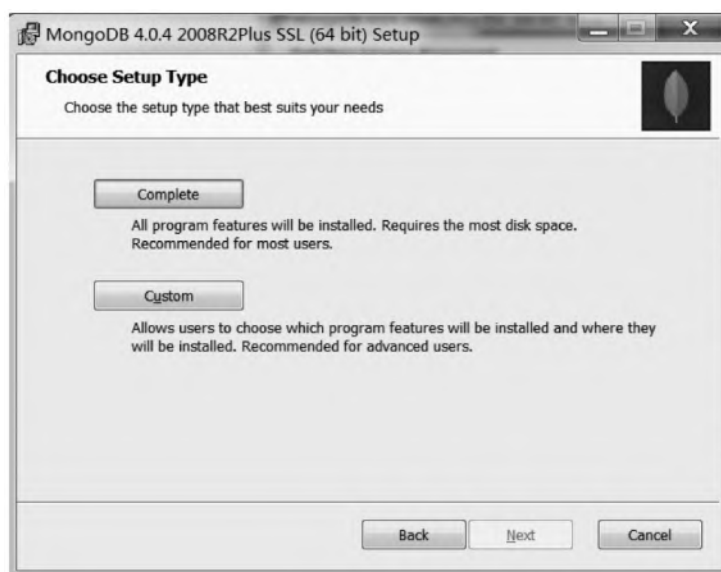
图C-1 MongoDB 4.0.4安装界面图（1）

(3) 勾选I accept the terms in the License Agreement，单击Next按钮，如图C-2所示。



图C-2 MongoDB 4.0.4安装界面图（2）

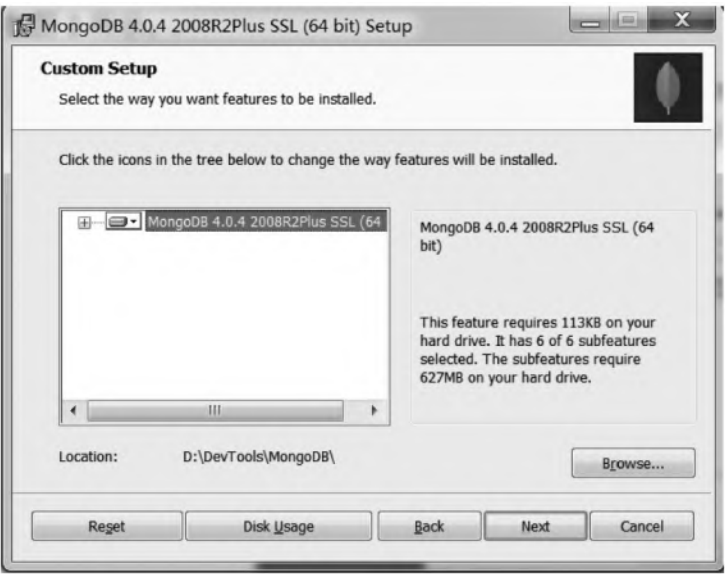
（4）本安装教程选择自定义安装，单击Custom按钮，如图C-3所示。



图C-3 MongoDB 4.0.4安装界面图（3）

（5）程序默认安装位置为 C:\Program Files\MongoDB\Server\4.0\，单击Browse可选择自定义安装目录，本

安装教程的自定义安装路径为D:\DevTools\MongoDB\，单击Next按钮，如图C-4所示。



图C-4 MongoDB 4.0.4安装界面图（4）

（6）单击Next按钮，如图C-5所示。



图C-5 MongoDB 4.0.4安装界面图（5）

(7) 不勾选Install MongoDB Compass, 单击Next按钮, 如图C-6所示。



图C-6 MongoDB 4.0.4安装界面图(6)

(8) 单击Install按钮, 如图C-7所示。



图C-7 MongoDB 4.0.4安装界面图(7)

(9) 安装大概需要3~4分钟，单击Finish按钮，至此完成了MongoDB 4.0.4社区版的安装，如图C-8所示。



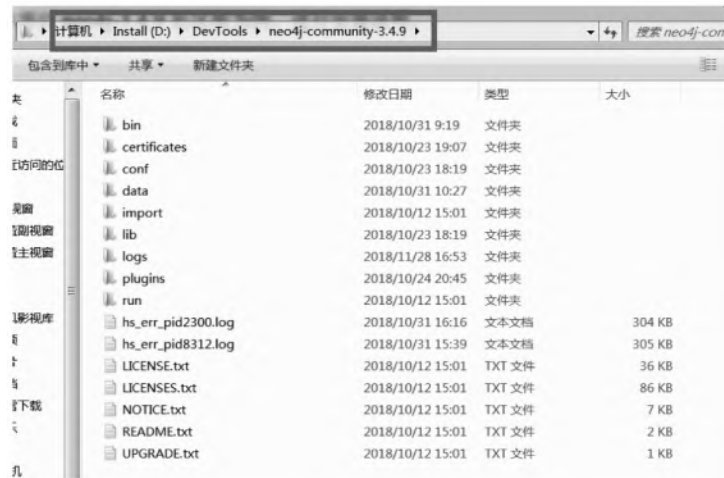
图C-8 MongoDB 4.0.4安装界面图(8)

附录D Neo4j安装步骤

Neo4j官网链接地址为：<https://neo4j.com/>，下载链接地址为：<https://neo4j.com/download-center/#releasesreleases/%23releases>。

本节以neo4j-3.4.9社区版为例，进行安装说明。

(1) 解压zip文件，将解压后的文件复制到自己想要的安装目录下，本安装教程的目录为：D:\DevTools\neo4j-community-3.4.9，如图D-1所示。



图D-1 MongoDB 4.0.4安装界面图

(2) Neo4j环境配置，单击“计算机-属性-高级系统设置”，单击“环境变量”按钮。在“系统变量”栏下单击“新建”按钮，创建新的系统环境变量，如图D-2所示。



图D-2 Neo4j环境配置图（1）

（3）新建NEO4J_HOME变量，对应的变量值为Neo4j的安装目录，本教程安装目录为：D:\DevTools\neo4j-community-3.4.9，单击“确定”按钮，如图D-3所示。



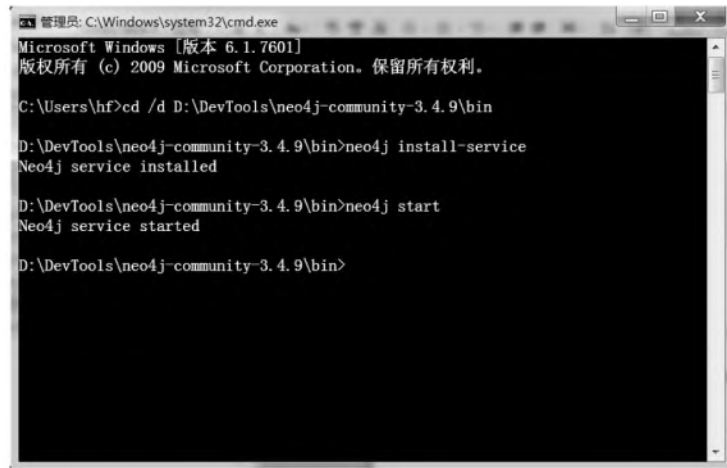
图D-3 Neo4j环境配置图（2）

(4) 新建PATH变量，对应的变量值为Neo4j的安装目录的bin，本教程安装目录为：D:\DevTools\neo4j-community-3.4.9\bin，单击“确定”按钮，如图D-4所示。



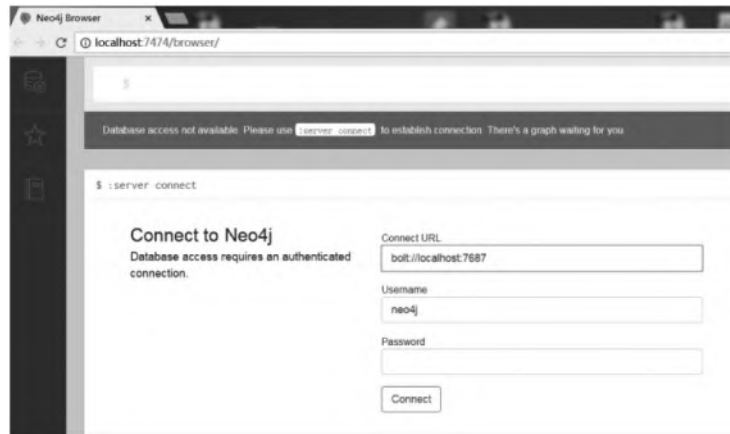
图D-4 Neo4j环境配置图 (3)

(5) 检验Neo4j是否安装正确，可以打开cmd窗口，切换到安装目录的bin目录下：cd/d D:\DevTools\neo4j-community-3.4.9\bin，然后启动Neo4j服务。先后输入：neo4j install-service 和 neo4j start，即可启动Neo4j服务，如图D-5所示。



图D-5 Neo4j启动图

(6) 打开浏览器，在浏览器中输入网址：
<http://localhost:7474/browser/>，即可进入Neo4j界面，如图D-6所示。



图D-6 Neo4j界面

附录E jdk安装步骤

jdk 下 载 链 接 地 址 为 :
<https://www.oracle.com/technetwork/java/javase/downloads/index.html>。

本节以jdk-8u111为例，进行安装说明。

- (1) 右击“安装包”，单击“以管理员身份运行”。
- (2) 单击“下一步”按钮，如图E-1所示。



图E-1 jdk-8u111安装界面图（1）

(3) jdk 默 认 安 装 位 置 为 C:\Program Files\Java\jdk1.8.0_111\，单击更改可选择自定义安装目录，本安装教程的自定义安装路径为D:\DevTools\jdk1.8，单击“下一步”按钮，如图E-2所示。



图E-2 jdk-8u111安装界面图（2）

（ 4 ） jre 默 认 安 装 位 置 为 C:\Program Files\Java\jre1.8.0_111\，单击“更改可选择自定义安装目录”，本安装教程的自定义安装路径为D:\DevTools\jre1.8，单击“下一步”按钮，如图E-3所示。



图E-3 jdk-8u111安装界面图（3）

(5) 单击“关闭”按钮，至此完成了jdk-8u111的安装，如图E-4所示。



图E-4 jdk-8u111安装界面图（4）

(6) jdk环境配置，单击“计算机-属性-高级系统设置”→“环境变量”。在“系统变量”栏下单击“新建”按钮，创建新的系统环境变量，如图E-5所示。



图E-5 jdk环境配置图（1）

(7) 新建JAVA_HOME变量，对应的变量值为jdk的安装目录，本教程安装目录为：D:\DevTools\jdk1.8，单击“确定”按钮，如图E-6所示。



图E-6 jdk环境配置图（2）

(8) 新建Path变量，对应的变量值为jdk安装目录的bin和jre\bin，本教程的变量值为：%JAVA_HOME%\bin;%JAVA_HOME%\jre\bin，单击“确定”按钮，如图E-7所示。



图E-7 jdk环境配置图（3）

（9）新建CLASSPATH变量，取值为安装目录的bin、lib\dt.jar和lib\tools.jar，本教程的变量值为：%JAVA_HOME%\lib;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar，单击“确定”按钮，如图E-8所示。



图E-8 jdk环境配置图（4）

（10）检验jdk是否安装正确，可以打开cmd窗口，输入java-version，如果能够查看到Java版本信息，说明安装正确，如图E-9所示。

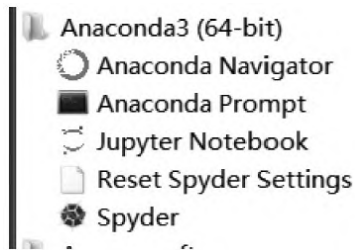


图E-9 java-version命令图

附录F 第三方包安装步骤

本节以第三方包imblearn为例，进行安装说明。

(1) 打开Anaconda文件夹下的Anaconda Prompt窗口，如图F-1所示。

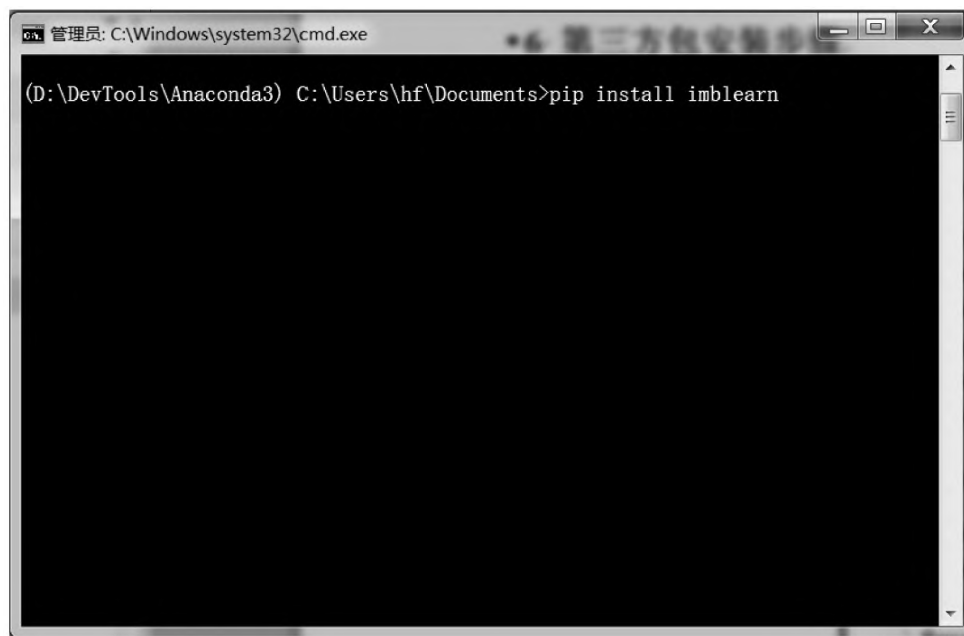


图F-1 Anaconda Prompt界面图

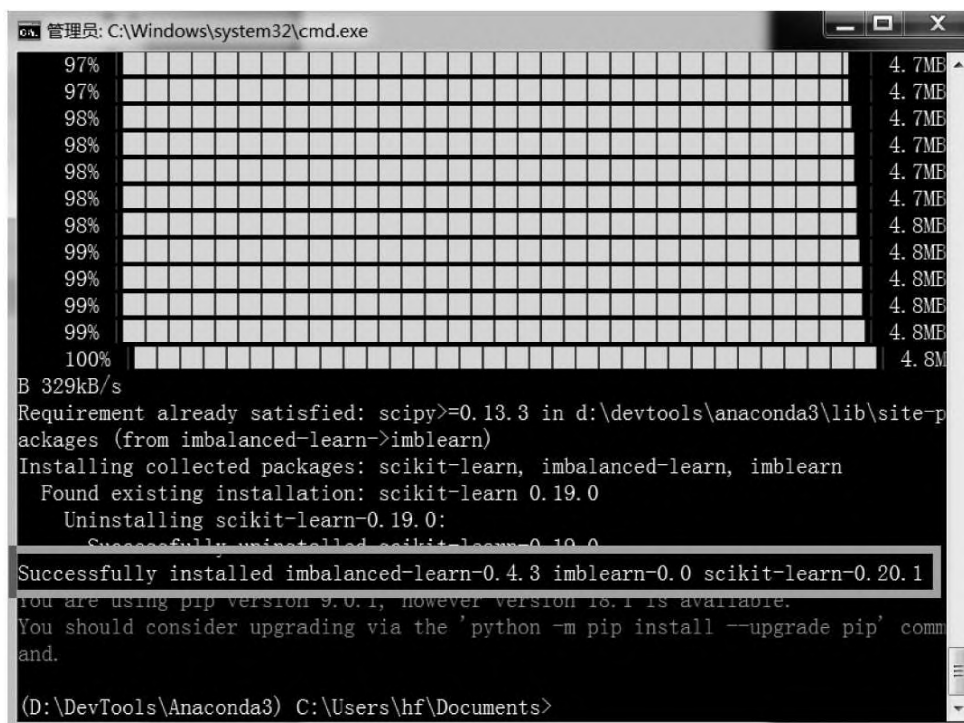
(2) 在窗口输入：`pip install imblearn`，如图F-2所示。

(3) 等待安装，提示 `Successfully installed imbalanced-learn-0.4.3`，即为安装成功，如图F-3所示。

对应的第三方包的安装，都是打开Anaconda Prompt命令窗口，输入：`pip install` 第三方包名。例如，`pip install jieba`，`pip install wordcloud`等。



图F-2 imblearn安装界面图



图F-3 imblearn安装成功界面图

以某电商企业数据为基础，以案例为主线
从0到1，教你用专业的方法展现数据之美

Python数据可视化之 Matplotlib与 Pyecharts

资源文件
视频教学
PPT课件

王国平 编著



清华大学出版社

Python数据可视化之

Matplotlib与 Pyecharts

王国平 编著



清华大学出版社
北京

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

图书在版编目（CIP）数据

Python数据可视化之Matplotlib与Pyecharts / 王国平编著. —北京：清华大学出版社，2020.5

ISBN 978-7-302-55355-7

I. ①P… II. ①王… III. ①软件工具—程序设计 IV. ①TP311.561

中国版本图书馆CIP数据核字（2020）第062130号

责任编辑：王金柱

封面设计：王翔

责任校对：闫秀华

责任印制：沈露

出版发行：清华大学出版社

网 址：<http://www.tup.com.cn>, <http://www.wqbook.com>

地 址：北京清华大学学研大厦A座

邮 编：100084

社总机：010-62770175

邮 购：010-62786544

投稿与读者服务： 010-62776969, c-service@tup.tsinghua.edu.cn

质 量 反 馈： 010-62772015, zhiliang@tup.tsinghua.edu.cn

印 装 者： 三河市君旺印务有限公司

经 销： 全国新华书店

开 本： 190mm×260mm

印 张： 16.25

字 数： 416千字

版 次： 2020年6月第1版

印 次： 2020年6月第1次印刷

产品编号： 085521-01

作者简介

王国平，毕业于上海海洋大学，硕士，从业近十年，主要从事数据可视化、数据挖掘和大数据分析等工作。精通Tableau、Power BI、SPSS、Python等软件与开发工具，已出版《精通Tableau商业数据分析与可视化》《Microsoft Power BI数据可视化与数据分析》等多本专著。

内容简介

本书以某上市电商企业的客户数据、订单数据、股价数据为基础，由浅入深、循序渐进地介绍Python可视化技术，重点介绍Matplotlib与Pyecharts在数据可视化应用中的基本功能和使用技巧。全书以案例为主线，既包括软件的操作与应用，又融入了数据可视化的基础知识，绘图案例大多选自工作实践，可使读者真正掌握专业的可视化方法与技巧，提升数据分析的整体能力。本书配套资源包含案例采用的数据源文件、源代码和教学视频，供读者在阅读本书时进行操作练习和参考。

本书可作为管理、经济、社会人文等领域的人员学习Python软件进行大数据可视化分析的参考书，也可以作为大中专院校相关专业的教学用书或参考书。

前言

“让每个人都成为数据分析师”是大数据时代的要求，数据可视化技术的出现恰恰从侧面缓解了专业数据分析人才的缺乏。Tableau、Microsoft、IBM等IT厂商纷纷加入数据可视化的阵营，在降低数据分析门槛的同时，为分析结果提供了更炫的展现效果。

但是，这些可视化工具存在不足之处，其中比较大的缺点是视图定制化水平有限，不能根据分析师的想法进行个性化定制。Python中的部分包恰好弥补了这个不足。Python是开源免费、简单易学、用途广泛的编程语言。本书将围绕如何使用Python对大数据进行可视化这一话题展开，希望能对正在选型中的个人和企业有所帮助。

研究表明人类大脑处理图形的速度要比文字快几万倍，如何将海量的数据转换成可视化的图形是数据分析的必修课。Matplotlib和Pyecharts是Python中常用的两个可视化库，其功能强大，可以方便地绘制折线图、条形图、柱形图、散点图等基础图形，还可以绘制复杂的图形，如日历图、树形图、聚类图等。

Matplotlib是Python数据可视化库的泰斗，尽管已有十多年的历史，但仍然是Python社区中使用广泛的绘图库，它的设计与Matlab非常相似，提供了一整套和Matlab相似的命令API，适合交互式制图，还可以将它作为绘图控件，嵌入其他应用程序中。

Pyecharts是一款将Python与Echarts相结合的数据可视化工具，可以高度灵活地配置，轻松搭配出精美的视图。其中，Echarts是百度开源的一个数据可视化库，而Pyecharts将Echarts与Python进行有机对接，方便在Python中直接生成各种美观的图形。

本书首先介绍大数据可视化分析的一些基础知识和主要技术，然后通过实际案例重点讲解Matplotlib和Pyecharts在数据可视化分析过程中的技巧及方法，包括图形参数设置、绘制基本图形、绘制高级图形等。希望本书的出版能够改变目前国内市场相关图书匮乏，而且书中案例较少的现状。

本书由浅入深、循序渐进地介绍Python可视化技术，并且结合案例重点介绍Python在数据可视化方面的应用和使用技巧。全书以案例为主线，既介绍软件应用与操作的方法和技巧，又融入了可视化的基础知识，使读者通过学习本书能够轻松掌握可视化的方法。下载资源中包含每个案例采用的数据源文件，供读者在阅读本书时进行操作练习。

本书可作为管理、经济、社会人文等相关从业人员学习Python软件进行大数据可视化分析的参考书，也可以作为高校计算机相关专业本科生、研究生的教材或教学参考书。

截至2019年11月，Matplotlib的版本为3.1.1，Pyecharts的版本为1.5.1。本书正是基于以上版本编写的，全面且详细地介绍它们在数据可视化分析中的应用。

本书主要内容

第一部分（第1~5章）介绍大数据可视化基础。

第1章介绍大数据可视化的技术挑战、技术难点以及可视化工具的必备特性。

第2章介绍Hadoop集群的安装及配置、集群案例数据集以及连接集群的工具。

第3章介绍大数据可视化软件Tableau、Zeppelin和Python及其可视化案例。

第4章介绍Python环境的安装、如何搭建代码开发环境以及如何连接各类数据源。

第5章介绍Python主要的可视化库，如Matplotlib、Pyecharts、Seaborn等。

第二部分（第6~8章）介绍Matplotlib数据可视化。

第6章介绍Matplotlib的图形参数设置，如线条、坐标轴、图例等。

第7章介绍使用Matplotlib绘制基础图形，如直方图、饼图、散点图等。

第8章介绍使用Matplotlib绘制高级图形，如树形图、误差条形图等。

第三部分（第9~12章）介绍Pyecharts数据可视化。

第9章介绍Pyecharts的图形参数配置，如全局配置项和系列配置项。

第10章介绍使用Pyecharts绘制常用视图，如折线图、条形图、箱形图等。

第11章介绍使用Pyecharts绘制高级视图，如日历图、仪表盘、环形图等。

第12章通过实际案例介绍Pyecharts与Django的集成，包括Django框架等。

本书的特色

（1）精心构建的学习体系

本书为读者构建了一个科学合理、循序渐进的学习体系，首先介绍如何构建Hadoop集群，并导入企业数据案例集，以方便后续的上机演练；然后介绍如何运用Matplotlib和Pyecharts可视化工具实现数据可视化；最后介绍如何在Web上展示自己的数据。代码注释详细，解说步骤清晰，十分易于上手。

（2）学以致用，马上提升职场竞争力

全书以某上市电商企业的客户数据、订单数据、股价数据为基础进行讲解，所有案例基本上都围绕该企业的数据可视化展开，如销售额的分析、商品收益率的分析、利润额的分析、客户教育水平的分析、企业股价变动分析等，有的放矢，掌握专业技能，并应对工作需求。

（3）以案例为主线，提供丰富的配书资源

全书以案例为主线，既包括软件的操作方法与应用技巧，又融入了数据可视化的基础知识。为了方便读者使用本书，还提供了源代码、PPT课件和教学视频，读者扫描本书提供的二维码即可下载，可随时随地观看，大幅提升学习效率。

源代码、教学视频与PPT课件下载

为了方便读者更好地使用本书，本书还免费提供了以下资源：



源文件与程序代码：读者扫描右侧的二维码下载后直接调用即可上机演练。

PPT教学课件：方便培训或教学使用，同样，读者可扫描右侧的二维码下载。

教学视频：读者扫描本书各章提供的二维码即可在移动设备上观看，随时随地学习，充分利用碎片时间。

如果下载有问题或需要技术支持，请联系booksaga@126.com，邮件主题为“Python数据可视化之Matplotlib与Pyecharts”。

本书的读者对象

本书的内容和案例适用于互联网、电商、咨询等行业数据分析用户以及媒体、网站等数据可视化用户，可供高等院校相关专业的学生以及从事大数据可视化的研究者参考使用，也可作为Python软件培训和自学的教材。

由于编者水平有限，书中难免存在错误和不妥之处，请广大读者批评指正。

编 者
2020年1月

目 录

[前言](#)

[第一部分 大数据可视化基础](#)

[第1章 大数据可视化概述](#)

[1.1 大数据时代的技术挑战](#)

[1.2 数据可视化的技术难点](#)

[1.3 可视化工具的必备特性](#)

[第2章 搭建大数据开发环境](#)

[2.1 集群的安装及网络配置](#)

[2.1.1 Hadoop集群概述](#)

[2.1.2 集群软件及其版本](#)

[2.1.3 集群网络环境配置](#)

[2.2 集群案例数据集简介](#)

[2.2.1 数据字段说明](#)

[2.2.2 数据导入说明](#)

[2.2.3 运行环境说明](#)

[2.3 连接Hive的图形界面工具](#)

[2.3.1 DBeaver](#)

[2.3.2 Oracle SQL Developer](#)

[2.3.3 DbVisualizer](#)

[2.3.4 Squirrel SQL Client](#)

[第3章 大数据可视化工具](#)

[3.1 Tableau](#)

[3.1.1 Tableau简介](#)

[3.1.2 Tableau连接Hive](#)

[3.1.3 Tableau连接Spark](#)

[3.2 Zeppelin](#)

[3.2.1 Zeppelin简介](#)

[3.2.2 Zeppelin连接Hive](#)

[3.2.3 Zeppelin连接Spark](#)

[3.3 Python在数据可视化中的应用](#)

[3.3.1 Python简介](#)

[3.3.2 Python连接Hive](#)

[3.3.3 Python可视化案例](#)

[第4章 Python可视化编程基础](#)

[4.1 Python环境安装](#)

[4.1.1 Anaconda](#)

[4.1.2 PyCharm](#)

[4.2 Python代码开发工具](#)

[4.2.1 Spyder](#)

[4.2.2 Jupyter Notebook](#)

[4.2.3 Jupyter Lab](#)

[4.3 Python连接数据源](#)

[4.3.1 连接单个文件数据](#)

[4.3.2 连接关系型数据库](#)

[4.3.3 连接Hadoop集群](#)

[第5章 Python数据可视化库](#)

[5.1 Matplotlib](#)

[5.1.1 Matplotlib库简介](#)

[5.1.2 Matplotlib可视化案例](#)

[5.2 Pyecharts](#)

[5.2.1 Pyecharts库简介](#)

[5.2.2 Pyecharts可视化案例](#)

[5.3 Seaborn](#)

[5.3.1 Seaborn库简介](#)

[5.3.2 Seaborn可视化案例](#)

[5.4 ggplot](#)

[5.4.1 ggplot库简介](#)

[5.4.2 ggplot可视化案例](#)

5.5 Bokeh

5.5.1 Bokeh库简介

5.5.2 Bokeh可视化案例

5.6 Pygal

5.6.1 Pygal库简介

5.6.2 Pygal可视化案例

5.7 Plotly

5.7.1 Plotly库简介

5.7.2 Plotly可视化案例

5.8 其他可视化库

5.8.1 Altair

5.8.2 PyQtGraph

5.8.3 NetworkX

第二部分 Matplotlib数据可视化

第6章 Matplotlib图形参数设置

6.1 Matplotlib主要参数配置

6.1.1 线条的设置

6.1.2 坐标轴的设置

6.1.3 图例的设置

6.2 绘图参数文件及主要函数

6.2.1 修改绘图参数文件

6.2.2 绘图主要函数简介

6.3 Matplotlib参数配置案例

第7章 Matplotlib基础绘图

7.1 直方图的绘制

7.1.1 直方图及其参数说明

7.1.2 实例：每日利润额的数值分布

7.2 折线图的绘制

7.2.1 折线图及其参数说明

7.2.2 实例：每周商品销售业绩分析

7.3 条形图的绘制

[7.3.1 条形图及其参数说明](#)

[7.3.2 实例：不同省份利润额的比较](#)

[7.4 饼图的绘制](#)

[7.4.1 饼图及其参数说明](#)

[7.4.2 实例：不同类型商品销售额比较](#)

[7.5 散点图的绘制](#)

[7.5.1 散点图及其参数说明](#)

[7.5.2 实例：销售额与利润额的关系](#)

[7.6 箱形图的绘制](#)

[7.6.1 箱形图及其参数说明](#)

[7.6.2 实例：销售经理业绩比较分析](#)

[第8章 Matplotlib高级绘图](#)

[8.1 树形图的绘制](#)

[8.1.1 树形图及其应用场景](#)

[8.1.2 实例：不同省份销售额的比较分析](#)

[8.2 误差条形图的绘制](#)

[8.2.1 误差条形图及其应用场景](#)

[8.2.2 实例：门店业绩考核达标情况分析](#)

[8.3 火柴杆图的绘制](#)

[8.3.1 火柴杆图及其应用场景](#)

[8.3.2 实例：不同省份送货准时性分析](#)

[8.4 甘特图的绘制](#)

[8.4.1 甘特图及其应用场景](#)

[8.4.2 实例：企业信息化项目进度管理](#)

[8.5 自相关图](#)

[8.5.1 自相关图及其应用场景](#)

[8.5.2 实例：股票价格的自相关分析](#)

[8.6 图形整合](#)

[8.6.1 图形整合及其应用场景](#)

[8.6.2 实例：区域销售额与利润额分析](#)

[第三部分 Pyecharts数据可视化](#)

第9章 Pyecharts图形参数配置

9.1 全局配置项

9.1.1 基本元素配置项

9.1.2 坐标轴配置项

9.1.3 原生图形配置项

9.2 系列配置项

9.2.1 样式类配置项

9.2.2 标记类配置项

9.2.3 其他类配置项

9.3 运行环境

9.3.1 生成HTML

9.3.2 生成图片

9.3.3 Jupyter Notebook

9.3.4 Jupyter Lab

第10章 Pyecharts基础绘图

10.1 折线图的绘制

10.1.1 折线图及其参数配置

10.1.2 实例：各门店销售业绩比较分析

10.2 条形图的绘制

10.2.1 条形图及其参数配置

10.2.2 实例：各省市商品订单数量分析

10.3 箱形图的绘制

10.3.1 箱形图及其参数配置

10.3.2 实例：不同类型商品的收益分析

10.4 涟漪散点图的绘制

10.4.1 涟漪散点图及其参数配置

10.4.2 实例：不同收入等级客户价值分析

10.5 K线图的绘制

10.5.1 K线图及其参数配置

10.5.2 实例：企业股票价格趋势分析

10.6 双坐标轴图的绘制

[10.6.1 双坐标轴图及其参数配置](#)

[10.6.2 实例：区域销售业绩及数量分析](#)

[第11章 Pyecharts高级绘图](#)

[11.1 日历图的绘制](#)

[11.1.1 日历图及其参数配置](#)

[11.1.2 实例：企业股票每日交易量分析](#)

[11.2 漏斗图的绘制](#)

[11.2.1 漏斗图及其参数配置](#)

[11.2.2 实例：华东地区各省市利润额分析](#)

[11.3 仪表盘的绘制](#)

[11.3.1 仪表盘及其参数配置](#)

[11.3.2 实例：企业2019年销售业绩完成率](#)

[11.4 环形图的绘制](#)

[11.4.1 环形图及其参数配置](#)

[11.4.2 实例：不同教育群体的购买力分析](#)

[11.5 雷达图的绘制](#)

[11.5.1 雷达图及其参数配置](#)

[11.5.2 实例：不同区域销售业绩的比较](#)

[11.6 旭日图的绘制](#)

[11.6.1 旭日图及其参数配置](#)

[11.6.2 实例：绘制我的家庭树旭日图](#)

[11.7 主题河流图的绘制](#)

[11.7.1 主题河流图及其参数配置](#)

[11.7.2 实例：不同类型商品销售情况分析](#)

[11.8 词云的绘制](#)

[11.8.1 词云及其参数配置](#)

[11.8.2 实例：商品类型关键词词云](#)

[11.9 玫瑰图的绘制](#)

[11.9.1 玫瑰图及其参数配置](#)

[11.9.2 实例：不同职业群体的购买力分析](#)

[第12章 Web端的数据可视化](#)

[12.1 搭建Django开发环境](#)

[12.1.1 Django框架简介](#)

[12.1.2 Django开发环境](#)

[12.2 Pyecharts与Django集成案例](#)

[12.2.1 创建项目运行环境](#)

[12.2.2 配置项目参数文件](#)

[12.2.3 测试项目运行效果](#)

[附录 集群各节点的参数配置](#)

[Hadoop的参数配置](#)

[Hive的参数配置](#)

[Spark的参数配置](#)

[Zeppelin的参数配置](#)

[集群的启动与关闭](#)

[参考文献](#)

[返回总目录](#)

第一部分

大数据可视化基础

本部分我们将介绍大数据可视化技术的基础知识，包括大数据开发环境的搭建、常见的大数据可视化工具、Python可视化编程基础以及Python中几个重要的可视化分析库。

由于本书后续的可视化分析是基于Hadoop集群讲解的，因此我们首先需要搭建集群，然后简单介绍集群的一些基础知识和案例数据集，以及几种连接Hadoop Hive的图形界面工具，并且通过实际案例介绍Tableau、Zeppelin和Python三类大数据常用的可视化工具。此外，还将讲解Python的一些基础知识，包括软件的安装、代码开发环境、如何访问常见的数据源以及Python中几类比较重要的可视化库。

第1章

大数据可视化概述



“让每个人都成为数据分析师”是大数据时代的要求，数据可视化技术的出现恰恰从侧面缓解了专业数据分析人才的缺乏。Tableau、Microsoft、IBM等IT厂商纷纷加入数据可视化的阵营，在降低数据分析门槛的同时，为分析结果提供了更炫的展现效果。

1.1 大数据时代的技术挑战

大数据的出现正在引发全球范围内技术与商业变革的深刻变化。在技术领域，以往更多依靠模型的方法，现在可以借用规模庞大的数据，用基于统计的方法，使语音识别、机器翻译等技术在大数据时代取得了突破性的进展。

既有技术架构和路线已经无法高效处理海量的数据。对于相关企业组织来说，如果投入巨大而采集的信息无法及时处理与反馈，就会得不偿失。可以说，大数据时代对人类的数据驾驭能力提出了新挑

战，也为人们获得更为深刻、全面的洞察能力提供了前所未有的空间。

大数据时代主要有以下4个技术挑战：

第一个挑战是数据量大。

大数据的起始计量单位是PB（1000TB）、EB（100万TB）或ZB（10亿TB）。目前，企业面临数据量的大规模增长，预测到2020年，全球数据量将扩大50倍。如今，大数据的规模尚在不断变化，单一数据集的规模范围从几十TB到数PB不等。导致我们无法通过目前主流的软件工具收集、管理、处理数据并整理成为帮助企业达到经营决策目的的资讯。

第二个挑战是数据类型繁多。

包括网络日志、音频、视频、图片、地理位置信息等，多种类型的数据对数据处理能力提出了更高要求。数据多样性的增加主要由新型多结构数据和多种数据类型（包括网络日志、社交媒体、互联网搜索、手机通话记录及传感器数据等）造成。其中，越来越多的传感器被安装在火车、汽车和飞机上，每个传感器都增加了数据的多样性。

第三个挑战是数据价值密度低。

大数据结构非常复杂，有结构化的，也有非结构化的，增长速度飞快，单条数据的价值密度极低。此外，随着物联网的广泛应用，信息感知无处不在。信息海量，但价值密度较低，如何通过强大的机器算法迅速地完成数据的价值“提纯”，是大数据时代亟待解决的难题。

第四个挑战是高速性。

描述的是数据被创建和移动的速度。在高速网络时代，通过实现软件性能优化的高速计算机处理器和服务器的创建实时数据流已成为流行趋势。企业不仅需要了解如何快速创建数据，还必须知道如何将数据快速处理、分析并返回给用户，以满足用户的实时需求。

1.2 数据可视化的技术难点

大数据具有多层结构，意味着会呈现多变的形式和类型。相较于传统的业务数据，大数据存在不规则和模糊不清的特性，造成很难甚至无法使用传统应用软件进行分析。传统业务数据随着时间的演变已经拥有标准的格式，能够被标准商务智能软件识别。目前，企业面临的挑战是处理并从各种形式呈现的复杂数据中挖掘价值。

传统数据可视化工具仅将数据加以组合，通过不同展现方式提供给用户，用于发现数据之间的关联信息。近年来，随着云和大数据时代的来临，数据可视化产品已经不再满足于使用传统数据可视化工具对数据仓库中的数据抽取、归纳并简单地展现。新型数据可视化产品必须满足互联网爆发的大数据需求，必须快速收集、筛选、分析、归纳、展现决策者所需要的信息，并根据新增数据进行实时更新，这是大数据可视化的技术难点。

大数据技术的有效可视化不应该只是为管理层绘制漂亮的图片。专家表示，企业可通过考虑布局、迭代设计、吸引用户和了解业务需求来改善结果。开展数据可视化项目的企业提供了以下几个注意事项：

(1) 了解业务。分析之前与业务人员进行深入沟通，了解他们希望获取什么信息。在构思不同的仪表板时，应该始终考虑最终用户，

如管理层、分析师、IT人员和业务人员希望从不同类型的可视化分析中获取什么，只有这样，大数据的可视化才有实际价值。

（2）注重个性化。应该确保仪表板向最终用户显示个性化信息，以及为最终用户提供离线访问，这将让可视化走得更长远。注意仪表板就像一本书，它需要考虑用户的实际需求，而不仅仅是强制列出所有可访问的数据。

（3）尽可能简化。由于大数据可视化工具的功能非常丰富，因此通常会导致分析师构建过于复杂的可视化图表，导致难以收集可行的见解，优秀的分析师应尽可能简化可视化，确保最终产品不是徒有炫酷外表而不能满足实际需求。

（4）从用户角度。应该使用颜色、形状、大小和布局来显示可视化的设计和使用。用颜色来突出希望用户关注的方面，而大小可以有效地说明数量，但过多使用可能会导致混乱，应该有选择地使用这些元素。

（5）选择合适的方法。不同的情况需要采用不同的可视化方法。例如，许多数据可视化专家不建议使用饼图，这是因为人眼和头脑可以更容易地测量长度或位置之间的差异，而很难识别角度差异。

1.3 可视化工具的必备特性

数据可视化的历史可以追溯到20世纪50年代计算机图形学的早期，人们利用计算机创建了首批图形图表。到了1987年，一篇题目为“Visualization in Scientific Computing”（科学计算中的可视化，即科学可视化）的报告成为数据可视化领域发展的里程碑，它强调了基于计算机可视化技术新方法的必要性。

随着人类采集的数据种类和数量的增长、计算机运算能力的提升，越来越多高级计算机图形学技术与方法应用于处理和可视化这些规模庞大的数据集。20世纪90年代初期，“信息可视化”成为新的研究领域，旨在为许多应用领域对于抽象异质性数据集的分析工作提供支持。

当前，数据可视化是一个既包含科学可视化又包含信息可视化的新概念。数据可视化是可视化技术在非空间数据上的新应用，使得人们不再局限于通过关系数据表观察和分析数据信息，还能以更直观的方式看到数据与数据之间的结构关系。

数据可视化是关于数据视觉表现形式的研究。这种数据视觉表现形式被定义为一种以某种概要形式抽取出来的信息，包括相应信息单位的各种属性和变量。

在大数据时代，数据可视化工具必须具备以下4个特性：

- **实时性：** 数据可视化工具必须适应大数据时代数据量爆炸式增长的需求，必须快速收集、分析数据，并对数据信息进行实时更新。
- **简单操作：** 数据可视化工具满足快速开发、易于操作的特性，能满足互联网时代信息多变的特点。
- **更丰富的展现：** 数据可视化工具需要具有更丰富的展现方式，能充分满足数据展现的多维度要求。
- **多种数据集成支持方式：** 数据的来源不局限于数据库，数据可视化工具将支持团队协作数据、数据仓库、文本等多种方式，并能够通过互联网进行展现。

数据可视化的思想是将数据库中每一个数据项作为单个图元元素，通过抽取的数据构成数据图像，同时将数据的各个属性值加以组合，并以多维数据的形式通过图表、三维等方式展现数据之间的关联

信息，使用户能从不同维度和不同组合对数据库中的数据进行观察，从而对数据进行更深入的分析 and 挖掘。

第2章

搭建大数据开发环境



本书的可视化分析是基于Hadoop集群展开的，因此首先需要搭建集群，我们这里基于3台虚拟机搭建了一个由3个节点（master、slave1、slave2）构成的Hadoop完全分布式集群，节点安装的操作系统为CentOS 6.5，Hadoop版本选择2.5.2。

本章将简单介绍一些Hadoop集群的基础知识，包括集群中存储的案例数据集，以及几种常用的连接集群的图形界面工具。

2.1 集群的安装及网络配置

2.1.1 Hadoop集群概述

Hadoop在2006年成为雅虎项目，随后晋升为顶级Apache开源项目。它是一种通用的分布式系统基础架构，具有多个组件：Hadoop分布式文件系统（Hadoop Distributed File System，HDFS），将文件以Hadoop本机格式存储并在集群中并行化；YARN，协调应用程序运行

时的调度程序；MapReduce，实际并行处理数据的算法。此外，通过一个Thrift客户端，用户可以编写MapReduce或者Python代码，本书就是使用Python对集群中的数据进行可视化分析的。

Hadoop分布式文件系统是一种文件系统实现，类似于NTFS、EXT3、EXT4等，它将存储在集群中的文件分成块，每块默认为64MB，比一般文件系统块大得多，并分布在多台机器上，每块又会有多块冗余备份（默认为3），以增强文件系统的容错能力，在具体实现中主要有以下几个部分：

（1）名称节点（NameNode）

名称节点的职责在于存储整个文件系统的元数据，这是一个非常重要的角色。元数据在集群启动时会加载到内存中，元数据的改变也会写到磁盘的系统映像文件中，同时还会维护对元数据的编辑日志。HDFS存储文件时是将文件划分成逻辑上的块存储的，对应关系都存储在名称节点上，如果有损坏，整个集群的数据就会不可用。

我们可以采取一些措施备份名称节点的元数据，例如将名称节点目录同时设置到本地目录和一个NFS目录，这样任何元数据的改变都会写入两个位置进行冗余备份，使得使用中的名称节点关机后，可以使用NFS上的备份文件恢复文件系统。

（2）第二名称节点（SecondaryNameNode）

这个角色的作用是定期通过编辑日志合并命名空间映像，防止编辑日志过大。不过第二名称节点的状态滞后于主名称节点，如果主名称节点突然关闭，那么必定会有一些文件损失。

（3）数据节点（DataNode）

这是HDFS中具体存储数据的地方，一般有多台机器。除了提供存储服务外，还定时向名称节点发送存储的块列表。名称节点没有必要永久保存每个文件、每个块所在的数据节点，这些信息会在系统启动后由数据节点重建。

Hadoop一般采用MapReduce计算框架，在系统架构上，它是一种主从架构，由一个单独的JobTracker节点和多个TaskTracker节点共同组成，核心是将任务分解成小任务，由不同计算者同时参与计算，并将各个计算者的计算结果合并，得出最终结果。模型本身非常简单，一般只需要实现两个接口即可，关键在于怎样将实际问题转化为MapReduce任务。

Hadoop的MapReduce主要由以下两部分组成：

（1）作业跟踪节点（JobTracker）

负责调度构成一个作业的所有任务，这些任务分布在不同的TaskTracker节点上，监控它们的执行，以及重新执行已经失败的任务等。

（2）任务跟踪节点（TaskTracker）

负责具体的任务执行。TaskTracker通过“心跳”的方式告知JobTracker其状态，并由JobTracker根据报告的状态为其分配任务。TaskTracker会启动一个新JVM运行任务，当然JVM实例也可以被重用。

Hadoop在大数据领域的应用前景很大，不过因为是开源技术，实际应用过程中存在很多问题。于是出现了各种Hadoop发行版，国外目前主要有3家创业公司在做这项业务：Cloudera、Hortonworks和MapR。

Cloudera和MapR的发行版是收费的，它们基于开源技术，提高了稳定性，同时强化了一些功能，定制化程度较高，核心技术是不公开

的，收入主要来自软件。Hortonworks则走向另一条路，它将核心技术完全公开，用于推动Hadoop社区的发展。这样做的好处是，如果开源技术有很大提升，他们的受益就会很大，因为定制化程度较少，自身不会受到技术提升的冲击。

2.1.2 集群软件及其版本

本书使用的Hadoop集群是基于3台虚拟机搭建的，它是由3个节点（master、slave1、slave2）构成的Hadoop完全分布式集群，节点使用的操作系统为CentOS 6.5，Hadoop版本为2.5.2。

首先，需要下载并安装VMware，这里我们选择的是VMware Workstation Pro 15.1.0，这是一款先进的虚拟化软件，能够提高生产效率，是为各类用户设计的桌面虚拟化解决方案，是开展业务不可或缺的利器，具体安装过程可参考网上的相关教程，这里不进行介绍。

然后，我们需要下载并安装CentOS 6.5系统。CentOS是一个基于Red Hat Linux提供的可自由使用源代码的企业级Linux发行版本，每个版本的CentOS都会获得10年的支持。具体安装过程可参考网上的相关教程，这里也不进行具体介绍。

本书使用的Hadoop集群上安装的软件及其版本如下：

```
apache-hive-1.2.2-bin.tar.gz
hadoop-2.5.2.tar.gz
jdk-7u71-linux-x64.tar.gz
mysql-5.7.20-linux-glibc2.12-x86_64.tar.gz
scala-2.10.4.tgz
spark-1.4.0-bin-hadoop2.4.tgz
sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz
zeppelin-0.7.3-bin-all.tgz
```

其中，集群主节点master上安装的软件如下：

```
apache-hive-1.2.2
hadoop-2.5.2
jdk-7u71
mysql-5.7.20
scala-2.10.4
spark-1.4.0
sqoop-1.4.6
zeppelin-0.7.3
```

集群主节点/etc/profile文件的配置如下：

```
export JAVA_HOME=/usr/java/jdk1.7.0_71/
export HADOOP_HOME=/home/dong/hadoop-2.5.2
export SCALA_HOME=/home/dong/scala-2.10.4
export SPARK_HOME=/home/dong/spark-1.4.0-bin-hadoop2.4
export HIVE_HOME=/home/dong/apache-hive-1.2.2-bin
export SQOOP_HOME=/home/dong/sqoop-1.4.6.bin__hadoop-
2.0.4-alpha
export PYTHONPATH=/home/dong/spark-1.4.0-bin-
hadoop2.4/Python
export RPATH=/home/dong/spark-1.4.0-bin-hadoop2.4/R
export ZEPPELIN_HOME=/home/dong/zeppelin-0.7.3-bin-all
export
PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$SCALA_HOME/bin:$JAVA
_HOME/bin:$SPARK_HOME/bin:$HIVE_HOME/bin:$SQOOP_HOME/bin:/usr
/local/mysql/bin:$ZEPPELIN_HOME/bin:$PATH
```

此外，集群两个从节点slave1与slave2上安装的软件如下：

```
hadoop-2.5.2
jdk-7u71
scala-2.10.4
spark-1.4.0
```


集群两个从节点/etc/profile文件的配置如下：

```
export JAVA_HOME=/usr/java/jdk1.7.0_71/
export HADOOP_HOME=/home/dong/hadoop-2.5.2
export SCALA_HOME=/home/dong/scala-2.10.4
export SPARK_HOME=/home/dong/spark-1.4.0-bin-hadoop2.4
export PYTHONPATH=/home/dong/spark-1.4.0-bin-hadoop2.4/Python
export RPATH=/home/dong/spark-1.4.0-bin-hadoop2.4/R
export PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$SCALA_HOME/bin:$JAVA_HOME/bin:$SPARK_HOME/bin:$PATH
```

2.1.3 集群网络环境配置

为了使得集群既能相互之间进行通信，又能够进行外网通信，需要为节点添加网卡，上网方式均采用桥接模式，外网IP设置为自动获取，通过此网卡进行外网访问，配置应该按照当前主机的上网方式进行合理配置，如果不与主机通信，上网方式就可以采用NAT，这样选取默认配置就行，内网IP设置为静态IP。

1. 配置集群节点网络

Hadoop集群各节点的网络IP配置如下：

```
master: 192.168.1.7
slave1: 192.168.1.8
slave2: 192.168.1.9
```

下面给出固定master虚拟机IP地址的方法，slave1和slave2与此类似：

```
vi /etc/sysconfig/network-scripts/ifcfg-eth0
TYPE="Ethernet"
UUID="b8bbe721-56db-426c-b1c8-38d33c5fa61d"
ONBOOT="yes"
NM_CONTROLLED="yes"
BOOTPROTO="static"
IPADDR=192.168.1.7
NETMASK=255.255.255.0
GATEWAY=192.168.1.1
DNS1=192.168.1.1
DNS2=114.144.114.114
```

为了不直接使用IP，可以通过设置hosts文件达到3个节点之间相互登录的效果，3个节点的设置都相同。配置hosts文件，在文件尾部添加如下行，保存后退出：

```
vi /etc/hosts
192.168.1.7 master
192.168.1.8 slave1
192.168.1.9 slave2
```

2. 关闭防火墙和SELinux

为了节点间的正常通信，需要关闭防火墙，3个节点的设置相同，集群处于局域网中，因此关闭防火墙一般不存在安全隐患。

查看防火墙状态的命令：

```
service iptables status
```

防火墙即时生效，重启后复原，命令如下：

开启: `service iptables start`。

关闭: `service iptables stop`。

如果需要永久性生效, 重启后不会复原, 命令如下:

开启: `chkconfig iptables on`。

关闭: `chkconfig iptables off`。

关闭SELinux的方法:

临时关闭SELinux: `setenforce 0`。

临时打开SELinux: `setenforce 1`。

查看SELinux的状态: `getenforce`。

开机关闭SELinux:

编辑 `/etc/selinux/config` 文件, 将 SELinux 的值设置为 `disabled`, 下次开机SELinux就不会启动了。

3. 免密钥登录设置

设置master节点和两个slave节点之间的双向SSH免密通信。下面以master节点SSH免密登录slave节点设置为例, 进行SSH设置介绍(以下操作均在master机器上操作)。

(1) 首先生成master的rsa密钥: `$ssh-keygen -t rsa`。

(2) 设置全部采用默认值, 按回车键。

(3) 将生成的rsa追加写入授权文件: `$cat ~/.ssh/id_rsa.pub`
`>> ~/.ssh/authorized_keys`。

(4) 给授权文件权限: `$chmod 600 ~/.ssh/authorized_keys`。

(5) 进行本机SSH测试: `$ssh maste r`。正常免密登录后, 所有的SSH第一次登录都需要密码, 此后都不需要密码。

(6) 将master上的authorized_keys传到slave1和slave2:

```
scp ~/.ssh/authorized_keys
root@slave1:~/.ssh/authorized_keys
scp ~/.ssh/authorized_keys
root@slave2:~/.ssh/authorized_keys
```

(7) 登录slave1的操作: `$ssh slave1`, 输入密码登录。

(8) 退出slave1: `$exit`。

(9) 进行免密SSH登录测试: `$ssh slave1`。

(10) 同理, 登录slave2进行相同的操作。

2.2 集群案例数据集简介

本节以某上市电商企业的客户数据、订单数据、股价数据为基础进行数据可视化的讲解, 当然实际工作中的数据分析需求应该更加繁杂, 但是我们可以先结合业务背景将需求整理成相应的指标, 然后抽取出数据, 再应用本书中介绍的数据可视化方法, 从而实现我们的可视化分析需求。

2.2.1 数据字段说明

我们选取该上市电商企业的客户数据、订单数据、股价数据中的部分指标作为分析的字段，分别存储在customers、orders和stocks三张表中。下面逐一进行说明。

客户表customers包含客户属性的基本信息，例如客户ID、性别、年龄、学历、职业等12个字段，具体见表2-1。

表2-1 客户表字段说明

序 号	变 量 名	说 明
1	cust_id	客户 ID
2	gender	性别
3	age	年龄
4	education	学历
5	occupation	职业
6	income	收入
7	telephone	手机号码
8	marital	婚姻状况
9	email	邮箱地址
10	address	家庭地址
11	retire	是否退休
12	custcat	客户等级

订单表orders包含客户订单的基本信息，例如订单ID、订单日期、门店名称、支付方式、发货日期等24个字段，具体见表2-2。

表2-2 订单表字段说明

序 号	变 量 名	说 明
1	order_id	订单 ID
2	order_date	订单日期
3	store_name	门店名称
4	pay_method	支付方式
5	deliver_date	发货日期
6	planned_days	计划发货天数
7	cust_id	客户 ID
8	cust_name	姓名
9	cust_type	类型
10	city	城市
11	province	省市
12	region	地区
13	product_id	产品 ID
14	product	产品名称
15	category	类别
16	subcategory	子类别
17	sales	销售额
18	amount	数量
19	discount	折扣
20	profit	利润额
21	manager	销售经理
22	return	是否退回

股价表stocks包含A企业近3年来股价的走势信息，包含交易日期、开盘价、最高价、最低价、收盘价等7个字段，具体见表2-3。

表2-3 股价表字段说明

序 号	变 量 名	说 明
1	trade_date	交易日期
2	open	开盘价
3	high	最高价
4	low	最低价
5	close	收盘价
6	adj_close	复权收盘价
7	volume	成交量

2.2.2 数据导入说明

企业的客户表、订单表和股价表的指标及数据都整理好后，接下来的工作就是将数据导入Hadoop集群中，这个过程分成两步：新建表和导入数据。注意这个过程都是在Hive中进行的，所以需要启动Hadoop集群和Hive。

在新建表之前需要先新建数据库，SQL语句如下：

```
create database sales
```

然后通过use sales语句使用sales数据库，再使用下面的3条SQL语句创建customers、orders和stocks三张表：

```
create table customers(cust_id string,gender string,age
int,education string,occupation string,income
string,telephone string,marital string,email string,address
string,retire string,custcat string) row format delimited
fields terminated by ','
create table orders(order_id string,order_date
string,store_name string,pay_method string,deliver_date
string,planned_days int,cust_id string,cust_name
string,cust_type string,city string,province string,region
string,product_id string,product string,category
```

```
string,subcategory string,sales float,amount int,discount
float,profit float,manager string,return int) partitioned by
(dt string) row format delimited fields terminated by ','
create table stocks(trade_date string,open float,high
float,low float,close float,adj_close float,volume int) row
format delimited fields terminated by ','
```

注意，由于企业的订单数据一般较多，因此我们将orders表定义成了分区表，分区字段是年份dt，而customers表和stocks表都是非分区表。如果想深入了解分区表与非分区表的区别与联系，可以参阅相关大数据的图书。

表创建完成后，需要将数据导入相应的表中，在Hive中可以通过load data命令实现，例如导入customers表中的数据的命令如下：

```
load data local inpath '/home/dong/sales/customers.txt'
overwrite into table customers
```

当然，也可以使用Sqoop中的sqoop import命令实现，这里就不进行详细介绍了。

对于分区表数据的导入，这个过程相对比较复杂，可以通过insert语句将非分区表的数据插入分区表中的方式实现，两张表的表结构要一致。例如，orders_1存储的是2019年的订单数据，需要将其导入orders表中，SQL语句为：

```
insert into table orders partition(dt) select
order_id,order_date,store_name,pay_method,deliver_date,planned_days,cust_id,cust_name,cust_type,city,province,region,product_id,product,category,subcategory,sales,amount,discount,profit,manager,return,dt from orders_1 where dt=2019
```


其他年份的数据也可以通过类似的方法导入orders表，注意在导入完成后需要验证一下数据是否正常导入，可以选择一种后面将要介绍的连接Hive的图形界面工具或者Hive的查询数据命令。

2.2.3 运行环境说明

现在大数据比较火热，企业的数据基本都存放在Hadoop环境中。因此，为了更好地贴近实际工作，使读者学以致用，本书中使用的案例数据也存放在Hadoop集群中，一个主节点、两个从节点的虚拟环境。当然，这个环境和企业的真实环境可能有一定的差异，例如数据量的问题等，读者可以结合实际情况对代码进行适当的修改。

此外，对于Hadoop环境的搭建，具体的搭建过程比较复杂，由于篇幅所限，这里就不详细介绍，毕竟本书的重点是介绍大数据环境下的数据可视化技术，读者可以参考网络上的资料或相关图书，只要懂一些Linux的基础命令操作，并花费一定的时间，基本都可以成功搭建。

2.3 连接Hive的图形界面工具

在日常工作中，为什么使用客户端界面工具而不用命令行使用Hive呢？原因是通过界面工具查看分析Hive中的数据要方便很多，业务人员没有权限通过命令行连接Hive，领导喜欢在界面工具上查看Hive中的数据。

本节讲解如何通过数据库客户端界面工具DBeaiver、Oracle SQL Developer、DbVisualizer和Squirrel SQL Client等工具连接Hadoop集群的Hive数据库。

2.3.1 DBeaver

DBeaver是一个通用的数据库管理工具和SQL客户端，支持MySQL、Oracle、DB2、MSSQL、Hive等数据库，它提供一个图形界面用来查看表结构、执行查询、导出数据等。

连接Hadoop集群Hive的工具还有很多，推荐使用DBeaver的原因是DBeaver简单易用，支持各种关系型数据库，还有就是DBeaver的快捷键和Eclipse一样，比如注释、删除、复制等操作。

1. 下载和安装DBeaver

DBeaver分为社区版和企业版，其中社区版是免费的，可以在官网上下载最新的社区版DBeaver，下载地址：<https://dbeaver.io/download/>，这里下载的是Windows 64位免安装社区版，如图2-1所示，读者可以根据实际情况下载对应版本。

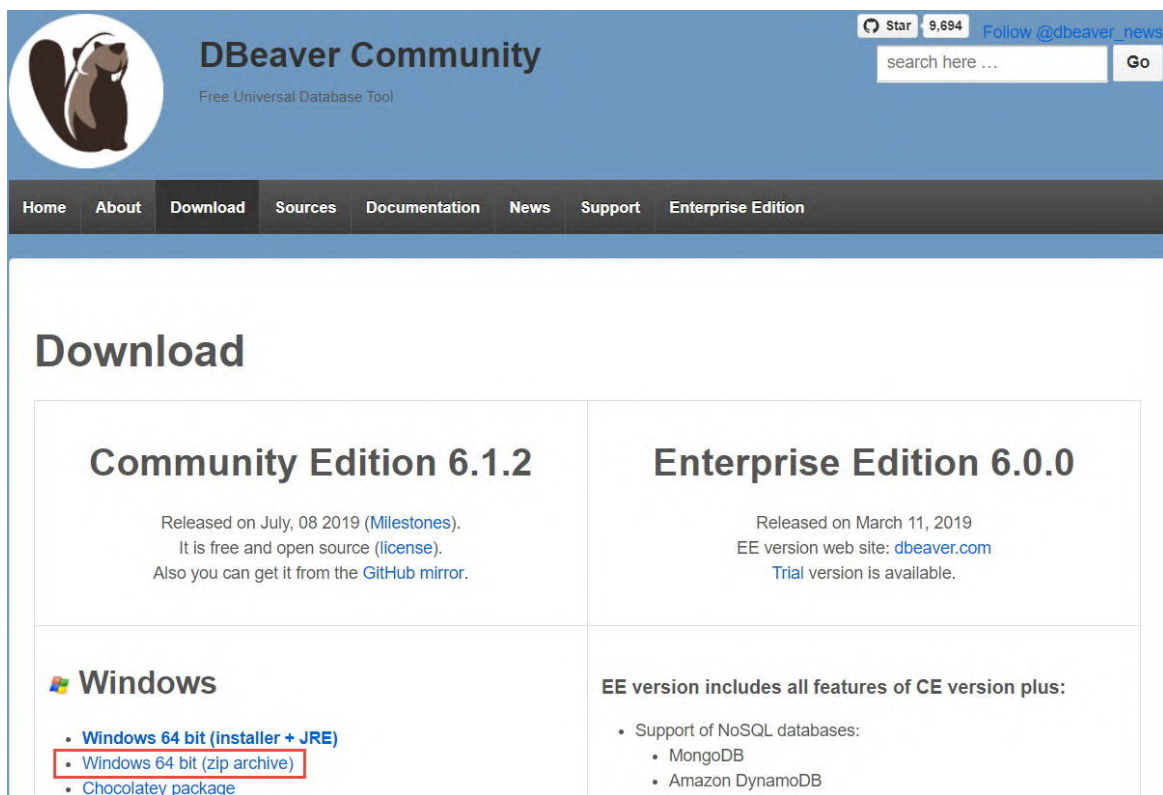


图2-1 下载DBeaver

由于笔者下载的是免安装版，因此解压后，直接单击dbeaver.exe就可以使用。

2. 启动Hadoop集群

测试连接前先启动Hadoop和Hive的相关服务。

(1) 启动Hadoop集群。

(2) 启动Hive，如果想远程连接Hive，那么还需要启动hiveserver2。

(3) 创建Hive测试表，如果已经有了，就可以省略。

3. 连接集群的具体步骤

DBeaver连接关系型数据库比较简单，但是连接Hive要下载和配置驱动程序，过程相对比较复杂。下面介绍连接Hive的具体步骤。

步骤 01

新建数据库连接。打开DBeaver，在界面中依次单击“文件”→“新建”→“数据库连接”，然后单击“下一步”按钮，如图2-2所示。

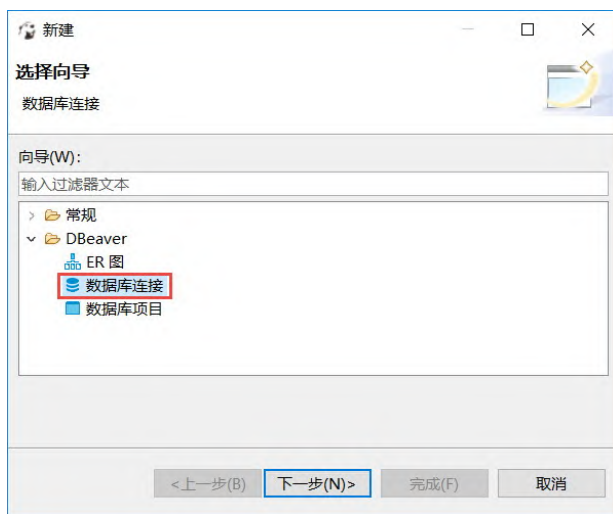


图2-2 选择向导

步骤 02

选择新连接类型。这里我们选择Apache Hive，单击“下一步”按钮，如图2-3所示。从这里看到，DBeaver支持的数据库类型是很丰富的。



图2-3 选择新连接类型

步骤 03

通用JDBC连接设置。在常规界面，填写JDBC URL、主机、端口、数据库/模式、用户名和密码等信息，如图2-4所示。



图2-4 通用JDBC连接设置

步骤 04

编辑驱动设置。单击“编辑驱动设置”按钮，在URL模板中根据Hadoop集群的权限配置添加相应的设置项，这里我们添加auth=noSasl，然后单击“添加工件”按钮，如图2-5所示。

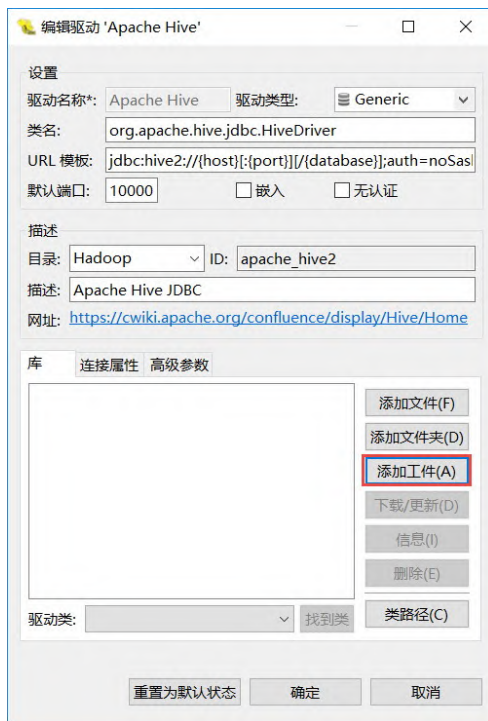


图2-5 添加工件

步骤 05

配置maven依赖。默认Hive的驱动版本是最新的RELEASE，由于集群的Hive版本是1.2.2，因此需要手动增加驱动。

下面分别配置Hive和Hadoop的驱动。首先我们配置Hive的驱动，在Group Id中输入“org.apache.hive”，在Artifact Id中输入“hive-jdbc”，在版本中输入集群对应的版本“1.2.2”，如图2-6所示。

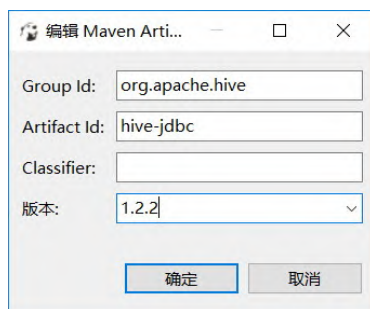


图2-6 配置Hive驱动

接下来配置 Hadoop 的驱动，在 Group Id 中输入“org.apache.hadoop”，在Artifact Id中输入“hadoop-core”，在版本中使用默认值“RELEASE”，如图2-7所示。

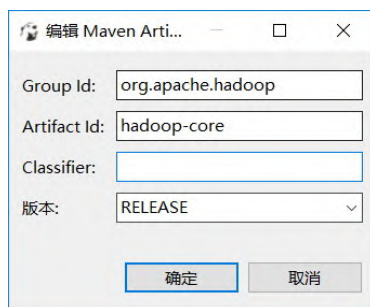


图2-7 配置Hadoop驱动

然后，单击“下载/更新”按钮，将会自动下载Hive和Hadoop的驱动程序，如图2-8所示。

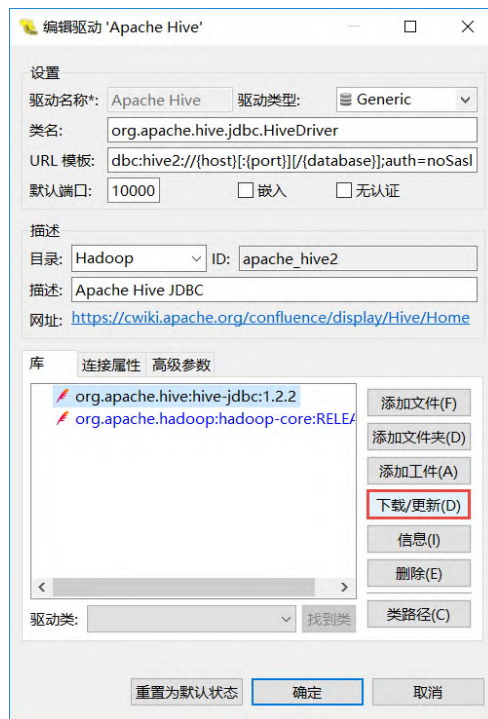


图2-8 下载更新驱动

单击界面最下方的“找到类”按钮并选择“org.apache.hive.jdbc.HiveDriver”，上方的类名会自动补齐，如图2-9所示。

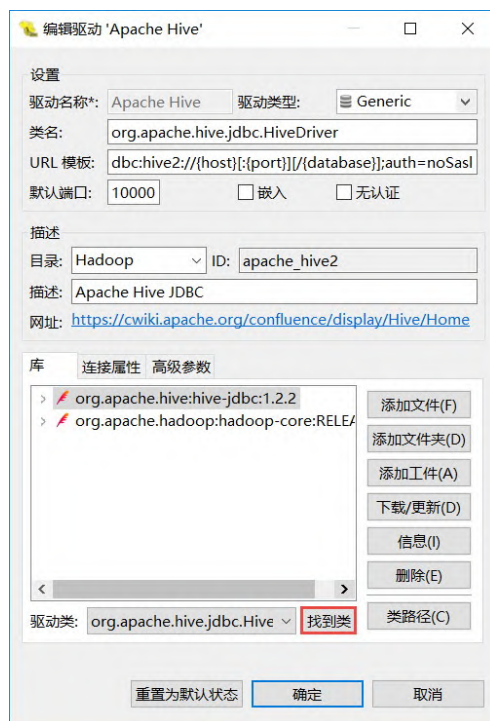


图2-9 配置驱动类

4. 测试连接是否正常

首先，需要右击连接的名称“192.168.1.7_Hadoop”，然后在下拉菜单中选择“编辑连接”选项，如图2-10所示。

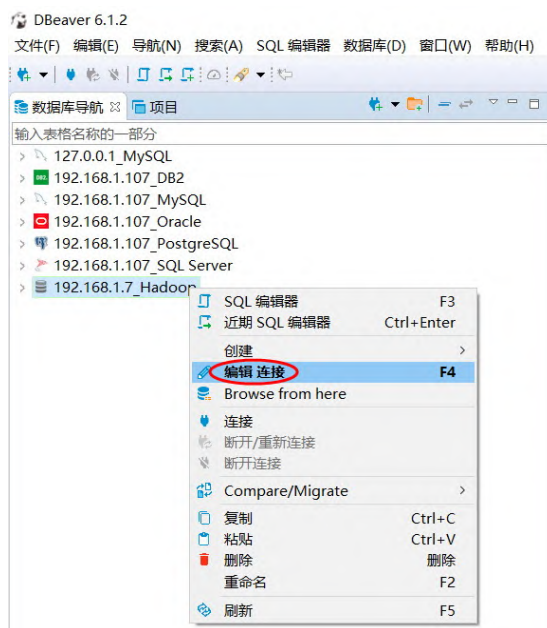


图2-10 编辑连接

在连接配置界面，选择“连接设置”选项，输入JDBC URL、主机、数据库/模式、用户名和密码等信息，如图2-11所示。

单击“测试链接”按钮，如果弹出如图2-12所示的成功信息，就说明Hive正常连接，否则需要检查连接设置，并重新进行连接过程。

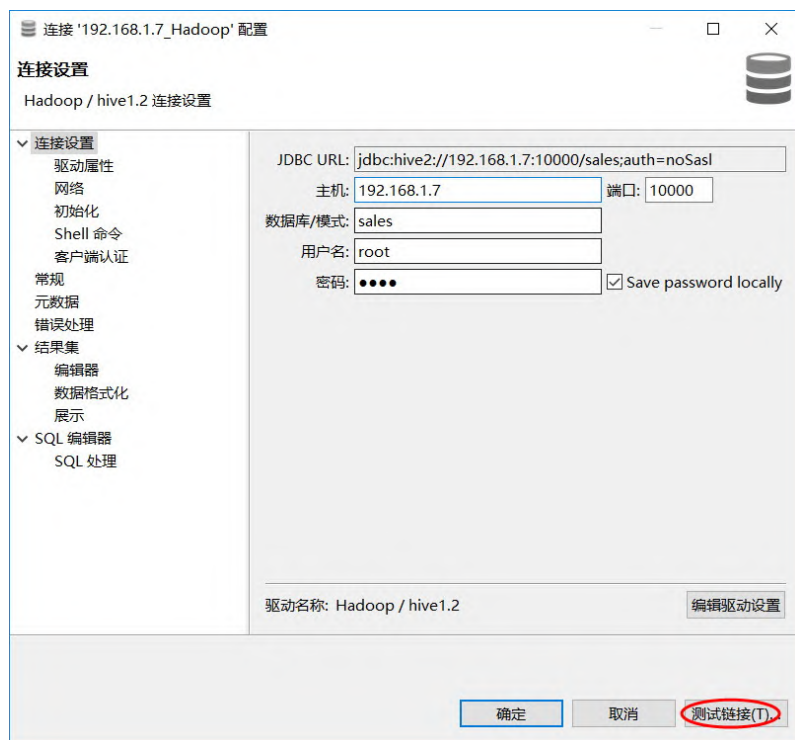


图2-11 连接配置

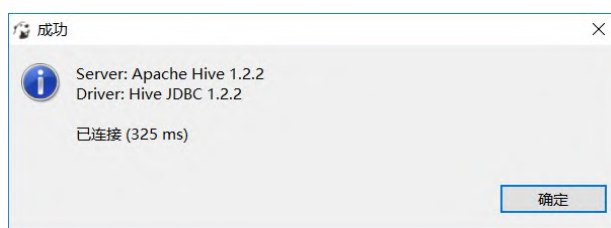


图2-12 连接成功

5. 不同职业客户平均年龄分布

我们可以在SQL界面输入想要执行的语句，例如我们要统计客户中不同职业的平均年龄，SQL语句为“select occupation, round(avg(age), 2) as avg_cust from customers group by occupation;”，SQL语句的执行结果如图2-13所示。

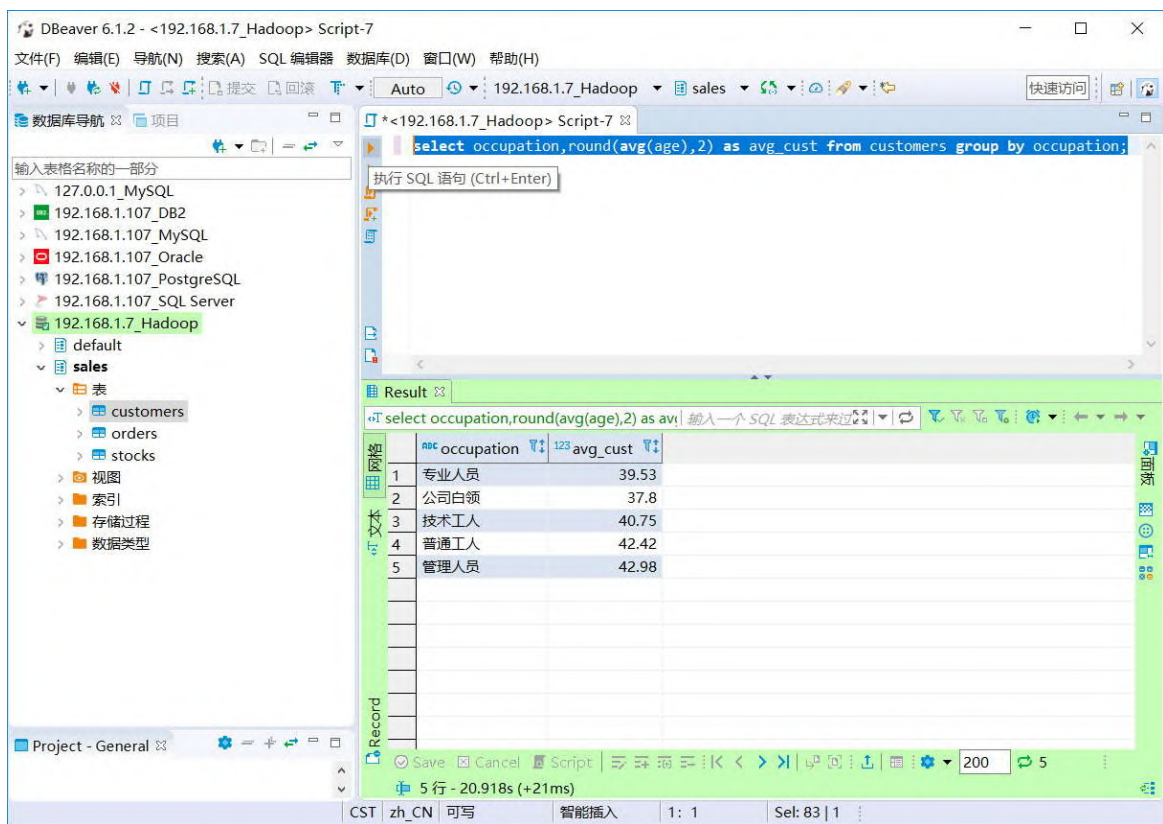


图2-13 执行SQL语句

从结果可以看出：A企业客户中，管理人员的平均年龄为42.98岁，普通工人的平均年龄为42.42岁，技术工人的平均年龄为40.75岁，专业人员的平均年龄为39.53岁，公司白领的平均年龄为37.8岁，不同职业客户的平均年龄差异比较明显。

此外，在Hadoop集群会显示SQL语句的具体执行过程以及运行时间，集群总计花费了3秒170毫秒，如图2-14所示。

```
Starting Job = job_1566113139239_0006, Tracking URL = http://master:18088/proxy/application_1566113139239_0006/
Kill Command = /home/dong/hadoop-2.5.2/bin/hadoop job -kill job_1566113139239_0006
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2019-08-18 18:14:36,690 Stage-1 map = 0%, reduce = 0%
2019-08-18 18:14:43,075 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.41 sec
2019-08-18 18:14:49,354 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 3.17 sec
MapReduce Total cumulative CPU time: 3 seconds 170 msec
Ended Job = job_1566113139239_0006
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 3.17 sec HDFS Read: 117676 HDFS Write: 154 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 170 msec
OK
```

图2-14 Hadoop集群执行过程

2.3.2 Oracle SQL Developer

Oracle SQL Developer支持常见的数据库类型，包括MySQL、Oracle、DB2、MSSQL、Hive等数据库，前提是要导入相应数据库的JAR包，而且是免费的，主要难点是下载和配置Oracle SQL Developer和Hive的JAR包，具体步骤如下：

步骤 01 下载和安装Oracle SQL Developer。首先需要到Oracle官网下载Oracle SQL Developer，下载之前需要注册Oracle账户。

步骤 02 准备连接的JAR包。使用Oracle SQL Developer连接Hive之前，需要找到集群Hive对应版本JDBC连接的JAR包，由于集群的Hive版本是Apache Hive 1.2.2，因此需要的JAR包如图2-15所示。

准备工作完成后，将连接Hive需要的JAR包上传到Oracle SQL Developer→“工具”→“首选项”→“数据库”→“第三方JDBC驱动程序”下，如图2-16所示。

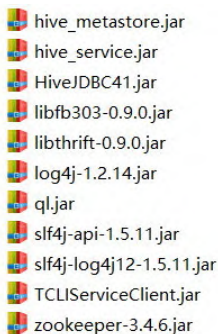


图2-15 需要的JAR包

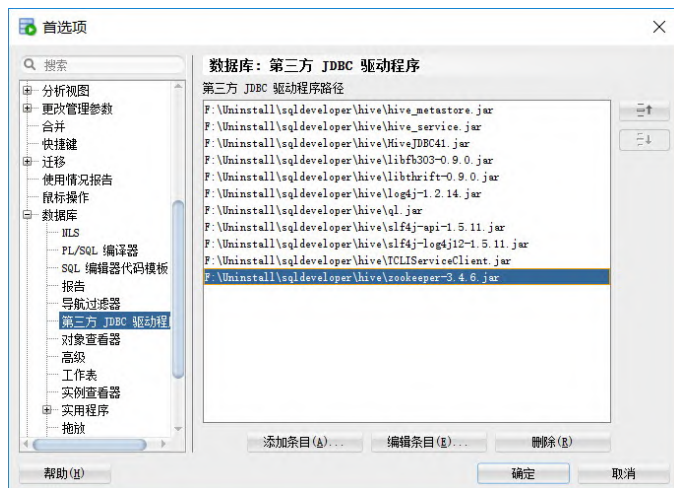


图2-16 第三方JDBC驱动程序

步骤 03

配置Oracle SQL Developer。关闭并重启Oracle SQL Developer，重启后新建Hive连接，如图2-17所示，如果出现Hive选项，就证明配置成功，如图2-18所示。



图2-17 新建连接

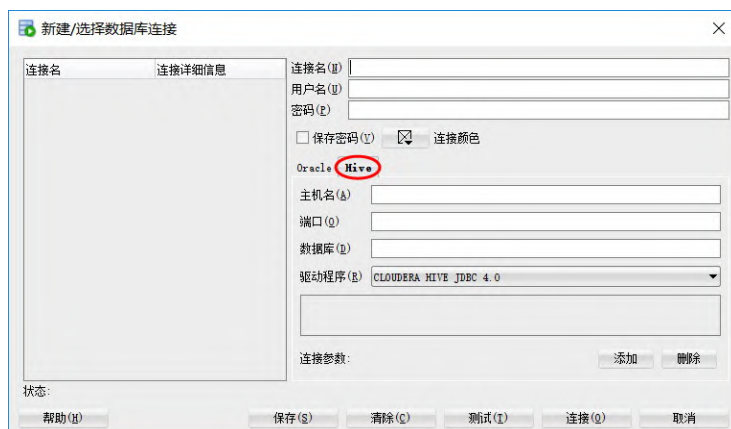


图2-18 Hive连接界面

配置相关连接参数，包括连接名、用户名、密码、主机名、端口、数据库和驱动程序，如图2-19所示。

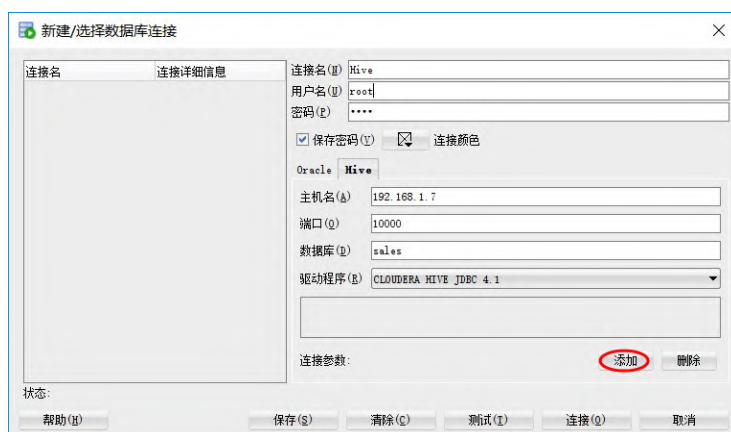


图2-19 配置连接参数

此外，还需要配置Hive的连接参数，单击“连接参数”后的“添加”按钮，弹出“添加参数”对话框，如图2-20所示。

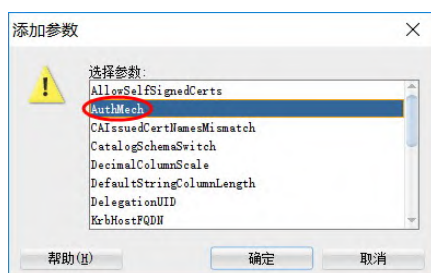


图2-20 配置连接类型参数

目前HiveServer 2支持多种用户安全认证方式：NOSASL、KERBEROS、LDAP、PAM、CUSTOM等。由于我们的集群权限设置的是NOSASL，因此AuthMech的参数需要设置为0，如图2-21所示。

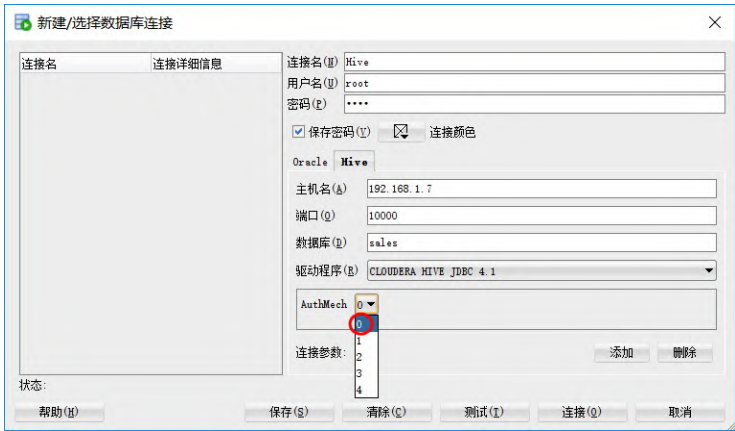


图2-21 配置连接集群权限

步骤 04 测试配置是否正常。单击“保存”按钮，软件将保存连接的配置，如图2-22所示。注意在测试之前需要开启Hadoop集群以及HiveServer 2。

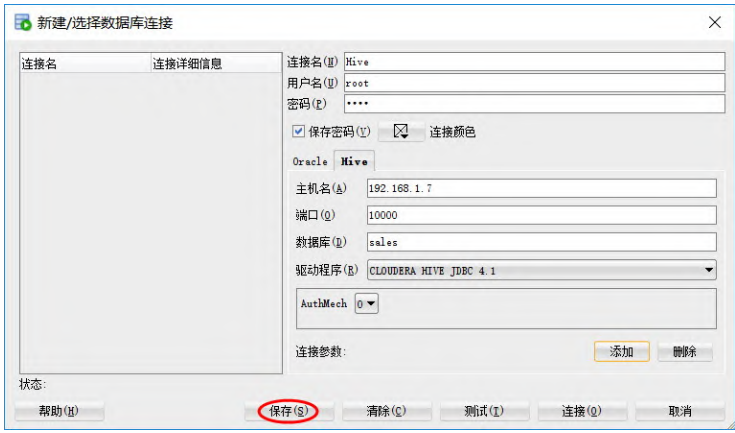


图2-22 保存连接

我们还可以单击“测试”按钮，检查是否可以正常连接Hive，如果弹出如图2-23所示的对话框，就说明可以正常连接，否则需要重新配置连接过程。

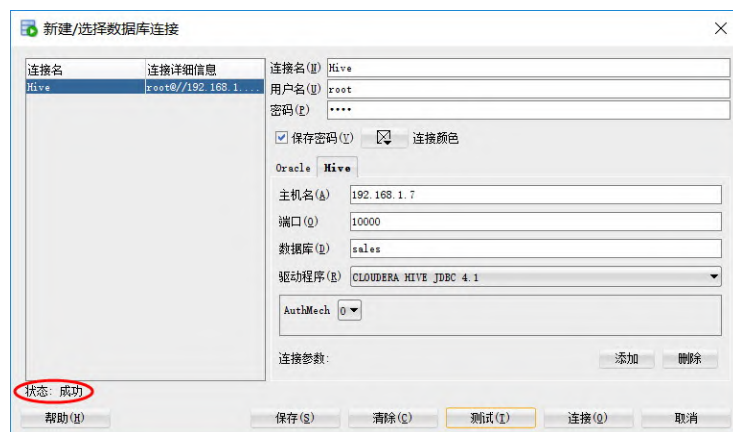


图2-23 测试连接

步骤 05

不同教育背景客户平均年龄分布。在图2-24中左侧单击配置好的Hive连接，连接成功后，我们可以在连接下查看数据库和数据库中的表。

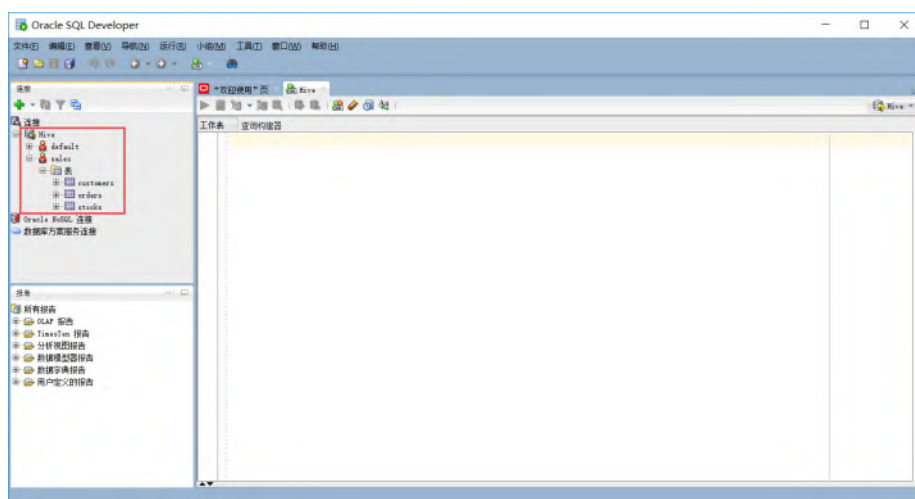


图2-24 查看数据库和表

我们可以在界面中输入想要执行的语句，例如要统计客户中不同教育背景的平均年龄分布，SQL语句为“select education,round(avg(age),2) as avg_cust from customers group by education;”，SQL语句执行结果如图2-25所示。

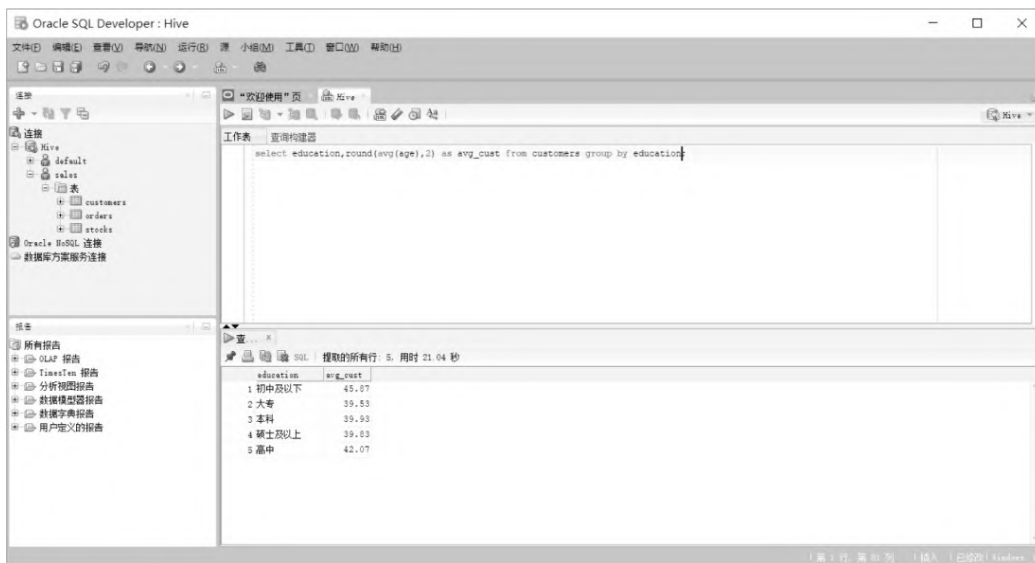


图2-25 执行SQL语句

从结果可以看出：初中及以下的平均年龄为45.87岁，高中的平均年龄为42.07岁，本科的平均年龄为39.83岁，硕士及以上的平均年龄为39.83岁，大专的平均年龄为39.53岁，不同教育背景客户的平均年龄差异比较明显。

2.3.3 DbVisualizer

DbVisualizer是基于JDBC的跨平台数据库操作工具，可以快速连接需要的数据库，包括MySQL、Oracle、DB2、MSSQL、Hive等数据库，连接Hive的具体步骤如下：

步骤 01

下载和安装DbVisualizer。我们可以到DbVisualizer的官网下载，地址是<http://www.dbvis.com/>，这里下载的是DbVisualizer 10.0版本，如图2-26所示。具体的安装步骤比较简单，选择默认安装即可，这里不进行详细介绍。

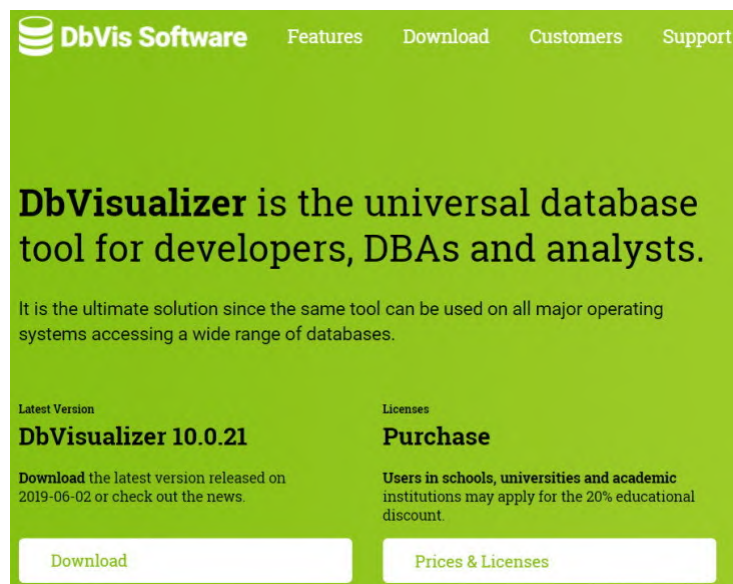


图2-26 下载DbVisualizer

步骤 02

准备连接的JAR包。在DbVisualizer的安装目录下有一个专门存放驱动的jdbc文件夹，在该文件夹下新建hive文件夹，如图2-27所示。

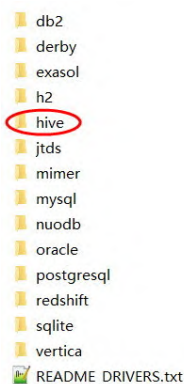


图2-27 新建hive文件夹

复制Hadoop集群的相关JAR包文件到新建的hive文件夹中，具体的包如图2-28所示。

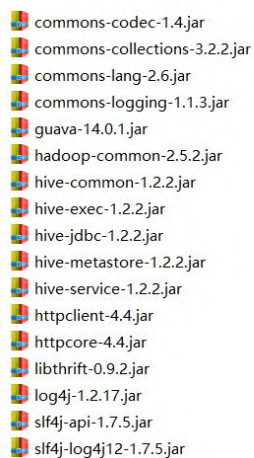


图2-28 添加依赖的包

图2-28中的JAR包分别位于以下文件夹中：

- hadoop-2.5.2/share/hadoop/common/hadoop-common-2.7.5.jar
- hadoop-2.5.2/share/hadoop/common/lib/
- apache-hive-1.2.2-bin/lib

步骤 03

配置DbVisualizer。打开DbVisualizer，此时会自动加载刚才添加的JAR包，也可以在Tools/Driver Manager中配置，如图2-29所示。

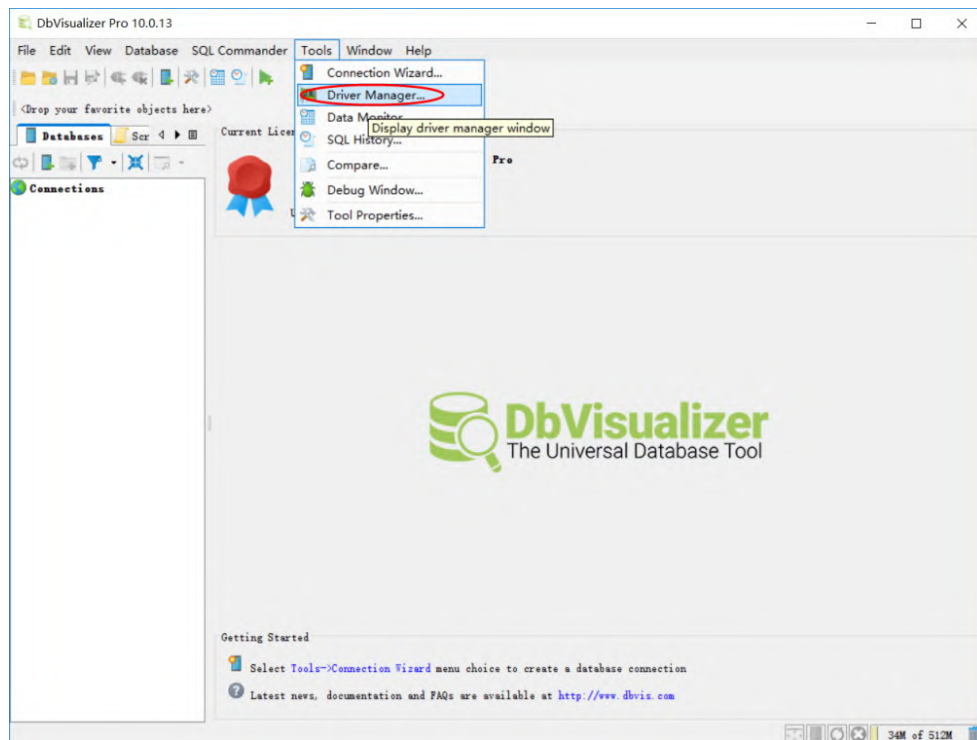


图2-29 加载依赖的包

加载Hive的JAR包结果如图2-30所示，可以根据需要进行核查和修改等。

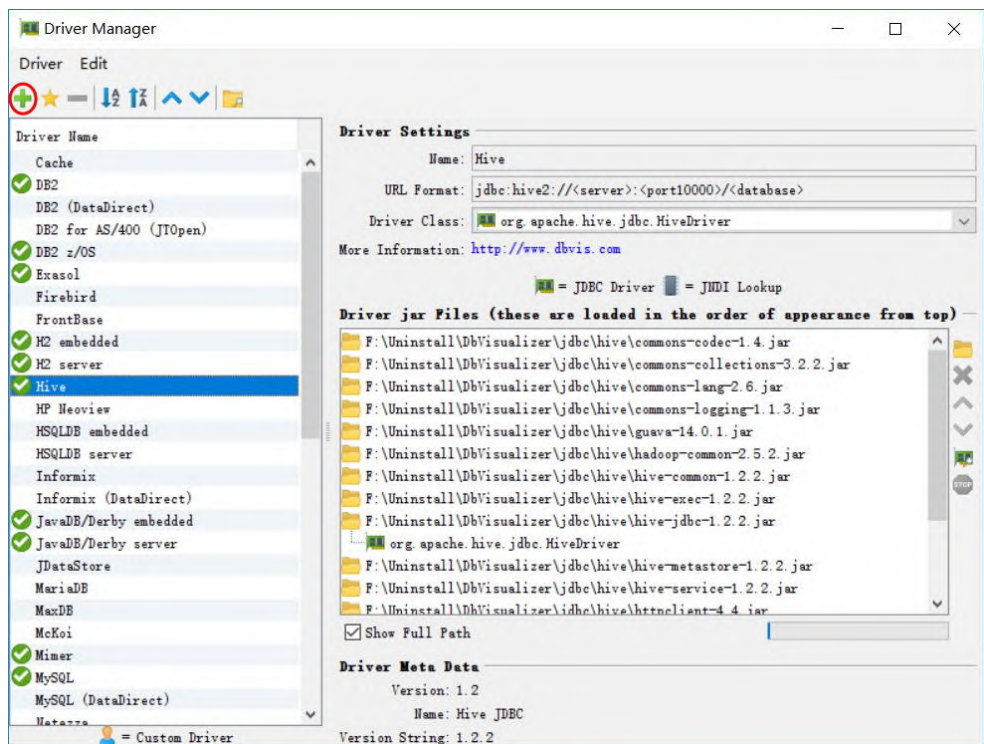


图2-30 核查依赖的包

步骤 04

测试配置是否正常。关闭DbVisualizer，然后重新打开DbVisualizer，弹出New Connection Wizard对话框，如图2-31所示。

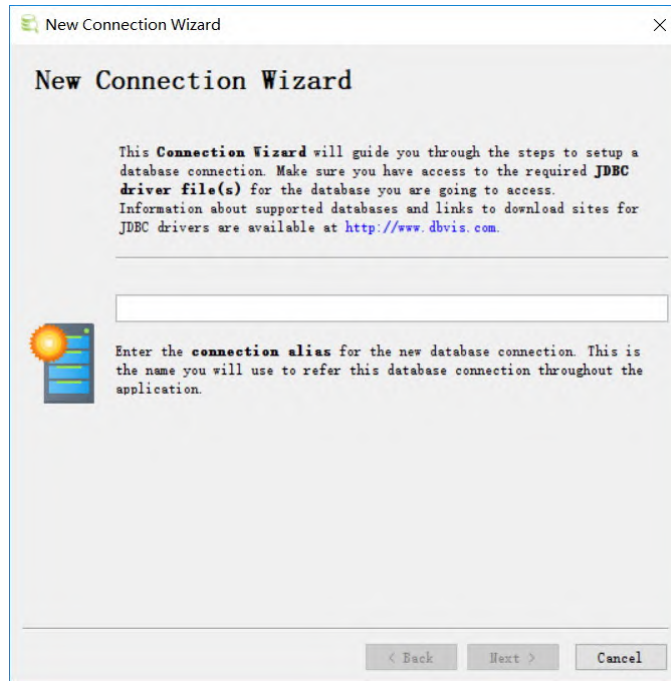


图2-31 添加新的连接

输入连接的名称，再单击Next按钮，弹出选择数据库驱动界面，这里我们选择Hive，单击Next按钮，如图2-32所示。

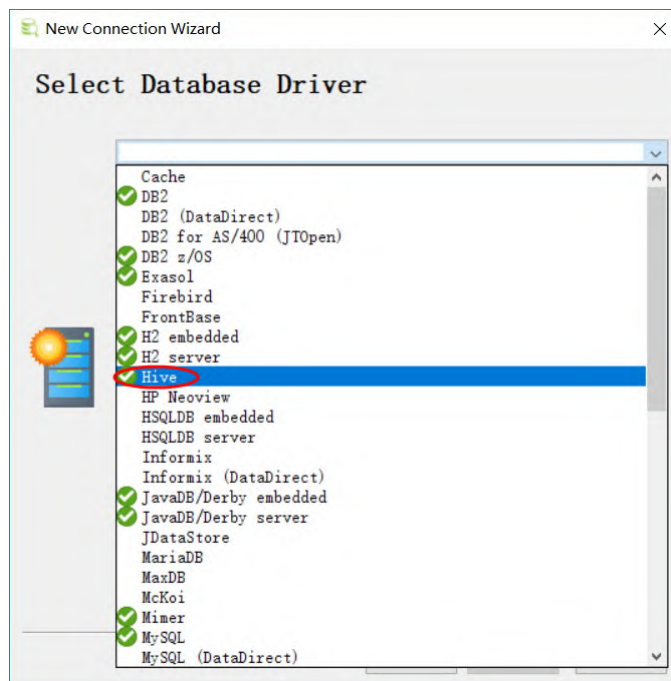


图2-32 选择连接名

在Settings Format下拉框中选择Database URL，这个比较重要，否则无法正常连接Hive，如图2-33所示。

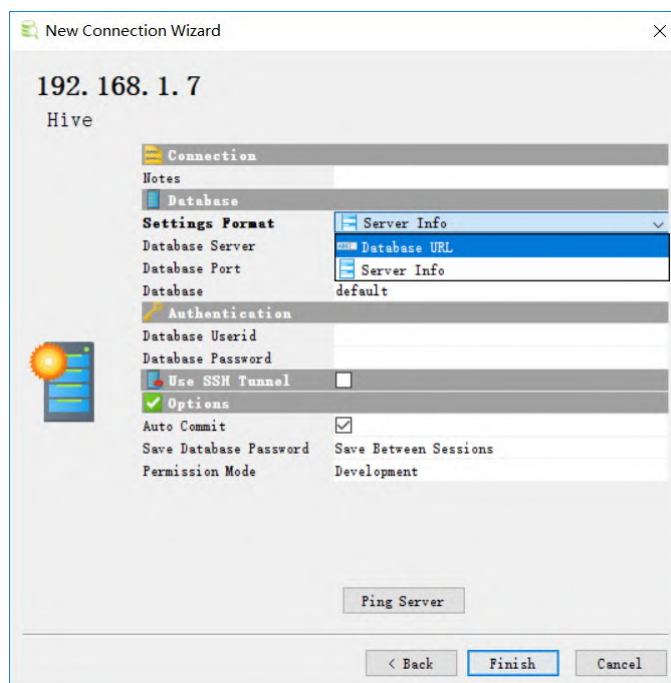


图2-33 选择连接方式

然后，在 Database URL 中输入“jdbc:hive2://192.168.1.7:10000/sales;auth=noSasl”，分别在 Database Userid和Database Password中输入账户和密码，如图2-34所示。

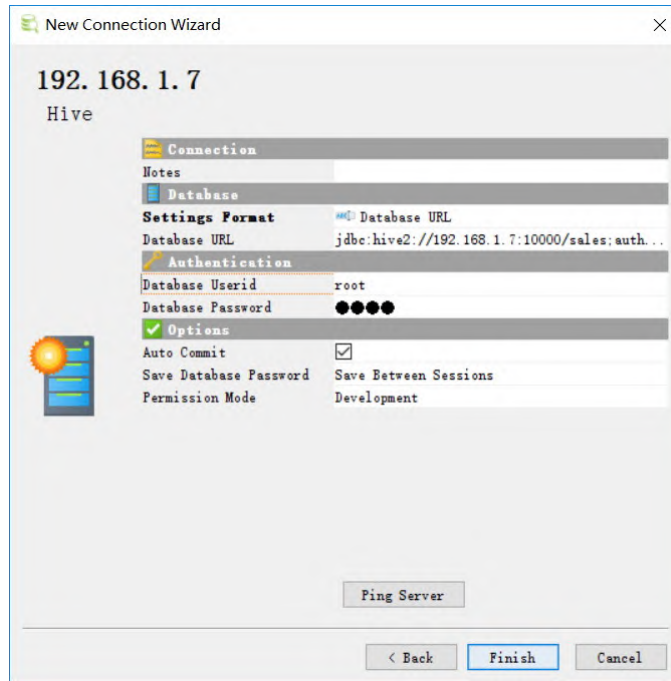


图2-34 配置连接参数

步骤 05

不同性别客户平均年龄分布。单击图2-34中的Finish按钮，如果弹出如图2-35所示的界面，就说明正常连接Hive，该界面显示Hadoop集群中的数据库和表。

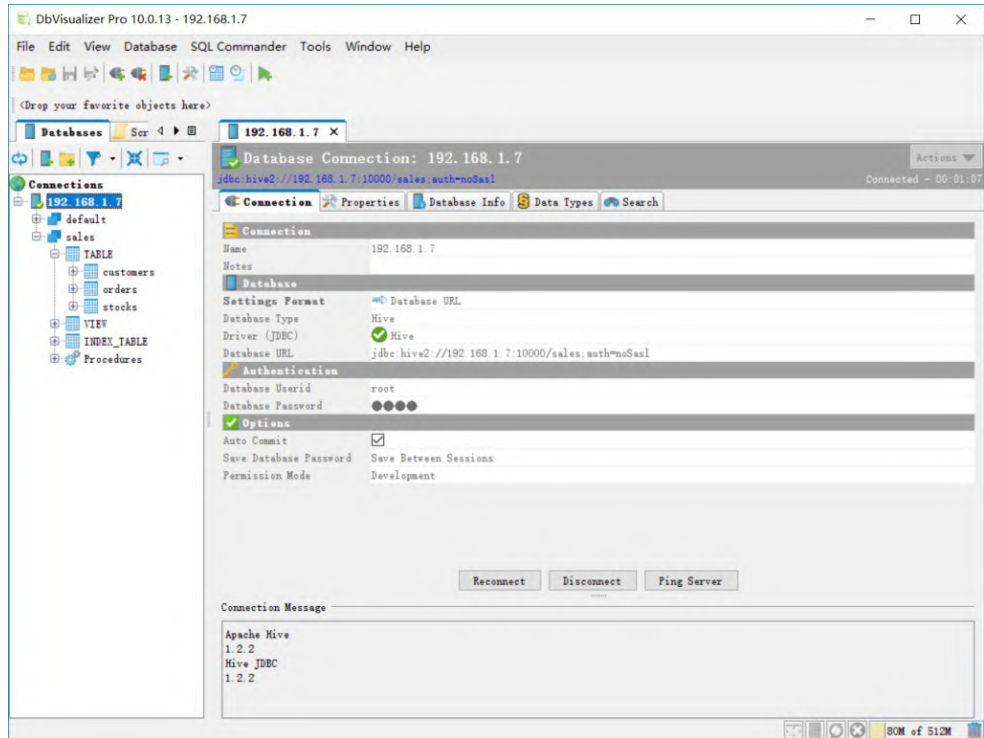


图2-35 查看数据库和表

我们可以在界面中输入想要执行的语句，例如要统计不同性别客户的平均年龄分布，SQL语句为“select gender, round(avg(age), 2) as avg_cust from customers group by gender;”，SQL语句执行结果如图2-36所示。

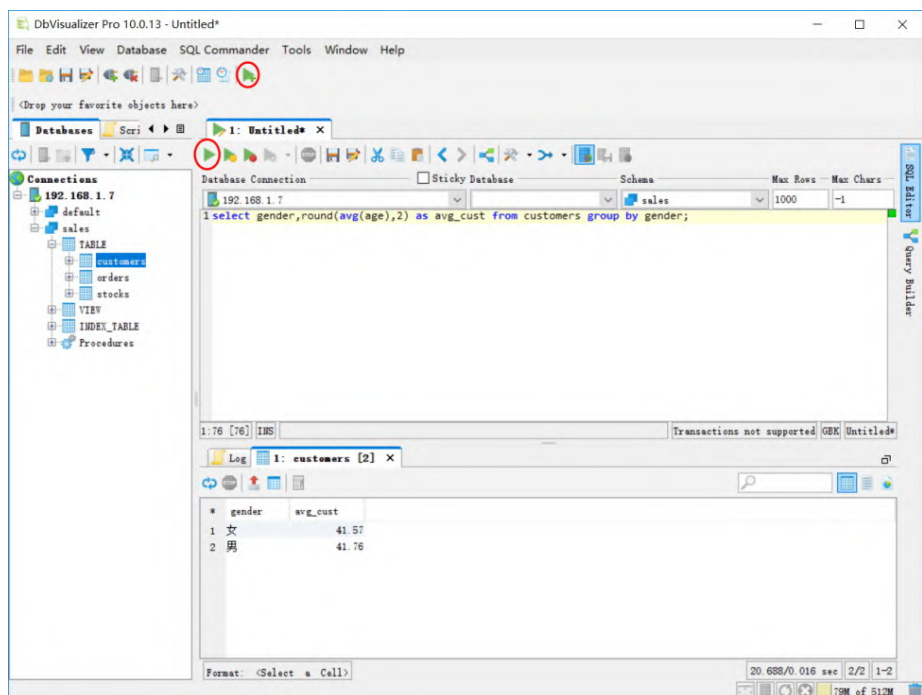


图2-36 执行SQL语句

从结果可以看出：男性客户的平均年龄为41.76岁，女性客户的平均年龄为41.57岁，不同类型客户的平均年龄差异不是很明显。

2.3.4 Squirrel SQL Client

Squirrel SQL Client是一个用Java写的数据库客户端工具，它通过一个统一的用户界面来操作MySQL、MSSQL、Hive等支持JDBC访问的数据库，具体连接Hive的步骤如下：

步骤 01

下载和安装Squirrel SQL Client。软件可以直接从官网下载：<http://www.squirrelsql.org>，截至2019年8月，最新版本为3.9.1，单击Download Squirrel SQL Client链接进行下载，如图2-37所示。

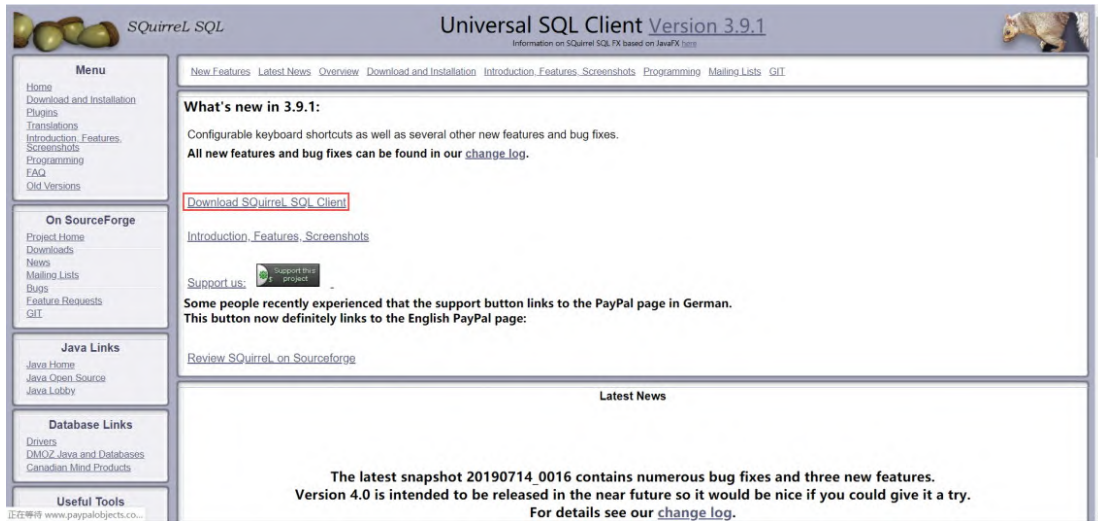


图2-37 下载Squirrel SQL Client

Squirrel SQL Client有安装版本和免安装版本，我们这里选择的是免安装版本，单击Plain zips the latest release for Windows/Linux/MacOS X/others链接进行下载，如图2-38所示。



图2-38 选择免安装版本

然后，选择squirrelsql-3.9.1-optional.zip，该版本可以根据需要添加数据库的驱动扩展，如图2-39所示。

Home / 1-stable / 3.9.1-plainzip

Name	Modified	Size	Downloads / Week
Parent folder			
squirrelsql-3.9.1-optional.zip	2019-02-24	57.1 MB	181
squirrelsql-3.9.1-standard.zip	2019-02-24	48.9 MB	239
squirrelsql-3.9.1-base.zip	2019-02-24	38.4 MB	20
Totals: 3 Items		144.5 MB	440

图2-39 选择驱动可扩展版本

下载完成后，解压该文件，在文件夹中双击squirrel-sql.bat，第一次打开Squirrel SQL Client，界面是空白的，如图2-40所示。

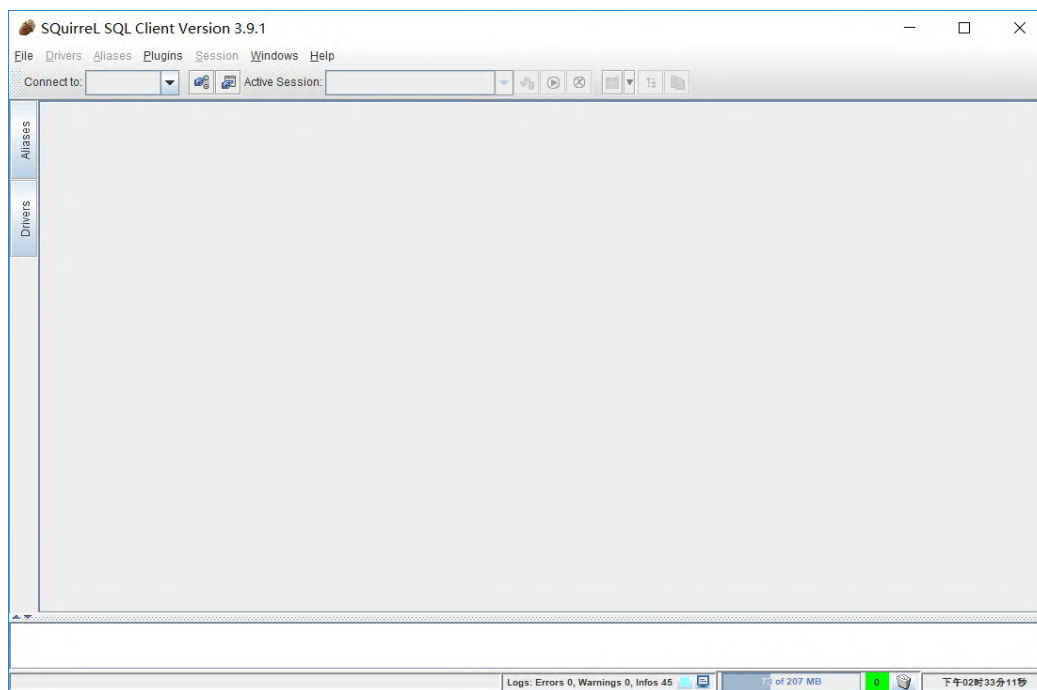


图2-40 打开Squirrel SQL Client

步骤 02

准备连接的JAR包。在Squirrel的安装目录下新建hive文件夹，如图2-41所示。

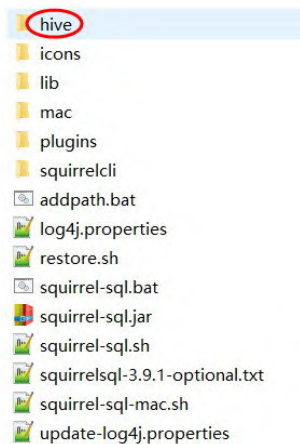


图2-41 新建hive文件夹

复制Hadoop集群的相关JAR包到新建的hive文件夹中，具体的包如图2-42所示。

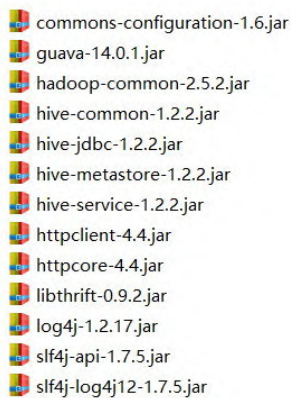


图2-42 配置连接JAR包

图2-42中的JAR包分别位于以下文件夹中：

- hadoop-2.5.2/share/hadoop/common/hadoop-common-2.7.5.jar
- hadoop-2.5.2/share/hadoop/common/lib/
- apache-hive-1.2.2-bin/lib

步骤 03

配置Squirrel SQL Client。连接Hive数据库，首先配置数据库连接的驱动类型，选择界面左侧的Drivers，然后单击“+”按钮，如图2-43所示。

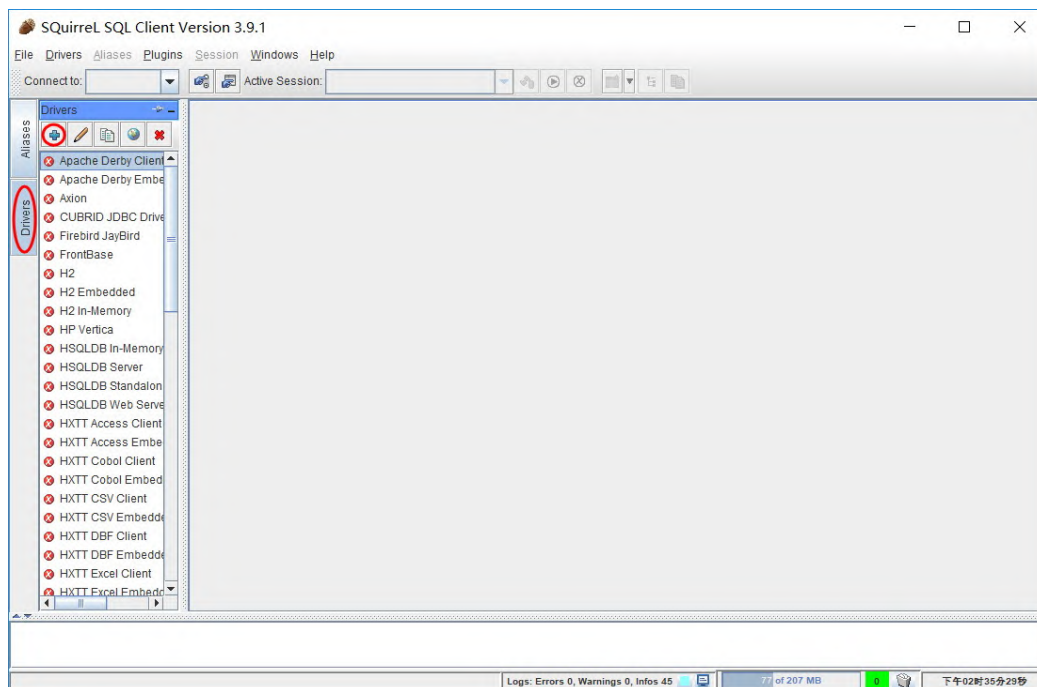


图2-43 配置数据库驱动类型

在配置界面，Name可以随意填写，这里输入“hive”，Example URL是连接集群的重要信息，根据实际情况填写，这里输入“jdbc:hive2://192.168.1.7:10000/sales;auth=noSasl”，即通过JDBC连接HiveServer 2，后面是服务器地址、端口、数据库及授权方式，Website URL可以不用填写。然后选择Hive连接所需要的JAR包，切换至Extra Class Path选项卡，再单击Add按钮，如图2-44所示。

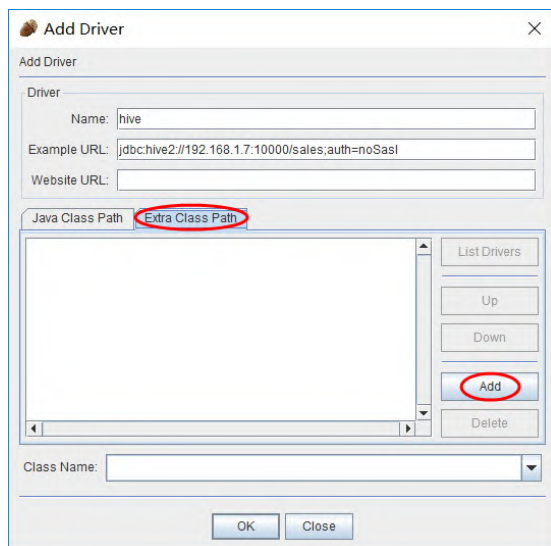


图2-44 配置数据库参数

打开Squirrel安装目录下新建的hive文件夹，选择Hive连接所需要的JAR包，然后单击“打开”按钮，如图2-45所示。

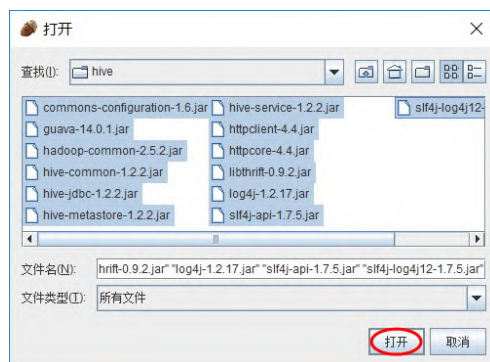


图2-45 选择连接的JAR包

在 Class Name 选项 中 输入 “org.apache.hive.jdbc.HiveDriver”，然后单击OK按钮，如图2-46所示，Hive驱动程序的配置过程到此结束。

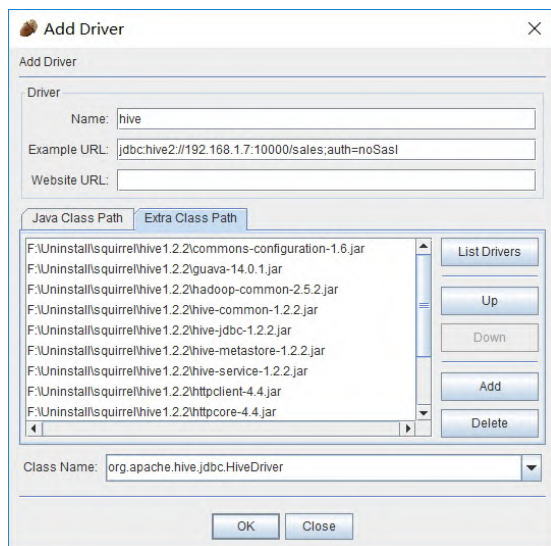


图2-46 配置类名

步骤 04 测试配置是否正常。配置完成后，需要测试配置是否正确，选择界面左侧的Aliases，单击“+”按钮，如图2-47所示。

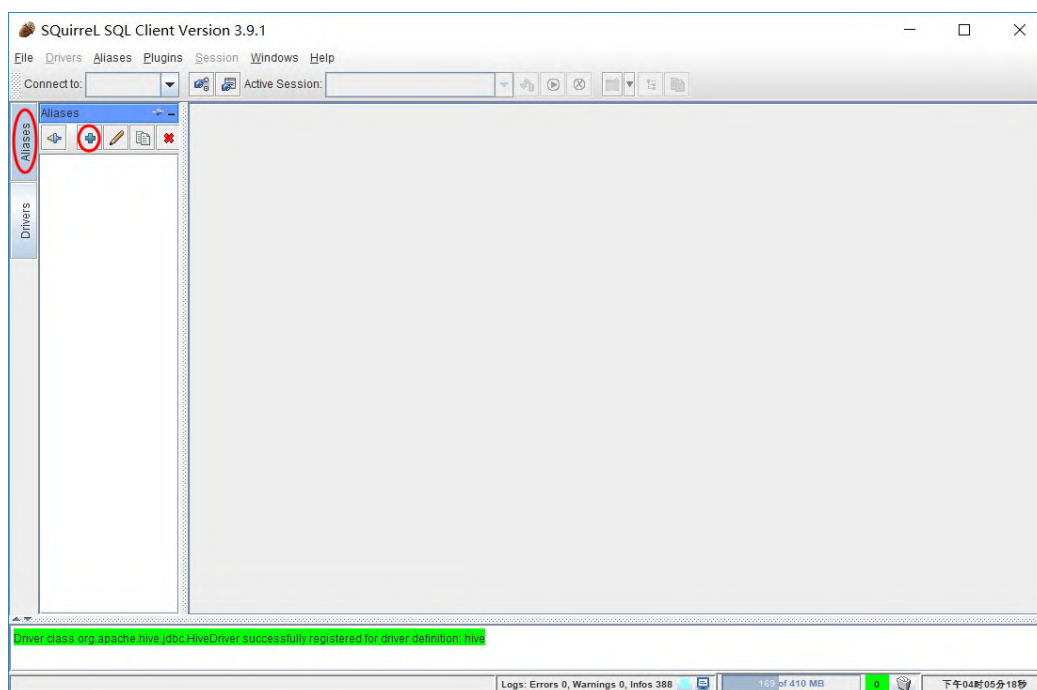


图2-47 测试配置是否正确

在弹出的连接对话框中，Name即连接的名称，可以随意填写，这里输入服务器的地址“192.168.1.7”，在Driver下拉框中选择我们配置好的hive，输入连接所需要的URL、User Name和Password，最后单击Test按钮，弹出连接信息对话框，单击Connect按钮，如果弹出Connection successful，就说明成功连接Hive，如图2-48所示。

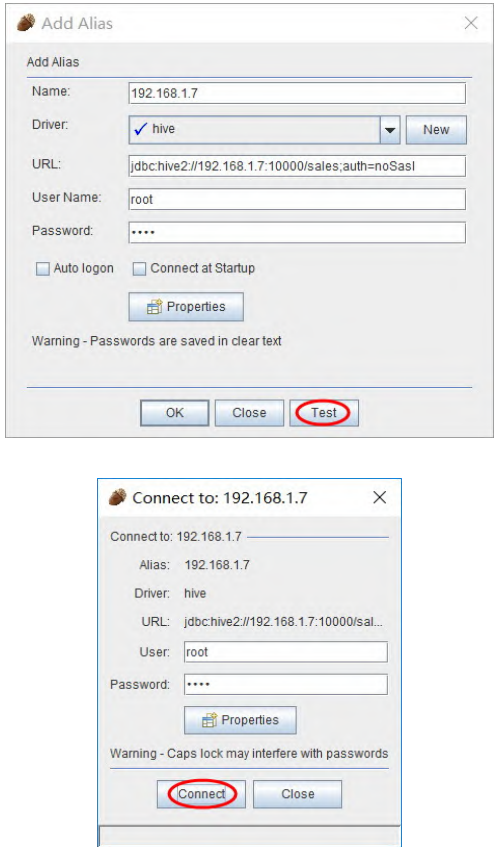


图2-48 配置连接参数

步骤 05 不同类型客户平均年龄分布。在图2-49左侧单击配置好的Hive连接，连接成功后，可以在Objects下查看数据库和数据库中的表。

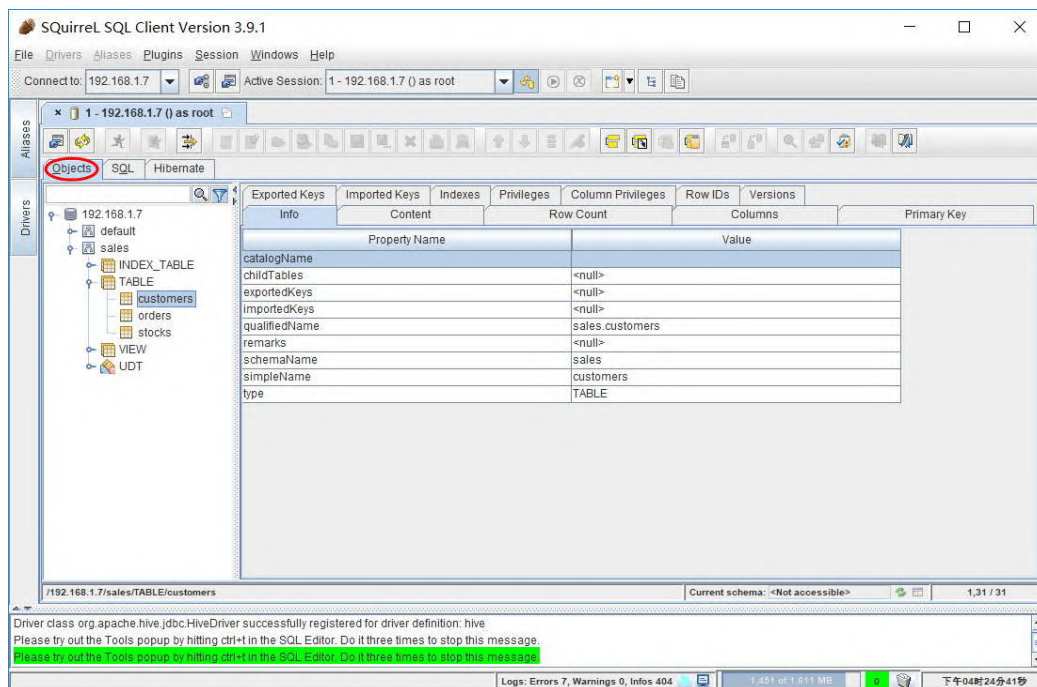


图2-49 查看数据库和表

我们可以在界面中输入想要执行的语句，例如要统计不同价值类型客户的平均年龄分布，SQL语句为“select custcat,round(avg(age),2) as avg_cust from customers group by custcat;”，SQL语句执行结果如图2-50所示。

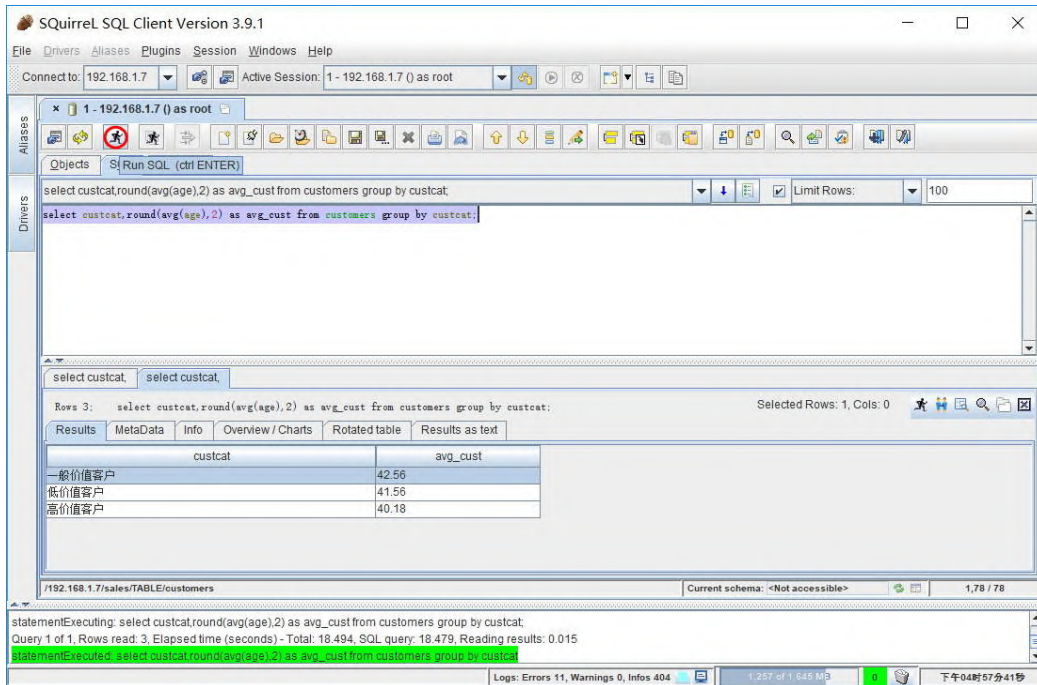


图2-50 执行SQL语句

从结果可以看出：一般价值客户的平均年龄为42.56岁，低价值客户的平均年龄为41.56岁，高价值客户的平均年龄为40.18岁，不同类型客户的平均年龄差异不是很明显。

此外，在Hadoop集群会显示SQL语句的具体执行过程以及运行时间，集群总计花费了3秒270毫秒，如图2-51所示。

```
Starting Job = job_1566113139239_0005, Tracking URL = http://master:18088/proxy/application_1566113139239_0005/
Kill Command = /home/dong/hadoop-2.5.2/bin/hadoop job -kill job_1566113139239_0005
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2019-08-18 16:33:13,924 Stage-1 map = 0%, reduce = 0%
2019-08-18 16:33:20,162 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.46 sec
2019-08-18 16:33:26,393 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 3.27 sec
MapReduce Total cumulative CPU time: 3 seconds 270 msec
Ended Job = job_1566113139239_0005
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 3.27 sec HDFS Read: 117658 HDFS Write: 117 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 270 msec
OK
```

图2-51 Hadoop执行过程

第3章

大数据可视化工具



本章将通过某上市电商企业的客户数据、订单数据、股价数据详细介绍Tableau、Zeppelin和Python三类大数据常用可视化工具，包括如何连接到Hive和Spark。

3.1 Tableau

3.1.1 Tableau简介

Tableau公司成立于2003年，是由斯坦福大学的3位校友Christian Chabot（首席执行官）、Chris Stole（开发总监）以及Pat Hanrahan（首席科学家）在远离硅谷的西雅图注册成立的。其中，Chris Stole是计算机博士；Pat Hanrahan是皮克斯动画工作室的创始成员之一，曾负责视觉特效渲染软件的开发，两度获得奥斯卡最佳科学技术奖，至今仍在斯坦福担任教授职位，教授计算机图形课程。

Tableau公司主要面向企业数据提供可视化服务，是一家商业智能软件提供商。企业运用Tableau授权的数据可视化软件对数据进行处理和展示，不过Tableau的商品并不局限于企业，其他机构甚至个人都能很好地运用Tableau软件进行数据分析工作。Tableau在抢占细分市场，也就是大数据处理末端的可视化市场，目前市场上并没有太多这样的商品。

Tableau系列包含7种工具：Tableau Desktop、Tableau Prep、Tableau Online、Tableau Server、Tableau Public、Tableau Mobile、Tableau Reader，其中可以连接Hadoop平台的工具是Tableau Desktop。“所有人都能学会的业务分析工具”是Tableau官方网站上对Tableau Desktop的描述。确实，Tableau Desktop非常简单易用，这也是该软件的最大特点。使用者不需要精通复杂的编程和统计原理，只需要把数据直接拖曳到工作簿中，通过一些简单的设置就可以得到想要的可视化图形。

Tableau Desktop的学习成本很低，使用者可以快速上手，这无疑对日渐追求高效率和成本控制的企业来说具有巨大吸引力，特别适合日常工作中需要绘制大量报表、经常进行数据分析或需要制作图表的人使用。简单、易用并没有妨碍Tableau Desktop拥有强大的性能，它不仅能完成基本的统计预测和趋势预测，还能实现数据源的动态更新。Tableau Desktop的开始界面如图3-1所示。

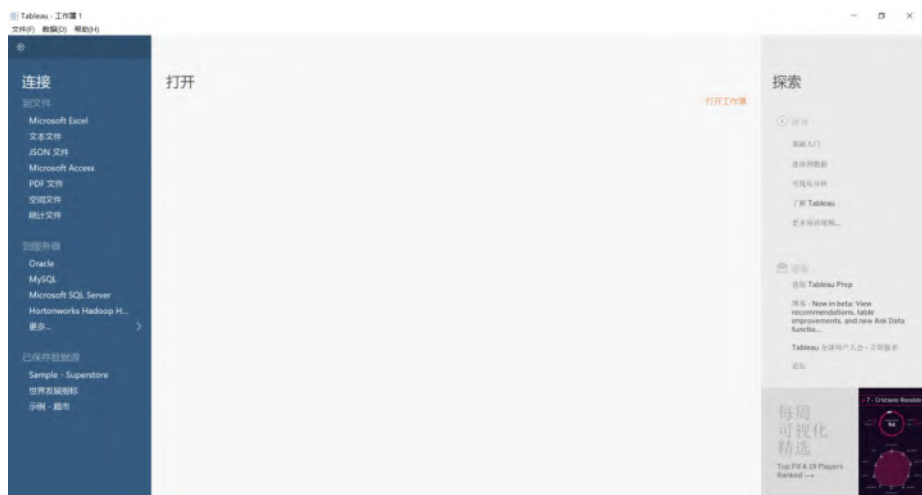


图3-1 Tableau Desktop的开始界面

Tableau Desktop不同于SPSS，SPSS作为统计分析软件，比较偏重于统计分析，使用者需要有一定的数理统计基础，虽然功能强大且操作简单、友好，但输出的图表与办公软件的兼容性及交互方面有所欠缺。Tableau Desktop是一款专业的数据可视化软件，用来辅助人们进行视觉化思考，并没有SPSS强大的统计分析功能。

总之，快速、易用、可视化是Tableau Desktop最大的特点，能够满足大多数企业、政府机构数据分析和展示的需要，以及部分大学、研究机构可视化项目的要求，而且特别适合企业使用，毕竟Tableau自己的定位是业务分析和商业智能。在简单、易用的同时，Tableau Desktop极其高效，数据引擎的速度极快，处理上亿行数据只需几秒就可以得到结果，用其绘制报表的速度也比程序员制作传统报表快10倍以上。

3.1.2 Tableau连接Hive

Tableau连接Hadoop集群的方法与MS Power BI类似，驱动程序可以通用。下面我们通过分析不同类型客户的平均年龄的案例详细介绍

如何通过Tableau连接Hive，并对集群中的数据表进行可视化分析，具体步骤如下：

步骤 01 打开Tableau Desktop软件，然后找到Hive的对应连接接口，这里我们选择“更多”下的Hortonworks Hadoop Hive选项，如图3-2所示。

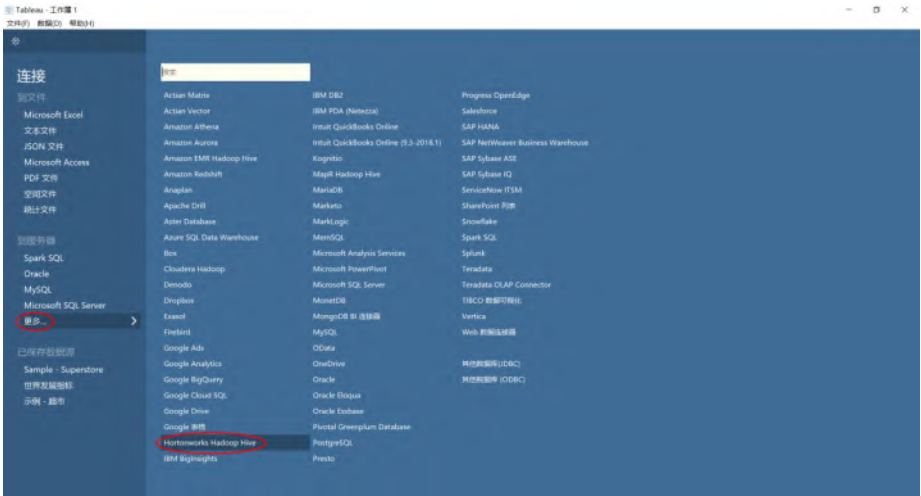


图3-2 选择数据源

步骤 02 在弹出的界面中输入Hadoop集群的服务器地址和服务器登录信息，服务器登录信息包括身份验证方式、传输方式、用户名和密码等，然后单击“登录”按钮，如图3-3所示。

Hortonworks Hadoop Hive

服务器(V): 192.168.1.7 端口(P): 10000

输入服务器登录信息:

身份验证: 用户名和密码

传输: 二进制

用户名(U): root

密码:

☐ 需要 SSL(L)

初始 SQL(I)...

登录

图3-3 输入连接参数

步骤 03

在Tableau的数据源界面，需要选择架构（数据库），在搜索框中输入“sales”，这是我们需要分析的数据所在的数据库名，然后单击界面右侧的“搜索”按钮，并选择合适的匹配模式，这里选择“精确”单选按钮，如图3-4所示。

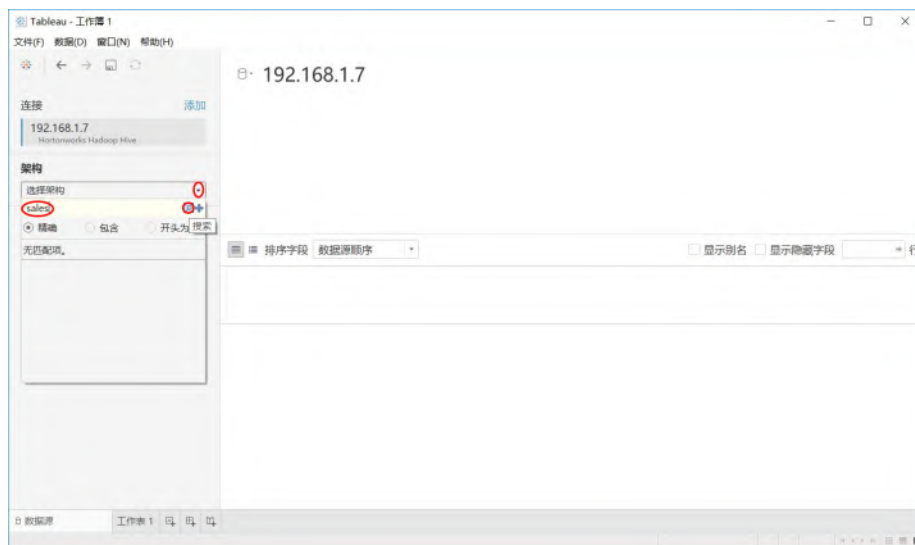


图3-4 选择架构

步骤 04

在搜索框中输入需要分析的表，如果是多个表，就需要重复操作，由于只需要分析客户表，因此输入“customers”，然后单击右侧的“搜索”按钮，并选择合适的匹配模式，这里选择“精确”单选按钮，如图3-5所示。



图3-5 搜索需要连接的表

步骤 05 在搜索框下方将会显示customers表，可以将其拖曳到右侧相应的区域，为后续可视化分析做准备，如图3-6所示。

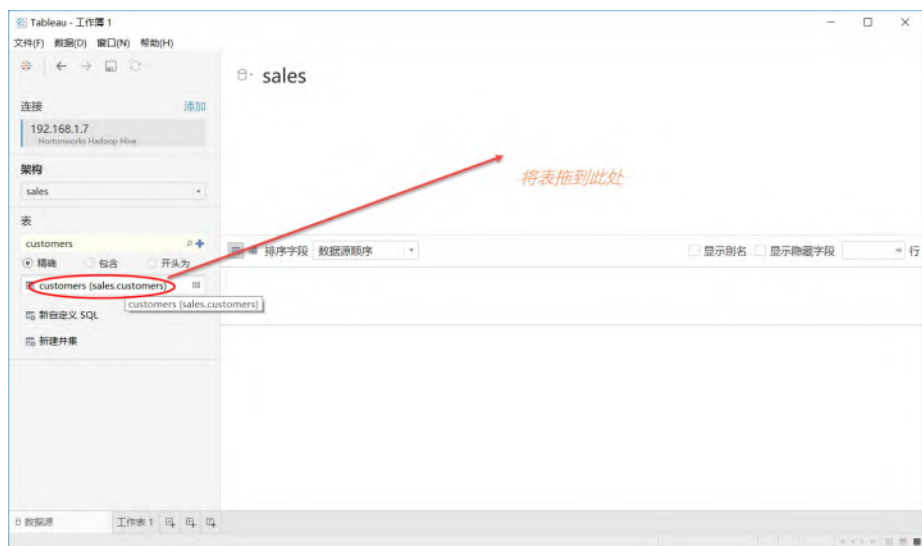


图3-6 拖曳需要连接的表

步骤 06 查看界面右下方的customers表中的数据是否有异常，如果没有异常，就单击界面左下方的“工作表1”选项，进入可视化视图的制作界面，如图3-7所示。

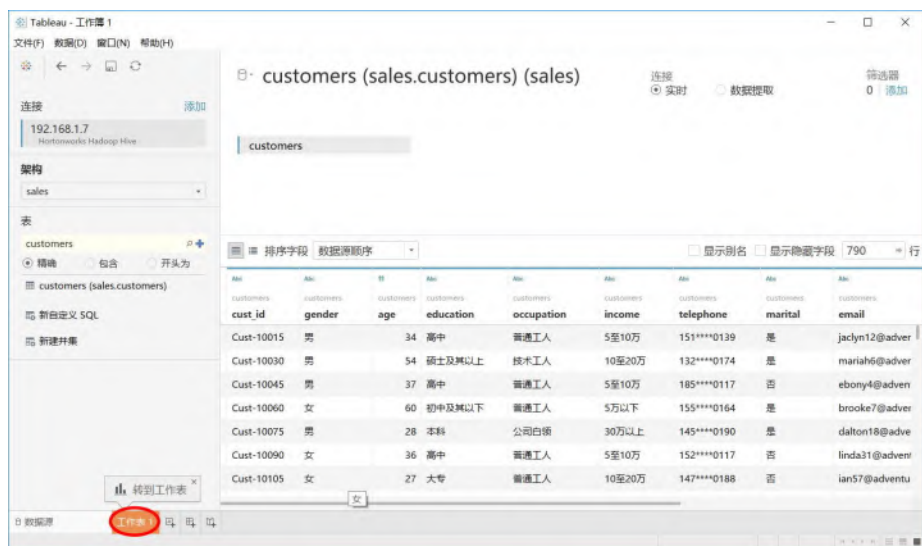


图3-7 查看需要连接的表

步骤 07 将custcat字段拖曳到“列”功能区，将平均值（age）字段拖曳到“行”功能区，度量类型选择“平均值”，注意其默认类型是“总和”，如图3-8所示。

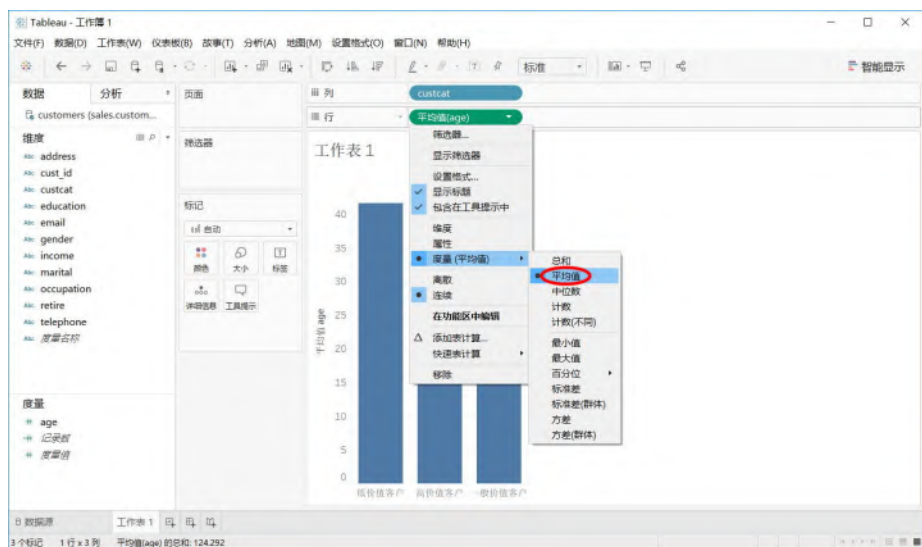


图3-8 设置可视化字段

步骤 08 最后，对视图进行适当的调整，例如颜色、标签、标题等，最终的效果如图3-9所示。

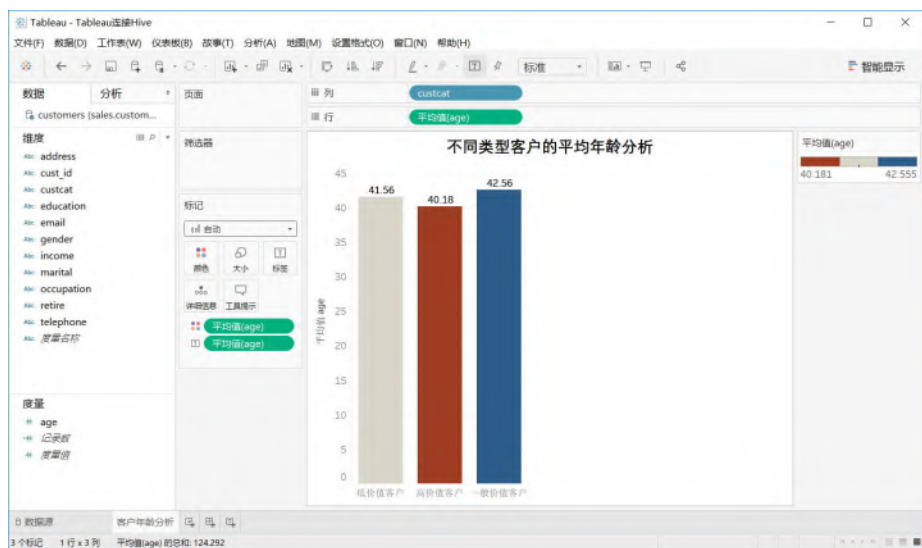


图3-9 调整可视化视图

从上面制作的条形图可以看出：一般价值客户的平均年龄为42.56岁，低价值客户的平均年龄为41.56岁，高价值客户的平均年龄为40.18岁，不同价值类型客户的平均年龄没有明显的差异。

3.1.3 Tableau连接Spark

Tableau连接Spark的方法与MS Power BI类似，驱动程序可以通用。下面我们通过分析企业现有客户的职业类型案例详细介绍如何通过Tableau连接Spark，并对集群中的数据表进行可视化分析，具体步骤如下：

步骤 01 打开Tableau Desktop软件，找到Spark对应的连接接口，这里我们选择“更多”下的Spark SQL选项，如图3-10所示。

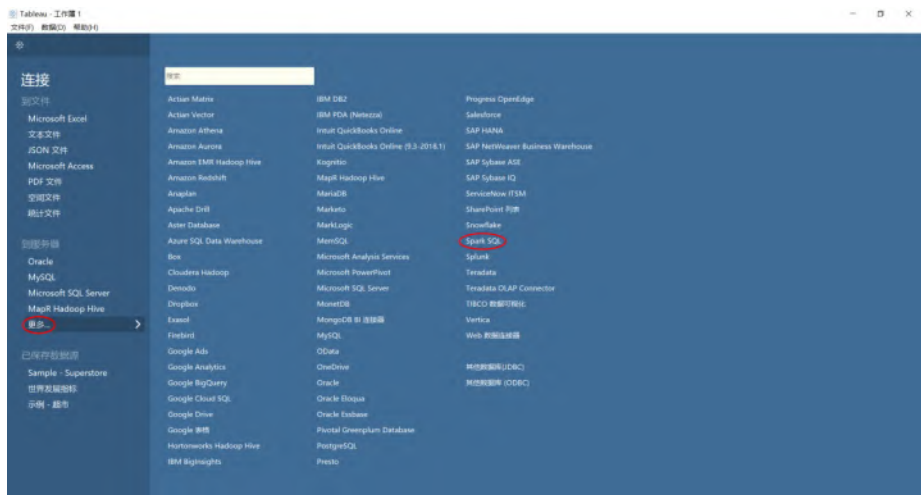


图3-10 连接数据源

步骤 02

在弹出的界面中输入Hadoop集群的服务器地址和服务器登录信息，服务器登录信息包括Spark类型、身份验证方式、传输方式、用户名和密码等，然后单击“登录”按钮，如图3-11所示。



图3-11 输入连接参数

步骤 03

在Tableau的数据源界面，选择架构（数据库），在搜索框中输入“sales”，这是我们需要分析的数据所在的数据库名，然后单击右侧的“搜索”按钮，并选择合适的匹配模式，这里选择“精确”单选按钮，如图3-12所示。



图3-12 选择架构

步骤 04

在搜索框中输入我们需要分析的表，如果是多个表，就需要重复操作，由于我们只需要分析客户表，因此输入“customers”，然后单击右侧的“搜索”按钮，并选择合适的匹配模式，这里选择“精确”单选按钮，如图3-13所示。



图3-13 搜索需要连接的表

步骤 05 在搜索框下方将会显示customers表，可以将其拖曳到右侧相应的区域，为后续可视化分析做准备，如图3-14所示。

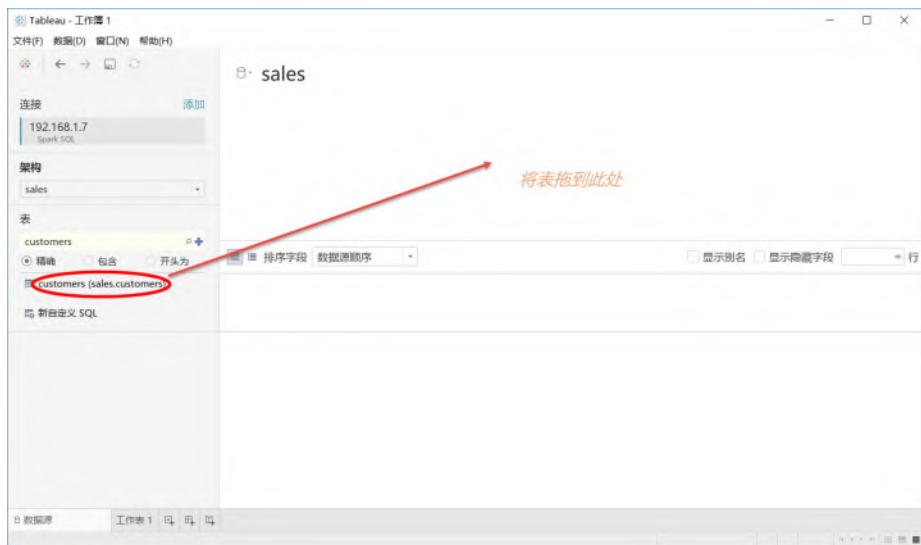


图3-14 拖曳需要连接的表

步骤 06 查看界面右下方的customers表中的数据是否有异常，如果没有异常，就单击界面左下方的“工作表1”选项，如图3-15所示，进入可视化视图的制作界面。

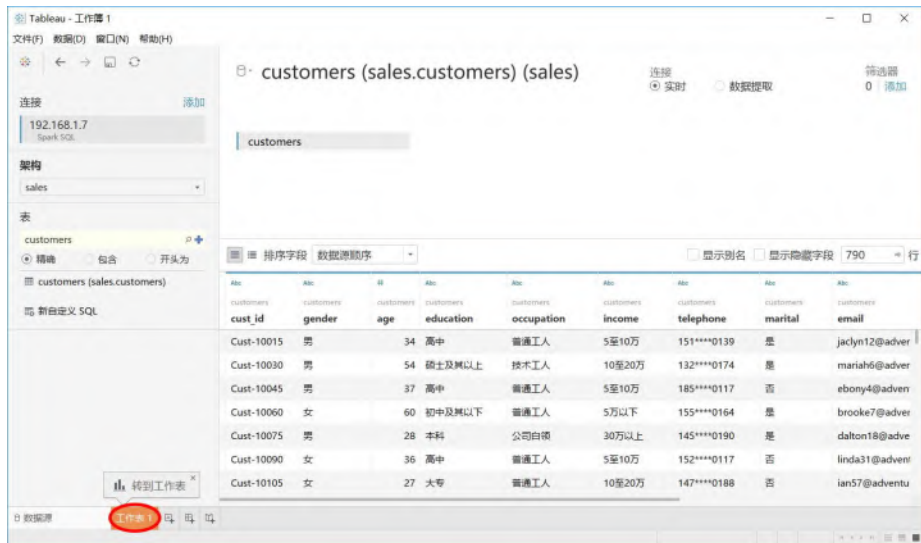


图3-15 对查看需要连接的表

步骤 07

将occupation（客户职业类型）字段拖曳到“行”功能区，将“总和（记录数）”字段拖曳到“列”功能区，注意度量类型选择“总和”，其默认计算类型也是“总和”，如图3-16所示。

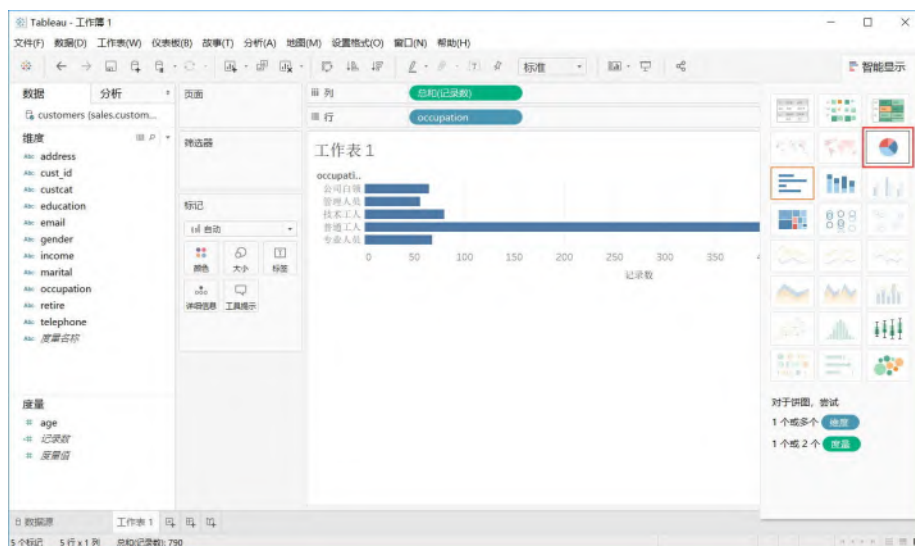


图3-16 设置可视化字段

步骤 08

对视图进行适当的调整，例如颜色、标签、标题等，最终的效果如图3-17所示。

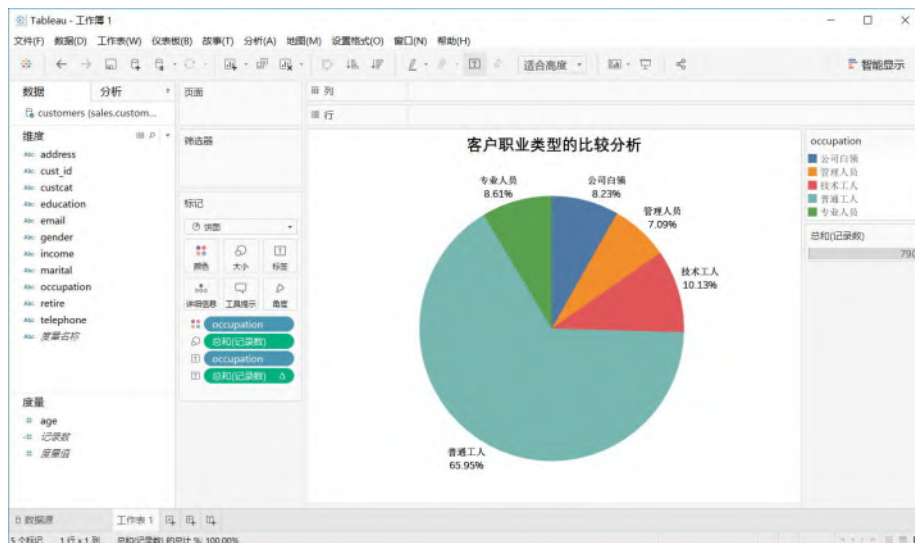


图3-17 调整可视化视图

从上面制作的饼图可以看出：该商品的客户类型中，普通工人占到了65.95%，其次技术工人占10.13%，专业人员占8.61%，公司白领占8.23%，管理人员占7.09%，职业类型存在较大的差异。

3.2 Zeppelin

3.2.1 Zeppelin简介

Apache Zeppelin是基于Web的笔记本，支持SQL、Scala等数据驱动的交互式数据分析和协作文档，通过JDBC可以支持PostgreSQL、MySQL、Amazon Redshift、Hive等数据库。在部署方面，支持单个用户，也支持多用户，可以满足数据摄取、数据发现、数据分析、数据可视化与协作等。

Zeppelin中核心的概念是Interpreter，Interpreter是一个插件，允许用户使用一个指定的语言或数据处理器。每一个Interpreter都属于一个InterpreterGroup，同一个InterpreterGroup的Interpreters可以相互引用，例如SparkSqlInterpreter可以引用SparkInterpreter以获取SparkContext，因为它们属于同一个InterpreterGroup。当前已经实现的Interpreter有Spark解释器、Python解释器、SparkSQL解释器、JDBC、Markdown和Shell等。

Interpreter接口中有3个重要的方法：Open、Close、Interpret，另外还有Cancel、getProgress、Completion等方法。

- Open：初始化部分，只会调用一次。
- Close：关闭释放资源的接口，只会调用一次。
- Interpret：运行一段代码并返回结果，同步执行方式。
- Cancel：可选的接口，用于结束interpret方法。
- getProgress：用于获取interpret的百分比进度。

- Completion: 基于游标位置获取结束列表, 通过这个接口可以实现自动结束。

此外, Zeppelin也存在一定的不足, 优点和缺点主要如下:

优点:

- (1) 提供restful和webSocket两种接口方式。
- (2) 使用Spark解释器, 进行接口编程, 可以自己操作SparkContext。
- (3) 扩展性好, 可以方便地自定义解释器。

缺点:

- (1) 没有提供JAR包的方式运行Spark任务。
- (2) 只有同步的方式运行程序, 等待时间较长。

3.2.2 Zeppelin连接Hive

利用Zeppelin可以通过Hive来操作Hadoop集群中的数据。下面将具体介绍操作步骤。

1. 启动集群和Spark相关进程

下面需要启动集群和Spark的相关进程, 主要步骤如下:

- (1) 启动Hadoop:

```
/home/dong/hadoop-2.5.2/sbin/start-all.sh
```

(2) 后台运行Hive:

```
nohup hive --service metastore > metastore.log 2>&1 &
```

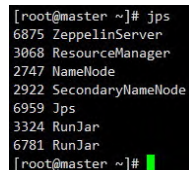
(3) 启动Hive的hiveserver2:

```
hive --service hiveserver2 &
```

(4) 启动Zeppelin:

```
/home/dong/zeppelin-0.7.3-bin-all/bin/zeppelin-daemon.sh  
start
```

(5) 查看启动的进程，输入jps，确认已经启动了如图3-18所示的7个进程。



```
[root@master ~]# jps  
6875 ZeppelinServer  
3068 ResourceManager  
2747 NameNode  
2922 SecondaryNameNode  
6959 Jps  
3324 RunJar  
6781 RunJar  
[root@master ~]#
```

图3-18 查看启动进程

2. 配置Hive解释器

首先，在浏览器中输入`http://192.168.1.7:7080/#/`，建议使用谷歌浏览器，进入集群的Zeppelin登录页面，如图3-19所示。

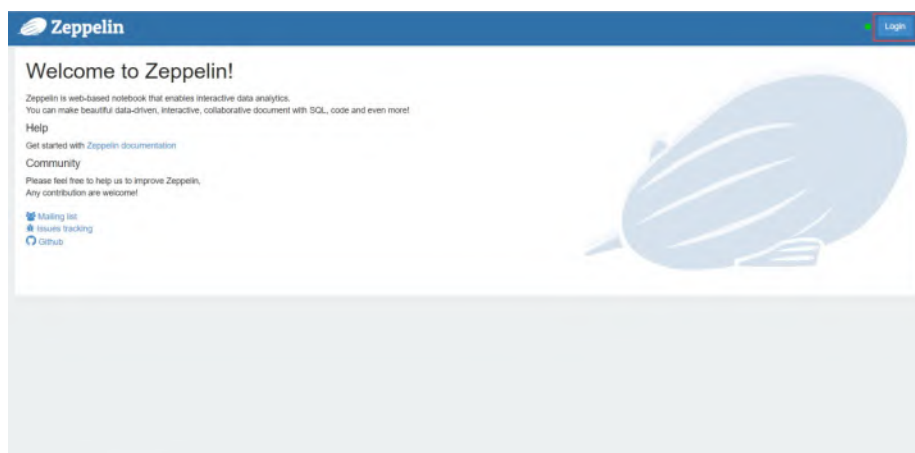


图3-19 登录页面

然后，单击界面右上方的Login按钮，在弹出的Login页面输入账户名和密码，如图3-20所示。

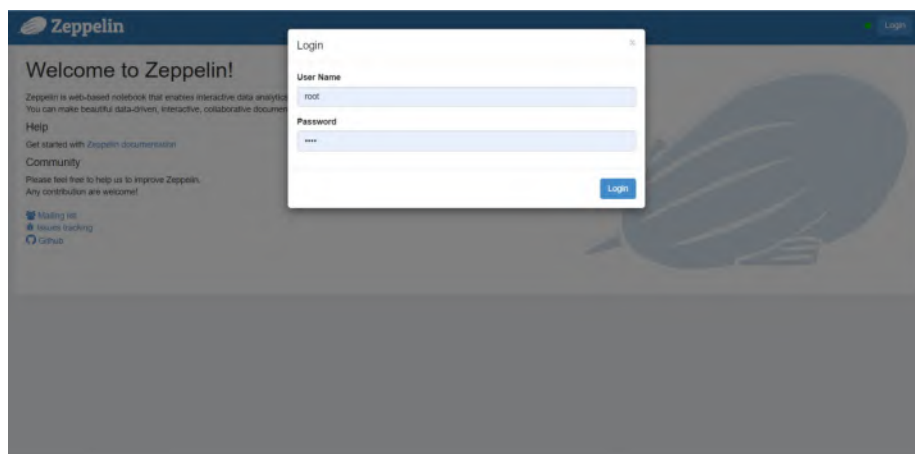


图3-20 输入账户名和密码

登录后的效果如图3-21所示，通过单击账户下的Interpreter选项，进入添加Hive的解释器界面。

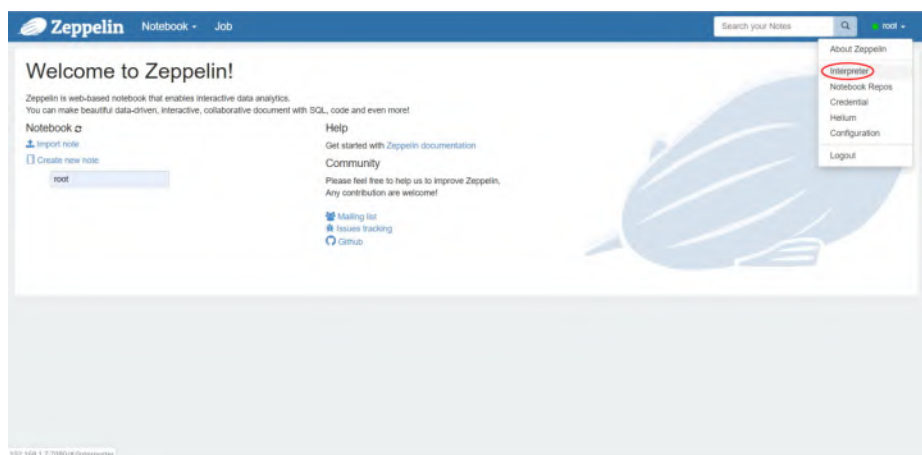


图3-21 添加解释器

单击界面右上方的Create按钮，如图3-22所示。

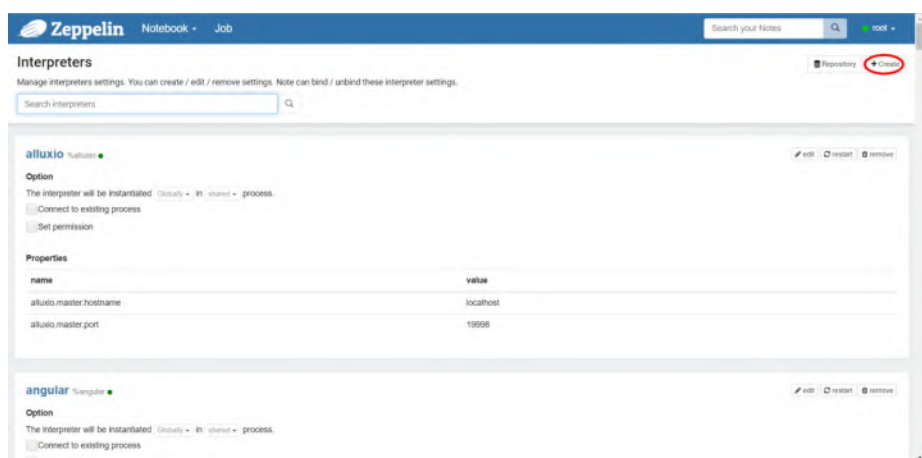


图3-22 创建解释器

在创建新的解释器页面，在Interpreter Name中输入解释器的名称，这里可以输入hive，Interpreter group需要选择jdbc。下面修改JDBC解释器的配置和添加相应的依赖，未配置前的参数设置如图3-23所示。

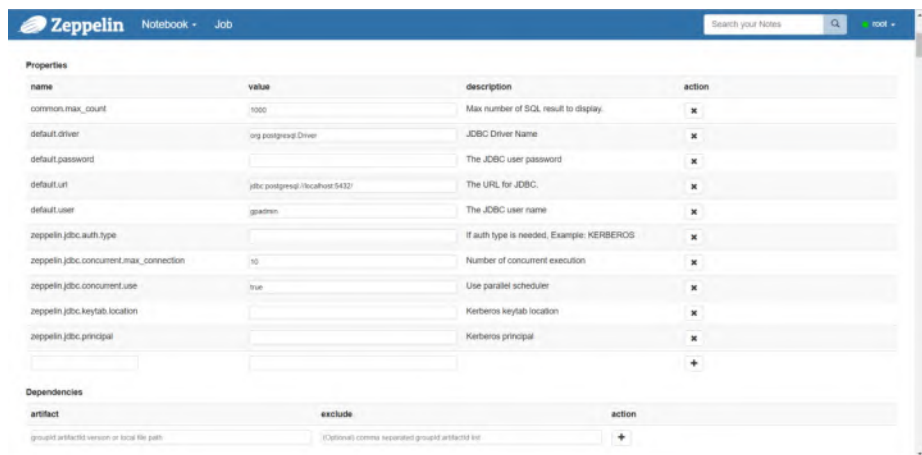


图3-23 未配置前的参数

需要根据实际Hadoop集群的配置修改JDBC解释器的配置，尤其要注意default.url，否则无法正常连接集群，配置后的参数列表如图3-24所示。

Properties	
name	value
common.max_count	1000
default.driver	org.apache.hive.jdbc.HiveDriver
default.password	root
default.url	jdbc:hive2://192.168.1.7:10000/sales;auth=noSasl
default.user	root
zeppelin.interpreter.localRepo	/home/dong/zeppelin-0.7.3-bin-all/local-repo/2EHRJ2ZVF
zeppelin.interpreter.output.limit	102400
zeppelin.jdbc.auth.type	
zeppelin.jdbc.concurrent.max_connection	10
zeppelin.jdbc.concurrent.use	true
zeppelin.jdbc.keytab.location	
zeppelin.jdbc.principal	

图3-24 配置后的参数

此外，还需要添加Hive、Hadoop和MySQL的相应依赖，其中Hive和Hadoop必须要与集群的版本一致，MySQL可以不一致，如图3-25所示。配置结束后，单击界面左下方的Save按钮保存设置。

Dependencies	
artifact	exclude
org.apache.hive:hive-jdbc:1.2.2	
org.apache.hadoop:hadoop-common:2.5.2	
mysql:mysql-connector-java:5.1.26	

图3-25 配置相应依赖

注意

如果报org.apache.hive.jdbc.HiveDriver的错误信息，就需要复制Hive的lib文件夹下的以hive开头的JAR包到Zeppelin的lib文件夹下。

3. 案例：2019年A公司的股票趋势分析

首先需要在Zeppelin的开始页面单击Create new note链接，进入创建笔记的过程，如图3-26所示。

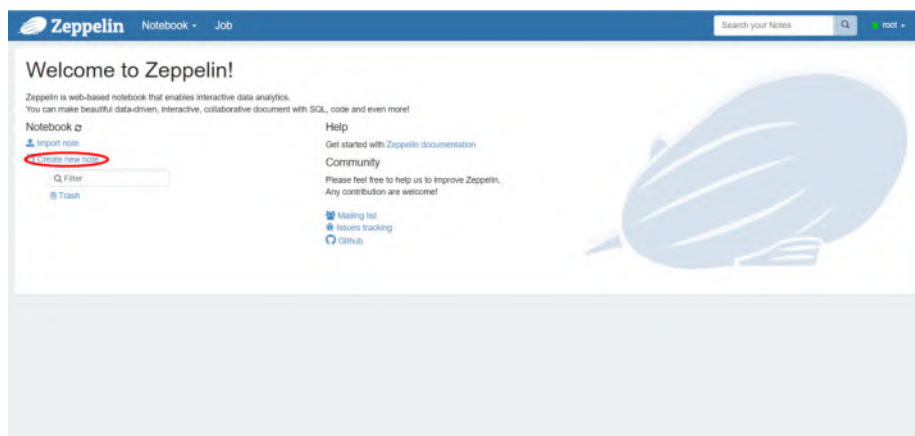


图3-26 创建笔记

在Note Name中输入名称，这里我们可以输入“2019年某公司的股票趋势分析”，然后在解释器的下拉框中选择“hive”，再单击Create Note按钮，如图3-27所示。

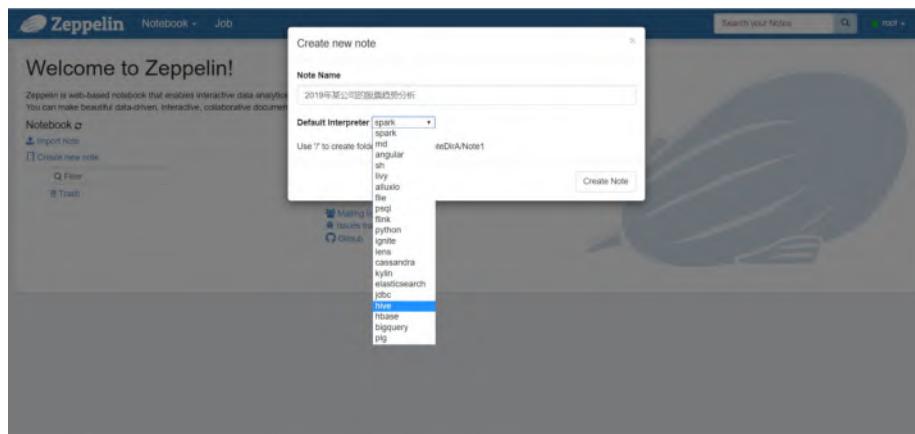


图3-27 选择解释器

在空白区域输入SQL代码，注意前面需要添加“%hive”，然后单击“执行”按钮，或者使用Shift+Enter快捷键，如图3-28所示。

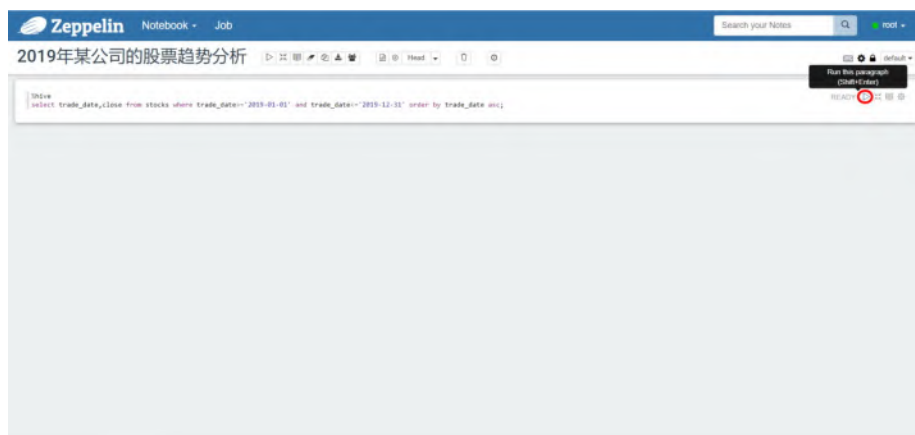
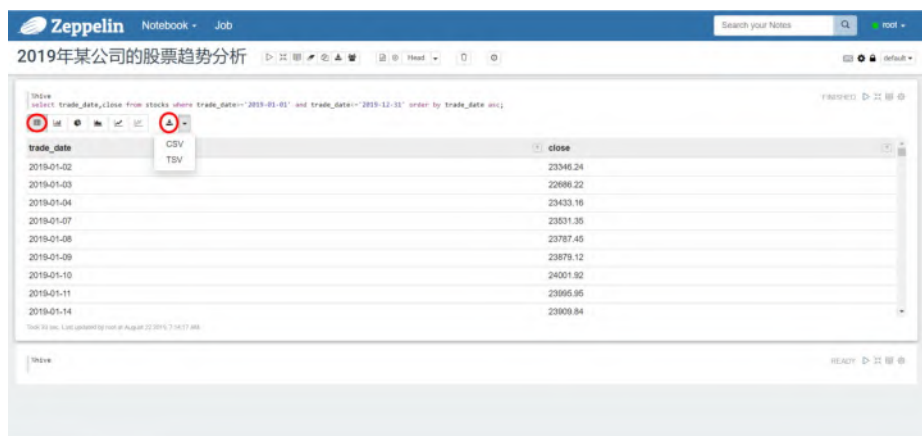


图3-28 执行SQL语句

SQL语句的查询速度与集群的配置有关，查询结果默认使用表格形式显示，我们还可以将结果导出到本地再进行更加深入的分析，如图3-29所示。



2019年某公司的股票趋势分析

```

show
select trade_date,close from stocks where trade_date>='2019-01-01' and trade_date<='2019-12-31' order by trade_date asc;

```

trade_date	close
2019-01-02	23346.24
2019-01-03	22686.22
2019-01-04	23433.16
2019-01-07	23531.35
2019-01-08	23787.45
2019-01-09	23879.12
2019-01-10	24001.82
2019-01-11	23995.95
2019-01-14	23909.84

图3-29 导出运行结果

Zeppelin的可视化视图是一些常见的条形图、饼图、面积图和折线图等，由于股票价格数据比较适合使用折线图展示，因此需要单击Line Chart按钮，效果如图3-30所示。

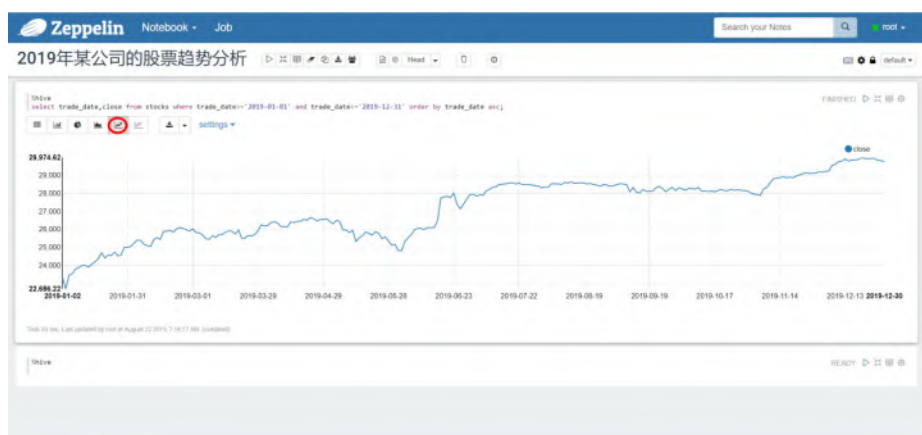


图3-30 查看可视化视图

此外，在实际工作中，可视化视图中经常需要进行分类展示，Zeppelin有专门的设置选项，单击settings按钮，将需要分组的字段拖曳到Groups框中即可，如图3-31所示。由于这里没有分类字段，因此该选项不需要设置。

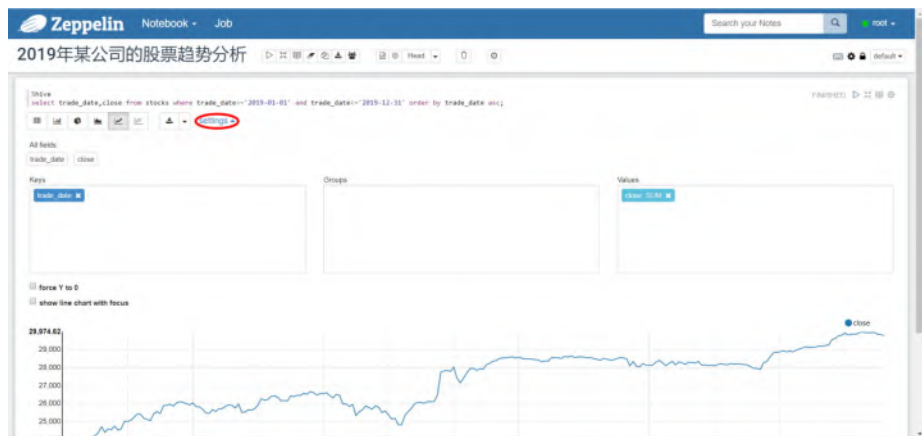


图3-31 分类展示可视化视图

从股价的走势圖可以看出：在2019年，該公司的股票價格基本呈現上升趨勢，下半年的股價明顯高於上半年的股價，這可能與公司在2019年6月份宣布收購一家新公司有關。

3.2.3 Zeppelin连接Spark

利用Zeppelin可以通过Spark来操作Hadoop集群中的数据。下面将具体介绍。

1. 启动集群和Spark相关进程

下面需要启动集群和Hive的相关进程，主要步骤如下：

(1) 启动Hadoop：

```
/home/dong/hadoop-2.5.2/sbin/start-all.sh
```

(2) 启动Spark：

```
/home/dong/spark-1.4.0-bin-hadoop2.4/sbin/start-all.sh
```

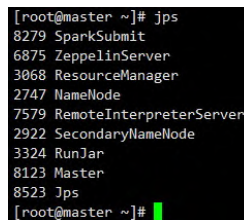
(3) 后台运行Hive:

```
nohup hive --service metastore > metastore.log 2>&1 &
```

(4) 启动Zeppelin:

```
/home/dong/zeppelin-0.7.3-bin-all/bin/zeppelin-daemon.sh  
start
```

(5) 查看启动的进程，输入jps，确认已经启动了如图3-32所示的9个进程。



```
[root@master ~]# jps  
8279 SparkSubmit  
6875 ZeppelinServer  
3068 ResourceManager  
2747 NameNode  
7579 RemoteInterpreterServer  
2922 SecondaryNameNode  
3324 RunJar  
8123 Master  
8523 Jps  
[root@master ~]#
```

图3-32 查看启动进程

2. 配置Spark解释器

配置Spark解释器的过程与配置Hive解释器的过程基本相同，也是通过单击账户下的Interpreter选项添加Spark解释器，如图3-33所示。

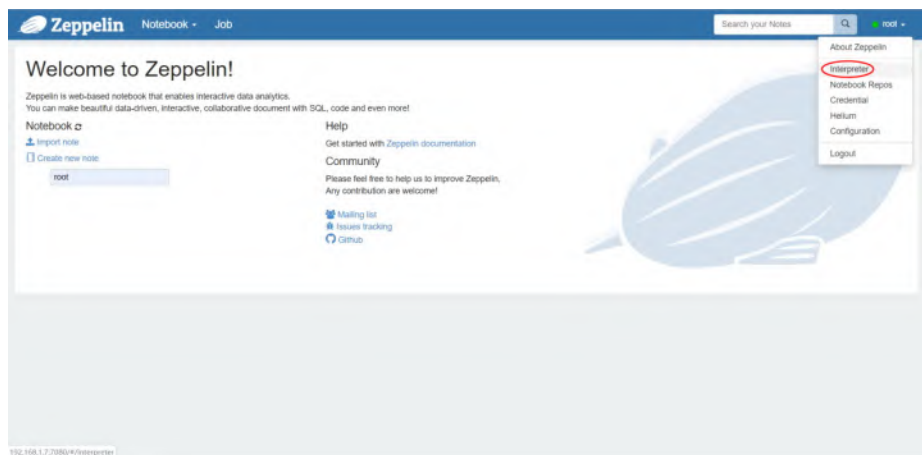


图3-33 添加解释器

单击界面右上方的Create按钮，如图3-34所示。

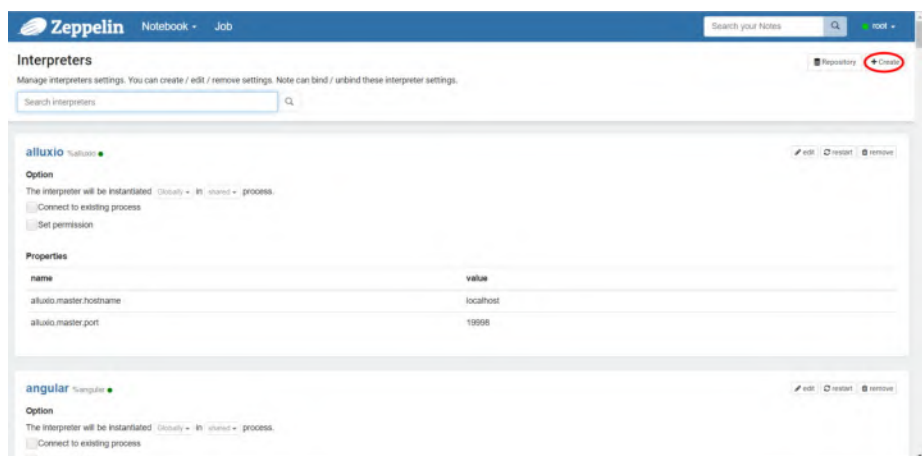
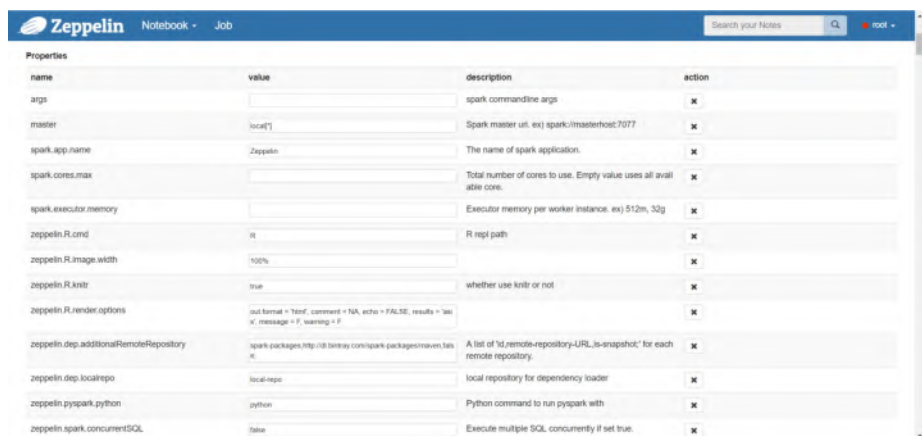


图3-34 创建解释器

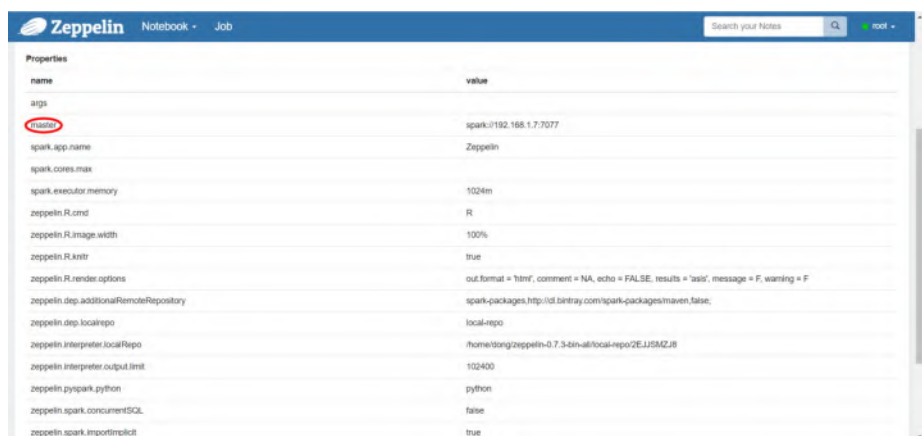
在创建新的解释器页面，在Interpreter Name中输入解释器的名称，这里可以输入spark，Interpreter group需要选择spark。然后修改JDBC解释器配置选项，没有配置之前的参数如图3-35所示。



name	value	description	action
args		spark commandline args	✕
master	local[*]	Spark master url. ex) spark://masterhost:7077	✕
spark.app.name	Zeppelin	The name of spark application.	✕
spark.cores.max		Total number of cores to use. Empty value uses all available core.	✕
spark.executor.memory		Executor memory per worker instance. ex) 512m, 32g	✕
zeppelin.R.cmd	R	R repl path	✕
zeppelin.R.image.width	100%		✕
zeppelin.R.knitr	true	whether use knitr or not	✕
zeppelin.R.render.options	out.format = 'html', comment = NA, echo = FALSE, results = 'asis', message = F, warning = F		✕
zeppelin.dep.additionalRemoteRepository	spark-packages,http://08.bintay.com/spark-packages/maven,fast	A list of 'id,remote-repository-URL,js-snapshot,' for each remote repository.	✕
zeppelin.dep.localrepo	local-repo	local repository for dependency loader	✕
zeppelin.pyspark.python	python	Python command to run pyspark with	✕
zeppelin.spark.concurrentSQL	false	Execute multiple SQL concurrently if set true.	✕

图3-35 未配置前的参数

需要根据实际Hadoop集群的配置修改JDBC解释器的配置，尤其要注意master选项的设置，否则无法正常连接集群，配置后的参数列表如图3-36所示。



name	value
args	
master	spark://192.168.1.7:7077
spark.app.name	Zeppelin
spark.cores.max	
spark.executor.memory	1024m
zeppelin.R.cmd	R
zeppelin.R.image.width	100%
zeppelin.R.knitr	true
zeppelin.R.render.options	out.format = 'html', comment = NA, echo = FALSE, results = 'asis', message = F, warning = F
zeppelin.dep.additionalRemoteRepository	spark-packages,http://08.bintay.com/spark-packages/maven,false;
zeppelin.dep.localrepo	local-repo
zeppelin.interpreter.localRepo	home/tong/zeppelin-0.7.3-bin-all/local-repo/ZEJUSAZJB
zeppelin.interpreter.output.limit	102400
zeppelin.pyspark.python	python
zeppelin.spark.concurrentSQL	false
zeppelin.spark.importImplicit	true

图3-36 配置后的参数

3. 案例：2019年A公司的股票交易量分析

首先需要在Zeppelin的开始页面单击Create new note链接，然后进入创建笔记界面，如图3-37所示。

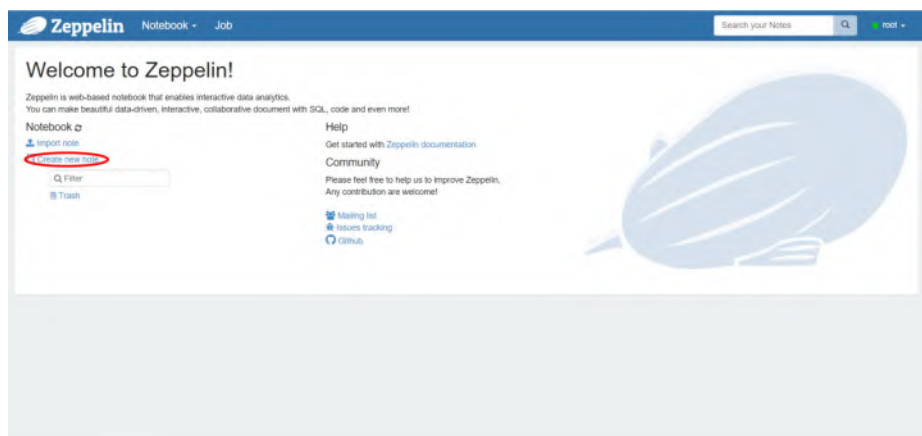


图3-37 创建笔记

在Note Name中输入名称，这里可以输入“2019年某公司的股票交易量分析”，然后在解释器的下拉框中选择spark，再单击Create Note按钮，如图3-38所示。

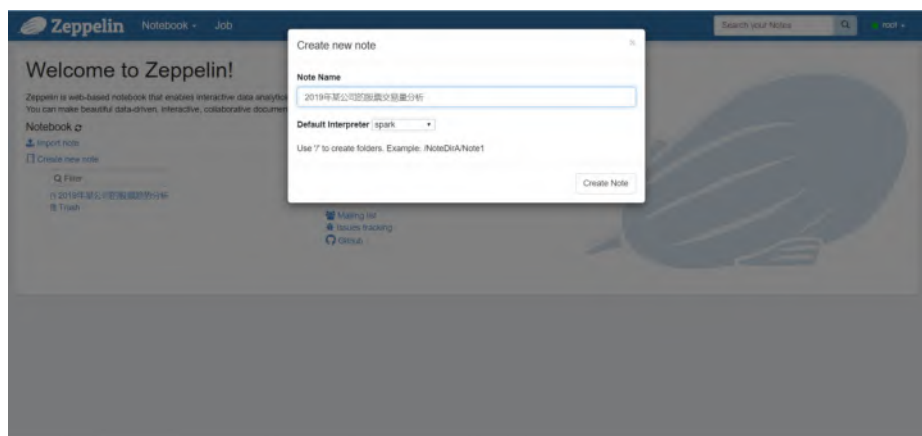


图3-38 选择解释器

在空白区域输入SQL代码，注意前面需要添加“%spark.sql”，然后单击“执行”按钮，或者使用Shift+Enter快捷键。SQL语句的查询速度与集群的配置有关，查询结果默认使用表格形式显示，我们还可以将结果导出到本地再进行更加深入的分析，如图3-39所示。

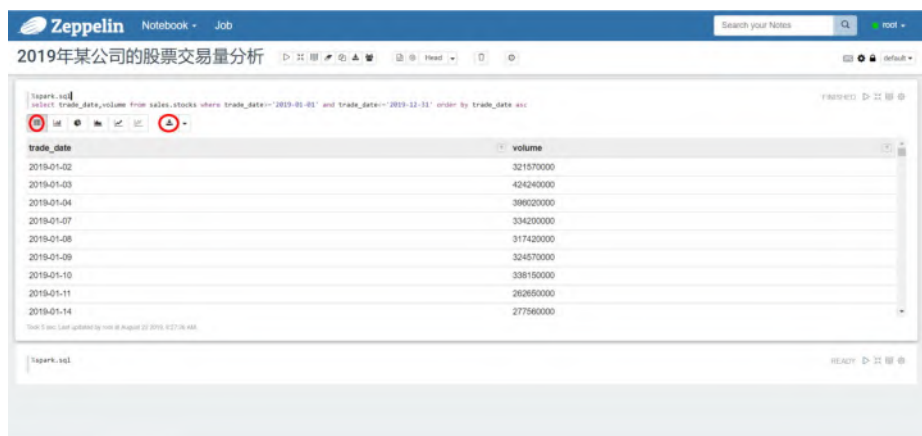


图3-39 导出运行结果

Zeppelin的可视化视图都是一些比较常见的图形，如条形图、饼图、面积图、折线图和散点图等，由于股票价格数据比较适合使用折线图展示，因此需要单击Line Chart按钮，效果如图3-40所示。

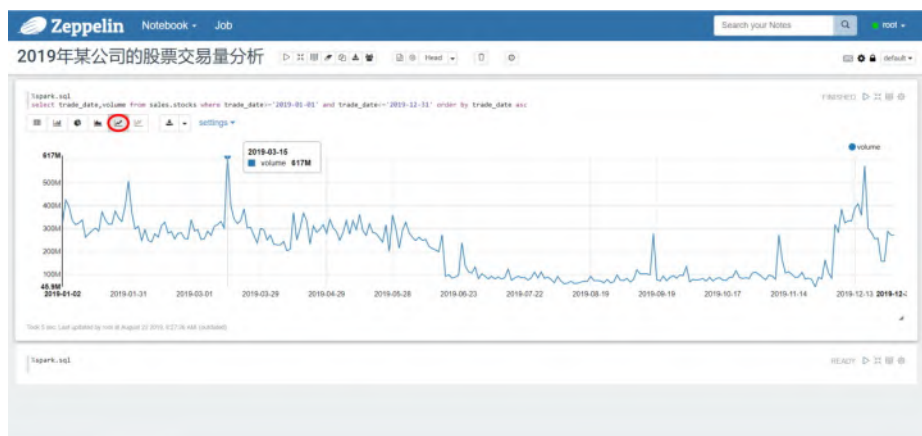


图3-40 查看可视化视图

从股票交易量的走势图可以看出：在2019年，除了12月份外，该公司的股票交易量基本呈现下降趋势，下半年的交易量明显低于上半年的交易量，这与前面的股票收盘价走势分析刚好相反，具体原因还需要结合公司的经营状况进行深入的分析。

3.3 Python在数据可视化中的应用

3.3.1 Python简介

Python是一门简单易学且功能强大的编程语言。它拥有高效的高级数据结构，并且能够用简单而又高效的方式进行面向对象编程。Python拥有优雅的语法和动态类型，再结合它的解释性，使其在大多数平台的许多领域成为编写脚本或开发应用程序的理想语言。

1989年，为了度过圣诞假期，吉多·范罗苏姆（Guido van Rossum）开始编写Python语言编译器。Python这个名字来自Guido喜爱的电视连续剧《Monty Python's Flying Circus》。他希望新的语言Python能够满足他在C和Shell之间创建全功能、易学、可扩展的语言的愿景。

2018年8月，IEEE Spectrum发布了第5届顶级编程语言交互排行榜：Python卫冕成功。2018年，Python与第二名C++拉开差距（两种语言的交互性排名差距为1.6），C和Java则分据第三和第四，如图3-41所示。

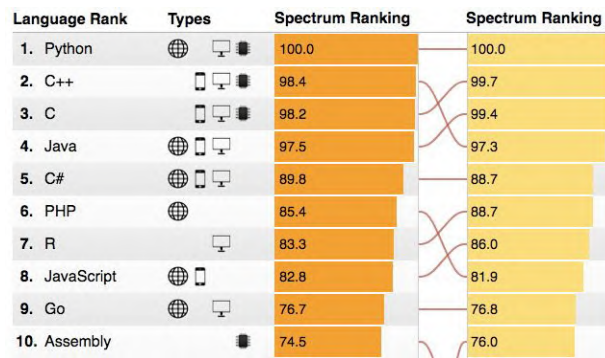


图3-41 编程语言排行榜

虽然Python易于使用，但它却是一门完整的编程语言，与Shell脚本或批处理文件相比，它为编写大型程序提供了更多的结构和支持。另一方面，Python提供了比C更多的错误检查，并且作为一门高级语

言，它支持高级的数据结构类型，例如灵活的数组和字典。Python语言的主要特点如下：

- 语法简单：Python的语法非常优雅，甚至没有像其他语言的大括号、分号等特殊符号，代表了一种极简主义的设计思想，阅读Python程序像是在读英语。
- 编程易学：Python入手非常快，学习曲线非常低，可以直接通过命令行交互环境来学习Python编程。
- 开源免费：Python具有功能强大的库，而且由于Python的开源特性，第三方库也非常多，例如在Web开发、爬虫、科学计算等方面。
- 跨平台：Python的代码几乎不用修改就可以运行在各类平台上。

此外，Python可以用于数据分析、搭建网站、开发游戏、自动化测试等，其在数据分析中的应用包括以下三个方面：

- Python编程实现：数据提取存储、数据预处理、数据清洗。
- Python网络爬虫：从互联网获取数据。
- 第三方库的使用：数据处理、数据可视化、建立模型。

本书的研究重点是Python的数据可视化，包括Matplotlib与Pyecharts两个第三方库，但是偶尔也会使用一些其他库，如Pandas、NumPy等。

3.3.2 Python连接Hive

Python借助impyla包可以连接到Hadoop集群的Hive。下面具体介绍其步骤。

首先需要启动Hadoop集群和Hive的相关进程，主要步骤如下：

(1) 启动Hadoop:

```
/home/dong/hadoop-2.5.2/sbin/start-all.sh
```

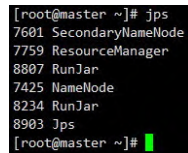
(2) 后台运行Hive:

```
nohup hive --service metastore > metastore.log 2>&1 &
```

(3) 启动Hive的hiveserver2:

```
hive --service hiveserver2 &
```

(4) 查看启动的进程，输入jps，确认已经启动了如图3-42所示的7个进程。



```
[root@master ~]# jps
7601 SecondaryNameNode
7759 ResourceManager
8807 RunJar
7425 NameNode
8234 RunJar
8903 RunJar
8903 Jps
[root@master ~]#
```

图3-42 查看启动的进程

然后安装3个包：impyla、thriftpy和thriftpy2，如果安装包的时候报错，就需要下载离线安装包再安装，注意要和Python版本相匹配。

这样就能成功地安装PyHive了，测试代码如下：

```

from impala.dbapi import connect
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cur=conn.cursor()
cur.execute('select * from orders where order_date="2019-11-11"')
print(cur.fetchall())

```

测试程序执行后的结果如图3-43所示，展示该产品在2019年11月11日的订单信息，注意我们采用的是样本数据集，企业的真实订单量应该是很大的。

```

[13]: from impala.dbapi import connect
      conn = connect(host='192.168.1.7', port=10000, database='sales',auth_mechanism='NOSASL',user='root')
      cur = conn.cursor()
      cur.execute('select * from orders where order_date="2019-11-11"')
      print(cur.fetchall())

[('CN-2019-102984', '2019-11-11', '众兴店', '其它', '2019-11-15', 3, 'Cust-18160', '邱梦', '消费者', '郸城', '河南', '中南', 'Prod-10003132', 'Jiffy_搭扣信封_每套_50_个', '办公用品', '信封', 119.28, 3, 0.0, 31.92, '王倩倩', 1, '2019'), ('CN-2019-102985', '2019-11-11', '庐江路', '微信', '2019-11-15', 6, 'Cust-14335', '马惠英', '公司', '南宁', '广西', '中南', 'Prod-10003039', '爱普生_收据打印机_耐用', '技术', '设备', 2186.8, 4, 0.0, 612.08, '王倩倩', 0, '2019'), ('CN-2019-102986', '2019-11-11', '众兴店', '其它', '2019-11-16', 6, 'Cust-19510', '钟庆缘', '小型企业', '杭州', '浙江', '华东', 'Prod-10002744', 'Eldon_盖_单宽度', '办公用品', '收纳具', 144.06, 3, 0.0, 17.22, '杨洪光', 1, '2019'), ('CN-2019-102987', '2019-11-11', '人民路店', '其它', '2019-11-16', 6, 'Cust-19510', '钟庆缘', '小型企业', '杭州', '浙江', '华东', 'Prod-10003981', 'Green_Bar_羊皮纸_优质', '办公用品', '纸张', 155.68, 2, 0.0, 24.64, '杨洪光', 0, '2019'), ('CN-2019-102988', '2019-11-11', '众兴店', '支付宝', '2019-11-17', 6, 'Cust-18910', '贾柏', '小型企业', '塘沽', '天津', '华北', 'Prod-1000334', 'Hewlett_无线传真机_数字化', '技术', '复印机', 8862.7, 5, 0.0, 1506.4, '张怡莲', 1, '2019'), ('CN-2019-102989', '2019-11-11', '人民路店', '信用卡', '2019-11-17', 6, 'Cust-20710', '姜松', '小型企业', '上海', '上海', '华东', 'Prod-10003358', 'Advantus_灯泡_一包多件', '家具', '用具', 375.76, 4, 0.0, 75.04, '杨洪光', 1, '2019'), ('CN-2019-102990', '2019-11-11', '众兴店', '信用卡', '2019-11-17', 6, 'Cust-19450', '冯巧', '消费者', '济南', '山东', '华东', 'Prod-10000292', 'Wilson_Jones_装订机盖_实惠', '办公用品', '装订机', 104.44, 2, 0.0, 34.44, '杨洪光', 0, '2019'), ('CN-2019-102991', '2019-11-11', '定远路店', '其它', '2019-11-17', 6, 'Cust-19450', '冯巧', '消费者', '济南', '山东', '华东', 'Prod-10002438', 'Hon_圆桌_白色', '家具', '桌子', 4352.04, 3, 0.4, -72.66, '杨洪光', 0, '2019'), ('CN-2019-102992', '2019-11-11', '临泉路', '信用卡', '2019-11-17', 6, 'Cust-19450', '冯巧', '消费者', '济南', '山东', '华东', 'Prod-10003466', 'Cuisinart_烤面包机_银色', '办公用品', '器具', 1011.92, 4, 0.0, 343.84, '杨洪光', 0, '2019'), ('CN-2019-102993', '2019-11-11', '人民路店', '微信', '2019-11-17', 6, 'Cust-19450', '冯巧', '消费者', '济南', '山东', '华东', 'Prod-10003809', 'Office_Star_摇椅_红色', '家具', '椅子', 197.274, 3, 0.0, 118.02, '杨洪光', 0, '2019'), ('CN-2019-102994', '2019-11-11', '众兴店', '信用卡', '2019-11-17', 6, 'Cust-18910', '贾柏', '小型企业', '塘沽', '天津', '华北', 'Prod-10003289', 'Tenex_隔板_单宽度', '办公用品', '收纳具', 510.44, 2, 0.0, 127.4, '张怡莲', 0, '2019'), ('CN-2019-102995', '2019-11-11', '燎原店', '信用卡', '2019-11-17', 6, 'Cust-20710', '姜松', '小型企业', '上海', '上海', '华东', 'Prod-10000670', 'Office_Star_凳子_可调', '家具', '椅子', 6586.72, 8, 0.0, 1777.44, '杨洪光', 0, '2019')]

```

图3-43 订单样本数据

3.3.3 Python可视化案例

为了让大家更好地理解本书后面的内容，这里提前举一个案例，使用的是公司的订单（orders）表，该表位于我们搭建的Hadoop集群中，而且是分区表，分区字段是年份dt。我们需要展示2019年各个区域的销售情况。下面简单介绍其步骤。

步骤 01

通过使用连接Hive的图形界面工具查看orders表中的相关字段信息，并测试程序中的SQL语句是否正常执行，如图3-44所示。

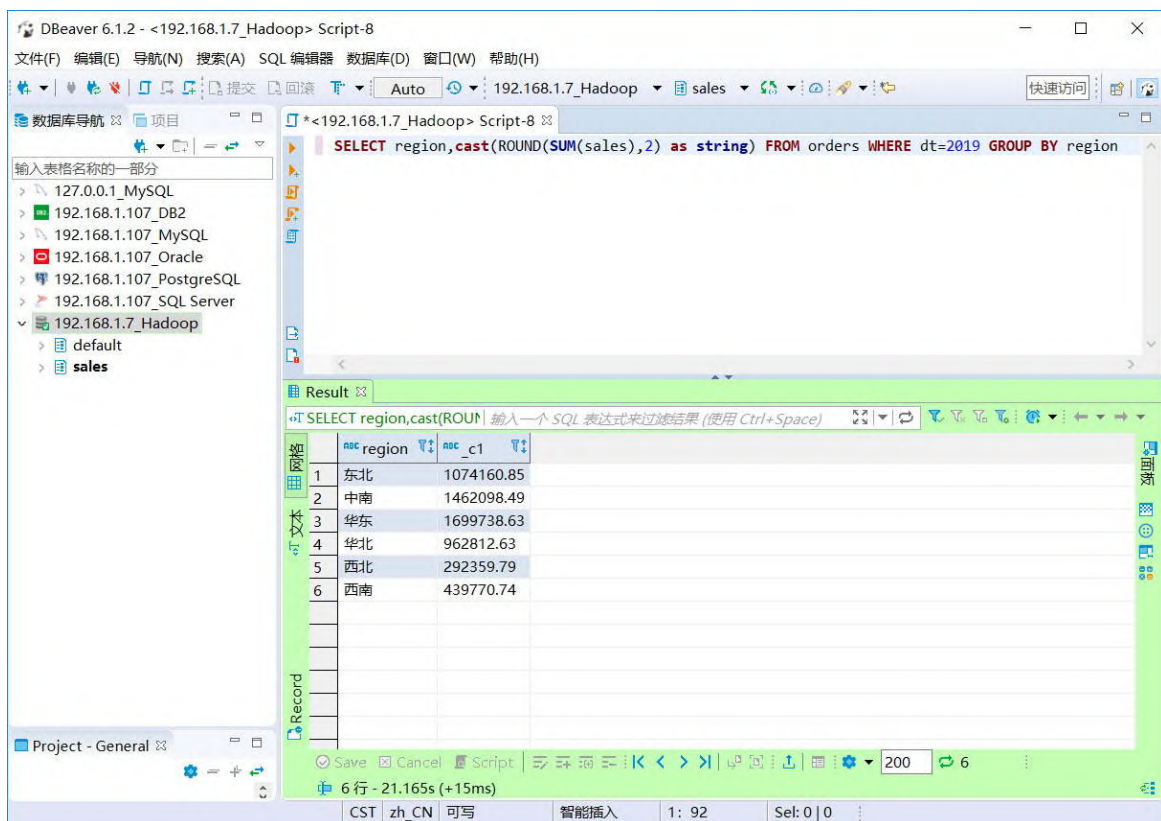


图3-44 区域销售额统计

步骤 02

利用Python中的相关包进行程序的编写，这里我们使用的是pyecharts包，Python可视化程序如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Page, Pie
from impala.dbapi import connect
```

```

#连接Hadoop数据库
v1=[]
v2=[]
v3=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism=

'NOSASL',user='root')
cursor=conn.cursor()
#读取Hadoop数据
sql_num="SELECT region,cast(ROUND(SUM(sales),2) as string)
FROM orders WHERE dt=2019 GROUP BY region"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])

#画饼图
def pie() -> Pie:
    c=(
        Pie()
        .add("",[list(z) for z in zip(v1, v2)],center=
["50%", "55%"],)
        .set_colors(["blue", "green", "purple", "red",
"silver", "orange"]) #设置颜色
        .set_global_opts(
            title_opts=opts.TitleOpts(title="利润额比较分析",
subtitle="2019年企业经营状况"),

legend_opts=opts.LegendOpts(orient="horizontal",
pos_top="5%", pos_left="30%"),
            #设置位置horizontal水平vertical垂直
            toolbox_opts=opts.ToolboxOpts()
        )

.set_series_opts(label_opts=opts.LabelOpts(formatter="{b}:
{c}"))
    )
    return c

#第一次渲染时调用load_javascript文件
pie().load_javascript()

#展示数据可视化图表
pie().render_notebook()

```

步骤 03

执行上面的程序，可以产生比较美观的图形，如图3-45所示。当然该图形还可以进一步美化，例如大小、颜色、标签等，这些内容会在本书后面的章节进一步说明，这里就不深入介绍了。

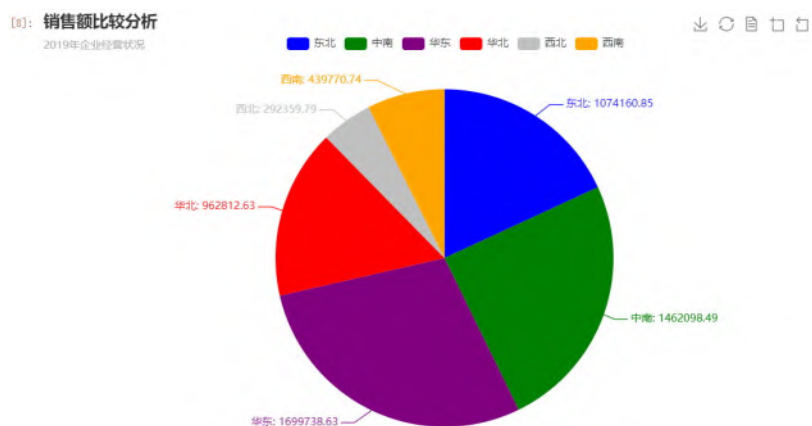


图3-45 区域销售额分析

从图形可以看出：在2019年全年，该产品在华东地区的销售额最大，为1 074 160.85元，其次是中南地区，为1 462 098.49元，其中销售额最少的是西北地区，为292 359.79元。

第4章

Python可视化编程基础



本章我们将介绍Python可视化编程的基础知识，包括软件的安装、如何搭建代码开发环境以及如何连接各类常用的数据源。

4.1 Python环境安装

目前，Python分为2.X和3.X两个版本。Python的3.X版本是一个较大的升级版本，为了不带入过多的累赘，在设计的时候没有考虑向下兼容。

2018年3月，Python开发者宣布Python 2.7将于2020年1月1日终止支持。用户如果想要在这个日期之后继续得到与Python 2.7有关的支持，就需要付费给商业供应商。

工欲善其事，必先利其器。Python的学习过程少不了代码开发环境，代码开发环境可以帮助开发者加快开发速度，提高效率。Python的开发环境较多，如Anaconda、PyCharm、Eclipse等。

下面介绍两种比较常用的Python开发环境：Anaconda和PyCharm。注意，本书中使用的环境是基于Python 3.7.4的Anaconda（截至2019年11月份的最新版本）。这里推荐读者使用Anaconda，尤其是对于刚刚入门的Python初学者，这是由于它已经内置了很多Python常用的第三方包，我们不必再考虑令人烦恼的兼容性问题。

4.1.1 Anaconda

Anaconda是一个基于Python的数据处理和科学计算平台，内置了许多非常有用的第三方库，装上Anaconda，就相当于把Python和一些常用的库（如NumPy、Pandas、Matplotlib等）自动安装好了，使得安装比常规Python安装要简单容易。

如果选择非集成环境Python，那么还需要使用`pip install`命令一个一个地安装各种库，安装起来比较复杂，尤其是对于初学者来说，这个过程是非常痛苦的，此外还需要考虑兼容性，需要去Python官网（<https://www.Python.org/downloads/windows/>）选择对应的版本。

此外，由于每个人的计算机系统的开发环境存在差异，可能会遇到一些无法解决的问题。下面是一个用户经常遇到的问题：如果不使用Anaconda，安装好Python 3.8后，在命令提示符中输入`pip install jupyterlab`，可以正常下载包，但是不能正常安装，会报以下的错误信息：`distutils.errors.DistutilsError: Setup script exited with error: Microsoft Visual C++ 14.0 is required. Get it with "Microsoft Visual C++ Build Tools": https://visualstudio.microsoft.com/downloads/`，虽然下载和安装了指定的安装包，并且安装过程较长，但是还是没有解决问题。

Anaconda是专注于数据分析的Python发行版本，包含Conda、Python等190多个科学包及其依赖项。Anaconda的优点总结起来就8个

字：省时省心、分析利器。

- **省时省心：** Anaconda通过管理工具包、开发环境、Python版本大大简化了工作流程，不仅可以方便地安装、更新、卸载工具包，而且安装时能自动安装相应的依赖包，同时还能使用不同的虚拟环境隔离不同要求的项目。
- **分析利器：** 在Anaconda官网中是这么宣传自己的：适用于企业级大数据分析的Python工具。其包含720多个数据科学相关的开源包，在数据可视化、机器学习、深度学习等多方面都有涉及，不仅可以进行数据分析，甚至可以用在大数据和人工智能领域。

Anaconda的安装过程比较简单，可以选择默认安装或者自定义安装，为了避免配置环境和安装pip的麻烦，建议勾选添加环境变量和安装pip选项。下面介绍具体的安装步骤。

步骤 01

进 入 官 网

(<https://www.anaconda.com/download/#windows>) 下载对应的版本，这里选择的是Windows 64bit，如图4-1所示。

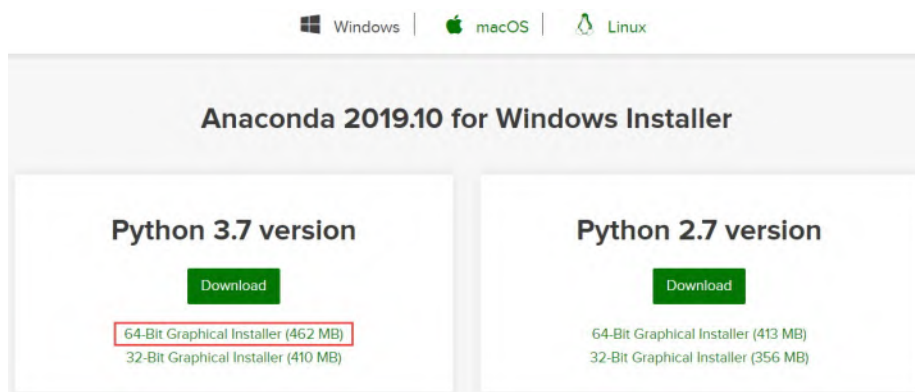


图4-1 下载Anaconda

步骤 02

以管理员身份运行下载好的Anaconda3-2019.10-Windows-x86_64.exe文件，在Welcome to Anaconda3 2019.10(64-bit)Setup界面单击Next按钮，在License Agreement界面单击I Agree按钮，在Select Installation Type界面单击Next按钮，如图4-2~图4-4所示。然后在Choose Install Location界面单击Browse按钮，选择安装目录，选择好之后单击Next按钮，如图4-5所示。接下来在Advanced Installation Options界面单击Install按钮，等待安装完成，在Installing界面单击Next按钮，接下来在Anaconda3 2019.10（64-bit）界面单击Next按钮，在Thanks for installing Anaconda3!界面单击Finish按钮，即可完成Anaconda的安装，如图4-6~图4-9所示。

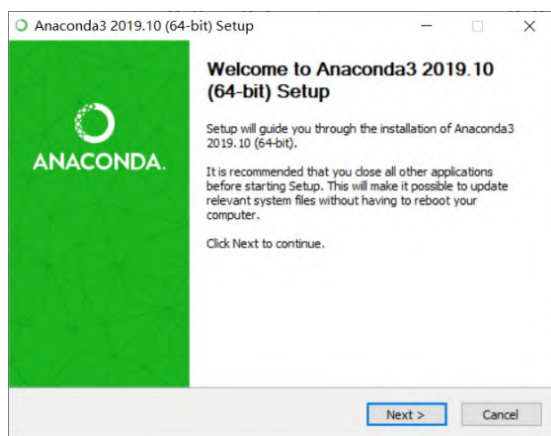


图4-2 单击Next按钮

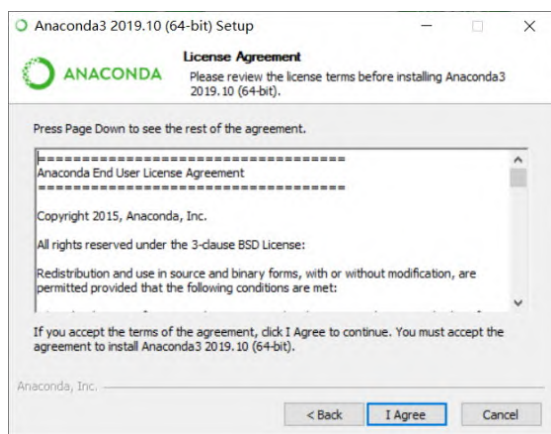


图4-3 单击I Agree按钮

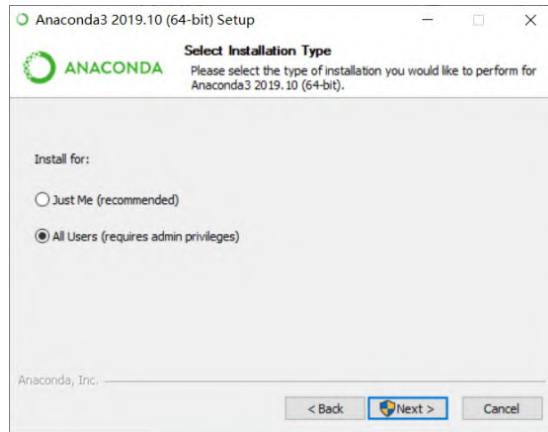


图4-4 单击Next按钮

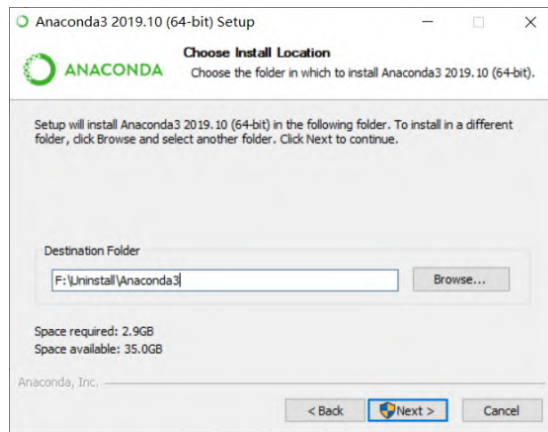


图4-5 选择安装目录

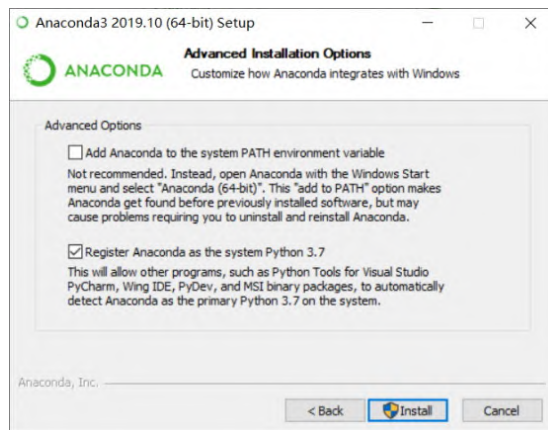


图4-6 单击Install按钮

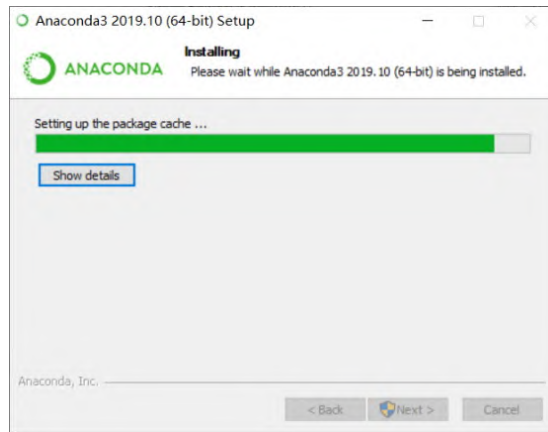


图4-7 单击Next按钮

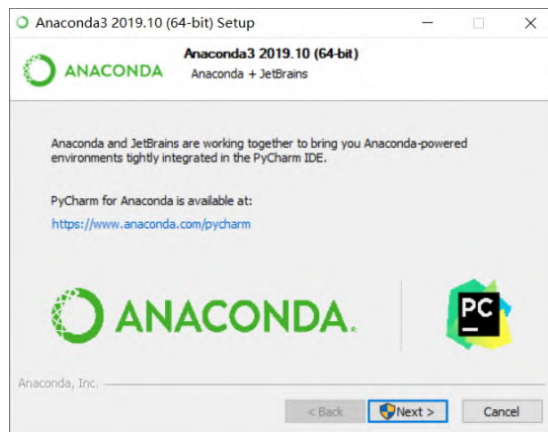


图4-8 单击Next按钮

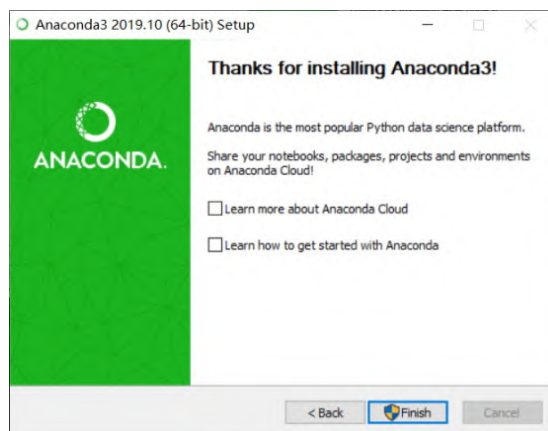
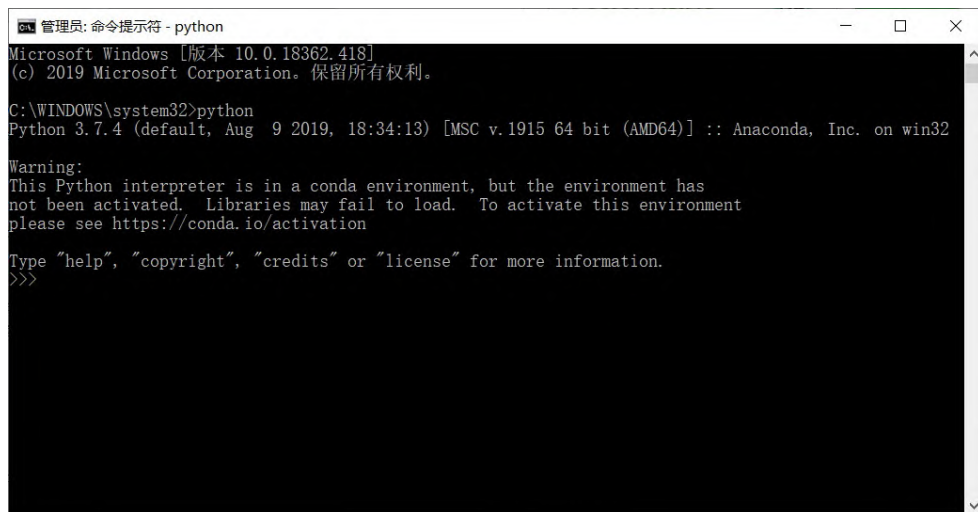


图4-9 单击Finish按钮

步骤 03

按 Windows+R 快捷键，进入命令提示符界面，输入 python，如果出现 Python 版本信息，就说明安装成功（如果看不到，那么尝试先进入安装目录，再输入 python），如图 4-10 所示。



```
管理员: 命令提示符 - python
Microsoft Windows [版本 10.0.18362.418]
(c) 2019 Microsoft Corporation. 保留所有权利。

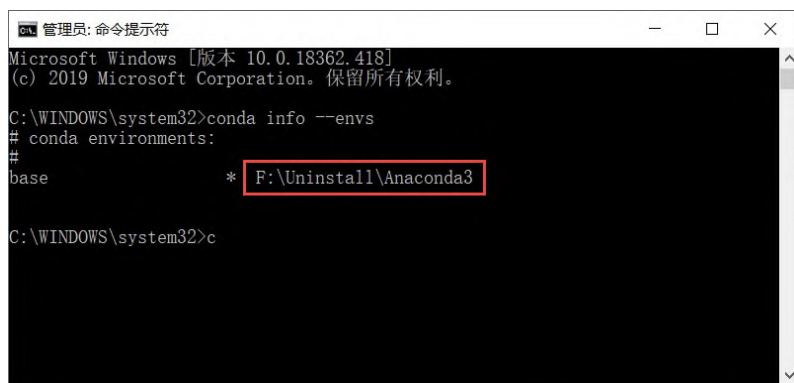
C:\WINDOWS\system32>python
Python 3.7.4 (default, Aug 9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32

Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environment
please see https://conda.io/activation

Type "help", "copyright", "credits" or "license" for more information.
>>>
```

图 4-10 查看 Python 版本

如果出现警告信息：This Python interpreter is in a conda environment, but the environment has not been activated. Libraries may fail to load. To activate this environment please see <https://conda.io/activation>，是因为安装的是 Anaconda 中带的 Python 版本，Python 处于 Conda 环境中，使用 Python 需要激活。可以在命令提示符界面输入 `conda info - envs` 命令，查看 Anaconda 的位置，复制 base 后面的位置信息，如图 4-11 所示。



```
管理员: 命令提示符
Microsoft Windows [版本 10.0.18362.418]
(c) 2019 Microsoft Corporation. 保留所有权利。

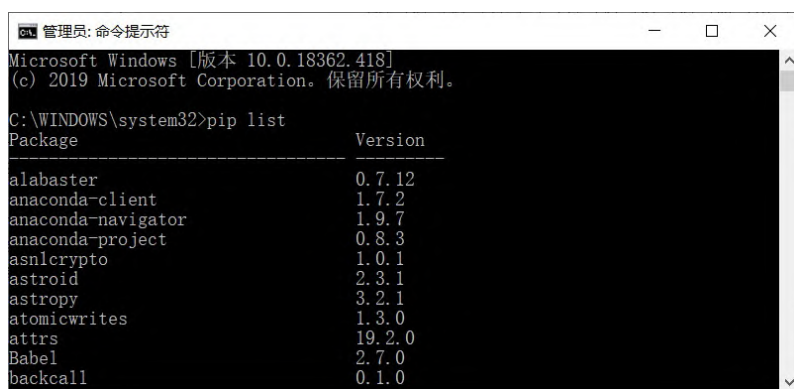
C:\WINDOWS\system32>conda info --envs
# conda environments:
#
base                * F:\Uninstall\Anaconda3

C:\WINDOWS\system32>c
```

图4-11 查看Anaconda的位置

先输入`conda init`对配置进行初始化，然后输入`conda activate F:\Uninstall\Anaconda3`，根据自己的位置修改`F:\Uninstall\Anaconda3`，其为自己计算机上base后面的位置信息，执行程序后就可以激活Python环境，没有警告信息了。

如果不能正常显示Python的版本信息，就需要在计算机的环境变量path中添加Python路径。然后进入命令提示符界面，切换盘符到Anaconda安装目录，输入`pip list`（或`conda list`）命令就可以查询已经安装了哪些包，如图4-12所示。

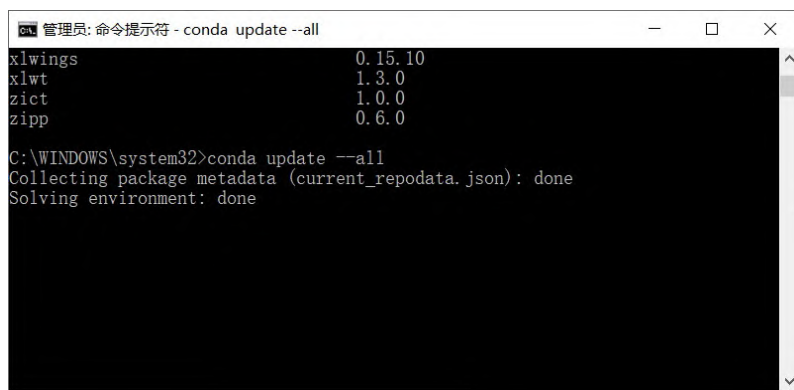


```
管理员: 命令提示符
Microsoft Windows [版本 10.0.18362.418]
(c) 2019 Microsoft Corporation. 保留所有权利。

C:\WINDOWS\system32>pip list
Package                                Version
-----
alabaster                              0.7.12
anaconda-client                        1.7.2
anaconda-navigator                    1.9.7
anaconda-project                      0.8.3
asn1crypto                            1.0.1
astroid                                2.3.1
astropy                                3.2.1
atomicwrites                          1.3.0
attrs                                  19.2.0
Babel                                  2.7.0
backcall                              0.1.0
```

图4-12 查看安装的包

初次安装的包一般版本比较老，为了避免之后使用报错，可以输入`conda update --all`命令，对所有的包进行更新，在提示是否更新的时候输入y（Yes）让更新继续，等待完成即可，如图4-13所示。



```
管理员: 命令提示符 - conda update --all
xlwings                0.15.10
xlwt                   1.3.0
zict                   1.0.0
zipp                   0.6.0

C:\WINDOWS\system32>conda update --all
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

图4-13 更新所有的包

由于Python第三方包比较多，因此很多人喜欢用Python，但是调包有的时候会出现问题，安装包不是失败就是很慢，很影响工作进度。当我们在命令提示符窗口中使用命令：`pip install包名`时，常常会出现安装失败的现象，你会看到下载的进度条，但是最后下载到百分之几十的时候，窗口中就会出现一堆红字，显示Python第三方库下载出问题了。我们在命令提示符窗口进行pip安装时，默认下载的是国外资源，由于网速不稳定甚至没有网速，就会导致安装包时出现错误，解决办法有两种：

(1) 首先搜索所需要的安装包，然后去国外的网站下载这个安装包（注意：要下载的安装包的名称就是在命令提示符窗口进行pip安装时程序自动搜索下载的那个安装包名）。下载后，使用`cd`命令将路径切换到下载包的文件夹中安装即可。在命令提示符窗口安装本地文件包时，可能会在窗口中看到系统自动安装相关必备的其他包，也可能会有下载失败的情况，这时下载不下来的包也可以去国外网站下载，然后在本地安装即可。

(2) 这是一种一劳永逸的方法，选择国内镜像源，相当于从国内的一些机构下载所需要的Python第三方库，这样速度就很快了。那么如何选择国内镜像源，又如何配置呢？

首先找到C:\Users\Administrator\AppData\Roaming这个路径的文件夹，有人会找不到，这是因为文件夹被隐藏了，解决办法如下：

以Windows 10系统为例，打开C盘，单击界面左上角的“查看”，勾选“隐藏的项目”，然后进入“用户”文件夹，双击计算机的登录用户名，例如shang。这样就能看到AppData文件夹了。

找到路径后，首先在该路径下新建文件夹，命名为pip，然后在pip文件夹中新建一个TXT格式的文本文档，打开文本文档，将下面这些代码复制到文本文档中，关闭并保存。然后将TXT格式的文本文档重新命名为pip.ini，这样就创建了一个配置文件。

```
[global]
timeout=60000
index-url=https://pypi.tuna.tsinghua.edu.cn/simple
[install]
use-mirrors=true
mirrors=https://pypi.tuna.tsinghua.edu.cn
```

文档中的链接地址还可以更换为以下地址：

- 阿里云： <http://mirrors.aliyun.com/pypi/simple/>。
- 中国科技大学：
<https://pypi.mirrors.ustc.edu.cn/simple/>。
- 豆瓣（douban）：<http://pypi.douban.com/simple/>。
- 清华大学：<https://pypi.tuna.tsinghua.edu.cn/simple/>。
- 中国科学技术大学：
<http://pypi.mirrors.ustc.edu.cn/simple/>。

这样再使用pip进行包安装时就默认选择国内源，速度很快。

此外，在Anaconda中可以使用以下命令创建多个版本的Python环境：

```
conda create -n env_name list of packages
```

其中，-n代表name；env_name是需要创建的环境名称；list of packages则是列出在新环境中需要安装的工具包，可以不用设置，默认安装一些基础包。

例如，现在的Python版本是3.7，但是需要使用Python 2.7和Python 3.6的环境，则在Anaconda prompt中输入以下语句即可：

```
conda create -n py36 Python=2.7  
conda create -n py36 Python=3.6
```

新环境安装结束后，会在Anaconda3的envs文件夹下生成两个文件，如图4-14所示。

名称	修改日期	类型	大小
py27	2019/10/18 10:18	文件夹	
py36	2019/10/18 10:21	文件夹	
.conda_envs_dir_test	2019/10/18 10:17	CONDA_ENVS_D...	0 KB

图4-14 查看envs文件夹

Python多环境操作的一些常用语句如下：

- 显示所有环境的语句：conda env list。
- 激活新配置环境的语句：conda activate py27。
- 退出新配置环境的语句：conda deactivate。
- 删除配置的新环境的语句：conda env remove -n py27。

当工作中需要分享代码的时候，也需要将运行环境分享给大家，执行以下命令可以将当前环境下的相关信息存入名为env的YAML文件中：

```
conda env export > env.yaml
```

同样，当执行别人的代码时，也需要配置相应的Python环境，这时可以用对方分享的YAML文件创建一个一模一样的运行环境，语句如下：

```
conda env create -f env.yaml
```

4.1.2 PyCharm

PyCharm是一个比较常见的Python代码开发环境，可以帮助用户在使用Python语言开发时提高效率，比如调试、语法高亮、Project管理、代码跳转、智能提示、自动完成、单元测试、版本控制等。

PyCharm是一个专注于Python的集成开发环境，分为专业版、教育版和社区版。专业版是收费的，只能试用一个月；教育版是免费的，是专门针对学生和老师设计的；社区版适合个人或小团队开发使用，但是一些功能没法使用，比如Web开发、Python探查等，对于初学者来说社区版的功能足够满足需求。

在开始安装PyCharm之前，需确保计算机上已经安装了Java 1.8以上的版本，并且已配置好环境变量。下面介绍具体安装步骤。

步骤 01

进入官网：

<http://www.jetbrains.com/pycharm/download/#section=windows>，下载社区版，如图4-15所示。

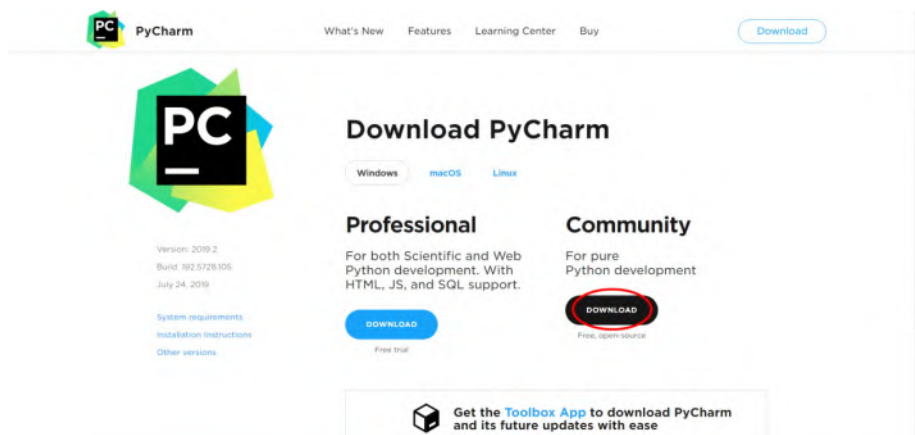


图4-15 下载PyCharm社区版

步骤 02

双击下载的pycharm-community-2019.2.exe文件开始安装，在欢迎界面单击Next按钮，如图4-16所示。

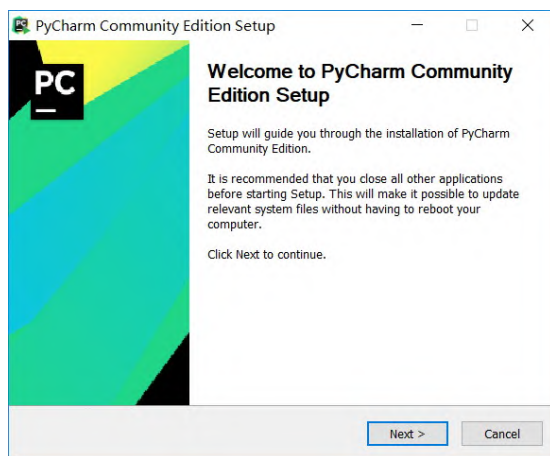


图4-16 欢迎页面

步骤 03

在选择安装理解界面，如果要修改安装路径，可以单击Browse按钮选择安装路径，切换好安装路径后再单击Next按钮，如图4-17所示。

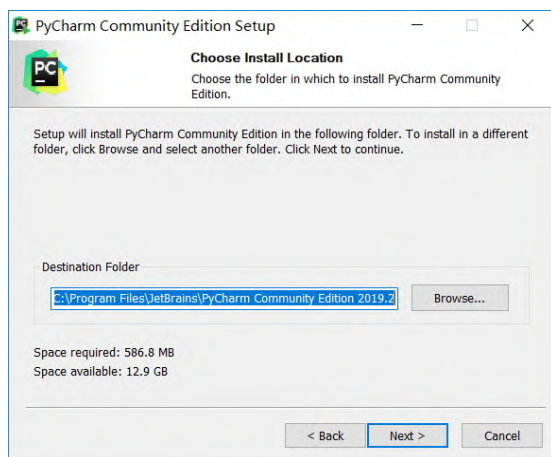


图4-17 选择安装路径

步骤 04

在软件配置界面，根据实际需要勾选相应复选框，配置完成后，单击Next按钮，如图4-18所示。在安装界面单击Install按钮，等待安装完成即可。

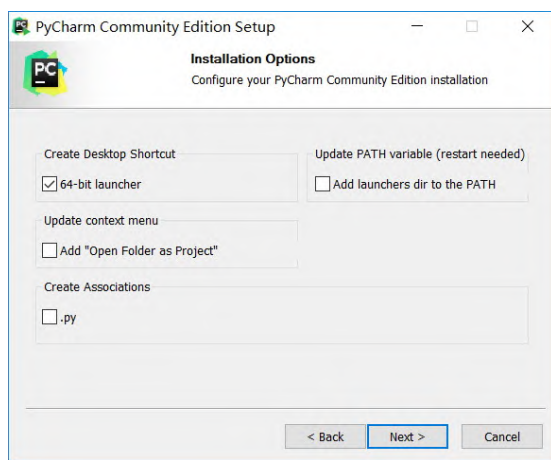


图4-18 配置软件参数

如果有特殊的需要，可参考各个选项的功能：

- 创建快捷方式：根据当前系统进行选择。
- 将PyCharm的启动目录添加到环境变量中，需要重启，如果需要使用命令行操作PyCharm，就勾选该复选框。

- 添加鼠标右键菜单，使用打开项目的方式打开文件夹。如果经常需要下载一些别人的代码查看，可以勾选该复选框，这会增加鼠标右键菜单的选项。
- 将所有.py文件关联到PyCharm，也就是双击计算机上的.py文件，会默认使用PyCharm打开。不建议勾选该复选框，勾选后PyCharm每次打开的速度会比较慢。若要单独打开.py文件，则建议使用Notepad++等文本编辑器，打开速度会更快。

步骤 05

在开始菜单文件夹界面，选择默认设置JetBrains，然后单击Install按钮开始安装，如图4-19所示。

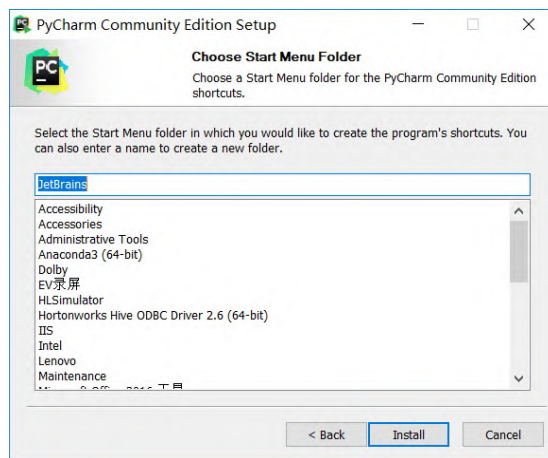


图4-19 开始菜单文件夹

步骤 06

安装完成后，单击Finish按钮关闭安装窗口，如图4-20所示。

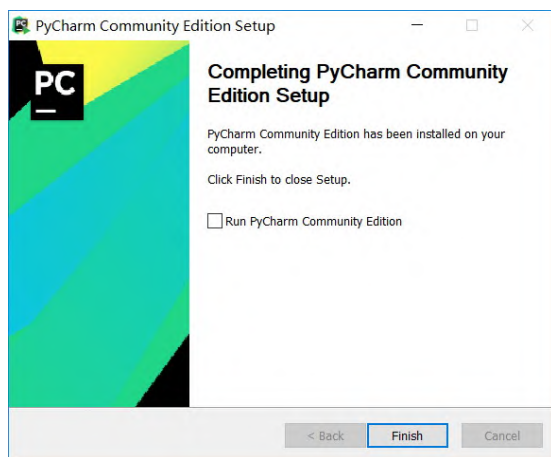


图4-20 软件安装结束

安装好PyCharm后，还需要配置其代码开发环境。首次启动PyCharm会弹出配置窗口，如图4-21所示。

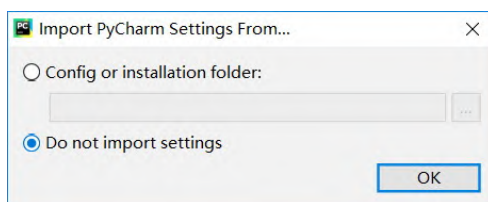


图4-21 软件配置窗口

如果之前使用过PyCharm并有相关的配置文件，就在此处选择导入；否则保持默认设置即可，然后单击OK按钮。在同意用户使用协议界面，勾选确认同意复选框，并单击Continue按钮，如图4-22所示。



图4-22 用户使用协议

确定是否需要进行数据共享，可以直接单击Don't send按钮，如图4-23所示。

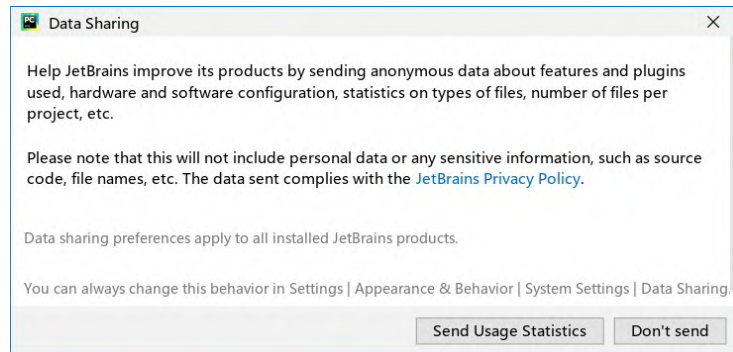


图4-23 数据共享设置

选择主题，左边为黑色主题，右边为白色主题，根据需要进行选择，这里我们选择Light类型，并单击Next:Featured plugins按钮继续后面的插件配置，如图4-24所示。

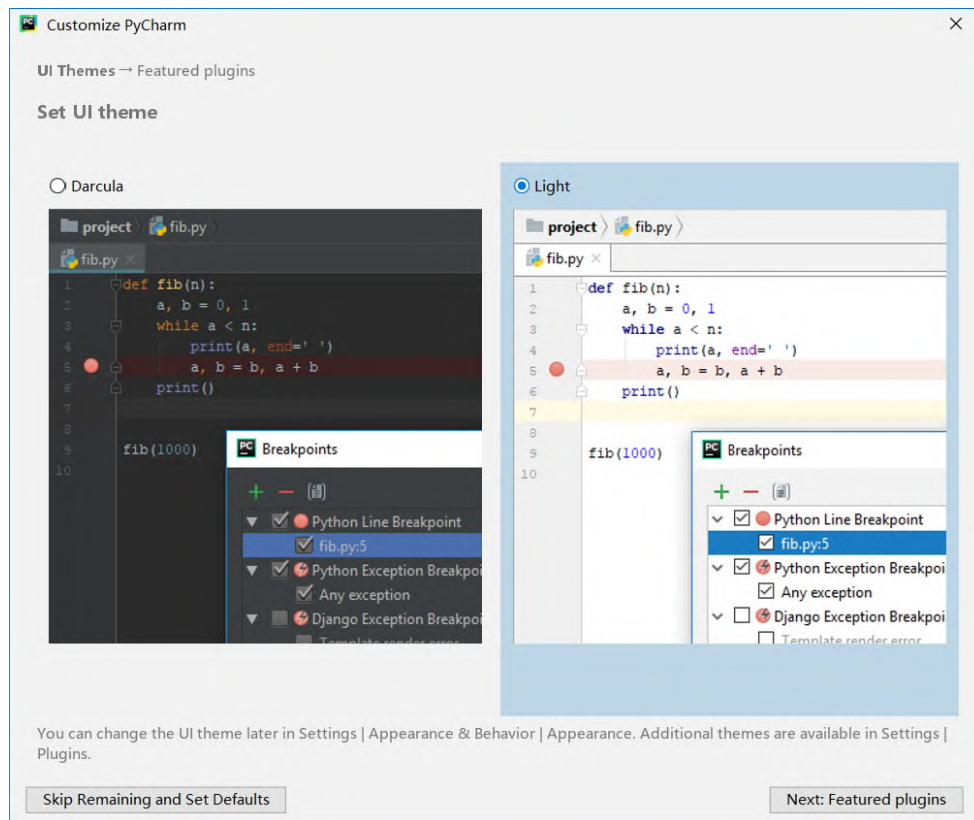


图4-24 选择软件主题

下载插件，可以根据需要下载，也可以不安装，单击Start using PyCharm按钮，如图4-25所示。

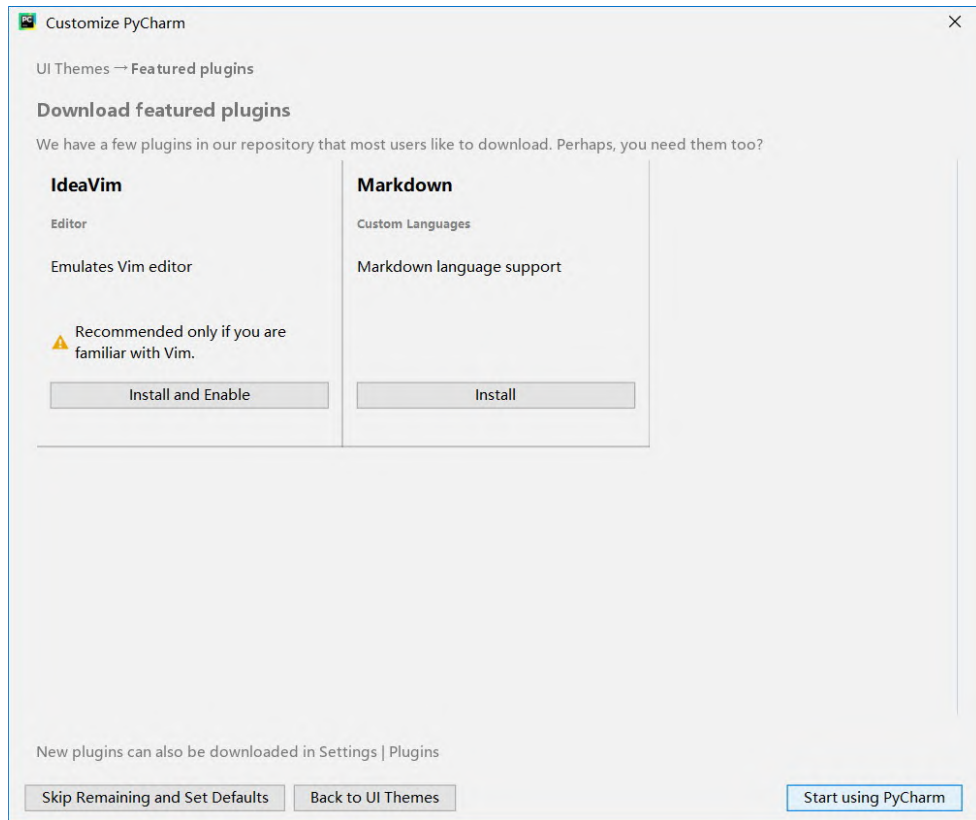


图4-25 下载相关插件

设置完成后，单击Create New Project选项，开始创建一个新的Python项目，如图4-26所示。

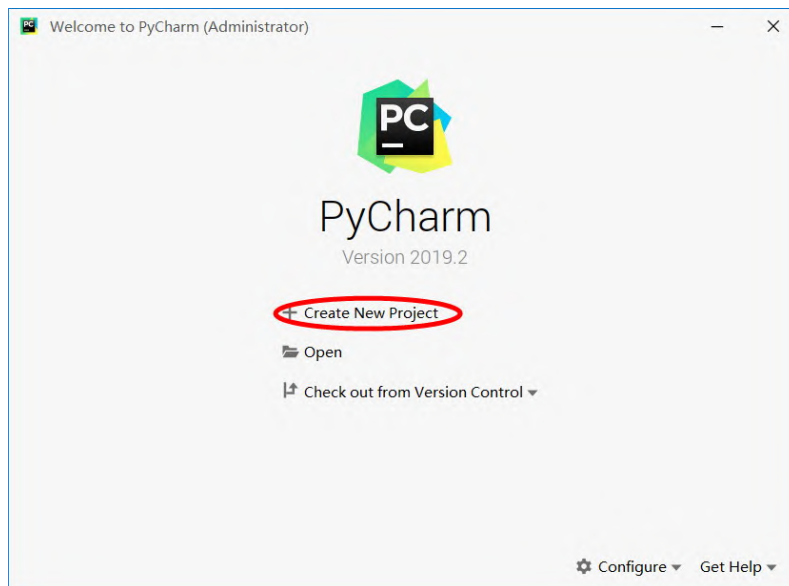


图4-26 创建新项目

在创建新项目界面。在Location中设置项目名称并选择解释器，注意：这里默认使用Python的虚拟环境，即第一个New environment using单选按钮，再单击Create按钮，如图4-27所示。

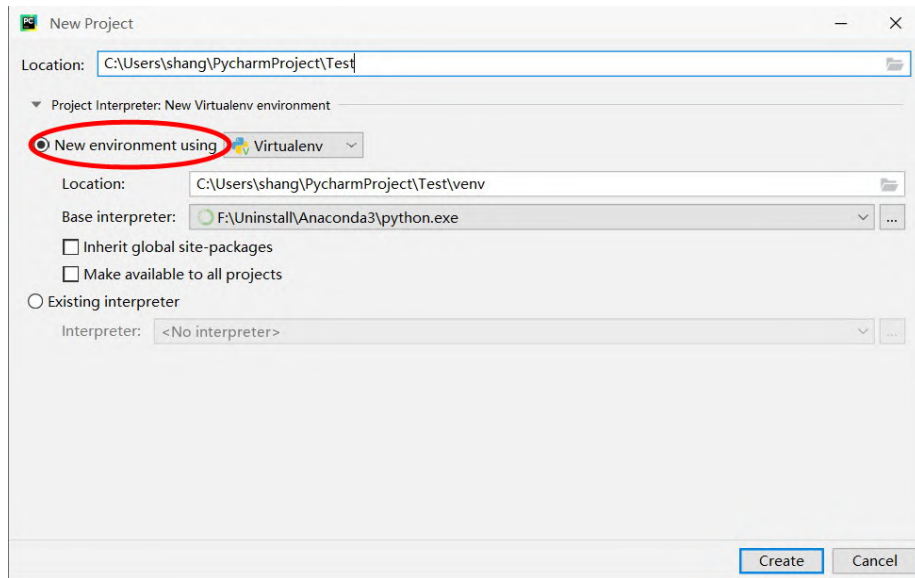


图4-27 配置新项目

如果不使用虚拟环境，一定要修改，就选择第二个Existing interpreter单选按钮，然后选择需要添加的解释器，再单击Create按钮，如图4-28所示。

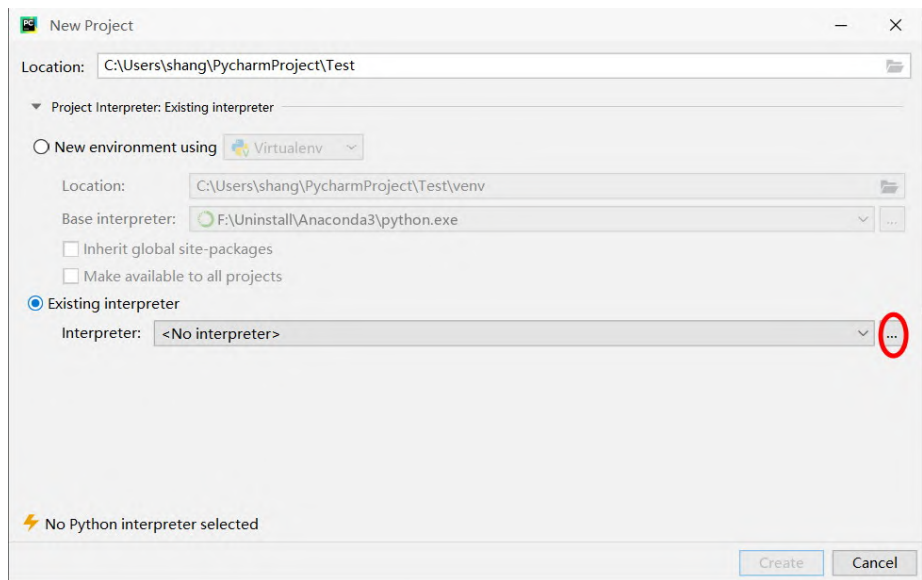


图4-28 配置解释器

在弹出的PyCharm欢迎界面，取消勾选Show tips on startup复选框，不用每次都打开欢迎界面，单击Close按钮，退出使用指导过程，如图4-29所示。

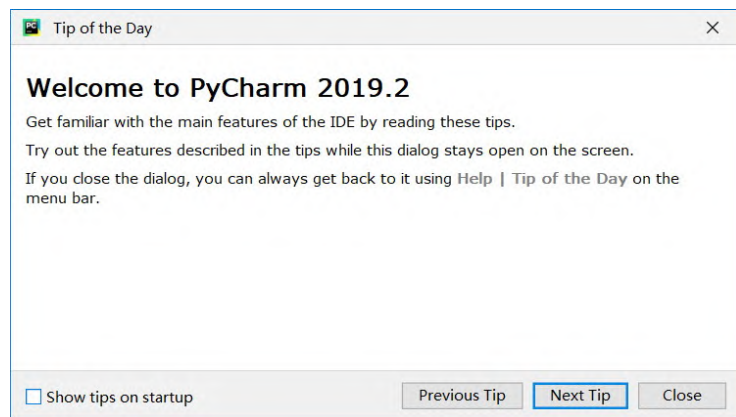


图4-29 欢迎界面

创建Python文件，在项目名称的位置右击，依次选择New→Python File选项，如图4-30所示。

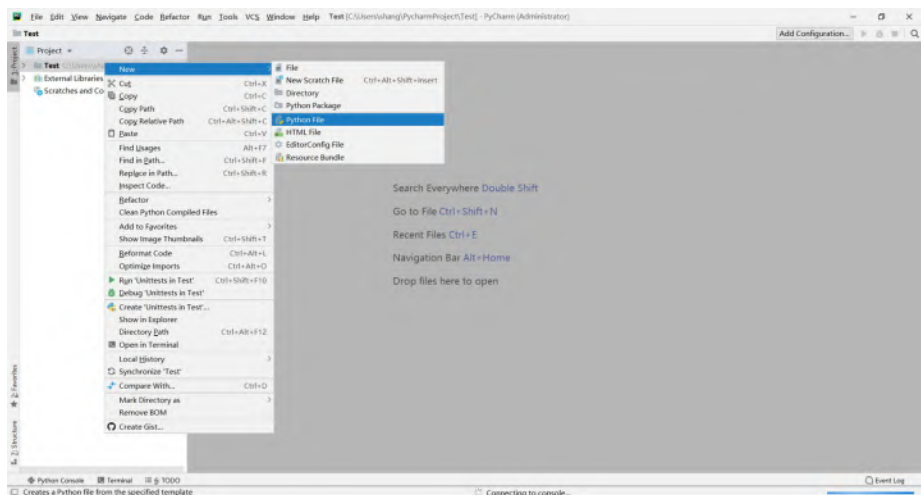


图4-30 新建Python文件

输入文件名称Hello，并按Enter键即可，如图4-31所示。

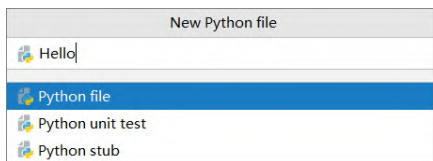


图4-31 输入文件名称

在文件中输入代码：`print("Hello Python!");`，然后在文件中任意空白位置右击，选择“Run 'Hello'”选项，在界面的下方显示Python代码的运行结果，PyCharm已经正常安装和配置，如图4-32所示。

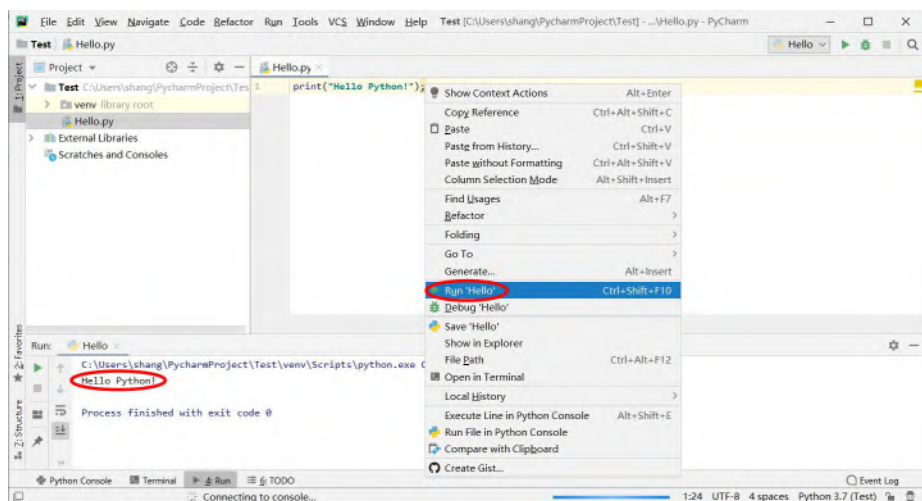


图4-32 运行Python代码

4.2 Python代码开发工具

Python的常用代码开发工具有Spyder、Jupyter Notebook和Jupyter Lab，由于本书研究的是数据可视化技术，需要经常展示一些图表，相对而言，个人认为Jupyter Lab这个工具比较适合。为什么合适，这个需要读者在研读完本书后才会理解，当然每个人有自己的编程工具喜好，可以根据个人实际情况进行选择。下面逐一进行介绍。

4.2.1 Spyder

安装Anaconda后，默认已经安装Spyder，因此不需要再单独安装。Spyder是Python的作者为它开发的一个简单的集成开发环境，与其他开发环境相比，它最大的优点就是模仿MATLAB的“工作空间”功能，可以方便地观察和修改数组的值。

Anaconda安装成功后，默认会将Spyder的启动程序添加到环境变量中，可以通过单击计算机的“开始”按钮，然后单击其快捷方式启动，启动程序为Spyder，如图4-33所示。

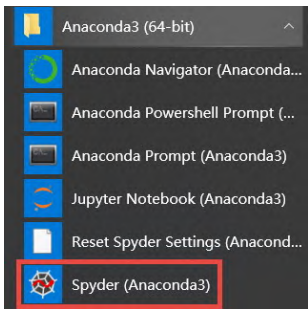


图4-33 Spyder启动程序

Spyder的界面由多个窗格构成，用户可以根据自己的喜好调整它们的位置和大小，如图4-34所示。

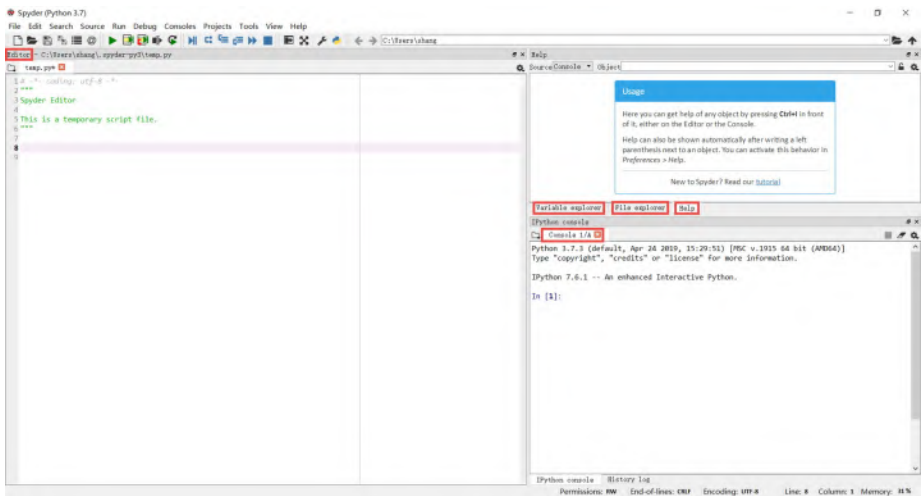


图4-34 Spyder界面

从图4-34中可以看到Editor、Console、Variable explorer、File explorer、Help等窗格。表4-1中列出了Spyder的主要窗格及其作用。

表4-1 Spyder主要窗格

窗格名称	作用
Editor	编辑程序，可用标签页的形式编辑多个程序文件
Console	在别的进程中运行的 Python 控制台
Variable explorer	显示 Python 控制台中的变量列表
File explorer	文件浏览器，用于打开程序文件或者切换当前路径
Help	查看对象的说明文档

在使用Spyder进行代码开发时，需要在Editor窗格的空白区域编写代码。编写完毕后，可以通过工具栏上的运行按钮运行程序，如果正常执行，就可以在界面右下方的Console窗格中看到结果和报错信息等，如图4-35所示。

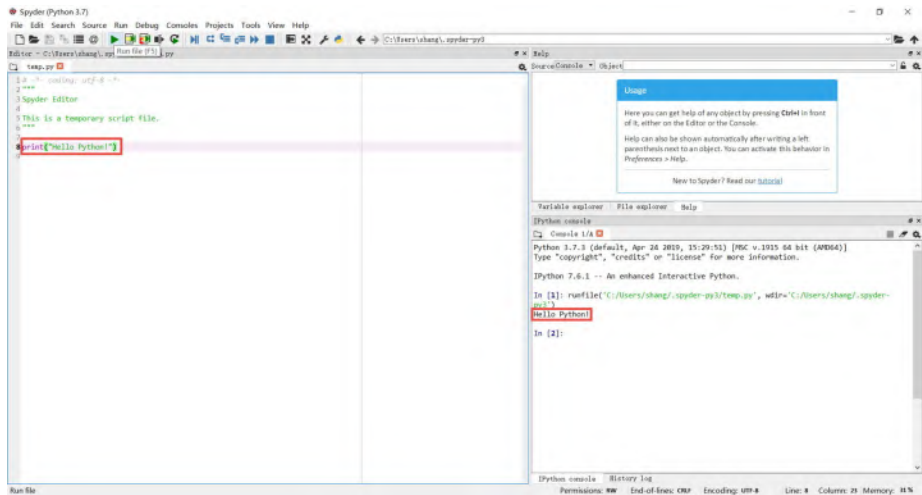


图4-35 运行示例程序

快捷键可以方便我们进行代码的开发和测试，Spyder的常用快捷键如表4-2所示。此外，可以通过Tools→Preferences→Keyboard Shortcut查看所有快捷键。

表4-2 Spyder主要快捷键

快 捷 键	中文名称
Ctrl+R	替换文本
Ctrl+I	单行注释，单次注释，双击取消注释
Ctrl+4	块注释，单次注释，双击取消注释
F5	运行程序
Ctrl+P	文件切换
Ctrl+L	清除 Shell
Ctrl+I	查看某个函数的帮助文档
Ctrl+Shift+V	调出变量窗口
Ctrl+Up	回到文档开头
Ctrl+Down	回到文档末尾

4.2.2 Jupyter Notebook

Jupyter Notebook是一个在浏览器中使用的交互式笔记本，可以实现将代码、文字完美结合起来，用户大多数是一些从事数据科学相关领域（机器学习、数据分析等）的人员。下面介绍Jupyter Notebook的常用功能。

Anaconda安装成功后，默认会将Jupyter Notebook的启动程序添加到环境变量中，启动程序为Jupyter Notebook，如图4-36所示。

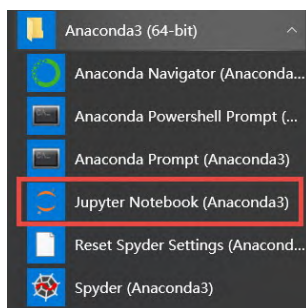


图4-36 Jupyter Notebook启动程序

启动前需要说明一个概念，Jupyter Notebook中有一个叫作工作空间（工作目录）的概念，也就是想在哪个目录进行之后的编程工作，就在哪个目录启动它。程序启动后，浏览器会自动打开Jupyter Notebook窗口，如图4-37所示，说明Jupyter Notebook安装成功。

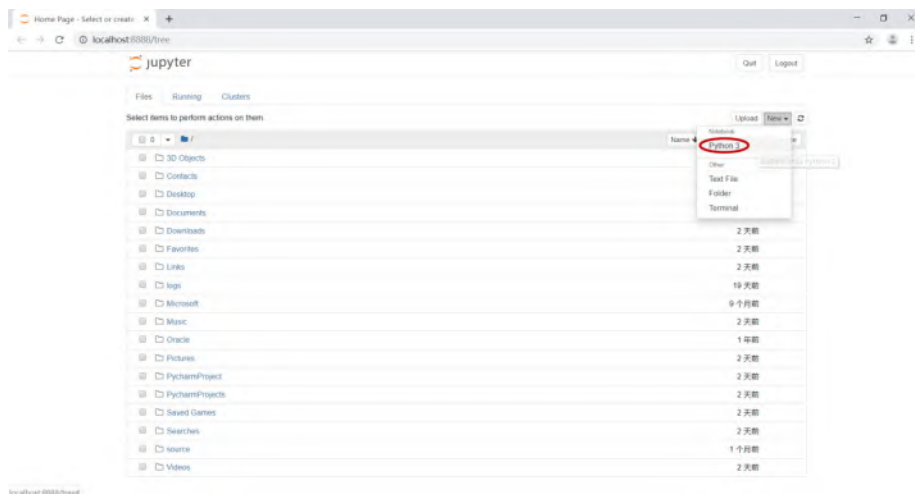


图4-37 Jupyter Notebook窗口

新建Notebook，打开的界面主要包含图4-37所示的几个菜单。我们单击New下的Python 3，创建一个Python 3的.ipynb文件，如图4-38所示。

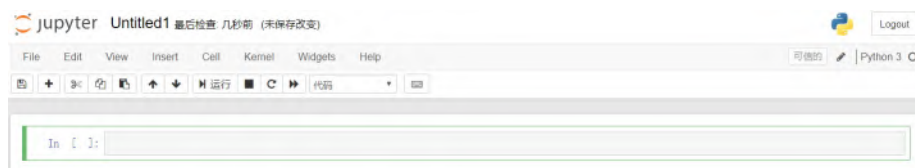


图4-38 新建Notebook

可以看到，每一个Notebook主要包含4个区域：文件名、菜单栏、工具栏和内容编辑。

要修改文件名，单击界面上方的文件名Untitled1，可以重命名当前Notebook的文件名，这里修改为Hello_Python，再单击“重命名”按钮即可，如图4-39所示。

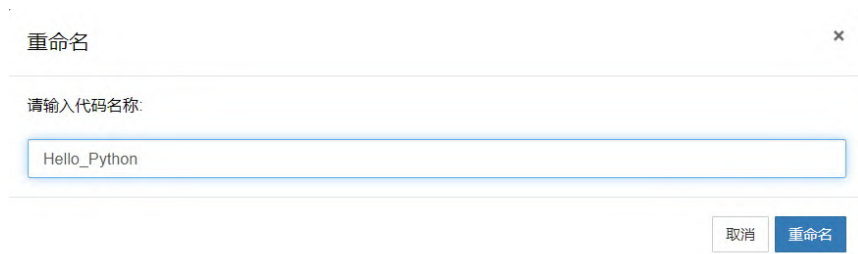


图4-39 输入代码名称

下面重点介绍几个常用的菜单栏及其作用。

File菜单中主要包含：创建新的Notebook、打开新的界面、复制当前Notebook、保存当前Notebook、重命名Notebook、保存还原点、恢复到指定还原点、查看Notebook预览、下载Notebook、关闭Notebook等。

注意Download as选项，它可以将当前Notebook转为HTML文件、LaTeX文件、Markdown文件、Notebook文件、PDF文件、RST文件、PY文件等，如图4-40所示。

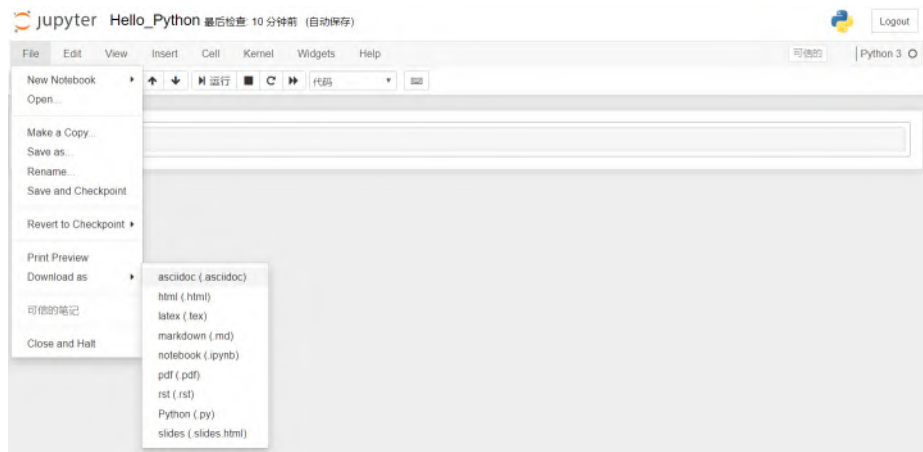


图4-40 Download as选项

Insert菜单中包含在当前位置的下方插入一个新的单元格、在当前位置的上方插入一个新的单元格，如图4-41所示。

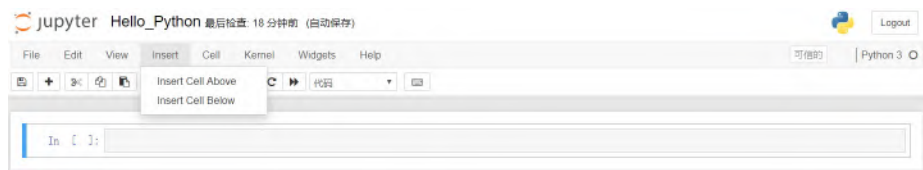


图4-41 Insert菜单

Cell菜单主要包含运行Cells、运行Cells并在之后插入新的Cell、运行所有的Cells、运行当前之上的所有Cells、运行当前之下的所有Cells、改变Cells的类型（Code、Markdown、Raw NBConvert）等，如图4-42所示。

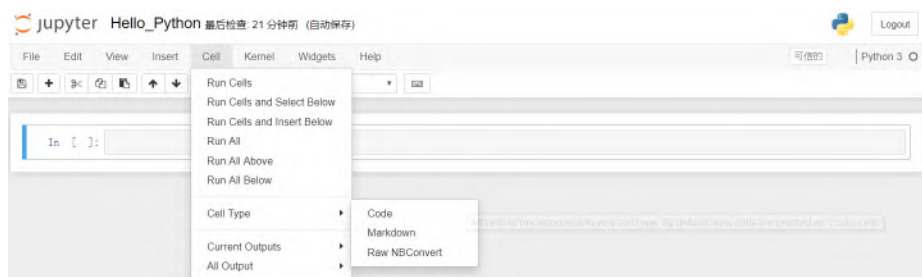


图4-42 Cell菜单

Kernel菜单主要包含中断Kernel、重启Kernel、重启Kernel并清除输出、重启Kernel并运行所有的Cells、重连Kernel、关闭Kernel、改变Kernel类型，如图4-43所示。

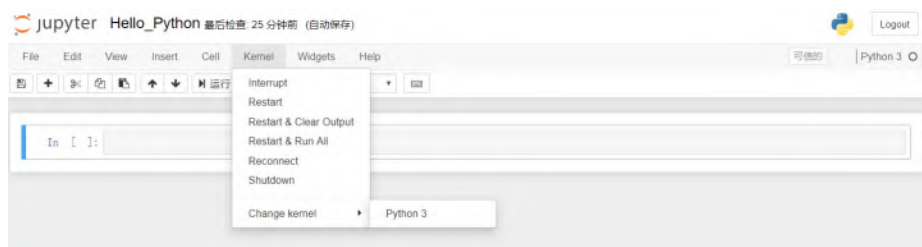


图4-43 Kernel菜单

此外，Help菜单主要包含用户交互引导、键盘快捷键、Notebook帮助、Python帮助、IPython帮助、NumPy帮助、SciPy帮助、Matplotlib帮助、Pandas帮助等。如果能记住一些常用的快捷键，那么对使用Jupyter Notebook的帮助还是很大的，如图4-44所示。



图4-44 键盘快捷键

工具栏上的图标依次为：保存还原点、在当前位置之下添加Cell、剪切当前Cell、复制选择的Cell、拷贝选择的Cell、上移选中的Cell、下移选中的Cell、运行Cell、中断Kernel、重启Kernel等，如图4-45所示。很明显，工具栏中的功能大多是菜单栏中一部分功能的体现，主要是为了方便寻找。

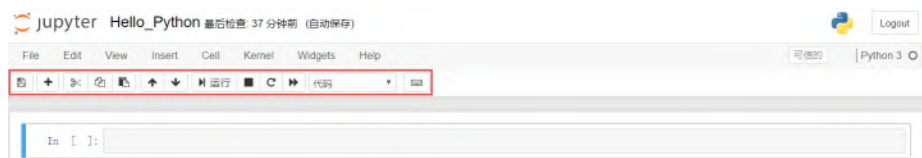


图4-45 工具栏

如果想要运行Python代码，其实很简单，因为Python代码最后都是在Cell中编写的。首先在Cell中编写Python代码，然后单击“运行”按钮，可以直接在界面下方看到结果。我们可以发现，第一个

Cell前面有 “In [1]:” 提示符，第二个Cell前面有 “In [2]:” 提示符，如图4-46所示。

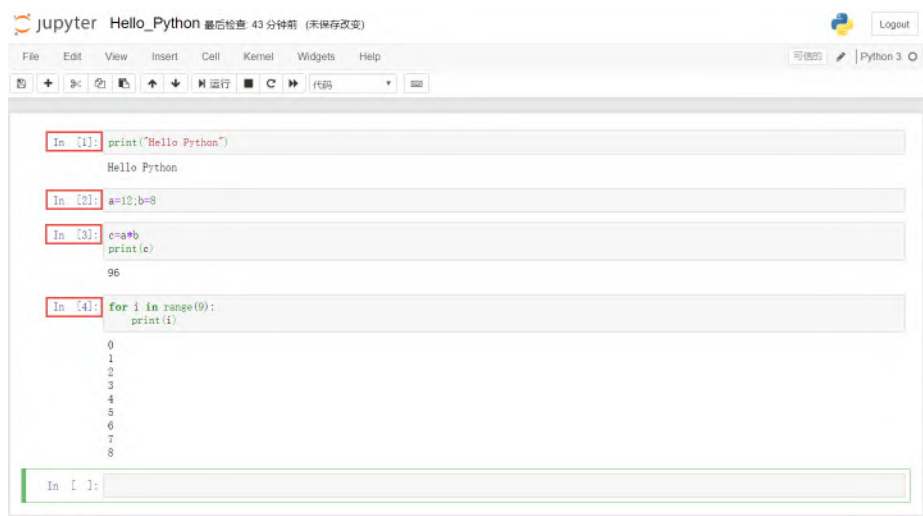


图4-46 运行代码

4.2.3 Jupyter Lab

Jupyter Lab源于IPython Notebook，是使用Python（R、Julia、Node等其他语言的内核）进行代码演示、数据分析、可视化、教学的很好的工具，对Python的愈加流行和在AI领域的领导地位有很大的推动作用，这是本书默认使用的代码开发工具。可以通过Try Jupyter网站（<https://jupyter.org/try>）使用Jupyter Lab，如图4-47所示。

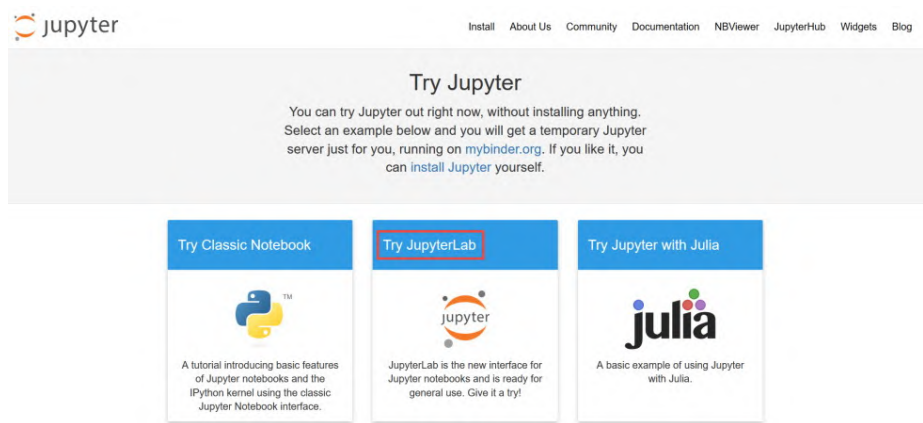


图4-47 Jupyter Lab简介

Jupyter Lab是Jupyter的一个拓展，提供了更好的用户体验，例如可以同时在一个浏览器页面打开编辑多个Notebook、IPython Console和Terminal终端，并且支持预览和编辑更多种类的文件，如代码文件、Markdown文档、JSON文件和各种格式的图片等，还可以使用Jupyter Lab连接Google Drive等云存储服务，从而极大地提升工作效率，如图4-48所示。

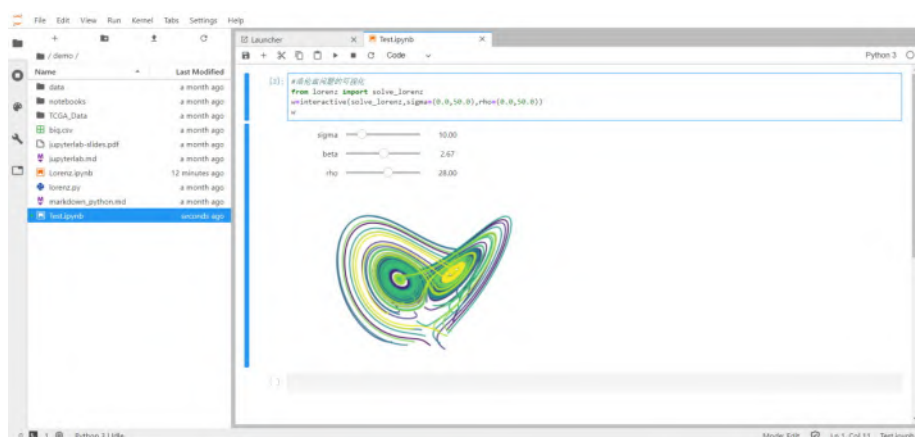


图4-48 演示Jupyter Lab

安装Anaconda后，默认安装了pip，但是没有安装Jupyter Lab，我们可以通过pip来安装Jupyter Lab。首先需要打开Anaconda Prompt，然后输入pip install jupyterlab进行安装，如图4-49所示。

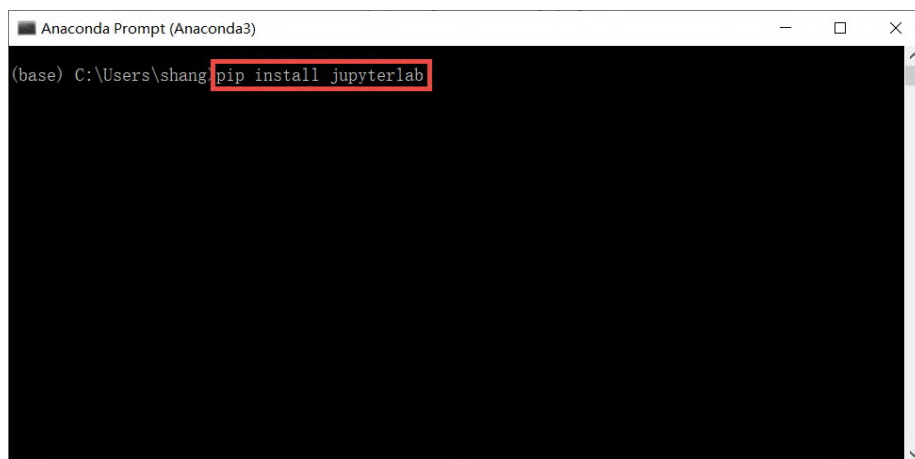


图4-49 安装Jupyter Lab

Jupyter Lab会继承Jupyter Notebook的配置，如地址、端口、密码等。运行Jupyter Lab的方式比较简单，只需要在Anaconda Prompt中输入jupyter lab即可，如图4-50所示。

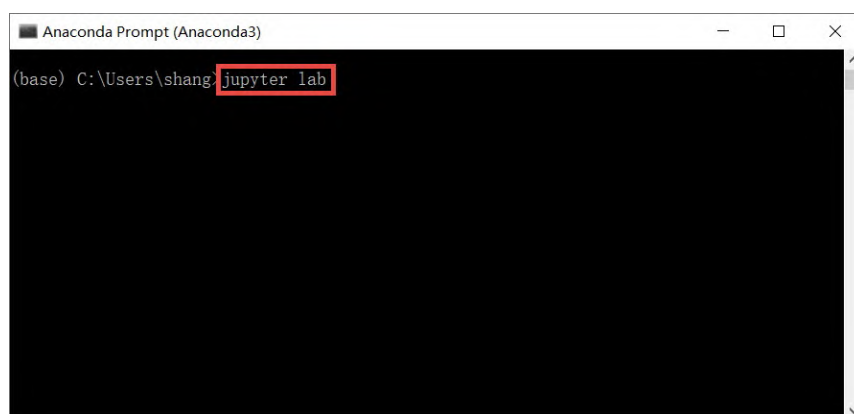


图4-50 启动Jupyter Lab

Jupyter Lab程序启动后，浏览器会自动打开编程窗口，默认地址是http://localhost:8888，如图4-51所示，说明Jupyter Lab安装成功。从图4-51中可以看出，Jupyter Lab界面左边是存放笔记本的工作路径，右边是我们需要创建的笔记本类型，当然还可以根据需要添加其他类型的Kernel，如R等。

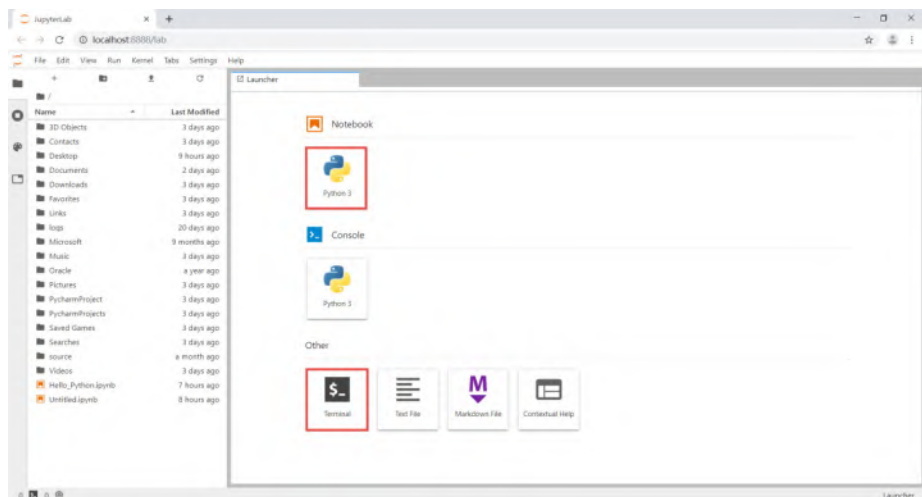


图4-51 Jupyter Lab界面

我们可以对Jupyter Lab的参数进行修改，如对远程访问、工作路径等进行设置，配置文件位于C盘系统用户名下的.jupyter文件夹中，文件名称为jupyter_notebook_config.py。

如果配置文件不存在，就需要自行创建，单击图4-51中Other下的Terminal，使用jupyter notebook--generate-config命令生成配置文件，并且会显示出文件的存储路径及名称，如图4-52所示。

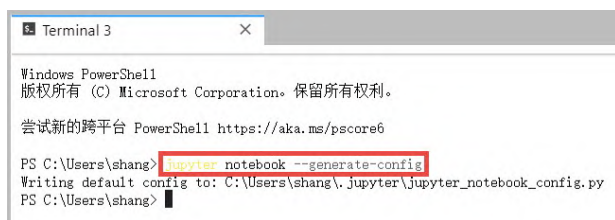


图4-52 配置Jupyter Lab

Jupyter Lab提供了一个命令来设置密码：jupyter notebook password，生成的密码存储在jupyter_notebook_config.json文件中，下方将会显示文件的路径及名称，如图4-53所示。



```
Terminal 3
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

尝试新的跨平台 PowerShell https://aka.ms/pscore6

PS C:\Users\shang> jupyter notebook --generate-config
Writing default config to: C:\Users\shang\.jupyter\jupyter_notebook_config.py
PS C:\Users\shang> jupyter notebook password
Enter password:
Verify password:
[NotebookPasswordApp] Wrote hashed password to C:\Users\shang\.jupyter\jupyter_notebook_config.json
PS C:\Users\shang>
```

图4-53 配置Jupyter Lab密码

如果需要允许远程登录，那么还需要在 `jupyter_notebook_config.py` 中找到下面几行代码，取消注释并根据项目的实际情况进行修改，修改后的配置如下：

```
c.NotebookApp.ip='*'
c.NotebookApp.open_browser=False
c.NotebookApp.port=8888
```

如果需要修改Jupyter Lab的默认工作路径，就需要找到下面的代码，取消注释并根据项目的实际情况进行修改，修改后的配置如下：

```
c.NotebookApp.notebook_dir=u'D:\\Python for Matplotlib and
pycharts'
```

待需要配置参数都修改后，需要重新启动Jupyter Lab才能生效，首先需要输入刚刚配置的密码，如图4-54所示。

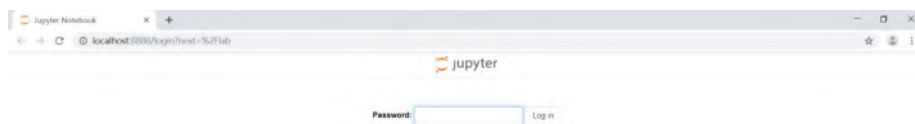


图4-54 输入密码

输入密码后，单击Log in按钮，在新的编程窗口中，工作路径发生了变化，现在呈现的就是D盘Python for Matplotlib and pyecharts文件夹下的文件，如图4-55所示。

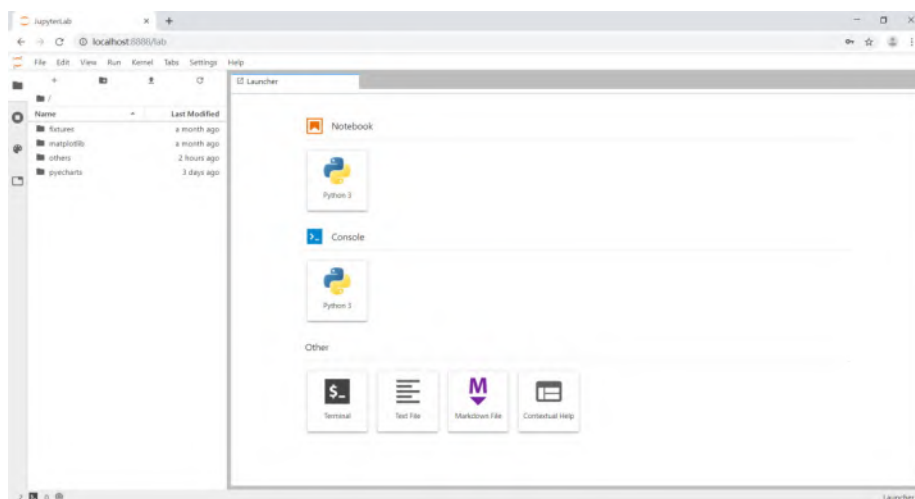


图4-55 登录Jupyter Lab

4.3 Python连接数据源

4.3.1 连接单个文件数据

1. 读取TXT文件数据案例

Python可以直接读取TXT文件数据，并对数据进行可视化分析，案例如下：

```
# -*- coding: utf-8 -*-
"""
连接TXT数据文件案例
"""

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB
```

```

from pyecharts import options as opts
from pyecharts.charts import Bar, Page
from pymysql import *
import Pandas as pd
import pymysql
from Matplotlib import pyplot as plt
import Matplotlib.dates as mdates
from datetime import datetime
import codecs
plt.rcParams['font.sans-serif']=['SimHei']    #中文字体设置

#读取本地文本文件
with open('D:/Python for Matplotlib and
pyecharts/data/orders.txt', mode='r', encoding='utf-8') as
file_object: # 打开TXT文件, 以'utf-8'编码读取
    lines=file_object.readlines()
    file1=[]
    row=[]
    for line in lines:
        row=line.split(",")    #指定空格作为分隔符对line进行切片
        file1.append(row)

v1=[]
v2=[]
v3=[]
for row1 in file1:
    v1.append(row1[0])
    v2.append(row1[1])
    v3.append(row1[2])

del v1[0]
del v2[0]
del v3[0]

int_v2=[]
for str_h in v2:
    a=int(float(str_h))
    int_v2.append(a)

int_v3=[]
for str_l in v3:
    a=int(float(str_l))
    int_v3.append(a)

#画图
def bar_base() -> Bar:

```

```

c= (
    Bar()
    .add_xaxis(v1)
    .add_yaxis("销售额", int_v2)
    .add_yaxis("利润额", int_v3)

    .set_global_opts(title_opts=opts.TitleOpts(title="企业销售业绩
分析", subtitle="12月销售业
绩"), toolbox_opts=opts.ToolboxOpts())
)
return c

#第一次渲染时调用load_javascript文件
bar_base().load_javascript()
#展示数据可视化图表
bar_base().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图4-56所示的可视化视图。

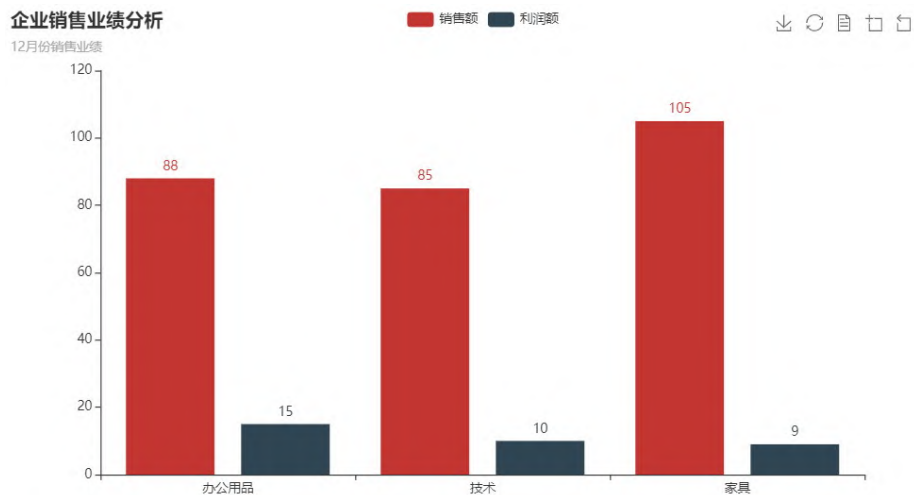


图4-56 销售额和利润额的条形图

2. 读取CSV文件数据案例

Python可以直接读取CSV文件数据，并对数据进行可视化分析，案例如下：

```

# -*- coding: utf-8 -*-
"""
连接CSV文件数据案例
"""

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Bar, Page
import csv

filename='D:/Python for Matplotlib and
pyecharts/data/orders.csv'
#打开文件并将结果文件对象存储在f中
with open(filename,encoding='utf-8') as f:
    #创建一个与该文件相关联的reader对象
    reader=csv.reader(f)
    #只调用一次next()方法，得到文件的第一行，将第一行数据中的每一个
元素存储在列表中
    header_row=next(reader)

    #print(reader)
    #print(header_row)

    #从文件中获取列数据
    v1=[]
    v2=[]
    v3=[]
    #遍历文件中余下的各行
    #reader对象从其当前所在的位置继续读取CSV文件，每次都自动返回当
前所处位置的下一行
    for row in reader:
        #转换为数字
        t_v1=row[0]
        v1.append(t_v1)
        t_v2=row[1]
        v2.append(t_v2)
        t_v3=row[2]
        v3.append(t_v3)

#画图形
def bar_toolbox() -> Bar:
    c=(

```

```

Bar()
.add_xaxis(v1)
.add_yaxis("销售额", v2)
.add_yaxis("利润额", v3)
#.reversal_axis()    #翻转XY轴，水平条形图
.set_global_opts(
    title_opts=opts.TitleOpts(title="12月份企业销售
业绩分析"),

    yaxis_opts=opts.AxisOpts(name="销售额与利润额"),
    xaxis_opts=opts.AxisOpts(name="产品类型"),
    toolbox_opts=opts.ToolboxOpts(),
    legend_opts=opts.LegendOpts(is_show=True),
)
)
return c

#第一次渲染时调用load_javascript文件
bar_toolbox().load_javascript()
#展示数据可视化图表
bar_toolbox().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图4-57所示的可视化视图。

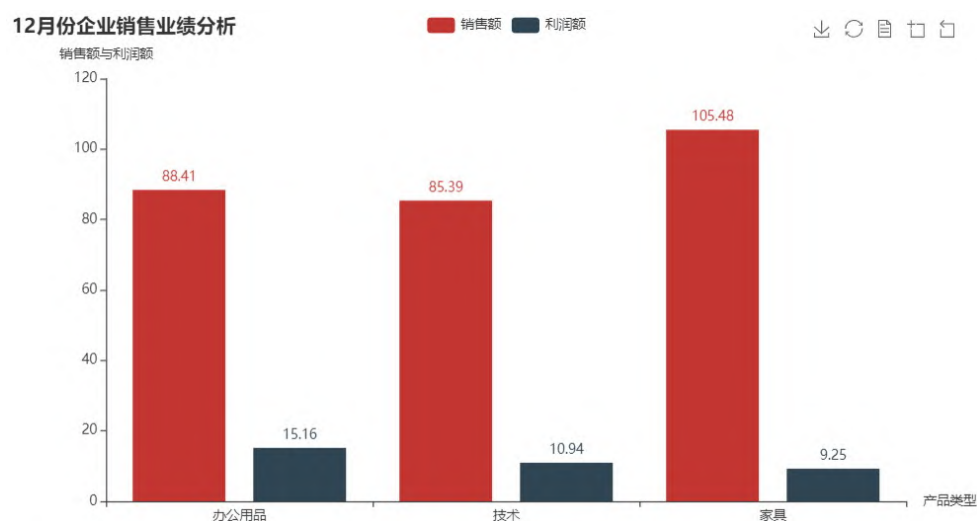


图4-57 销售额和利润额的条形图

3. 读取XLS文件数据案例

Python可以直接读取XLS文件数据，并对数据进行可视化分析，案例如下：

```
# -*- coding: utf-8 -*-
"""
连接XLS数据文件案例
"""

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Bar, Page
import Pandas as pd
import xlrd

# 打开文件
data=pd.read_excel('D:/Python for Matplotlib and
pyecharts/data/orders.xls')
#读取Excel文件，创建DataFrame
# 获取一系列的数值
v1=list(data['category'])
v2=list(data['x'])
v3=list(data['y'])

#画图形
def bar_toolbox() -> Bar:
    c=(
        Bar()
        .add_xaxis(v1)
        .add_yaxis("销售额", v2)
        .add_yaxis("利润额", v3)
        #.reversal_axis()    #翻转XY轴，水平条形图
        .set_global_opts(
            title_opts=opts.TitleOpts(title="12月份不同类型
商品销售情况分析"),
            yaxis_opts=opts.AxisOpts(name="销售额与利润额"),
            xaxis_opts=opts.AxisOpts(name="商品类型"),
            toolbox_opts=opts.ToolboxOpts(),
            legend_opts=opts.LegendOpts(is_show=True),
        )
    )
```

```

return c

#第一次渲染时调用load_javascript文件
bar_toolbox().load_javascript()
#展示数据可视化图表
bar_toolbox().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图4-58所示的可视化视图。

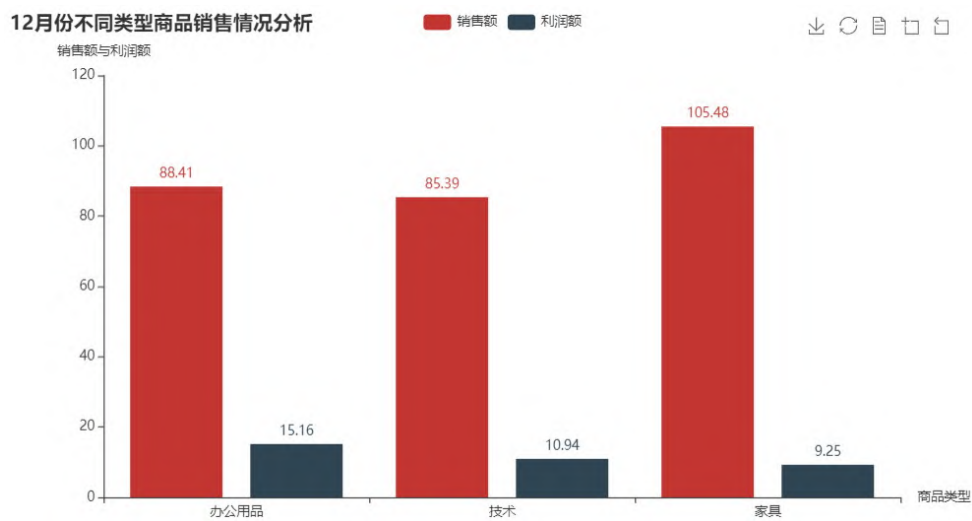


图4-58 销售额和利润额的条形图

4. 读取XLSX文件数据

Python可以直接读取XLSX文件数据，并对数据进行可视化分析，案例如下：

```

# -*- coding: utf-8 -*-
"""
连接XLSX数据文件案例
"""

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType

```



```

CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Bar, Page
import xlrd

# 打开文件
data=xlrd.open_workbook('D:/Python for Matplotlib and
pyecharts/data/orders.xlsx')
table=data.sheet_by_index(0)
# 获取总行数
nrows=table.nrows
# 获取总列数
ncols=table.ncols
# 获取一列的数值, 例如第1列
v1=table.col_values(0)
v2=table.col_values(1)
v3=table.col_values(2)

del v1[0]
del v2[0]
del v3[0]

#画图形
def bar_toolbox() -> Bar:
    c=(
        Bar()
        .add_xaxis(v1)
        .add_yaxis("销售额", v2)
        .add_yaxis("利润额", v3)
        #.reversal_axis()          #翻转XY轴, 水平条形图
        .set_global_opts(
            title_opts=opts.TitleOpts(title="12月份不同类型
商品销售情况分析"),
            yaxis_opts=opts.AxisOpts(name="销售额与利润额"),
            xaxis_opts=opts.AxisOpts(name="商品类型"),
            toolbox_opts=opts.ToolboxOpts(),
            legend_opts=opts.LegendOpts(is_show=True),
        )
    )
    return c

#第一次渲染时调用load_javascript文件
bar_toolbox().load_javascript()
#展示数据可视化图表
bar_toolbox().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图4-59所示的可视化视图。

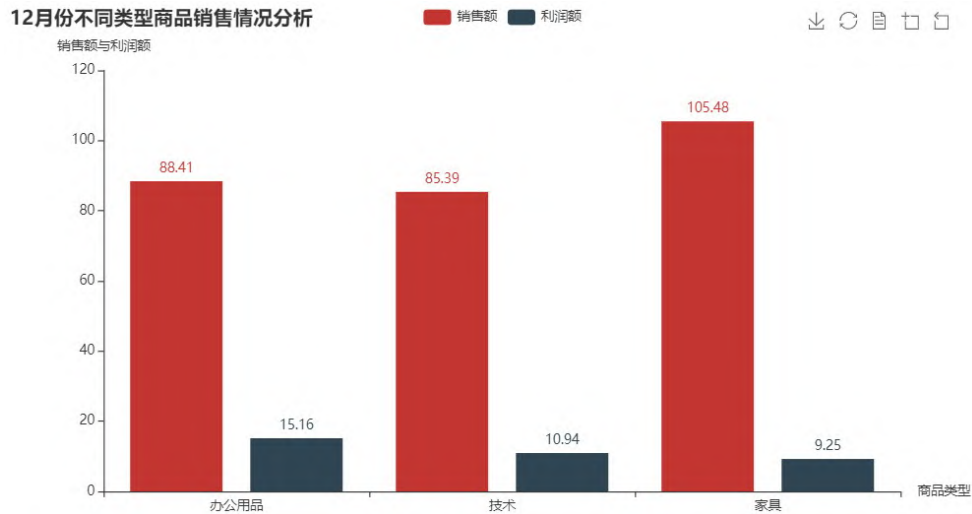


图4-59 销售额和利润额的条形图

4.3.2 连接关系型数据库

1. 读取MySQL数据库数据

Python可以直接读取MySQL数据库的数据，并对数据进行可视化分析，案例如下：

```
# -*- coding: utf-8 -*-
"""
连接MySQL数据库案例
"""

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
```

```

from pyecharts.charts import Bar, Page
from pymysql import *

#连接MySQL数据库
v1=[]
v2=[]
v3=[]

conn=pymysql.connect(host='127.0.0.1',port=3306,user='root',password='root',db='sales',charset='utf8')
cursor=conn.cursor()
#读取MySQL数据
sql_num="SELECT
category,ROUND(SUM(sales/10000),2),ROUND(SUM(profit/10000),2)
FROM orders where dt=2019 GROUP BY category"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])
    v3.append(s[2])

#画图形
def bar_toolbox() -> Bar:
    c=(
        Bar()
        .add_xaxis(v1)
        .add_yaxis("销售额", v2)
        .add_yaxis("利润额", v3)
        #.reversal_axis()    #翻转XY轴，水平条形图
        .set_global_opts(
            title_opts=opts.TitleOpts(title="2019年不同类型
商品销售情况分析"),
            yaxis_opts=opts.AxisOpts(name="销售额与利润额"),
            xaxis_opts=opts.AxisOpts(name="商品类型"),
            toolbox_opts=opts.ToolboxOpts(),
            legend_opts=opts.LegendOpts(is_show=True),
        )
    )
    return c

#第一次渲染时调用load_javascript文件
bar_toolbox().load_javascript()
#展示数据可视化图表
bar_toolbox().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图4-60所示的可视化视图。

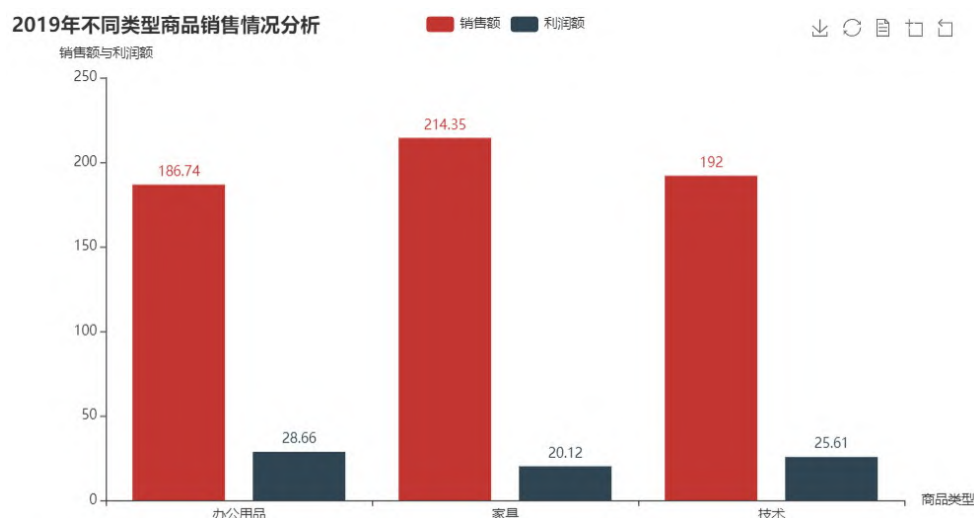


图4-60 销售额和利润额的条形图

2. 读取SQL Server数据库数据

Python可以直接读取SQL Server数据库的数据，并对数据进行可视化分析，案例如下：

```
# -*- coding: utf-8 -*-
"""
连接SQL Server数据库案例
"""

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Bar, Page
import pymssql

#连接SQL Server数据库
v1=[]
v2=[]
```

```

v3=[]

conn=pymssql.connect(host='192.168.1.107',user='sa',password=
'Wren2014',database='sales',charset='utf8')
    cursor=conn.cursor()
    #读取SQL Server数据
    sql_num="SELECT
manager,ROUND(SUM(sales/10000),2),ROUND(SUM(profit/10000),2)
FROM orders where dt=2019 GROUP BY manager"
    cursor.execute(sql_num)
    sh=cursor.fetchall()
    for s in sh:
        v1.append(s[0])
        v2.append(s[1])
        v3.append(s[2])

#画图形
def bar_toolbox() -> Bar:
    c=(
        Bar()
        .add_xaxis(v1)
        .add_yaxis("销售额", v2)
        .add_yaxis("利润额", v3)
        #.reversal_axis()    #翻转XY轴，水平条形图
        .set_global_opts(
            title_opts=opts.TitleOpts(title="2019年区域销售
经理业绩分析"),
            yaxis_opts=opts.AxisOpts(name="销售额与利润额"),
            xaxis_opts=opts.AxisOpts(name="销售经理"),
            toolbox_opts=opts.ToolboxOpts(),
            legend_opts=opts.LegendOpts(is_show=True),
        )
    )
    return c

#第一次渲染时调用load_javascript文件
bar_toolbox().load_javascript()
#展示数据可视化图表
bar_toolbox().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图4-61所示的可视化视图。

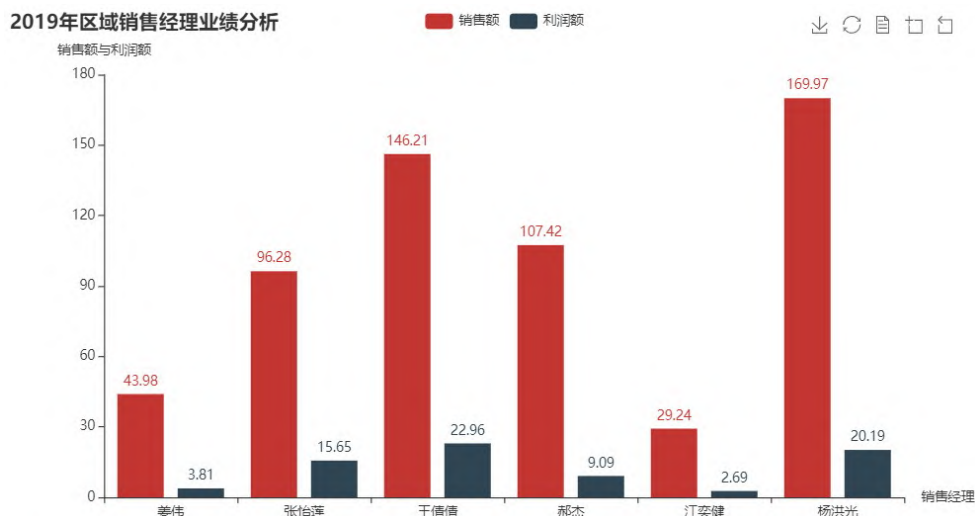


图4-61 销售额和利润额的条形图

4.3.3 连接Hadoop集群

Python可以直接读取Hadoop集群中的非结构化数据，并对数据进行可视化分析，需要安装依赖的包impyla、thrift、thriftty2，然后执行连接程序即可。

连接Hadoop集群，并生成有效订单与有效客户的直方图，案例代码如下：

```
# -*- coding: utf-8 -*-
"""
连接Hadoop集群案例
"""

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Bar, Page
from impala.dbapi import connect

#连接Hadoop数据库
```

```

v1=[]
v2=[]
v3=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()

#读取Hadoop表数据
sql_num="SELECT region,count(distinct
order_id),count(distinct cust_id) FROM orders WHERE dt=2019
and return=0 GROUP BY region"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])
    v3.append(s[2])

#画直方图
def bar_toolbox() -> Bar:
    c=(
        Bar()
        .add_xaxis(v1)
        .add_yaxis("有效订单数", v2)
        .add_yaxis("有效客户数", v3, is_selected=True)
#is_selected默认是False, 即不选中
        .set_global_opts(
            title_opts=opts.TitleOpts(title="有效订单与有效
客户分析", subtitle="2019年企业经营状况分析"),
            toolbox_opts=opts.ToolboxOpts(),
            legend_opts=opts.LegendOpts(is_show=True)
        )
    )
    return c

#第一次渲染时调用load_javascript文件
bar_toolbox().load_javascript()
#展示数据可视化图表
bar_toolbox().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图4-62所示的可视化视图。

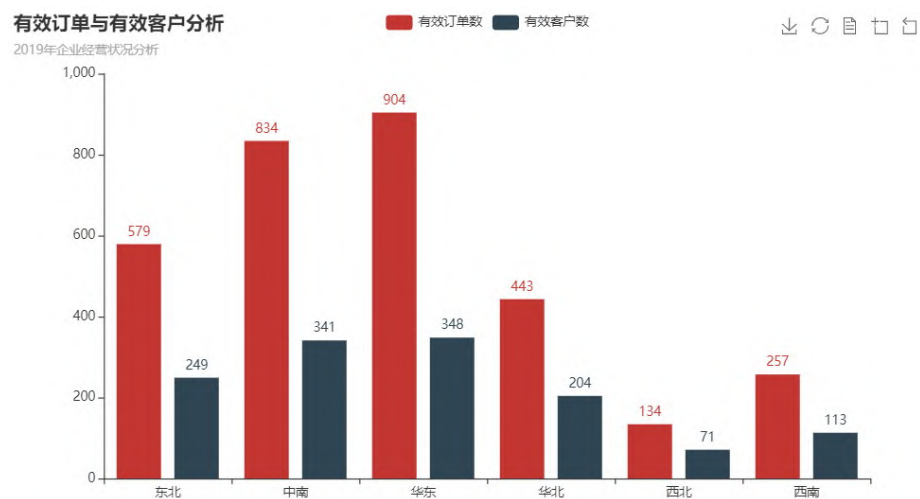


图4-62 销售额和利润额的条形图

第5章

Python数据可视化库



本章我们将结合实际案例介绍Python的主要数据可视化库，包括Matplotlib、Pycharts、Seaborn、ggplot、Bokeh、Pygal、Plotly、NetworkX等。

5.1 Matplotlib

5.1.1 Matplotlib库简介

Matplotlib是一个比较重要的Python绘图库，它基于NumPy的数组运算功能，绘图功能非常强大，已经成为Python中公认的数据可视化工具，通过Matplotlib可以很轻松地画一些简单或复杂的图形，几行代码即可生成线图、直方图、功率谱、条形图、错误图、散点图等。

Python绘图库众多，各有其特点，但是Matplotlib是基础的Python可视化库，如果需要学习Python数据可视化，那么一定要学习Matplotlib，然后学习其他库进行纵横向的拓展。Matplotlib的中文学

习资料比较丰富，其中最好的学习资料还是其帮助文档，地址为：<https://www.Matplotlib.org.cn/>，读者可以在上面查阅自己感兴趣的视图类型。

安装Anaconda后，会默认安装Matplotlib库，如果要单独安装，那么可以通过pip命令实现，前提是需要先安装pip包，命令为：pip install Matplotlib。

5.1.2 Matplotlib可视化案例

下面演示一个比较简单的Matplotlib数据可视化的例子。例如，需要按照组和性别统计某次考核的成绩，通过条形图对结果进行可视化，具体代码如下：

```
# -*- coding: utf-8 -*-#

import Numpy as np
import Matplotlib.pyplot as plt

#图形显示中文
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False

N=5      #组数
menMeans=(20, 35, 30, 35, 27)
womenMeans=(25, 32, 34, 20, 25)
menStd=(2, 3, 4, 1, 2)
womenStd=(3, 5, 2, 3, 3)
ind=np.arange(N)      #组的位置
width=0.35             #条形图的宽度

p1=plt.bar(ind, menMeans, width, yerr=menStd)
p2=plt.bar(ind, womenMeans, width,
            bottom=menMeans, yerr=womenStd)

plt.ylabel('得分')
plt.title('按照组和性别统计得分')
plt.xticks(ind, ('组1', '组2', '组3', '组4', '组5'))
```

```
plt.yticks(np.arange(0, 81, 10))
plt.legend((p1[0], p2[0]), ('男', '女'))

plt.show()
```

通过运行上面的代码可以绘制出学生考试成绩的条形图，如图5-1所示。从图中可以清楚地看出每个组的得分情况，以及每个组中男女的得分情况。

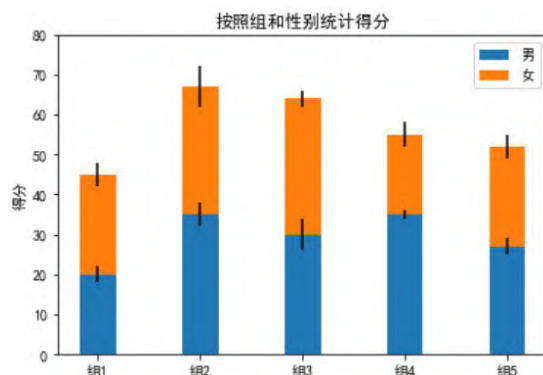


图5-1 得分的条形图

前面只是简单地举例说明Matplotlib的绘图过程，在本书的第二部分Matplotlib数据可视化中，我们还将深入讲解Matplotlib可视化方面的应用及技巧等。

5.2 Pyecharts

5.2.1 Pyecharts库简介

Pyecharts是一个用于生成Echarts图表的类库，可以与Python进行对接，方便在Python中直接使用数据生成图。Echarts是百度开源的一个数据可视化JS库，生成的图可视化效果非常棒，凭借着良好的交互性，精巧的图表设计，得到了众多开发者的认可。

Pyecharts分为v0.5.X和v1两个大版本，v0.5.X和v1间不兼容，v1是一个全新的版本。Pyecharts经过了半年的沉寂后，终于发布了新版本，新版本号将从v1.0.0开始，这是一个全新的、向下不兼容的Pyecharts版本，类似于Python 3与Python 2。不过，如果开发者以前接触过Pyecharts，新版本就很容易上手。

截至2019年11月，Pyecharts的最新版本为1.5.1，具有以下特点：

1. 全面拥抱Python 3和TypeHint

Pyecharts v1停止对Python 2.7及Python 3.4~3.5版本的支持和维护，仅支持Python 3.6+。如果还不知道什么是TypeHint的同学，建议尽早学习。

2. 弃用插件机制

Pyecharts v1废除原有的插件机制，包括地图包插件和主题插件，插件的本质是提供Pyecharts运行所需要的静态资源文件，所以现在开放了两种模式提供静态资源文件：online模式，使用Pyecharts官方提供的assetshost，或者部署自己的remotehost；local模式，使用自己本地开启的文件服务提供assetshost。

3. 更加轻量级

新版本的Pyecharts只依赖两个第三方库：jinja2和prettytable。这意味着Pyecharts总体的体积将变小，安装更加轻松，也可以很方便地进行离线安装，配合前面介绍的local模式。

4. 支持原生JavaScript

Pyecharts 0.5.X版本对原生JavaScript的支持还很局限，v1版本彻底打通了任督二脉，支持传入任意的JavaScript代码及任意的配置项回调函数。

5. 支持Jupyter Lab

Pyecharts对Jupyter Lab的支持一直是很多开发者关心的功能，毕竟Jupyter Lab号称是下一代的Notebook。Pyecharts v1开始支持在Jupyter Lab中渲染图表。

6. 代码风格重构

所有配置项均为面向对象程序设计（OOP），在新版本的Pyecharts中，一切皆选项，配置项种类更多，可操作性更强，可以画出更丰富的图表。Pyecharts官方画廊 [pyecharts/pyecharts-gallery](https://pyecharts.org/#/zh-cn/gallery)。

7. 支持Selenium/PhantomJS渲染图片

不是必需的，无此需求的开发者可忽略，并不会影响正常的使用。Pyecharts v1提供两种模式渲染图片：Selenium和PhantomJS，分别需要安装snapshot-selenium和snapshot-phantomjs。

8. 新增更多的图表类型

新增了图表类型和组件类型，如旭日图、百度地图等。

5.2.2 Pyecharts可视化案例

下面演示一个比较简单的Matplotlib数据可视化的例子。例如，需要按照组和性别统计某次考核的成绩，通过条形图对结果进行可视化，具体代码如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Bar, Page
from impala.dbapi import connect

bar=Bar()
bar.add_xaxis(["数学", "语文", "英语", "政治", "历史", "地理",
"物理", "化学", "生物"])
bar.add_yaxis("班级A", [134, 125, 127, 89, 95, 87, 85, 88,
89])
bar.add_yaxis("班级B", [131, 128, 129, 87, 92, 88, 86, 85,
92])
bar.set_global_opts(title_opts=opts.TitleOpts(title="学生考
试成绩比较分析", subtitle="2019年班级A和班级B期末考试"),
)

#第一次渲染时调用load_javascript文件
bar.load_javascript()

#展示数据可视化图表
bar.render_notebook()
```

通过运行上面的代码可以绘制出学生考试成绩的条形图，如图5-2所示，从图中可以清楚地看出班级A和班级B各科平均得分情况。

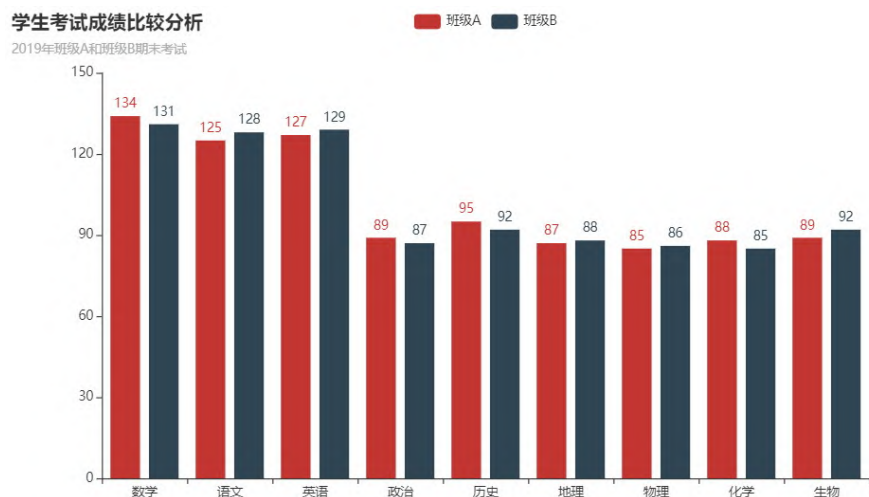


图5-2 学生考试成绩分析

上面只是简单地举例说明Pyecharts的绘图过程，在本书的第三部分Pyecharts数据可视化中，我们还将深入讲解Pyecharts在可视化方面的应用及技巧等。

5.3 Seaborn

5.3.1 Seaborn库简介

Seaborn同Matplotlib一样，也是Python进行数据可视化分析的重要第三方包。但Seaborn在Matplotlib的基础上进行了更高级的API封装，使得作图更加容易，图形更加漂亮。Seaborn是基于Matplotlib产生的一个模块，专攻统计可视化，可以和Pandas进行无缝链接，使初学者更容易上手。相对于Matplotlib，Seaborn语法更简洁，两者的关系类似于NumPy和Pandas之间的关系。

但是应该强调的是，应该把Seaborn视为Matplotlib的补充，而不是替代物。

安装Anaconda后，会默认安装Seaborn库，如果要单独安装，那么可以通过pip命令实现，前提是需要先安装pip包，代码为：pip install seaborn。

5.3.2 Seaborn可视化案例

下面演示一个Seaborn数据可视化的例子。例如，如果要分析某企业每个月利润额的分布情况，可以通过直方图和密度图的形式进行可视化分析，数据存储在本地的MySQL数据库中，数据库名为sales，使用的表为订单表orders，具体代码如下：

```
# -*- coding: utf-8 -*-

import Numpy as np
import Pandas as pd
import Matplotlib.pyplot as plt
from Pandas import Series, DataFrame
import seaborn as sns
import pymysql

plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False

#连接MySQL数据库，数据库为sales
v1=[]
v2=[]

conn=pymysql.connect(host='127.0.0.1',port=3306,user='root',password='root',db='sales',charset='utf8')
cursor=conn.cursor()

#读取数据库中的数据，数据表为orders
sql_num="SELECT
year(order_date),MONTH(order_date),ROUND(SUM(profit)/10000,2)
FROM orders GROUP BY year(order_date),MONTH(order_date)"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
```



```

v2.append(s[2])

#直方图和密度图
sns.distplot(v2,hist=True,kde=True,rug=True)          #前两个默
认就是True, rug是在最下方显示出频率情况, 默认为False

# bins=20 表示等分为20份的效果, 同样有label等参数
sns.kdeplot(v2,shade=True,color='r')      #shade表示线下颜色为
阴影, color表示颜色是红色
sns.rugplot(v2)          #在下方画出频率情况

```

通过运行上面的代码可以绘制出该企业近3年每个月利润额数据的直方图和密度图, 如图5-3所示, 从图中可以看出利润额基本呈现正态分布, 平均值在5.80附近。

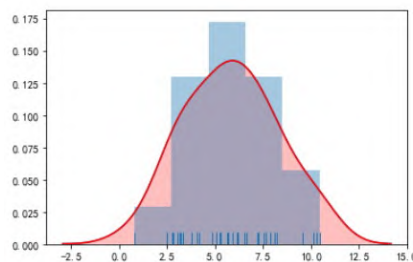


图5-3 企业月利润额分析

5.4 ggplot

5.4.1 ggplot库简介

ggplot是基于R的ggplot 2和Python的绘图系统。它的构建是为了用最少的代码快速绘制既专业又美观的图表。ggplot与Python中的Pandas有着共生关系。如果打算使用ggplot, 那么最好将数据保存在DataFrames中, 即若想使用ggplot, 则先将数据转化为DataFrame形式。

注意，在安装好ggplot后，如果直接导入ggplot，即import ggplot，就会出现错误信息：AttributeError: module 'Pandas' has no attribute 'tslib'。这是由于ggplot环境配置的不兼容导致的，我们需要修改smoothers.py和utils.py两个文件，具体修改方法如下：

打开 smoothers.py ，该文件位于 Anaconda3\Lib\site-packages\ggplot\stats下，注释掉下面的代码，并增加相应的语句（注意下面仅展示了部分代码）：

```
from __future__ import (absolute_import, division,
print_function,unicode_literals)
import Numpy as np
from Pandas._libs.tslib import Timestamp      #需要添加的
#from Pandas.lib import Timestamp            #需要被注释掉的
的
import Pandas as pd
import statsmodels.api as sm
from statsmodels.nonparametric.smoothers_lowess import
lowess as smlowess
from statsmodels.sandbox.regression.predstd import
wls_prediction_std
from statsmodels.stats.outliers_influence import
summary_table
import scipy.stats as stats
import datetime

date_types=(
    Timestamp,                        #需要添加的
    #pd.tslib.Timestamp,              #需要被注释掉的
    pd.DatetimeIndex,
    pd.Period,
    pd.PeriodIndex,
    datetime.datetime,
    datetime.time
)
```

此外，还需要修改 utils.py 文件，该文件位于 Anaconda3\Lib\site-packages\ggplot 下，注释掉部分代码，并增加相应的语句（注意下面仅展示了部分代码）：

```
from __future__ import (absolute_import, division,
print_function,unicode_literals)
from Pandas._libs.tslib import Timestamp as ts          #需
要添加的
import Matplotlib.cbook as cbook
import Numpy as np
import Pandas as pd
import datetime

date_types=(
    ts,                                     #需要添加的
    #pd.tslib.Timestamp,                  #需要被注释掉的
    pd.DatetimeIndex,
    pd.Period,
    pd.PeriodIndex,
    datetime.datetime,
    datetime.time
)
```

通过上面对smoothers.py和utils.py两个文件的修改，我们就可以正常导入ggplot库了。

5.4.2 ggplot可视化案例

下面演示一个简单的ggplot数据可视化的例子。例如，需要分析某企业2019年销售额情况，可以通过按销售日期绘制折线图的方法进行可视化分析，具体代码如下：

```
# -*- coding: utf-8 -*-

import Numpy as np
```

```

import Pandas as pd
import Matplotlib.pyplot as plt
from Pandas import Series, DataFrame
import ggplot as gp
import pymysql
plt.rcParams['font.sans-serif']=['SimHei']    #中文字体设置

#连接MySQL数据库
v1=[]
v2=[]

conn=pymysql.connect(host='127.0.0.1',port=3306,user='root',p
assword='root',db='sales',charset='utf8')
cursor=conn.cursor()

#读取MySQL数据
sql_num="SELECT order_date,ROUND(SUM(sales)/10000,2) FROM
orders where dt=2019 GROUP BY order_date"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])

meat =pd.DataFrame(list(sh),index=v1,columns=['订单日
期','销售额'])
p=gp.ggplot(gp.aes(x='订单日期',y='销售
额'),data=meat)+gp.geom_point(color='red')+gp.geom_line(color
='blue')+gp.ggtitle(u'2019年企业销售额折线图')
print(p)

```

通过运行上面的代码可以绘制出该企业2019年销售额的折线图，如图5-4所示。从图中可以看出该企业的日销售额变化较大，热销的时候销售额接近7万元，淡季的时候几乎没有什么客户。

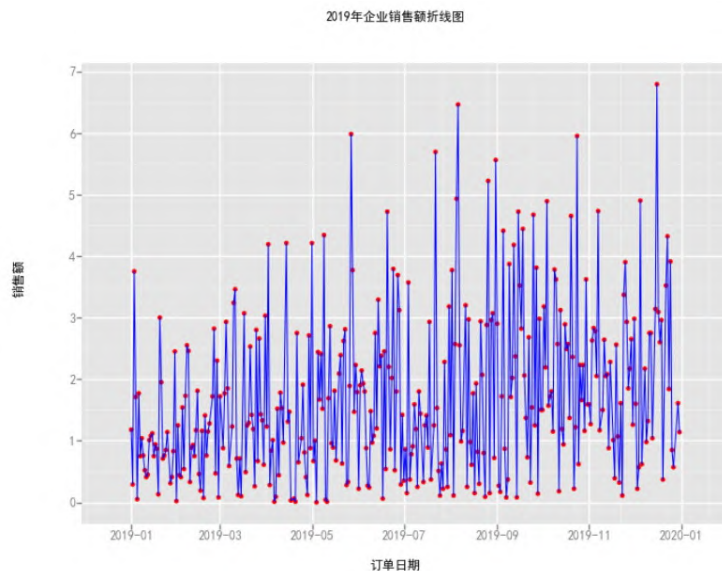


图5-4 企业销售额折线图

5.5 Bokeh

5.5.1 Bokeh库简介

Bokeh是基于JavaScript来实现的交互可视化库，它可以在Web浏览器中实现美观的视觉效果。但是它也有明显的缺点：其一是版本时常更新，重要的是有时语法还不向下兼容；其二是语法晦涩，与Matplotlib相比，可以说是有过之而无不及。

Bokeh的优势：

- Bokeh允许通过简单的指令快速创建复杂的统计图。
- Bokeh提供到各种媒体（如HTML、Notebook文档和服务器的输出）的输出。
- 我们也可以将Bokeh可视化嵌入Flask和Django程序。
- Bokeh可以转换写在其他库（如Matplotlib、Seaborn）中的可视化。

- Bokeh能灵活地将交互式应用、布局和不同样式选择用于可视化。

Bokeh面临的挑战:

- 与任何即将到来的开源库一样, Bokeh正在经历不断地变化和发展。所以, 今天写的代码可能将来并不能被完全再次使用。
- 与D3.js相比, Bokeh的可视化选项相对较少。因此, 短期内Bokeh无法挑战D3.js的霸主地位。

5.5.2 Bokeh可视化案例

下面演示一个简单的Bokeh数据可视化的例子。例如, 需要分析某企业2019年的经营状况, 可以通过绘制销售额和利润额折线图的方法进行可视化分析, 具体代码如下:

```
# -*- coding: utf-8 -*-

import Numpy as np
import Pandas as pd
import Matplotlib.pyplot as plt
from Pandas import Series, DataFrame
from bokeh.plotting import figure, show
import pymysql
plt.rcParams['font.sans-serif']=['SimHei']    #中文字体设置

#连接MySQL数据库
v1=[]
v2=[]
v3=[]

conn=pymysql.connect(host='127.0.0.1',port=3306,user='root',password='root',db='sales',charset='utf8')
cursor=conn.cursor()

#读取订单表数据
```

```

    sql_num="SELECT
MONTH(order_date),ROUND(SUM(sales)/10000,2),ROUND(SUM(profit)
/10000,2) FROM orders where dt=2019 GROUP BY
MONTH(order_date)"
    cursor.execute(sql_num)
    sh=cursor.fetchall()
    for s in sh:
        v1.append(s[0])
        v2.append(s[1])
        v3.append(s[2])

    p=figure(width=800, height=400, title="2019年企业经营状况分
析")
    p.xaxis.axis_label="月份"
    p.xaxis.axis_label_text_color="violet"
    p.yaxis.axis_label="销售额与利润额"
    p.yaxis.axis_label_text_color="violet"
    dashes=[12, 4]
    listx1=v1
    listy1=v2
    p.line(listx1, listy1, line_width=4, line_color="red",
line_alpha=0.3, line_dash=dashes, legend="销售额")
    listx2=v1
    listy2=v3
    p.line(listx2, listy2, line_width=4, legend="利润额")
    show(p)

```

通过运行上面的代码会弹出一个新的Web页面，绘制出该企业2019年销售额和利润额的折线图，如图5-5所示。从图中可以看出该企业的月销售额基本呈现上升趋势，但是月利润额的变化相对很小，为什么出现销售额增加而利润额不变的现象，还需要进行更加深入的分析。

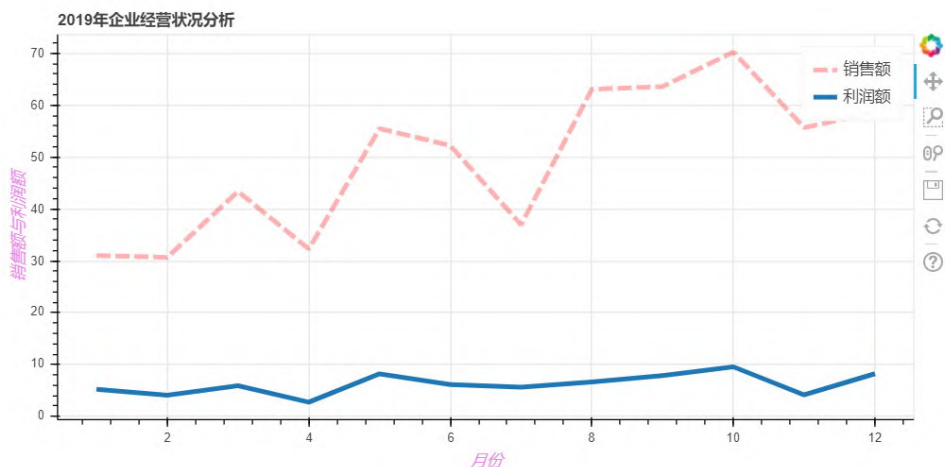


图5-5 销售额和利润额的折线图

5.6 Pygal

5.6.1 Pygal库简介

Pygal是Python中另一个简单易用的数据图库，它以面向对象的方式来创建各种数据图，而且使用Pygal可以非常方便地生成各种格式的数据图，包括PNG、SVG等。使用Pygal也可以生成XML etree、HTML表格。

对于需要在尺寸不同的屏幕上显示的图表，需要考虑使用Pygal来生成它们，因为它们将自动缩放，以适应观看者的屏幕，这样它们在任何设备上显示时都会很美观。Pygal绘制线图很简单，可以将图表渲染为一个SVG文件，使用浏览器打开SVG文件就可以查看生成的图表。

5.6.2 Pygal可视化案例

下面演示一个简单的Pygal数据可视化的例子。例如，需要分析2019年某企业每个门店的经营状况，可以通过绘制销售额和利润额折线图的方法进行可视化分析，具体代码如下：


```

# -*- coding: utf-8 -*-

import Numpy as np
import Pandas as pd
import Matplotlib.pyplot as plt
from Pandas import Series, DataFrame
import pygal
import pymysql

plt.rcParams['font.sans-serif']=['SimHei']    #中文字体设置

#连接MySQL数据库
v1=[]
v2=[]
v3=[]

conn=pymysql.connect(host='127.0.0.1',port=3306,user='root',p
assword='root',db='sales',charset='utf8')
cursor=conn.cursor()

#读取订单表数据
sql_num="SELECT
store_name,ROUND(SUM(sales)/10000,2),ROUND(SUM(profit)/10000,
2) FROM orders where dt=2019 GROUP BY store_name"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])
    v3.append(s[2])

line_chart=pygal.HorizontalLine()    #创建一个水平线图的实例化对
象
line_chart.title='销售额利润额'    #设置标题
line_chart.x_labels=v1    #注意，这里的是水平线图，那么x轴就变为
Y轴，Y轴变为X轴
# 下面添加两条线
line_chart.add('销售额', v2)
line_chart.add('利润额', v3)
line_chart.range=[0, 80]    #设置X轴的范围
line_chart.render_to_file('bar_chart.svg')    #将图像保存为
SVG文件，可通过浏览器查看

```

通过运行上面的代码会自动生成一个SVG文件，可以通过浏览器查看图表，绘制出2019年该企业每个门店的销售额和利润额的折线图，如图5-6所示。从图中可以看出门店的销售额差异较大，但是利润额的差异相对很小，这可能与每个门店的销售策略有关。

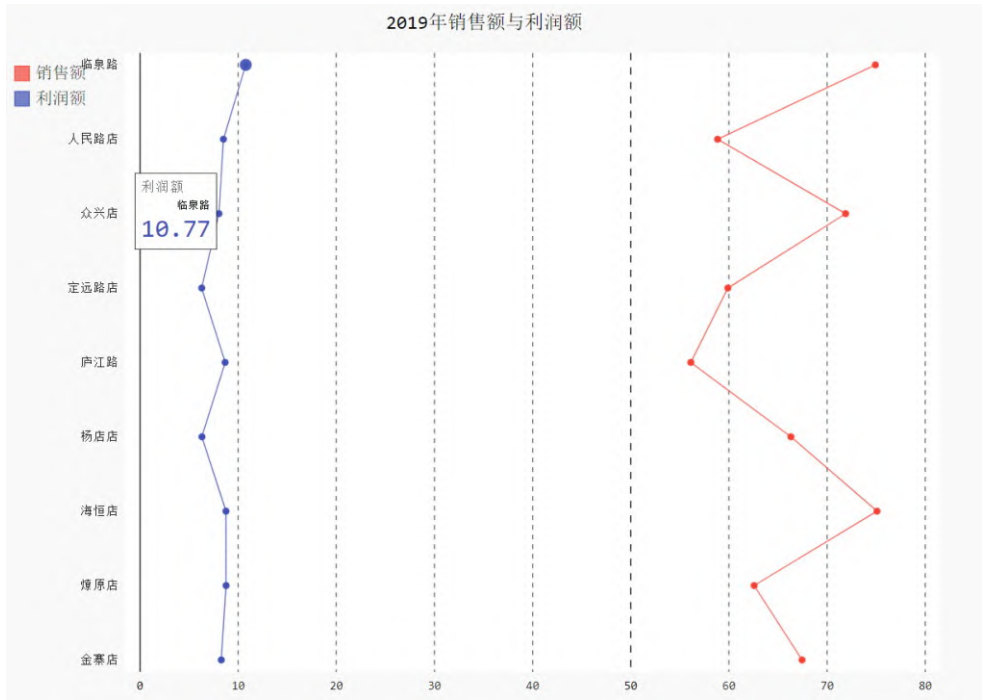


图5-6 销售额和利润额的折线图

5.7 Plotly

5.7.1 Plotly库简介

Plotly是Python中一个进行可视化交互的库。它不仅支持Python语言，还支持R语言。Plotly的优点是能提供Web在线交互，配色也好看。如果你是一名数据分析师，Plotly强大的交互功能能助你完成展示。

2019年7月，Plotly团队发布了Plotly.py 4.0版本，现在可从PyPI下载。此版本包括一些令人兴奋的新功能和更改，包括默认情况下切换到“离线”模式、Plotly Express作为库中的推荐入口点以及新的渲染框架，不仅兼容Jupyter Notebook，还兼容其他Notebook系统，例如Colab、Azure和Kaggle Notebook，以及PyCharm、VSCode、Spyder等流行的IDE。

用于在“在线”和“离线”模式下创建图形的功能。在“在线”模式下，数字被上传到Plotly的Chart Studio服务的实例，然后显示；而在“离线”模式下，数字在本地呈现。这种二元性是多年来混淆的常见原因，因此在第4版中，团队进行了一些重要的更改以帮助明确这一点。

在这个版本中，plotly包中唯一支持的操作模式是“离线”模式，它不需要互联网连接，没有账户，没有身份验证令牌，也没有任何类型的付款。对“在线”模式的支持已经转移到一个名为chart-studio的单独安装的软件包中。

5.7.2 Plotly可视化案例

下面演示一个简单的Plotly数据可视化的例子。例如，需要分析2019年某企业在全国各个区域的经营状况，可以通过绘制条形图的方法进行分析，具体代码如下：

```
# -*- coding: utf-8 -*-

import Numpy as np
import Pandas as pd
import Matplotlib.pyplot as plt
from Pandas import Series, DataFrame
#import chart_studio.plotly
import plotly.graph_objs as go
import pymysql
```

```

plt.rcParams['font.sans-serif']=['SimHei']    #中文字体设置

#连接MySQL数据库
v1=[]
v2=[]
v3=[]

conn=pymysql.connect(host='127.0.0.1',port=3306,user='root',password='root',db='sales',charset='utf8')
cursor=conn.cursor()

#读取订单表数据
sql_num="SELECT
region,ROUND(SUM(sales)/10000,2),ROUND(SUM(profit)/10000,2)
FROM orders where dt=2019 GROUP BY region"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])
    v3.append(s[2])

#按区域绘制条形图
date_sales=pg.Bar(x=v1, y=v2, name='销售额')
date_profit=pg.Bar(x=v1, y=v3, name='利润额')
data=[date_sales, date_profit]
layout=pg.Layout(barmode='group', title="2019年区域业绩分析")
fig=pg.Figure(data=data, layout=layout)
fig.write_html("2019年区域业绩分析.html")

```

通过运行上面的代码会自动生成一个HTML文件，可以通过浏览器查看图表，绘制出2019年该企业在全国各个区域的销售额和利润额的条形图，在图形的右上方有相应的编辑工具，如图5-7所示。

从图中可以看出在华东地区的销售额最多，其次是中南地区，但是在利润额方面，却是中南地区最多，其次是华东地区，这和每个地区的销售成本大小有关。在2019年，该企业在华东地区加大了营销推广力度，但是短期收效却不是很明显。

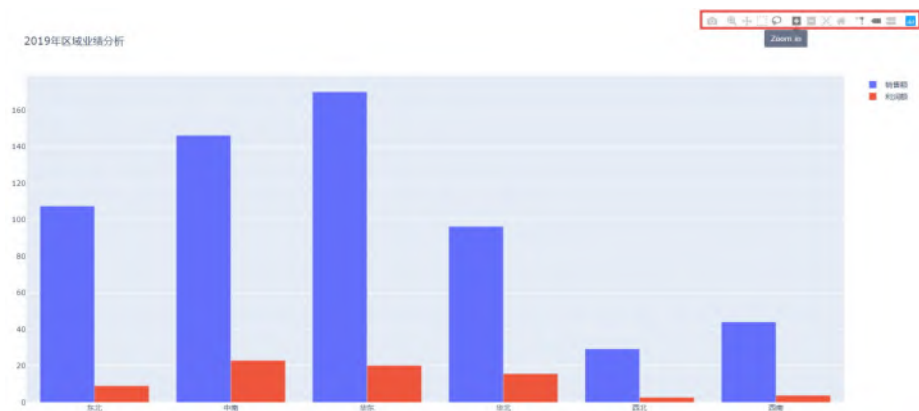


图5-7 销售额和利润额的条形图

5.8 其他可视化库

5.8.1 Altair

Altair是Python的一个公认的统计可视化库。它的API简单、友好、一致，并建立在强大的Vega - Lite（交互式图形语法）之上。Altair API不包含实际的可视化呈现代码，而是按照Vega - Lite规范发出JSON数据结构，由此产生的数据可以在用户界面中呈现，这种优雅的简单性产生了漂亮且有效的可视化效果，且只需很少的代码。

Altair可视化的数据源需要是DataFrame格式的，它可以由不同数据类型的列组成。DataFrame是一种整洁的格式，其中的行与样本相对应，而列与观察到的变量相对应。数据通过数据转换映射到使用组的视觉属性（位置、颜色、大小、形状、面板等）。

在可视化分析之前，首先需要通过pip命令安装altair包，否则程序会报缺少该包的错误。例如，使用altair包分析农产品的产量与平均增长率两者之间的关系，可以绘制散点图的方式进行可视化分析，具体代码如下：

```

#农产品的产量与平均增长率的散点图
import altair as alt
import Pandas as pd

data=pd.DataFrame({'农产品名称': ['粮食', '棉花', '油料', '肉类', '水产品'],
                    '产量(万吨)': [66160.7, 565.3, 3475.2, 8654.4, 6445.3],
                    '平均增长率': [2.1, 1.5, 1.0, 2.2, 3.3]})
c=alt.Chart(data)
c=c.mark_point(size=300)
c=c.encode(x='产量(万吨):Q',y='平均增长率:Q',
           color='农产品名称:N',
           tooltip=['农产品名称', '产量(万吨)', '平均增长率'])
c.serve()
c.display()

```

在Jupyter Lab中运行该代码会自动打开一个新的浏览器页面，并生成如图5-8所示的散点图。从图中可以清楚地看出农产品的产量与平均增长率的关系，此外单击页面右上方的 $\cdots$ 按钮，还可以将图片保存为指定的格式，以及查看源代码等。

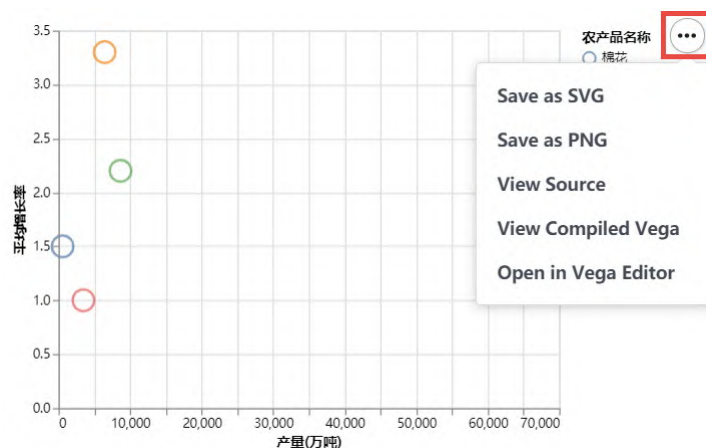


图5-8 农产品产量与平均增长率散点图

5.8.2 PyQtGraph

PyQtGraph是Python的图形和用户界面库，它充分利用PyQt和PtSide的高质量图形表现水平和NumPy的快速科学计算与处理能力，在数学、科学和工程领域都有广泛的应用。其主要目标是：为数据（绘图、视频等）提供快速可交互的图形显示，提供帮助快速开发应用程序的工具（例如Qt Designer中使用的属性树）。

PyQtGraph被大量应用于Qt GUI平台（通过PyQt或PySide），因为它的高性能图形以及NumPy可用于大量数据处理。特别需要注意的是，PyQtGraph使用了Qt的GraphicsView框架，它本身是一个功能强大的图形系统，我们将优化和简化的语句应用到这个框架中，以最小的工作量实现数据可视化。

对于绘图而言，PyQtGraph几乎不像Matplotlib那么完整或者成熟，但是运行速度更快。Matplotlib的目标更多是制作出版质量的图形，而PyQtGraph则用于数据采集和分析应用。Matplotlib对于Matlab程序员来说更直观。PyQtGraph对Python/Qt程序员更直观。Matplotlib并不包含许多PyQtGraph的功能，如图像交互、体绘制、参数树、流程图等。

在可视化分析之前，首先需要通过pip命令安装pyqtgraph包，否则程序会报缺少该包的错误。例如，使用pyqtgraph包绘制折线图，具体代码如下：

```
# -*- coding: utf-8 -*-#

import Numpy as np
import pyqtgraph as pg

#生成数据并绘图
data=np.random.normal(size=100)
pg.plot(data,title="绘制一个简单的可视化视图")
```

```
#启动Qt事件循环
if __name__ == '__main__':
    import sys
    if sys.flags.interactive != 1 or not hasattr(QtCore,
'PYQT_VERSION'):
        pg.QtGui.QApplication.exec_()
```

在Jupyter Lab中运行该代码会自动弹出一个新的页面，并生成如图5-9所示的折线图。从图中可以看出数据的波动性，以及基本的分布情况。

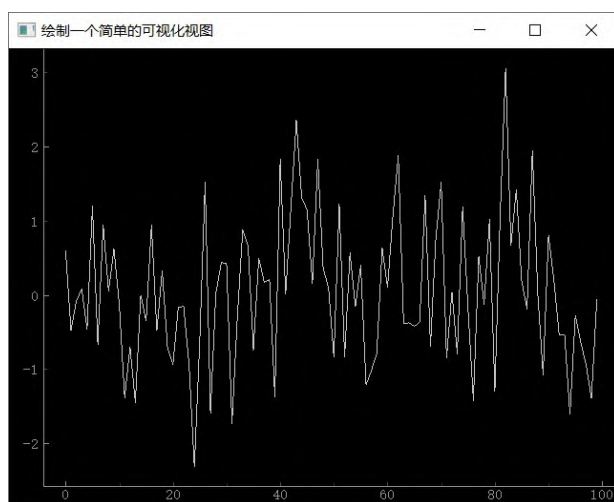


图5-9 折线图

5.8.3 NetworkX

NetworkX是一款Python的软件包，用于创造、操作复杂网络，以及学习复杂网络的结构、动力学及其功能。有了NetworkX就可以用标准或者不标准的数据格式加载或者存储网络，它可以产生许多种类的随机网络或经典网络，也可以分析网络结构、建立网络模型、设计新的网络算法、绘制网络等。

对于已经装了pip的环境，安装第三方模块很简单，只需要使用 `pip install networkx` 命令即可。在NetworkX中，顶点可以是任何可以哈希的对象，比如文本、图片、XML对象、其他的图对象、任意定制 的节点对象等。NetworkX画图参数如表5-1所示。

表5-1 NetworkX包参数

属 性	说 明
node_size	指定节点的尺寸大小（默认是 300，单位未知）
node_color	指定节点的颜色（默认是红色，可以用字符串简单标识颜色）
node_shape	节点的形状（默认是圆形，用字符串'o'标识，具体可查看手册）
Alpha	透明度（默认是 1.0，不透明，0 为完全透明）
Width	边的宽度（默认为 1.0）
edge_color	边的颜色（默认为黑色）
Style	边的样式（默认为实线，可选：solid dashed dotted,dashdot）
with_labels	节点是否带标签（默认为 True）
font_size	节点标签字体大小（默认为 12）
font_color	节点标签字体颜色（默认为黑色）
node_size	指定节点的尺寸大小（默认是 300，单位未知）

在可视化分析之前，首先需要通过pip命令安装networkx包，否则程序会报缺少该包的错误。例如，使用networkx包分析农产品的产量与平均增长率两者之间的关系，可以绘制散点图的方式进行可视化分析，具体代码如下：

```
# -*- coding: utf-8 -*-#

#中文字体设置
from Matplotlib import pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei']

import networkx as nx

#定义graph
nodes=['step A','step B','step C','step D','step E','step F']
edges=[('step A','step C'),('step A','step B'),('step A','step E'),('step B','step E'),('step B','step F'),('step
```

```

C','step F'),('step C','step E'),('step D','step F')]
G=nx.Graph()
G.add_nodes_from(nodes)
G.add_edges_from(edges)

#使用spring_layout布局
pos=nx.spring_layout(G)

#画网络关系图
plt.title('某项目各步骤间的网络关系图')
nx.draw_networkx(G)
plt.show()

```

在Jupyter Lab中运行上述代码，生成如图5-10所示的网络关系图。从图中可以看出步骤A、步骤B、步骤C和步骤E与其他步骤的联系比较紧密，在项目实施过程中要重点监控，以免影响其他步骤的正常进行。

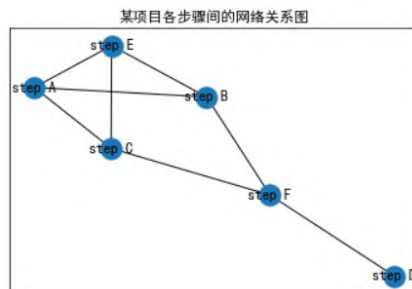


图5-10 网络关系图

第二部分

Matplotlib数据可视化

本部分我们将介绍Matplotlib，它是Python数据可视化库的泰斗，尽管已有十多年的历史，但是仍然是Python社区中使用广泛的绘图库。Matplotlib的设计与Matlab非常相似，提供了一整套和Matlab相似的命令API，适合交互式制图，还可以将它作为绘图控件嵌入其他应用程序中。

第6章

Matplotlib图形参数设置



本章介绍Matplotlib的主要参数配置，包括线条、坐标轴、图例等，以及绘图的参数文件及主要函数，并结合实际案例进行深入说明。

6.1 Matplotlib主要参数配置

6.1.1 线条的设置

在Matplotlib中，可以很方便地绘制各类图形，如果不在程序中设置参数，软件就会使用默认的参数，例如需要对输入的数据进行数据变换，并绘制曲线的案例，具体代码如下：

```
#导入绘图相关模块
import Matplotlib.pyplot as plt
import Numpy as np

#生成数据并绘图
```

```

x=np.arange(0,20,1)
y1=(x-9)**2 + 1
y2=(x+5)**2 + 8

#绘制图形
plt.plot(x,y1)
plt.plot(x,y2)

#输出图形
plt.show()

```

运行上述代码，生成如图6-1所示的简单视图。

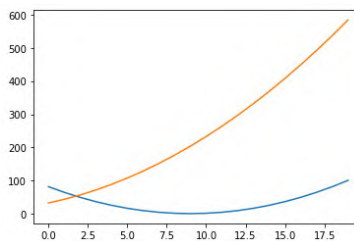


图6-1 简单曲线

上面绘制的曲线比较单调，我们可以设置线的颜色、线宽、样式以及添加点，并设置点的样式、颜色、大小。上述的数据变换案例优化后的代码如下：

```

#导入绘图相关模块
import Matplotlib.pyplot as plt
import Numpy as np

#生成数据
x=np.arange(0,20,1)
y1=(x-9)**2 + 1
y2=(x+5)**2 + 8

#设置线的颜色、线宽、样式
plt.plot(x,y1,linestyle='-.',color='red',linewidth=5.0)
#添加点，设置点的样式、颜色、大小
plt.plot(x,y2,marker='*',color='green',markersize=10)

```

```
#输出图形
plt.show()
```

运行上述代码，生成如图6-2所示的调整后的视图。

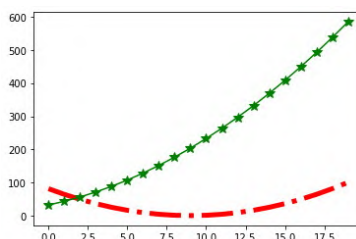


图6-2 调整后的曲线

此外，在Matplotlib中，我们可以手动设置线的颜色（color）、标记（marker）、线型（line）等参数。下面将对其进行详细介绍。

（1）线的颜色参数设置如表6-1所示。

表6-1 颜色的设置

字 符	颜 色
'b'	蓝色
'g'	绿色
'r'	红
'c'	青色
'm'	品红
'y'	黄色
'k'	黑
'w'	白色

（2）线的标记参数设置如表6-2所示。

表6-2 标记的设置

字 符	描 述
'.'	点标记
','	像素标记
'o'	圆圈标记
'v'	triangle_down 标记
'^'	triangle_up 标记
'<'	triangle_left 标记
'>'	triangle_right 标记
'1'	tri_down 标记
'2'	tri_up 标记
'3'	tri_left 标记
'4'	tri_right 标记
's'	方形标记
'p'	五角大楼标记
'*'	星形标记
'h'	hexagon1 标记
'H'	hexagon2 标记
'+'	加号标记
'x'	x 标记
'D'	钻石标记
'd'	thin_diamond 标记
' '	均标记
'_'	修身标记

(3) 线的类型参数设置如表6-3所示。

表6-3 线的设置

字 符	描 述
'_'	实线样式
'--'	虚线样式
'-.'	破折号-点线样式
':'	虚线样式

6.1.2 坐标轴的设置

Matplotlib坐标轴的刻度设置可以使用plt.xlim()和plt.ylim()函数，参数分别是坐标轴的最小值和最大值。例如，要绘制一条直

线，横轴和纵轴的刻度都在0~20之间，具体代码如下：

```
#导入绘图相关模块
import Matplotlib.pyplot as plt
import Numpy as np

#生成数据并绘图
x=np.arange(0,20,1)
y1=(x-9)**2 + 1
y2=(x+5)**2 + 8

#设置线的颜色、线宽、样式
plt.plot(x,y1,linestyle='-.',color='red',linewidth=5.0)
#添加点，设置点的样式、颜色、大小
plt.plot(x,y2,marker='*',color='green',markersize=10)

#设置x轴的刻度
plt.xlim(0,20)

#设置y轴的刻度
plt.ylim(0,400)

#输出图形
plt.show()
```

运行上述代码，生成如图6-3所示的视图。

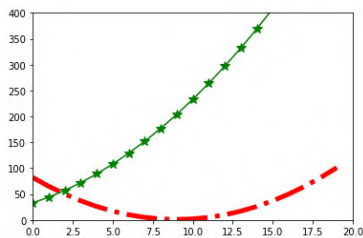


图6-3 添加坐标刻度

在Matplotlib中，可以使用plt.xlabel()函数对坐标轴的标签进行设置，其中参数xlabel设置标签的内容，size设置标签的大小，rotation设置标签的旋转度，horizontalalignment设置标签的左右位

置（分为center、right和left），verticalalignment设置标签的上下位置（分为center、top和bottom）。

例如，要绘制一条曲线，横轴的刻度在0~20之间，纵轴的刻度在0~400之间，并且为横轴和纵轴添加'x'和'y'标签，以及标签的大小、旋转度、位置等，具体代码如下：

```
#导入绘图相关模块
import Matplotlib.pyplot as plt
import Numpy as np

#生成数据并绘图
x=np.arange(0,20,1)
y1=(x-9)**2 + 1
y2=(x+5)**2 + 8

#设置线的颜色、线宽、样式
plt.plot(x,y1,linestyle='-.',color='red',linewidth=5.0)
#添加点，设置点的样式、颜色、大小
plt.plot(x,y2,marker='*',color='green',markersize=10)

#给x轴加上标签
plt.xlabel('x',size=15)

#给y轴加上标签
plt.ylabel('y',size=15,rotation=90,horizontalalignment='right',verticalalignment='center')

#设置x轴的刻度
plt.xlim(0,20)

#设置y轴的刻度
plt.ylim(0,400)

#输出图形
plt.show()
```

运行上述代码，生成如图6-4所示的视图。

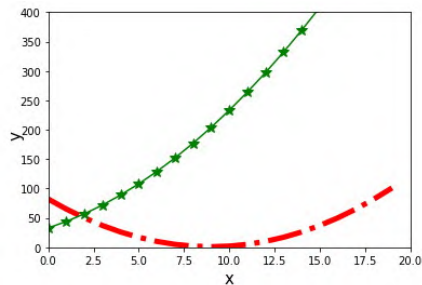


图6-4 添加坐标标签

6.1.3 图例的设置

图例是集中于地图一角或一侧的地图上各种符号和颜色所代表的内容与指标的说明，有助于更好地认识图形。

在Matplotlib中，图例的设置可以使用`plt.legend()`函数，函数参数如下：

```
plt.legend(loc, fontsize, frameon, ncol, title, shadow, markerfirst,
           markerscale, numpoints, fancybox, framealpha,
           borderpad, labelspacing, handlelength, bbox_to_anchor, *)
```

不带参数的调用`legend`会自动获取图例句柄及相关标签，例如上述数据变换的案例添加`plt.legend()`后的代码如下：

```
#导入绘图相关模块
import Matplotlib.pyplot as plt
import Numpy as np

#生成数据并绘图
x=np.arange(0,20,1)
y1=(x-9)**2 + 1
y2=(x+5)**2 + 8

#设置线的颜色、线宽、样式
```

```

plt.plot(x,y1,linestyle='-.',color='red',linewidth=5.0,label=
'convert A')
    #添加点，设置点的样式、颜色、大小

plt.plot(x,y2,marker='*',color='green',markersize=10,label='c
onvert B')

    #给x轴加上标签
plt.xlabel('x',size=15)

    #给y轴加上标签

plt.ylabel('y',size=15,rotation=90,horizontalalignment='right
',verticalalignment='center')

    #设置x轴的刻度
plt.xlim(0,20)
    #设置y轴的刻度
plt.ylim(0,400)

    #设置图例
plt.legend()

    #输出图形
plt.show()

```

运行上述代码，生成如图6-5所示的视图。

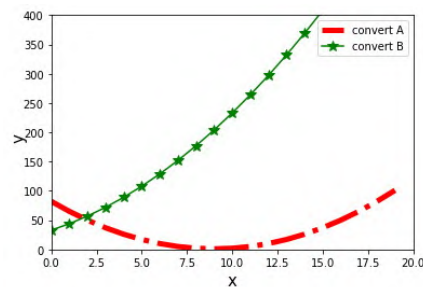


图6-5 添加视图图例

我们还可以重新定义图例的内容、位置、字体大小等参数，例如上述的 `plt.legend()` 函数可以修改为 `plt.legend(labels=['A',`

'B'],loc='upper left',fontsize=15)，运行结果如图6-6所示。

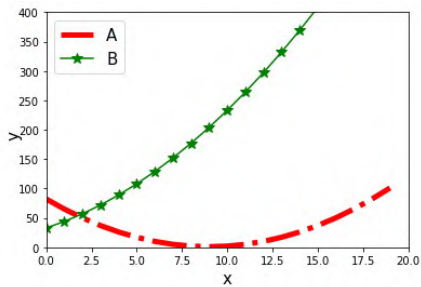


图6-6 调整图例后的视图

Matplotlib图例的主要参数配置如表6-4所示。

表6-4 图例参数配置

属 性	说 明
Loc	图例位置，如果使用了 <code>bbox_to_anchor</code> 参数，该项就无效
FontSize	设置字体大小
Frameon	是否显示图例边框
Ncol	图例的列的数量，默认为 1
Title	为图例添加标题
Shadow	是否为图例边框添加阴影
Markerfirst	True 表示图例标签在句柄右侧，False 反之
Markerscale	图例标记为原图标记中的多少倍大小
Numpoints	表示图例中的句柄上的标记点的个数，一般设为 1
Fancybox	是否将图例框的边角设为圆形
Framealpha	控制图例框的透明度
Borderpad	图例框内边距
Labelspacing	图例中条目之间的距离
Handlelength	图例句柄的长度
bbox_to_anchor	如果要自定义图例位置，就需要设置该参数

6.2 绘图参数文件及主要函数

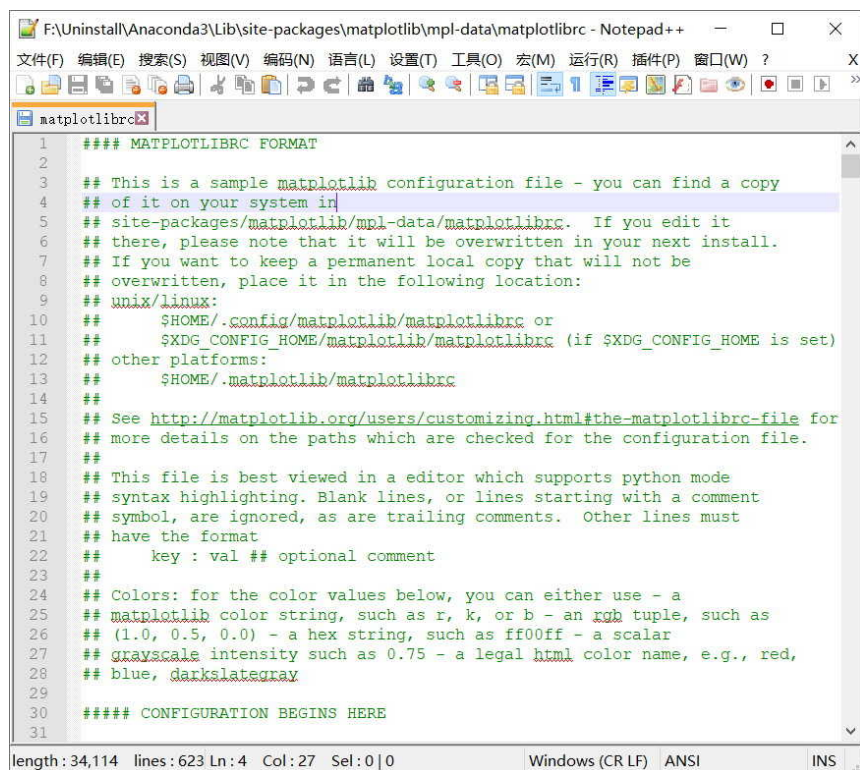
6.2.1 修改绘图参数文件

可以通过在程序中添加代码对参数进行配置，但是假设一个项目对于Matplotlib的特性参数总会设置相同的值，就没有必要在每次编写代码的时候都进行相同的配置，而应该在代码之外使用一个永久的文件设定Matplotlib参数的默认值。

在Matplotlib中，可以通过Matplotlibrc配置文件永久修改绘图参数，该文件中包含绝大部分可以变更的属性。Matplotlibrc通常在Python的site-packages目录下。不过在每次重装Matplotlib的时候，这个配置文件就会被覆盖，查看Matplotlibrc所在目录的代码为：

```
import Matplotlib
print(Matplotlib.Matplotlib_fname())
```

这里的路径是 F:\Uninstall\Anaconda3\lib\site-packages\Matplotlib\mpl-data\Matplotlibrc，具体路径由软件的安装位置决定，然后用Notepad打开Matplotlibrc文件，如图6-7所示。



```
1  ### MATPLOTLIBRC FORMAT
2
3  ## This is a sample matplotlib configuration file - you can find a copy
4  ## of it on your system in
5  ## site-packages/matplotlib/mpl-data/matplotlibrc. If you edit it
6  ## there, please note that it will be overwritten in your next install.
7  ## If you want to keep a permanent local copy that will not be
8  ## overwritten, place it in the following location:
9  ## unix/linux:
10 ##     $HOME/.config/matplotlib/matplotlibrc or
11 ##     $XDG_CONFIG_HOME/matplotlib/matplotlibrc (if $XDG_CONFIG_HOME is set)
12 ## other platforms:
13 ##     $HOME/.matplotlib/matplotlibrc
14 ##
15 ## See http://matplotlib.org/users/customizing.html#the-matplotlibrc-file for
16 ## more details on the paths which are checked for the configuration file.
17 ##
18 ## This file is best viewed in a editor which supports python mode
19 ## syntax highlighting. Blank lines, or lines starting with a comment
20 ## symbol, are ignored, as are trailing comments. Other lines must
21 ## have the format
22 ##     key : val ## optional comment
23 ##
24 ## Colors: for the color values below, you can either use - a
25 ## matplotlib color string, such as r, k, or b - an rgb tuple, such as
26 ## (1.0, 0.5, 0.0) - a hex string, such as ff00ff - a scalar
27 ## grayscale intensity such as 0.75 - a legal html color name, e.g., red,
28 ## blue, darkslategray
29
30 ##### CONFIGURATION BEGINS HERE
31
```

图6-7 Matplotlibrc文件

再根据自己的需要来修改里面相应的属性即可。注意，在修改后记得把前面的#去掉。配置文件包括以下配置项：

- axes: 设置坐标轴边界和颜色、坐标刻度值大小和网格。
- figure: 设置边界颜色、图形大小和子区。
- font: 设置字体集、字体大小和样式。
- grid: 设置网格颜色和线形。
- legend: 设置图例和其中文本的显示。
- line: 设置线条和标记。
- savefig: 可以对保存的图形进行单独设置。
- text: 设置字体颜色、文本解析等。
- xticks和yticks: 为x、y轴的刻度设置颜色、大小、方向等。

例如，在实际运用中，通常碰到中文显示为□□，那是因为没有给Matplotlib设置字体类型。如果不改变Matplotlibrc配置文件，在代码中只需要添加以下代码即可：

```
import Matplotlib.pyplot as plt
# 用来正常显示中文标签
plt.rcParams['font.sans-serif']=['SimHei']
# 用来正常显示负号
plt.rcParams['axes.unicode_minus']=False
```

如果不想这么麻烦，每次在使用Matplotlib的时候都要写上面的代码，就可以使用修改Matplotlibrc配置文件的方法。

6.2.2 绘图主要函数简介

Matplotlib中的pyplot模块提供一系列类似Matlab的命令式函数。每个函数可以对图形对象进行一些改动，比如新建一个图形对象、在图形中开辟绘制区、在绘制区画一些曲线、为曲线打上标签等。在Matplotlib.pyplot中，大部分状态是跨函数调用共享的。因此，它会跟踪当前图形对象和绘制区，绘制函数直接作用于当前绘制对象。

pyplot的基础图表函数如表6-5所示。

表6-5 基础图表函数

函 数	说 明
plt.plot()	绘制坐标图
plt.boxplot()	绘制箱形图
plt.bar()	绘制条形图
plt.barh()	绘制横向条形图
plt.polar()	绘制极坐标图
plt.pie()	绘制饼图
plt.psd()	绘制功率谱密度图
plt.specgram()	绘制谱图
plt.cohere()	绘制相关性函数
plt.scatter()	绘制散点图
plt.step()	绘制步阶图
plt.hist()	绘制直方图
plt.contour()	绘制等值图
plt.vlines()	绘制垂直图
plt.stem()	绘制柴火图
plt.plot_date()	绘制数据日期
plt.clabel()	绘制轮廓图
plt.hist2d()	绘制 2D 直方图
plt.quiverkey()	绘制颤动图
plt.stackplot()	绘制堆积面积图
plt.Violinplot()	绘制小提琴图

6.3 Matplotlib参数配置案例

下面将结合实际案例介绍Matplotlib绘图参数设置。本案例为了分析某企业2019年的销售额在全国各个地区的增长情况，本案例分别统计了每个地区在2018年和2019年的数据，并按照差额的大小进行了排序，绘制折线图的代码如下：


```

#导入可视化分析相关的包
import Matplotlib.pyplot as plt

#用来正常显示中文标签和负号
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False

#数据设置
x=['中南','东北','华东','华北','西南','西北']
y1=[223.65, 488.28, 673.34, 870.95, 1027.34, 1193.34]
y2=[214.71, 445.66, 627.11, 800.73, 956.88, 1090.24]

#设置输出的图片大小
figsize=10,8
figure, ax=plt.subplots(figsize=figsize)

#在同一幅图片上画两条折线
A=plt.plot(x,y1,'-r',label='2019年销售额',linewidth=5.0)
B=plt.plot(x,y2,'b-.',label='2018年销售额',linewidth=5.0)

#设置坐标刻度值的大小以及刻度值的字体
plt.tick_params(labelsize=15)
labels=ax.get_xticklabels() + ax.get_yticklabels()
[label.set_fontname('SimHei') for label in labels]

#设置图例并且设置图例的字体及大小
font1={'family' : 'SimHei','weight' : 'normal','size' :
15,}
legend=plt.legend(handles=[A,B],prop=font1)

#设置横纵坐标的名称以及对应字体格式
font2={'family' : 'SimHei','weight' : 'normal','size' :
20,}
plt.xlabel('地区',font2)
plt.ylabel('销售额',font2)

#输出图形
plt.show()

```

运行上述代码，生成如图6-8所示的视图。从图中可以看出：在2019年，该企业在6个区域的销售额增长额度由大到小依次是：西北、

西南、华北、华东、东北、中南。

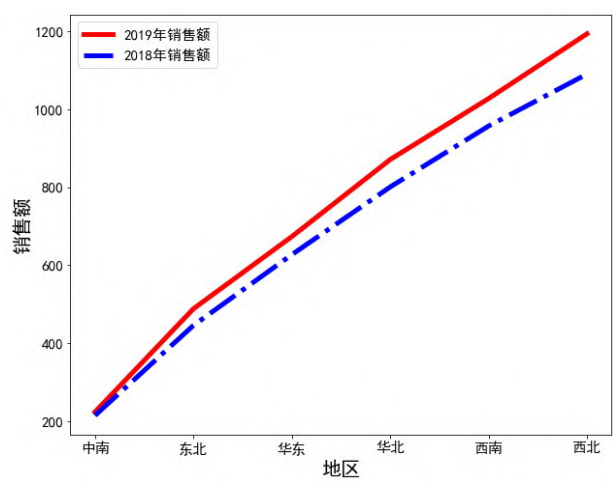


图6-8 各地区销售额分析

第7章

Matplotlib基础绘图



本章通过使用存储在集群中的实际案例数据介绍使用Matplotlib绘制一些基础图形，包括直方图、折线图、条形图、饼图、散点图、箱形图等。

7.1 直方图的绘制

7.1.1 直方图及其参数说明

Matplotlib绘制直方图使用`plt.hist()`函数，函数参数如下：

```
Matplotlib.pyplot.hist(x, bins=None, range=None, density=None, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, normed=None, *, data=None, **kwargs)
```

Matplotlib直方图的参数配置如表7-1所示。

表7-1 直方图参数配置

属 性	说 明
x	指定要绘制的直方图的数据
bins	指定直方图条形的个数
range	指定直方图数据的上下界，默认包含绘图数据的最大值和最小值
density	若为 True，则返回元组的第一个元素将是归一化的计数，以形成概率密度
weights	该参数可以为每一个数据点设置权重
cumulative	是否需要计算累计频数或频率
bottom	可以为直方图的每个条形添加基准线，默认为 0
histtype	指定直方图的类型，默认为 bar，还有 barstacked、step 等
align	设置条形边界值的对其方式，默认为 mid，还有 left 和 right
orientation	设置直方图的摆放方向，默认为垂直方向
rwidth	设置直方图条形宽度的百分比
log	是否需要将绘图数据进行对数变换
color	设置直方图的填充色
label	设置直方图的标签，可以通过 legend 展示其图例
stacked	当有多个数据时，是否需要将直方图呈堆叠摆放，默认水平摆放
normed	已经弃用，改用 density 参数

7.1.2 实例：每日利润额的数值分布

为了研究某企业的产品销售业绩情况，需要对每天的利润额进行分析，Python代码如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

import Numpy as np
```

```

import Matplotlib as mpl
import Matplotlib.pyplot as plt
from Matplotlib.font_manager import FontProperties
mpl.rcParams['font.sans-serif']=['SimHei']          #显示中文
plt.rcParams['axes.unicode_minus']=False          #正常显示负号
from impala.dbapi import connect

#连接Hadoop数据库
v1=[]
v2=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()

#读取Hadoop订单表数据
sql_num="SELECT order_date,ROUND(SUM(profit)/10000,2) FROM
orders WHERE dt=2019 GROUP BY order_date"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])

plt.figure(figsize=(15,8))          #设置图形大小
plt.hist(v2, bins=40, normed=0, facecolor="blue",
edgecolor="black", alpha=0.7)
# 显示横轴标签
plt.xlabel("区间")
# 显示纵轴标签
plt.ylabel("频数")
# 显示图标题
plt.title("利润额分布直方图")
plt.show()

```

在Jupyter Lab中运行上述代码，生成如图7-1所示的直方图。从图中可以看出该企业在2019年大部分天数是盈利的，利润额基本都在0万~1.0万之间，但是也有部分日期出现亏损的情况，大体在0万~0.5万之间。

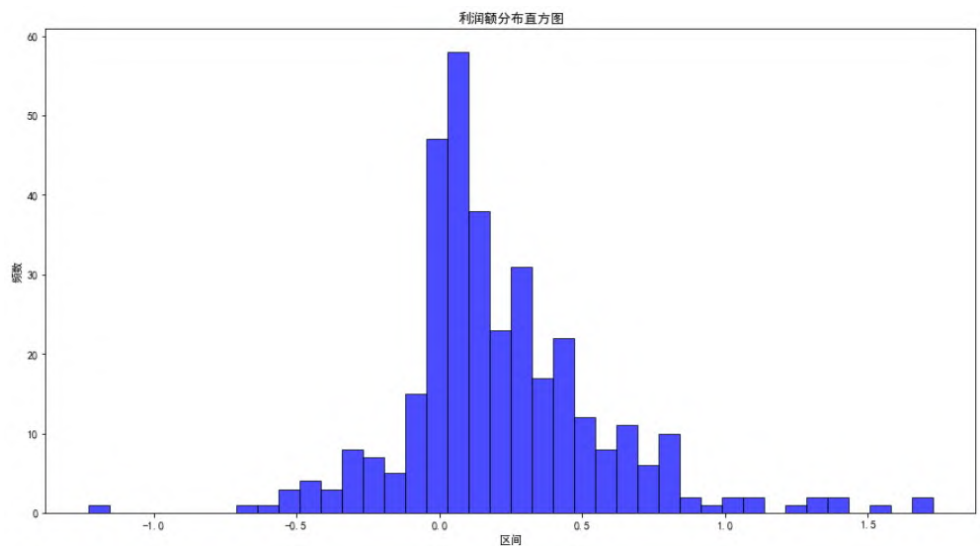


图7-1 利润额直方图

7.2 折线图的绘制

7.2.1 折线图及其参数说明

Matplotlib绘制折线图使用plt.plot()函数，函数参数如下：

```
plot([x], y, [fmt], data=None, **kwargs)
```

Matplotlib折线图的参数配置如表7-2所示。

表7-2 折线图参数配置

属 性	说 明
x、y	设置数据点的水平或垂直坐标
fmt	用一个字符串来定义图的基本属性，如颜色、点型、线型
data	带有标签的绘图数据

7.2.2 实例：每周商品销售业绩分析

电商企业的产品销售一般都具有周期性，为了深入研究某企业的销售额和利润额的变化情况，需要绘制企业每周的销售额和利润额折线图，Python代码如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

import Matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei'] #显示中文
plt.rcParams['axes.unicode_minus']=False #正常显示负号
from impala.dbapi import connect

#连接Hadoop数据库
v1=[]
v2=[]
v3=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()

#读取Hadoop订单表数据
sql_num="SELECT
weekofyear(order_date),ROUND(SUM(sales)/10000,2),ROUND(SUM(profit)/10000,2) FROM orders WHERE dt=2019 GROUP BY
weekofyear(order_date)"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])
    v3.append(s[2])

#画折线图
plt.plot(v1, v2)
plt.plot(v1, v3)
#设置纵坐标范围
plt.ylim((-1,21))
#设置横坐标角度，这里设置为45度
```

```
plt.xticks(rotation=45)
#设置横纵坐标名称
plt.xlabel("日期")
plt.ylabel("销售额与利润额")
#设置折线图名称
plt.title("2019年企业销售额与利润额分析")
plt.show()
```

在Jupyter Lab中运行上述代码，生成如图7-2所示的折线图。从图中可以看出该企业在2019年，每周的销售额相对于每周的利润额变化较大，尤其是在下半年，虽然销售额较上半年有较大幅度的上升，但是利润额基本没有什么变化，这可能与企业加大了营销推广力度有关，从而导致经营成本的大幅度增加。

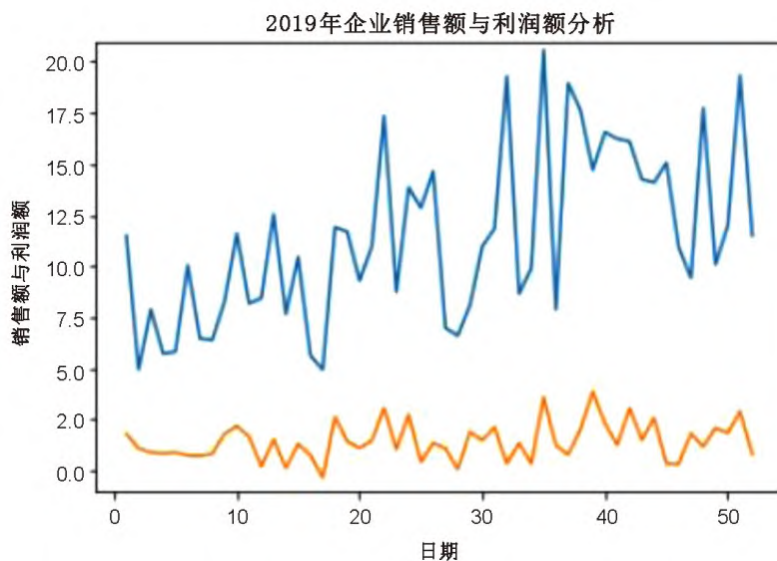


图7-2 销售额和利润额折线图

7.3 条形图的绘制

7.3.1 条形图及其参数说明

Matplotlib绘制条形图使用`plt.bar()`函数，函数参数如下：


```
Matplotlib.pyplot.bar(x, height, width=0.8, bottom=None,
*, align='center', data=None, **kwargs)
```

Matplotlib条形图的参数配置如表7-3所示。

表7-3 条形图参数配置

属 性	说 明
x	设置横坐标
height	条形的高度
width	直方图宽度，默认为 0.8
bottom	条形的起始位置
align	条形的中心位置
color	条形的颜色
edgecolor	边框的颜色
linewidth	边框的宽度
tick_label	下标的标签
log	y 轴使用科学计算法表示
orientation	是竖直条还是水平条

7.3.2 实例：不同省份利润额的比较

电商企业的产品销售往往会呈现区域性差异，为了深入研究某企业的产品在2019年是否具有区域差异性，绘制区域利润额的条形图，Python代码如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB
```

```

import Numpy as np
import Matplotlib as mpl
import Matplotlib.pyplot as plt
from Matplotlib.font_manager import FontProperties
mpl.rcParams['font.sans-serif']=['SimHei']      #显示中文
plt.rcParams['axes.unicode_minus']=False      #正常显示负号
from impala.dbapi import connect

#连接Hadoop数据库
v1=[]
v2=[]
v3=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()

#读取Hadoop订单表数据
sql_num="SELECT province,ROUND(SUM(profit)/10000,2) FROM
orders WHERE dt=2019 GROUP BY province"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])

plt.figure(figsize=(15,8))      #设置图形大小
plt.bar(v1, v2, alpha=0.5, width=0.4, color='blue',
edgecolor='red', label='利润额', lw=1)
plt.legend(loc='upper left')
plt.xticks(np.arange(31), v1, rotation=10)      #rotation控
制倾斜角度

#fontsize控制label和title字体大小
plt.ylabel('利润额', fontsize=10)
plt.title('2019年各省市销售额分析', fontsize=15)
plt.xlabel('销售区域', fontsize=10)

#设置坐标轴上数值的字体大小
plt.tick_params(axis='both', labelsize=10)
plt.show()

```

在Jupyter Lab中运行上述代码，生成如图7-3所示的条形图。从图中可以看出该企业在2019年，在全国31个省市的利润额存在较大的

差异，在山东省的利润额最大，其次是黑龙江省和广东省，但是在部分省市也出现亏损的情况，尤其是在辽宁省、浙江省和湖北省。

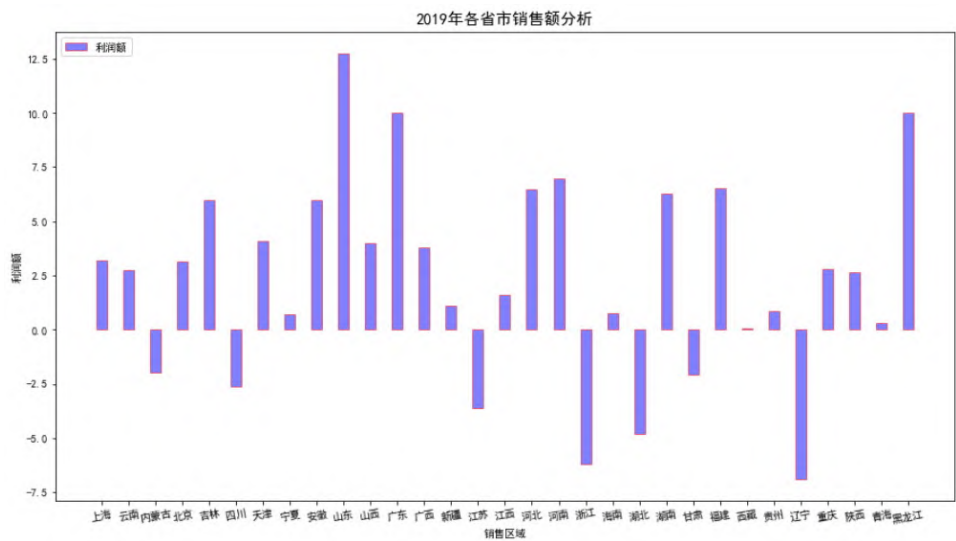


图7-3 利润额条形图

7.4 饼图的绘制

7.4.1 饼图及其参数说明

Matplotlib绘制饼图使用plt.pie()函数，函数参数如下：

```
Matplotlib.pyplot.pie(x, explode=None, labels=None,
colors=None, autopct=None, pctdistance=0.6, shadow=False,
labeldistance=1.1, startangle=None, radius=None,
counterclock=True, wedgeprops=None, textprops=None, center=
(0, 0), frame=False, rotatelabels=False, *, data=None)
```

Matplotlib饼图的参数配置如表7-4所示。

表7-4 饼图参数配置

属 性	说 明
x	每一块的比例，如果 $\text{sum}(x) > 1$ ，就会进行归一化处理
labels	每一块饼图外侧显示的说明文字
explode	每一块离开中心的距离
startangle	起始绘制角度，默认图是从 x 轴正方向逆时针画起的，若设定=90，则从 y 轴正方向画起
shadow	在饼图下面画一个阴影。默认为 False，即不画阴影
labeldistance	label 标记的绘制位置，相对于半径的比例，默认值为 1.1，若小于 1，则绘制在饼图内侧
autopct	控制饼图内的百分比设置
pctdistance	类似于 labeldistance，指定 autopct 的位置刻度，默认值为 0.6
radius	控制饼图半径，默认值为 1
counterclock	指定指针方向，可选，默认值为 True，即逆时针
wedgeprops	字典类型，可选，默认值为 None。参数字典传递给 wedge 对象用来画饼图
textprops	设置标签和比例文字的格式，字典类型，可选，默认值为 None
center	浮点类型的列表，可选，默认值为(0, 0)，图标中心位置
frame	布尔类型，可选，默认为 False。如果是 True，就绘制带有表的轴框架
rotatelabels	布尔类型，可选，默认为 False。如果为 True，就旋转每个 label 到指定的角度

7.4.2 实例：不同类型商品销售额比较

为了研究某企业不同类型商品的销售额是否存在一定的差异，绘制不同类型商品的饼图，Python代码如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

import Matplotlib.pyplot as plt
from impala.dbapi import connect
plt.rcParams['font.sans-serif']=['SimHei']
```

```

#连接Hadoop数据库
v1=[]
v2=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()

#读取Hadoop订单表数据
sql_num="SELECT category,ROUND(SUM(sales),2)FROM orders
WHERE dt=2019 GROUP BY category"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])

plt.figure(figsize=(15,8))          #设置饼图大小
labels=v1
explode =[0.1, 0.1, 0.1]           #每一块离开中心距离
plt.pie(v2, explode=explode,
labels=labels,autopct='%1.1f%%',textprops=
{'fontsize':15,'color':'black'})
plt.title('2019年不同类型产品的销售额分析',fontsize=20)
plt.show()

```

在Jupyter Lab中运行上述代码，生成如图7-4所示的饼图。从图中可以看出该企业在2019年，不同类型产品的销售额存在一定的差异，其中家具类产品的销售额占比达到了36.1%，技术类占到了32.4%，办公用品类占比为31.5%。

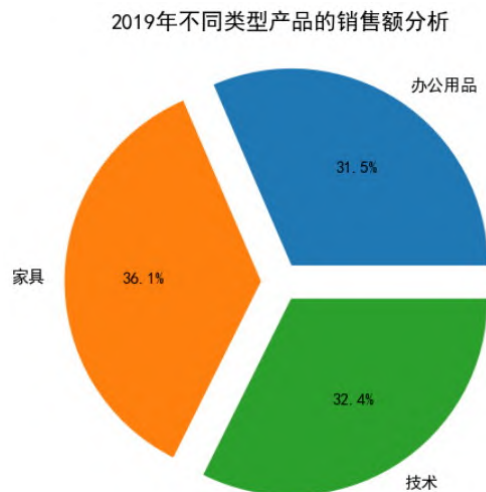


图7-4 不同类型产品的销售额

7.5 散点图的绘制

7.5.1 散点图及其参数说明

Matplotlib绘制散点图用到`plt.scatter()`函数，函数参数如下：

```
Matplotlib.pyplot.scatter(x, y, s=None, c=None,
marker=None, cmap=None, norm=None, vmin=None, vmax=None,
alpha=None, linewidths=None, verts=None, edgecolors=None, *,
data=None, **kwargs)
```

Matplotlib散点图的参数配置如表7-5所示。

表7-5 散点图参数配置

属 性	说 明
x、y	绘图的数据都是向量且必须长度相等
s	设置标记大小
C	设置标记颜色
marker	设置标记样式
cmap	设置色彩盘
norm	设置亮度，为 0~1 之间
vmin、vmax	设置亮度，如果 norm 已设置，该参数就无效
alpha	设置透明度，为 0~1 之间
linewidths	设置线条的宽度
edgecolors	设置轮廓颜色

7.5.2 实例：销售额与利润额的关系

为了研究某企业每天的销售额与利润额两者之间的关系，绘制销售额与利润额的散点图，Python代码如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

import Matplotlib.pyplot as plt
import Numpy as np
from impala.dbapi import connect
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False

#连接Hadoop数据库
v1=[]
v2=[]
v3=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()
```

```

#读取Hadoop订单表数据
sql_num="SELECT
order_date,ROUND(SUM(sales)/10000,2),ROUND(SUM(profit)/10000,
2) FROM orders WHERE dt=2019 GROUP BY order_date"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])
    v3.append(s[2])

plt.figure(figsize=(15,8))          #设置图形大小
plt.scatter(v2, v3, marker='o', s=75, alpha=0.5) #marker
点的形状, s点的大小, alpha点的透明度
plt.xlabel('销售额', fontsize=10)
plt.ylabel('利润额', fontsize=10)
plt.title('2019年销售额利润额分析', fontsize=15)
plt.grid(True)
plt.show()

```

在Jupyter Lab中运行上述代码，生成如图7-5所示的散点图。从图中可以看出该企业在2019年，每天的销售额与利润额两者之间的关系不大，即随着销售额的增加，利润额增加很少，甚至出现亏损的情况，这可能与不断增加的营销成本有关。

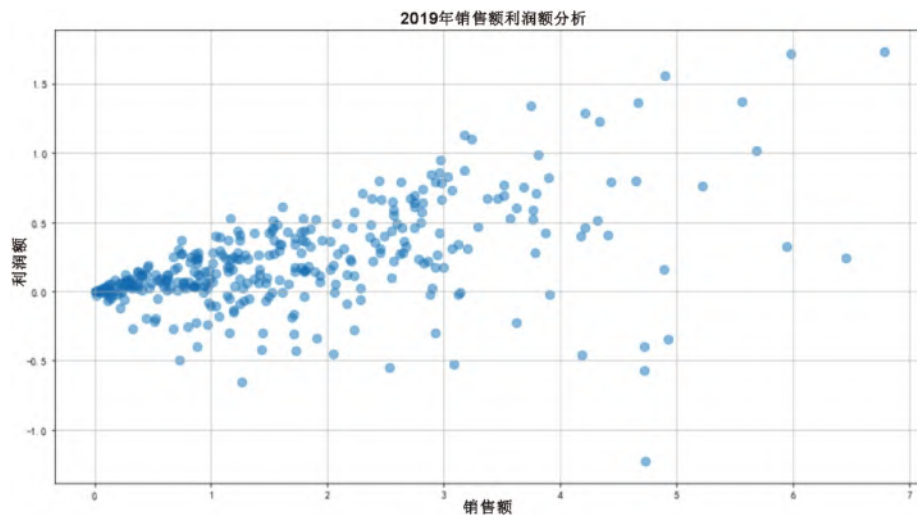


图7-5 销售额和利润额散点图

7.6 箱形图的绘制

7.6.1 箱形图及其参数说明

Matplotlib绘制箱线图使用`plt.boxplot()`函数，函数参数如下：

```
plt.boxplot(x, notch=None,  
sym=None,vert=None,whis=None,positions=None,  
widths=None,  
patch_artist=None,meanline=None,showmeans=None,showcaps=None,  
showbox=None,showfliers=None,boxprops=None,labels=None,flierp  
rops=None,medianprops=None,meanprops=None,  
capprops=None,whiskerprops=None)
```

Matplotlib箱型图的参数配置如表7-6所示。

表7-6 箱型图参数配置

属 性	说 明
x	指定要绘制箱线图的数据
notch	是否以凹口的形式展现箱线图，默认为非凹口
sym	指定异常点的形状，默认为“+”显示
vert	是否需要将箱线图垂直摆放，默认垂直摆放
whis	指定上下须与上下四分位的距离，默认为 1.5 倍的四分位差
positions	指定箱线图的位置，默认为[0,1,2...]
widths	指定箱线图的宽度，默认为 0.5
patch_artist	是否填充箱体的颜色
meanline	是否用线的形式表示均值，默认用点来表示
showmeans	是否显示均值，默认不显示
showcaps	是否显示箱线图顶端和末端的两条线，默认显示
showbox	是否显示箱线图的箱体，默认显示
showfliers	是否显示异常值，默认显示
boxprops	设置箱体的属性，如边框色、填充色等
labels	为箱线图添加标签，类似于图例的作用
flierprops	设置异常值的属性，如异常点的形状、大小、填充色等
medianprops	设置中位数的属性，如线的类型、粗细等
meanprops	设置均值的属性，如点的大小、颜色等
capprops	设置箱线图顶端和末端线条的属性，如颜色、粗细等
whiskerprops	设置须的属性，如颜色、粗细、线的类型等

7.6.2 实例：销售经理业绩比较分析

为了客观地评价销售经理的业绩情况，本例绘制每位销售经理在2019年销售业绩的箱形图，Python代码如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB
```

```

import Numpy as np
import Pandas as pd
import Matplotlib as mpl
import Matplotlib.pyplot as plt
mpl.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False
from impala.dbapi import connect

#读取Hadoop集群数据
v1=[]
v2=[]
v3=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cur=conn.cursor()
sql_num="SELECT manager,sales FROM orders WHERE dt=2019
and category='家具' and sales<=10000"
cur.execute(sql_num)
sh=cur.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])

data=np.transpose(pd.DataFrame([v1,v2]))
data.columns=['销售经理', '利润额']

group=data.销售经理.unique()
def group():
    df=[]
    group=data.销售经理.unique()
    for x in group:
        a=data.利润额[data.销售经理==x]
        df.append(a)
    return df
box1,box2,box3,box4,box5,box6=group()[0],group()[1],group()[2],group()[3],group()[4],group()[5]

#绘制箱线图并设置需要的参数
plt.figure(figsize=(15,7))

plt.boxplot([box1,box2,box3,box4,box5,box6],vert=False,showmeans=False,showbox=True)
plt.yticks([1, 2, 3, 4, 5, 6],['张怡莲', '王倩倩', '郝杰', '杨洪光', '江奕健',

```

```

'姜伟']])

plt.xticks(np.arange(0,10000,step=1000))
plt.xlabel('销售额',fontsize=15.0)
plt.ylabel('销售经理',fontsize=15.0)
plt.title('销售额经理的销售业绩分析',fontsize=20.0)
plt.yticks(fontsize=15.0)
plt.show()

```

在Jupyter Lab中运行上述代码，生成如图7-6所示的箱线图。从图中可以看出该企业在2019年，6名销售经理的销售业绩存在一定的差异，其中张怡莲的整体表现最好，剔除异常情况的影响，其每单的最大值接近7000元，平均值接近2000元，其次是姜伟，其每单的平均值大约在1600元，最差的是江奕健，每单的平均值仅为1000元左右。

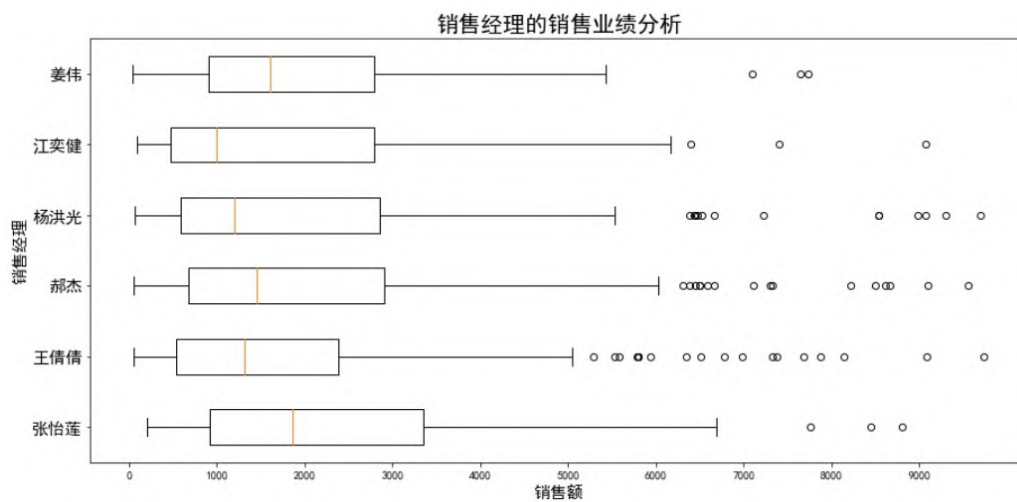


图7-6 销售经理业绩分析

第8章

Matplotlib高级绘图



本章通过使用存储在集群中的实际案例数据介绍使用Matplotlib绘制一些高级图形，包括树形图、误差条形图、火柴杆图、甘特图、自相关图、图形整合等。

8.1 树形图的绘制

8.1.1 树形图及其应用场景

树形图采用矩形表示层次结构的节点，父子层次关系用矩阵间的相互嵌套来表达。从根节点开始，空间根据相应的子节点数目被分为多个矩形，矩形面积大小对应节点属性。每个矩形又按照相应节点的子节点递归地进行分割，直到叶子节点为止。

树形图图形紧凑，同样大小的画布可以展现更多信息，以及成员间的权重，但是存在不够直观、不够明确、不像树图那么清晰、分类占比太小时不容易排布等缺点。

应用场景：

适合展现具有层级关系的数据，能够直观地体现同级之间的数据比较。

8.1.2 实例：不同省份销售额的比较分析

为了深入地研究某企业的产品是否具有区域差异性，这里绘制销售额的树形图，Python代码如下：

```
# -*- coding: utf-8 -*-

import Matplotlib.pyplot as plt
import squarify
from impala.dbapi import connect
plt.rcParams['font.sans-serif']=['SimHei']

#连接Hadoop数据库
v1=[]
v2=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()

#读取Hadoop订单表数据
sql_num="SELECT region, ROUND(SUM(sales/10000),2) FROM
orders WHERE dt=2019 GROUP BY region"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])

plt.figure(figsize=(15,8))
colors=
['steelblue','red','indianred','green','yellow','orange']    #
设置颜色数据
plot=squarify.plot(
    sizes=v2,                                     #指定绘图数据
```

```

        label=v1,                    #标签
        color=colors,                #指定自定义颜色
        alpha=0.6,                  #指定透明度
        value=v2,                   #添加数值标签
        edgecolor='white',          #设置边界框白色
        linewidth=8                  #设置边框宽度为3
    )

    plt.rc('font',size=15)           #设置标签大小
    plot.set_title('2019年企业销售额情况',fontdict=
{'fontsize':15}) #设置标题及大小
    plt.axis('off')                  #去除坐标轴
    plt.tick_params(top='off',right='off') #去除上边框和右边框
    刻度
    plt.show()

```

在Jupyter Lab中运行上述代码，生成如图8-1所示的树形图。从图中可以看出，该企业2019年在各区域的销售额差异较大，其中华东地区的销售额最多，为169.97万元；其次是中南地区，为146.21万元；最少的是西北地区，为29.24万元。

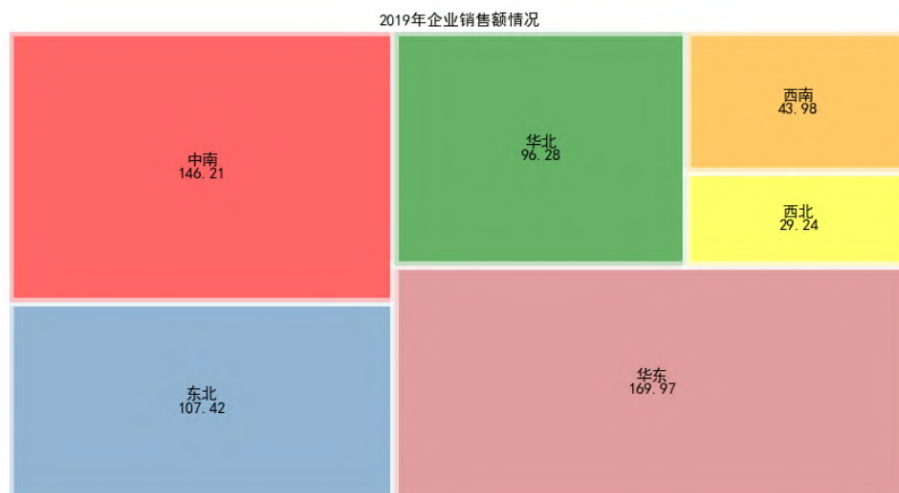


图8-1 销售额树形图

8.2 误差条形图的绘制

8.2.1 误差条形图及其应用场景

误差条形图是一类特殊的条形图，由带标记的线条组成，这些线条用于显示有关图中所显示的数据的统计信息。误差条形图类型的序列具有3个Y值，即平均值、下限误差值、上限误差值。

应用场景：

可以手动将值分配给每个点，但在大多数情况下，是从其他序列中的数据来计算的。Y值的顺序十分重要，因为值数组中的每个位置都表示误差条形图上的一个值。

8.2.2 实例：门店业绩考核达标情况分析

为了深入地研究某企业各个门店的销售业绩是否达标，绘制销售额的误差条形图，Python代码如下：

```
# -*- coding: utf-8 -*-

import Numpy as np
import Matplotlib as mpl
import Matplotlib.pyplot as plt
from impala.dbapi import connect
mpl.rcParams['font.sans-serif']=['SimHei']

#连接Hadoop数据库
v1=[]
v2=[]
v3=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()

#读取Hadoop订单表数据
sql_num="SELECT
```



```

store_name,SUM(sales)/10000,SUM(sales)/10000-60.00 FROM
orders WHERE dt=2019 GROUP BY store_name"
    cursor.execute(sql_num)
    sh=cursor.fetchall()
    for s in sh:
        v1.append(s[0])
        v2.append(s[1])
        v3.append(s[2])

    plt.figure(figsize=(10,6))          #设置图形大小
    plt.bar(v1, v2, yerr=v3, width=0.4, align='center',
ecolor='r', color='green', label='门店销售额');

    #添加坐标标签
    plt.xlabel('门店名称')
    plt.ylabel('销售额')
    plt.title('2019年门店业绩考核达标情况')
    plt.legend(loc='upper left')
    plt.show()

```

在Jupyter Lab中运行上述代码，生成如图8-2所示的误差条形图。从图中可以看出，该企业在2019年各个门店的销售额与业绩目标的差额。

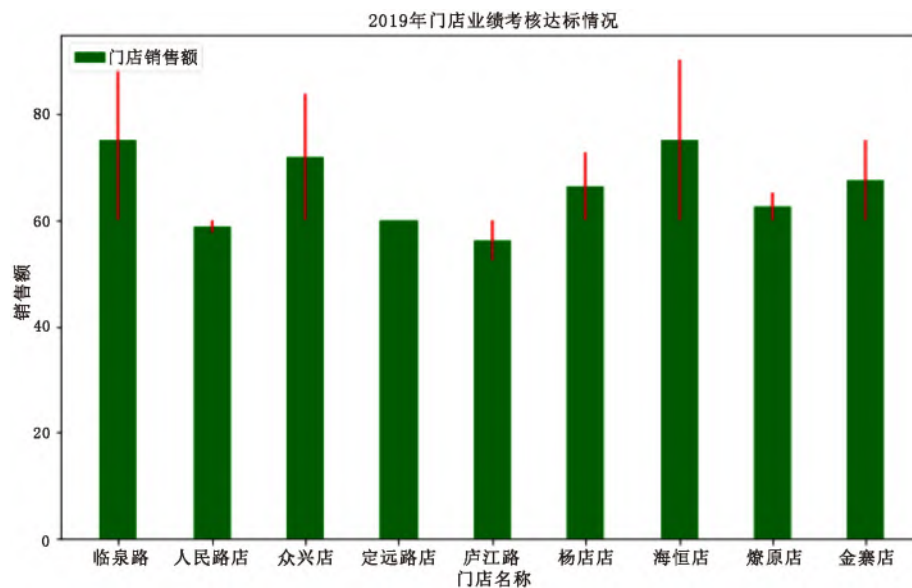


图8-2 销售额误差条形图

8.3 火柴杆图的绘制

8.3.1 火柴杆图及其应用场景

火柴杆图是用线条显示数据与x轴的距离，用一个小圆圈或者其他标记符号与线条相互连接，并在y轴上标记数据的值。

Matplotlib绘制火柴杆图用plt.step()函数，函数参数如下：

```
Matplotlib.pyplot.step(x, y, color, where, *)
```

应用场景：

需要美观地显示各种类型下的数值到x轴的距离。

8.3.2 实例：不同省份送货准时性分析

为了深入地研究某企业的产品送货准时性情况，绘制送货延迟时间的火柴杆图，Python代码如下：

```
# -*- coding: utf-8 -*-

import Matplotlib.pyplot as plt
import Numpy as np
from impala.dbapi import connect
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False

#连接Hadoop数据库
v1=[]
v2=[]
v3=[]
conn=connect(host='192.168.1.7', port=10000,
```

```

database='sales',auth_mechanism='NOSASL',user='root')
    cursor=conn.cursor()

    #读取Hadoop订单表数据
    sql_num="SELECT
province,avg(datediff(deliver_date,order_date)-planned_days)
FROM orders WHERE dt=2019 GROUP BY province"
    cursor.execute(sql_num)
    sh=cursor.fetchall()
    for s in sh:
        v1.append(s[0])
        v2.append(s[1])

    plt.figure(figsize=(15,7))          #设置图形大小
    label="平均延迟天数"
    markerline, stemlines, baseline=plt.stem(v1, v2,
label=label)

    plt.setp(markerline, color='red', marker='o')
    plt.setp(stemlines, color='blue', linestyle=':')
    plt.setp(baseline, color='grey', linewidth=3, linestyle='-
')

    plt.xlabel('省市自治区')
    plt.ylabel('平均延迟天数')
    plt.title('2019年各省市平均延迟天数')
    plt.legend()
    plt.show()

```

在Jupyter Lab中运行上述代码，生成如图8-3所示的火柴杆图。从图中可以看出，该企业2019年商品在青海、湖南、贵州和云南地区均有不同程度的延迟，平均延迟天数最久的是青海。

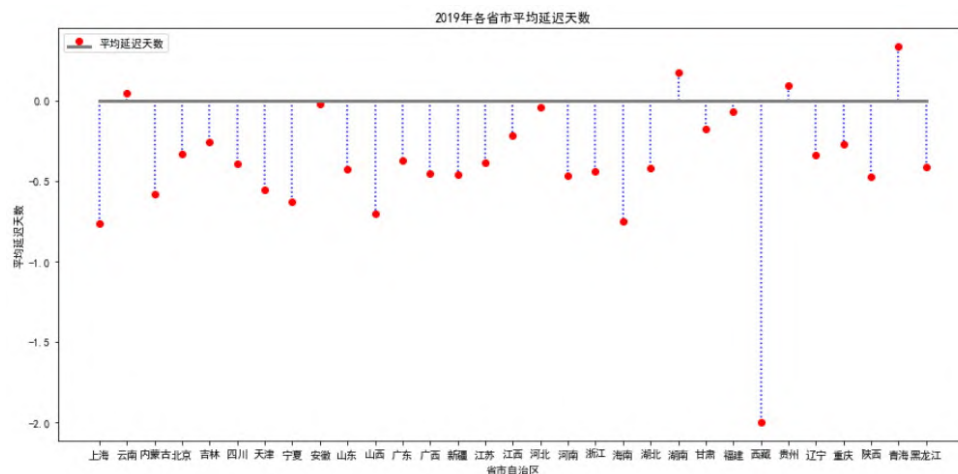


图8-3 平均延误天数火柴杆图

8.4 甘特图的绘制

8.4.1 甘特图及其应用场景

甘特图以图示的方式通过活动列表和时间刻度形象地表示任何特定项目的活动顺序与持续时间，即甘特图是将活动与时间联系起来的一种图表形式，显示每个活动的历时长短。它能够从时间上整体把握进度，很清晰地标识出每一项任务的起始与结束时间。这就不难理解甘特图的产生原因了：因为生产管理领域生产计划制定的需要而产生。

应用场景：

（1）项目管理：甘特图是在现代的项目管理中被广泛应用的一种图表形式。甘特图可以帮助我们预测时间、成本、数量及质量上的结果并回到开始，也能帮助我们考虑人力、资源、日期、项目中重复的要素和关键的部分，还能集成多张各方面的甘特图而成为一张总图。以甘特图的方式可以直观地看到任务的进展情况、资源的利用率等。

(2) 其他领域：如今甘特图不单单被应用到生产管理领域，随着生产管理的发展、项目管理的扩展，它被应用到了各个领域，如建筑、IT软件、汽车等所有把时间和任务进度联系到一起的领域。

8.4.2 实例：企业信息化项目进度管理

为了分析某企业信息化项目的进度情况，绘制项目进度的甘特图，Python代码如下：

```
# -*- coding: utf-8 -*-

from datetime import datetime
import sys
import Numpy as np
import Matplotlib.pyplot as plt
import Matplotlib.font_manager as font_manager
import Matplotlib.dates as mdates
import logging
from pylab import *
mpl.rcParams['font.sans-serif']=['SimHei']

class Gantt(object):
    #颜色色标: 参考http://colorbrewer2.org/
    RdYlGr=['#d73027', '#f46d43', '#fdae61','#fee08b',
            '#ffffbf', '#d9ef8b', '#a6d96a','#66bd63', '#1a9850']

    POS_START=1.0
    POS_STEP=0.5

    def __init__(self, tasks):
        self._fig=plt.figure(figsize=(15,10))
        self._ax=self._fig.add_axes([0.1, 0.1, .75, .5])

        self.tasks=tasks[::-1]

    def _format_date(self, date_string):
        try:
            date=datetime.datetime.strptime(date_string,
            '%Y-%m-%d %H:%M:%S')
        except ValueError as err:
```

```

        logging.error("String '{0}' can not be
converted to datetime object: {1}"
                        .format(date_string, err))
        sys.exit(-1)
    mpl_date=mdates.date2num(date)
    return mpl_date

def _plot_bars(self):
    i=0
    for task in self.tasks:
        start=self._format_date(task['start'])
        end=self._format_date(task['end'])
        bottom=(i * Gantt.POS_STEP) + Gantt.POS_START
        width=end - start
        self._ax.barh(bottom, width, left=start,
height=0.3,align='center',
label=task['label'],color=Gantt.RdYlGr[i])
        i += 1

    def _configure_yaxis(self):
        task_labels=[t['label'] for t in self.tasks]
        pos=self._positions(len(task_labels))
        ylocs=self._ax.set_yticks(pos)
        ylabels=self._ax.set_yticklabels(task_labels)
        plt.setp(ylabels, size='medium')

    def _configure_xaxis(self):
        self._ax.xaxis_date()
        rule=mdates.rrulewrapper(mdates.DAILY,
interval=20)
        loc=mdates.RRuleLocator(rule)
        formatter=mdates.DateFormatter("%d %b")

        self._ax.xaxis.set_major_locator(loc)
        self._ax.xaxis.set_major_formatter(formatter)
        xlabels=self._ax.get_xticklabels()
        plt.setp(xlabels, rotation=30, fontsize=10)

    def _configure_figure(self):
        self._configure_xaxis()
        self._configure_yaxis()

        self._ax.grid(True, color='gray')
        self._set_legend()
        self._fig.autofmt_xdate()

    def _set_legend(self):
        font=font_manager.FontProperties(size='small')

```

```

        self._ax.legend(loc='upper right', prop=font)

    def _positions(self, count):
        end=count * Gantt.POS_STEP + Gantt.POS_START
        pos=np.arange(Gantt.POS_START, end,
Gantt.POS_STEP)
        return pos

    def show(self):
        self._plotBars()
        self._configureFigure()
        plt.show()

if __name__ == '__main__':
    TEST_DATA=(
        {'label':'项目调研','start':'2019-02-01
12:00:00','end':'2019-03-15 18:00:00'},
        {'label':'项目准备','start':'2019-03-16
09:00:00','end':'2019-04-09 12:00:00'},
        {'label':'制定方案','start':'2019-04-10
12:00:00','end':'2019-04-30 18:00:00'},
        {'label':'项目实施','start':'2019-05-01
09:00:00','end':'2019-08-31 13:00:00'},
        {'label':'项目培训','start':'2019-09-01
09:00:00','end':'2019-09-21 13:00:00'},
        {'label':'项目验收','start':'2019-09-22
09:00:00','end':'2019-10-22 13:00:00'},
        {'label':'项目竣工','start':'2019-10-23
09:00:00','end':'2019-11-23 13:00:00'},
    )

    gantt=Gantt(TEST_DATA)
    plt.xlabel('项目日期')
    plt.ylabel('项目进度')
    plt.title('项目进度甘特图')
    plt.figure(figsize=(10,10))
    gantt.show()

```

在Jupyter Lab中运行上述代码，生成如图8-4所示的甘特图。从图中可以看出，该企业的信息化项目进度的具体过程及其时间进度。

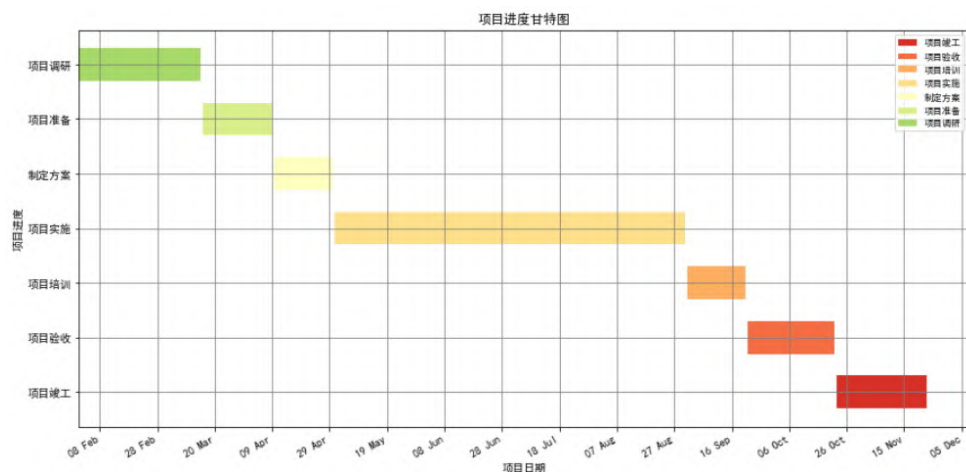


图8-4 项目进度甘特图

8.5 自相关图

8.5.1 自相关图及其应用场景

自相关也叫序列相关，是时间序列数据自身在不同时间点的互相关。由于时间序列的相关系数是指与之前的相同系列数据的相关性程度，因此被称为序列相关或自相关。时间序列的自相关系数也被称为自相关函数，自相关的图被称为相关图或自相关图。偏自相关函数用来度量暂时调整所有其他滞后项之后，时间序列中以 k 个时间单位分隔的观测值之间的相关性，即偏自相关是剔除干扰后的时间序列与先前相同时间步长的时间序列之间的相关系数。

应用场景：

直观地显示时间序列的当前序列值和过去序列值之间的相关性，并指示预测将来值时最有用的过去序列值。

8.5.2 实例：股票价格的自相关分析

为了深入地分析某企业的股票价格趋势，绘制股价的自相关图等，Python代码如下：

```
#导入相关的包
import Pandas as pd
import Matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf,
plot_pacf
from impala.dbapi import connect
plt.rcParams['font.sans-serif']=['SimHei']

#连接Hadoop数据库
v1=[]
v2=[]
v3=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()

#读取Hadoop订单表数据
sql_num="SELECT trade_date,close FROM stocks WHERE
year(trade_date)=2019 order by trade_date asc"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])
data=pd.DataFrame(v2,v1)

data.plot()
plt.title("股票收盘价的时序图")

# 绘制自相关图
plot_acf(data).show()
plt.title("股票收盘价的自相关图")

# 绘制偏自相关图
plot_pacf(data).show()
plt.title("股票收盘价的偏自相关图")
```

在Jupyter Lab中运行上述代码，生成如图8-5所示的时序图。从图中可以看出，该企业在2019年股票价格基本呈现上涨的趋势。



图8-5 时序图

生成如图8-6所示的企业2019年股票收盘价的自相关图。

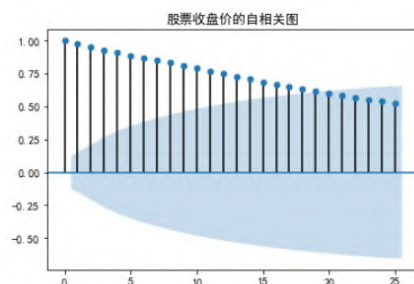


图8-6 自相关图

生成如图8-7所示的企业2019年股票收盘价的偏自相关图。

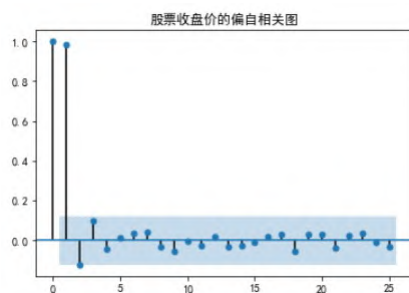


图8-7 偏自相关图

8.6 图形整合

8.6.1 图形整合及其应用场景

Matplotlib可以把很多张图画到一个显示界面，这就涉及面板切分成一个个子图。这是怎么做到的呢？Matplotlib的提供了两种方法：直接指定划分方式和按位置进行绘图。

第一种方法：直接指定划分方式——使用subplot函数

函数格式：plt.subplot(x,y,z)

Subplot函数前面的两个参数指定的是一个画板被分割成的行和列，后面一个参数则指定当前正在绘制图形的编号。例如：

```
# -*- coding: utf-8 -*-
import Matplotlib as mpl
import Matplotlib.pyplot as plt

t=np.arange(0.0,2.0,0.1)
s=np.sin(t*np.pi)
plt.subplot(2,2,1)           #要生成两行两列，这是第一个图
plt.plot(t,s,'b*')
plt.ylabel('y1')
plt.subplot(2,2,2)          #两行两列，这是第二个图
plt.plot(2*t,s,'r--')
plt.ylabel('y2')
plt.subplot(2,2,3)          #两行两列，这是第三个图
plt.plot(3*t,s,'m--')
plt.ylabel('y3')
plt.subplot(2,2,4)          #两行两列，这是第四个图
plt.plot(4*t,s,'k*')
plt.ylabel('y4')
plt.show()
```

执行上面的代码，生成如图8-8所示的整合图形。

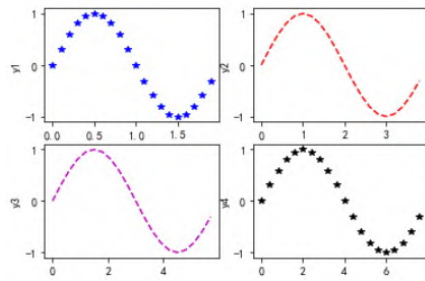


图8-8 整合图形

第二种方法：按位置绘图—使用subplots函数

函数格式：plt.subplots(x,y)

这个方法更直接，是指事先把画板分隔好，例如：

```
# -*- coding: utf-8 -*-
import Matplotlib as mpl
import Matplotlib.pyplot as plt

t=np.arange(0.0,2.0,0.1)
s=np.sin(t*np.pi)
c=np.cos(t*np.pi)
figure,ax=plt.subplots(2,2)
ax[0][0].plot(t,s,'r*')
ax[0][1].plot(t*2,s,'b--')
ax[1][0].plot(t,c,'g*')
ax[1][1].plot(t*2,c,'y--')
```

执行上面的代码，生成如图8-9所示的整合图形。

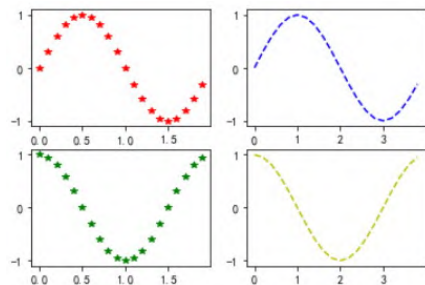


图8-9 整合图形

应用场景：

需要将多个图形有机地整合为一张图形，便于后续进行深入的比较分析。

8.6.2 实例：区域销售额与利润额分析

由于受区域经济环境、生活环境、文化环境等的影响，电商企业的产品销售往往会呈现区域性差异，为了深入地研究某企业的产品是否在2019年具有区域性差异，我们这里使用subplot函数进行可视化分析，Python代码如下：

```
# -*- coding: utf-8 -*-

import Matplotlib as mpl
import Matplotlib.pyplot as plt
from impala.dbapi import connect
mpl.rcParams['font.sans-serif']=['SimHei'] #显示中文
plt.rcParams['axes.unicode_minus']=False #正常显示负号

#连接Hadoop数据库
v1=[]
v2=[]
v3=[]
v4=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
```

```

cursor=conn.cursor()

#读取Hadoop订单表数据
sql_num="SELECT
region,ROUND(SUM(sales)/10000,2),ROUND(SUM(profit)/10000,2),R
OUND(SUM(amount),2) FROM orders WHERE dt=2019 GROUP BY
region"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])
    v3.append(s[2])
    v4.append(s[3])

figure()                                #绘制一张图片
plt.figure(figsize=(15,8))              #设置图形大小

subplot(231)
plt.plot(v1, v2)                         #v1、v2的折线图

subplot(232)
plt.bar(v1, v3)                          #v1、v3的条形图

subplot(233)
plt.barh(v2, v3, alpha=0.5, color='red', edgecolor='red',
lw=3)    #v2、v3的水平条形图

subplot(234)
plt.bar(v2, v3, alpha=0.5, width=1.6, color='yellow',
edgecolor='red', lw=1)    #v2、v3的条形图

subplot(235)
plt.boxplot(v2)                          #v2的箱线图

subplot(236)
plt.scatter(v2, v3)                       #v2、v3的散点图

plt.suptitle('2019年区域销售额比较分析', fontsize=15)
plt.show()

```

在Jupyter Lab中运行上述代码，生成如图8-10所示的复合图形。从图中可以看出，该企业在2019年各地区的销售额基本情况，其中华

东地区的销售额最多，而利润额最多的是中南地区；销售额多利润额不一定多，且可以看出销售额在145万附近时，利润额是最大的；6个地区的销售额最大值大约在170万，平均值在100万左右。

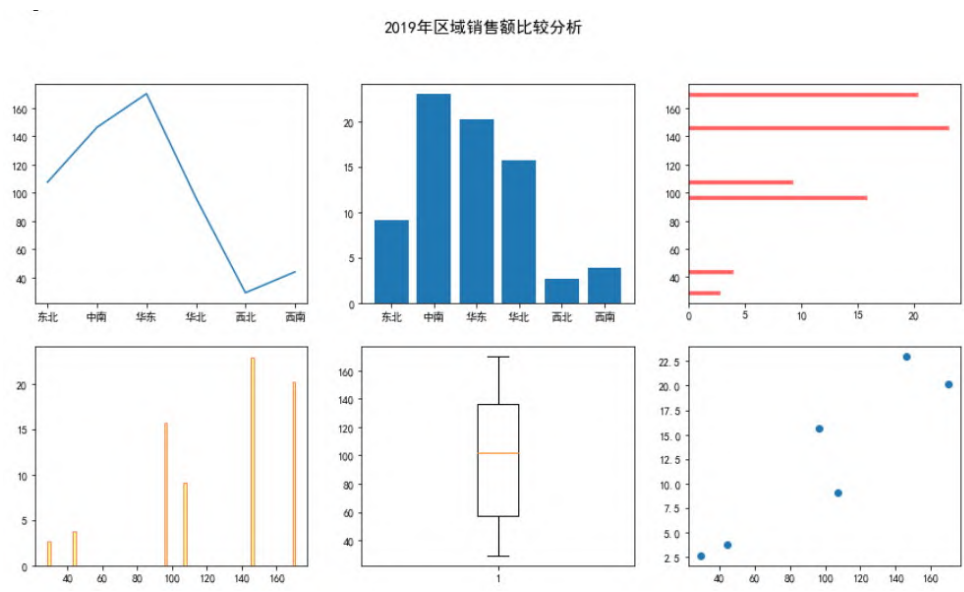


图8-10 区域销售额比较

第三部分

Pyecharts数据可视化

本部分我们将通过案例介绍Python中另一个非常重要的可视化包Pyecharts，这是一款将Python与Echarts相结合的强大数据可视化工具，可以高度灵活地配置，轻松搭配出精美的图表。本部分将介绍如何通过Pyecharts绘制可视化视图，包括图形的参数配置以及绘制一些常用的视图，包括折线图、条形图、箱形图、日历图、漏斗图、仪表盘、环形图等共计15种，此外，还会通过具体案例介绍Pyecharts如何与Django进行集成。

第9章

Pyecharts图形参数配置



图形的参数配置是数据可视化的基础，Pyecharts中的参数配置比较简单，可以分为全局配置项和系列配置项。本章我们将深入细致地列出每种配置。同时，还会简单地介绍Pyecharts的几种运行环境，读者可以根据实际工作需求选择适合自己的程序运行环境。

9.1 全局配置项

Pyecharts视图的全局配置项文件位于\Anaconda3\Lib\site-packages\pyecharts\options下的global_options.py文档中，可以通过set_global_options方法设置。

9.1.1 基本元素配置项

Pyecharts的基本元素配置项主要包括：InitOpts、ToolBoxFeatureOpts、ToolboxOpts、TitleOpts、DataZoomOpts、

LegendOpts、VisualMapOpts、TooltipOpts八项配置。

(1) InitOpts：初始化配置项如表9-1所示。

表9-1 初始化配置项

配 置 项	说 明
width	图表画布宽度
height	图表画布高度
chart_id	图表 ID，图表唯一标识，用于在多图表时区分图表
renderer	渲染风格，可选 canvas、svg
page_title	网页标题
theme	图表主题
bg_color	图表背景颜色
js_host	远程 js host

(2) ToolBoxFeatureOpts：工具箱工具配置项如表9-2所示。

表9-2 工具箱工具配置项

配 置 项	说 明
save_as_image	保存为图片
restore	配置项还原
data_view	数据视图工具，可以展现当前图表所用的数据，编辑后可以动态更新
data_zoom	数据区域缩放，目前只支持直角坐标系的缩放

(3) ToolboxOpts：工具箱配置项如表9-3所示。

表9-3 工具箱配置项

配 置 项	说 明
is_show	是否显示工具栏组件
orient	工具栏 icon 的布局朝向，可选：horizontal（水平）、vertical（竖直）
pos_left	工具栏组件离容器左侧的距离。left 的值可以是像 20 这样的具体像素值，可以是像'20%'这样相对于容器高宽的百分比，也可以是 'left'、'center'、'right'。如果 left 的值为 'left'、'center'、'right'，组件就会根据相应的位置自动对齐
pos_right	工具栏组件离容器右侧的距离。right 的值可以是像 20 这样的具体像素值，也可以是像 '20%' 这样相对于容器高宽的百分比
pos_top	工具栏组件离容器上侧的距离。top 的值可以是像 20 这样的具体像素值，可以是像 '20%' 这样相对于容器高宽的百分比，也可以是 'top'、'middle'、'bottom'。如果 top 的值为'top'、'middle'、'bottom'，组件就会根据相应的位置自动对齐
pos_bottom	工具栏组件离容器下侧的距离。bottom 的值可以是像 20 这样的具体像素值，也可以是像 '20%' 这样相对于容器高宽的百分比
feature	各工具配置项

（4）TitleOpts：标题配置项如表9-4所示。

表9-4 标题配置项

配置项	说明
title	主标题文本，支持使用 \n 换行
title_link	主标题跳转 URL 链接
title_target	主标题跳转链接方式默认值是 blank，可选参数为'self'、'blank'、'self'为当前窗口打开，'blank'为新窗口打开
subtitle	副标题文本，支持使用 \n 换行
subtitle_link	副标题跳转 URL 链接
subtitle_target	副标题跳转链接方式默认值是 blank，可选参数为'self'、'blank'、'self'为当前窗口打开，'blank'为新窗口打开
pos_left	title 组件离容器左侧的距离。left 的值可以是像 20 这样的具体像素值，可以是像 '20%' 这样相对于容器高宽的百分比，也可以是'left'、'center'、'right'。如果 left 的值为'left'、'center'、'right'，组件就会根据相应的位置自动对齐
pos_right	title 组件离容器右侧的距离。right 的值可以是像 20 这样的具体像素值，也可以是像 '20%' 这样相对于容器高宽的百分比
pos_top	title 组件离容器上侧的距离。top 的值可以是像 20 这样的具体像素值，可以是像 '20%' 这样相对于容器高宽的百分比，也可以是'top'、'middle'、'bottom'。如果 top 的值为'top'、'middle'、'bottom'，组件就会根据相应的位置自动对齐
pos_bottom	title 组件离容器下侧的距离。bottom 的值可以是像 20 这样的具体像素值，也可以是像 '20%' 这样相对于容器高宽的百分比
title_textstyle_opts	主标题字体样式配置项
subtitle_textstyle_opts	副标题字体样式配置项

(5) DataZoomOpts：区域缩放配置项如表9-5所示。

表9-5 区域缩放配置项

配置项	说明
is_show	是否显示组件。如果设置为 false，就不会显示，但是数据过滤的功能还存在
type_	组件类型，可选 "slider"、"inside"
is_realtime	拖曳时，是否实时更新系列的视图。如果设置为 false，就只在拖曳结束的时候更新
range_start	数据窗口范围的起始百分比，范围是 0~100，表示 0%~100%
range_end	数据窗口范围的结束百分比，范围是 0~100
start_value	数据窗口范围的起始数值。如果设置了 start，startValue 就会失效
end_value	数据窗口范围的结束数值。如果设置了 end，endValue 就会失效
orient	布局方式是横还是竖。不仅是布局方式，对于直角坐标系而言，也决定了默认情况控制横向数轴还是纵向数轴，可选值为：'horizontal'或'vertical'
xaxis_index	设置 dataZoom-inside 组件控制的 x 轴（xAxis，直角坐标系中的概念，参见 grid）。不指定时，当 dataZoom-inside.orient 为 'horizontal'时，默认控制和 dataZoom 平行的第一个 xAxis。如果是 number，就表示控制一个轴；如果是 Array，就表示控制多个轴
yaxis_index	设置 dataZoom-inside 组件控制的 y 轴（yAxis，是直角坐标系中的概念）。不指定时，当 dataZoom-inside.orient 为 'horizontal'时，默认控制和 dataZoom 平行的第一个 yAxis。如果是 number，就表示控制一个轴；如果是 Array，就表示控制多个轴
is_zoom_lock	是否锁定选择区域（或叫作数据窗口）的大小。如果设置为 true，就锁定选择区域的大小，也就是说，只能平移，不能缩放
pos_left	dataZoom-slider 组件离容器左侧的距离。left 的值可以是像 20 这样的具体像素值，可以是像 '20%' 这样相对于容器高宽的百分比，也可以是'left'、'center'、'right'。如果 left 的值为 'left'、'center'、'right'，组件就会根据相应的位置自动对齐
pos_top	dataZoom-slider 组件离容器上侧的距离。top 的值可以是像 20 这样的具体像素值，可以是像'20%'这样相对于容器高宽的百分比，也可以是 'top'、'middle'、'bottom'。如果 top 的值为'top'、'middle'、'bottom'，组件就会根据相应的位置自动对齐
pos_right	dataZoom-slider 组件离容器右侧的距离。right 的值可以是像 20 这样的具体像素值，也可以是像 '20%'这样相对于容器高宽的百分比，默认自适应
pos_bottom	dataZoom-slider 组件离容器下侧的距离。bottom 的值可以是像 20 这样的具体像素值，也可以是像'20%'这样相对于容器高宽的百分比，默认自适应

（6）LegendOpts：图例配置项如表9-6所示。

表9-6 图例配置项

配 置 项	说 明
type_	图例的类型。可选值：'plain'，为普通图例，默认就是普通图例；'scroll'：可滚动翻页的图例，当图例数量较多时可以使用。
selected_mode	图例选择的模式，控制是否可以通过单击图例改变系列的显示状态。默认开启图例选择，可以设成 false，关闭图例选择，除此之外，也可以设成'single'或者'multiple'，使用单选或者多选模式
is_show	是否显示图例组件
pos_left	图例组件离容器左侧的距离。left 的值可以是像 20 这样的具体像素值，可以是像 '20%' 这样相对于容器高宽的百分比，也可以是 'left'、'center'、'right'。如果 left 的值为'left'、'center'、'right'，组件就会根据相应的位置自动对齐
pos_right	图例组件离容器右侧的距离。right 的值可以是像 20 这样的具体像素值，也可以是像 '20%' 这样相对于容器高宽的百分比
pos_top	图例组件离容器上侧的距离。top 的值可以是像 20 这样的具体像素值，可以是像 '20%' 这样相对于容器高宽的百分比，也可以是 'top'、'middle'、'bottom'。如果 top 的值为'top'、'middle'、'bottom'，组件就会根据相应的位置自动对齐
pos_bottom	图例组件离容器下侧的距离。bottom 的值可以是像 20 这样的具体像素值，也可以是像 '20%' 这样相对于容器高宽的百分比
orient	图例列表的布局朝向，可选：'horizontal'或'vertical'
textstyle_opts	图例组件的字体样式

(7) VisualMapOpts：视觉映射配置项如表9-7所示。

表9-7 视觉映射配置项

配置项	说明
type_	映射过渡类型，可选："color"或"size"
min_	指定 visualMapPiecewise 组件的最小值
max_	指定 visualMapPiecewise 组件的最大值
range_text	两端的文本，如['High', 'Low']
range_color	visualMap 组件过渡颜色
range_size	visualMap 组件过渡 symbol 大小
orient	如何放置 visualMap 组件：水平'horizontal'或竖直'vertical'
pos_left	visualMap 组件离容器左侧的距离。left 的值可以是像 20 这样的具体像素值，可以是像 '20%' 这样相对于容器高宽的百分比，也可以是 'left'、'center'、'right'。如果 left 的值为 'left'、'center'、'right'，组件就会根据相应的位置自动对齐
pos_right	visualMap 组件离容器右侧的距离。right 的值可以是像 20 这样的具体像素值，也可以是像 '20%' 这样相对于容器高宽的百分比
pos_top	visualMap 组件离容器上侧的距离。top 的值可以是像 20 这样的具体像素值，可以是像 '20%' 这样相对于容器高宽的百分比，也可以是 'top'、'middle'、'bottom'。如果 top 的值为'top'、'middle'、'bottom'，组件就会根据相应的位置自动对齐
pos_bottom	visualMap 组件离容器下侧的距离。bottom 的值可以是像 20 这样的具体像素值，也可以是像 '20%' 这样相对于容器高宽的百分比
split_number	对于连续型数据，自动平均切分成几段，默认为 5 段。连续数据的范围需要 max 和 min 来指定
dimension	组件映射维度
is_calculable	是否显示拖曳用的手柄（手柄能拖曳调整选中范围）
is_piecewise	是否为分段型
pieces	自定义每一段的范围、每一段的文字以及每一段的特别样式
out_of_range	定义在选中范围外的视觉元素（用户可以和 visualMap 组件交互，用鼠标或触摸选择范围）
textstyle_opts	文字样式配置项

（8）TooltipOpts：提示框配置项如表9-8所示。

表9-8 提示框配置项

配置项	说明
is_show	是否显示提示框组件，包括提示框浮层和 axisPointer
trigger	触发类型，可选：'item'、'axis'、'none'
trigger_on	提示框触发的条件，可选：'mousemove'，鼠标移动时触发；'click'，鼠标单击时触发；'mousemove click'，鼠标同时移动和单击时触发；'none'，不在'mousemove'或'click'时触发
axis_pointer_type	指示器类型。可选：'line'，直线指示器；'shadow'，阴影指示器；'none'，无指示器；'cross'，十字准星指示器。其实这是一种简写，表示启用两个正交的轴的 axisPointer
background_color	提示框浮层的背景颜色
border_color	提示框浮层的边框颜色
border_width	提示框浮层的边框宽
textstyle_opts	文字样式配置项

9.1.2 坐标轴配置项

Pyecharts 的坐标轴配置项主要包括：AxisLineOpts、AxisTickOpts、AxisPointerOpts、AxisOpts、SingleAxisOpts五项个配置。

(1) AxisLineOpts：坐标轴轴线配置项如表9-9所示。

表9-9 坐标轴轴线配置项

配置项	说明
is_show	是否显示坐标轴轴线
is_on_zero	X 轴或者 Y 轴的轴线是否在另一个轴的 0 刻度上，只有在另一个轴为数值轴且包含 0 刻度时有效
on_zero_axis_index	当有双轴时，可以用这个属性手动指定在哪个轴的 0 刻度上
symbol	轴线两边的箭头。可以是字符串，表示两端使用同样的箭头；或者长度为 2 的字符串数组，分别表示两端的箭头。默认不显示箭头，即 'none'。两端都显示箭头可以设置为 'arrow'。只在末端显示箭头可以设置为 ['none', 'arrow']
linestyle_opts	坐标轴线风格配置项

(2) AxisTickOpts：坐标轴刻度配置项如表9-10所示。

表9-10 坐标轴刻度配置项

配 置 项	说 明
is_show	是否显示坐标轴刻度
is_align_with_label	类目轴中在 boundaryGap 为 true 的时候有效，可以保证刻度线和标签对齐
is_inside	坐标轴刻度是否朝内，默认朝外
length	坐标轴刻度的长度
linestyle_opts	坐标轴线风格配置项

(3) AxisPointerOpts: 坐标轴指示器配置项如表9-11所示。

表9-11 坐标轴指示器配置项

配 置 项	说 明
is_show	默认显示坐标轴指示器
type_	指示器类型。可选参数：默认为 'line"line'，表示直线指示器，'shadow'为阴影指示器，'none'为无指示器
label	坐标轴指示器的文本标签，坐标轴标签配置项
linestyle_opts	坐标轴线风格配置项

(4) AxisOpts: 坐标轴配置项如表9-12所示。

表9-12 坐标轴配置项

配置项	说明
type_	坐标轴类型。可选: 'value'、'category'、'time'。例如会根据跨度的范围来决定使用月、星期、日还是小时范围的刻度。'log'为对数轴, 适用于对数数据
name	坐标轴名称
is_show	是否显示 x 轴
is_scale	只在数值轴中 (type: 'value') 有效。设置成 true 后, 坐标刻度不会强制包含零刻度。在双数值轴的散点图中比较有用。在设置 min 和 max 之后该配置项无效
is_inverse	是否强制设置坐标轴分割间隔
name_location	坐标轴名称显示位置。可选: 'start'、'middle' 或者 'center'、'end'
name_gap	坐标轴名称与轴线之间的距离
name_rotate	坐标轴名字旋转, 角度值
interval	强制设置坐标轴分割间隔。因为 splitNumber 是预估的值, 实际根据策略计算出来的刻度可能无法达到想要的效果, 这时可以使用 interval 配合 min、max 强制设定刻度划分, 一般不建议使用。无法在类目轴中使用。在时间轴 (type: 'time') 中需要传时间戳, 在对数轴 (type: 'log') 中需要传指数值
grid_index	x 轴所在的 grid 的索引, 默认位于第一个 grid
position	x 轴的位置。可选 'top'、'bottom'; 默认 grid 中的第一个 x 轴在 grid 的下方 ('bottom'), 第二个 x 轴视第一个 x 轴的位置放在另一侧
offset	Y 轴相对于默认位置的偏移, 在相同的 position 上有多个 Y 轴的时候有用
split_number	坐标轴的分割段数, 需要注意的是这个分割段数只是一个预估值, 最后实际显示的段数会在这个基础上根据分割后坐标轴刻度显示的易读程度进行调整。默认值是 5

(续表)

配置项	说明
boundary_gap	坐标轴两边的留白策略，类目轴和非类目轴的设置和表现不一样。在类目轴中，boundaryGap 可以配置为 true 和 false。默认为 true，这时刻度只是作为分隔线，标签和数据点都会在两个刻度之间的带（band）中间。非类目轴包括时间、数值、对数轴，boundaryGap 是两个值的数组，分别表示数据最小值和最大值的延伸范围，可以直接设置数值或者相对的百分比，在设置 min 和 max 后无效
min_	坐标轴刻度最小值。可以设置成特殊值 'dataMin'，此时取数据在该轴上的最小值作为最小刻度。不设置时会自动计算最小值，保证坐标轴刻度的均匀分布。在类目轴中，也可以设置为类目的序数（类目轴 data 也可以设置为负数，如-3）
max_	坐标轴刻度最大值。可以设置成特殊值 'dataMax'，此时取数据在该轴上的最大值作为最大刻度。不设置时会自动计算最大值，保证坐标轴刻度的均匀分布。在类目轴中，也可以设置为类目的序数（类目轴 data 也可以设置为负数，如-3）
min_interval	自动计算坐标轴的最小间隔大小。例如，可以设置成 1，保证坐标轴分割刻度显示成整数。默认值是 0
max_interval	自动计算坐标轴的最大间隔大小。例如，在时间轴（type: 'time'）可以设置成 3600 * 24 * 1000，保证坐标轴分割刻度最大为一天
axisline_opts	坐标轴刻度线配置项
axistick_opts	坐标轴刻度配置项
axislabel_opts	坐标轴标签配置项
axispointer_opts	坐标轴指示器配置项
name_textstyle_opts	坐标轴名称的文字样式
splitarea_opts	分割区域配置项
splitline_opts	分割线配置项

（5）SingleAxisOpts：单轴配置项如表9-13所示。

表9-13 单轴配置项

配置项	说明
name	坐标轴名称
max_	坐标轴刻度最大值。可以设置成特殊值 'dataMax', 此时取数据在该轴上的最大值作为最大刻度。不设置时会自动计算最大值, 保证坐标轴刻度的均匀分布。在类目轴中, 也可以设置为类目的序数 (如类目轴 data 也可以设置为负数, 如-3)
min_	坐标轴刻度最小值。可以设置成特殊值 'dataMin', 此时取数据在该轴上的最小值作为最小刻度。不设置时会自动计算最小值, 保证坐标轴刻度的均匀分布。在类目轴中, 也可以设置为类目的序数 (类目轴 data 也可以设置为负数, 如-3)
pos_left	single 组件离容器左侧的距离。left 的值可以是像 20 这样的具体像素值, 可以是像'20%'这样相对于容器高宽的百分比, 也可以是 'left'、'center'、'right'。如果 left 的值为'left'、'center'、'right', 组件就会根据相应的位置自动对齐
pos_right	single 组件离容器右侧的距离。right 的值可以是像 20 这样的具体像素值, 也可以是像 '20%' 这样相对于容器高宽的百分比
pos_top	single 组件离容器上侧的距离。top 的值可以是像 20 这样的具体像素值, 可以是像 '20%' 这样相对于容器高宽的百分比, 也可以是'top'、'middle'、'bottom'。如果 top 的值为'top'、'middle'、'bottom', 组件就会根据相应的位置自动对齐
pos_bottom	single 组件离容器下侧的距离。bottom 的值可以是像 20 这样的具体像素值, 也可以是像 '20%' 这样相对于容器高宽的百分比
width	single 组件的宽度。默认自适应
height	single 组件的高度。默认自适应
orient	轴的朝向, 默认为水平朝向, 可以设置成 'vertical' 垂直朝向
type_	坐标轴类型。可选: 'value'、'category'、'time', 例如会根据跨度的范围来决定使用月、星期、日还是小时范围的刻度。'log'为对数轴, 适用于对数数据

9.1.3 原生图形配置项

Pyecharts 的原生图形配置项包括：GraphicGroup、GraphicItem、GraphicBasicStyleOpts、GraphicShapeOpts、GraphicImage、GraphicText、GraphicTextStyleOpts、GraphicRect 8项配置。

(1) GraphicGroup：原生图形元素组件如表9-14所示。

表9-14 原生图形元素组件

配置项	说明
graphic_item	图形的配置项
is_diff_children_by_name	根据其 children 中每个图形元素的 name 属性进行重绘
children	子节点列表，其中项都是一个图形元素定义。目前可以选择 GraphicText、GraphicImage、GraphicRect

(2) GraphicItem: 原生图形配置项如表9-15所示。

表9-15 原生图形配置项

配置项	说明
id_	id 用于在更新或删除图形元素时指定更新或删除哪个图形元素，不需要用时可以忽略
action	指定对图形元素的操作行为。可选: 'merge', 如果已有元素, 新的配置项和已有的设定进行 merge, 如果没有就新建; 'replace', 如果已有元素, 就删除, 并新建元素进行替代; 'remove': 删除元素
position	平移: 默认值是 [0, 0]。表示 [横向平移的距离, 纵向平移的距离]。右和下为正值
rotation	旋转: 默认值是 0。表示旋转的弧度值。正值表示逆时针旋转

(续表)

配置项	说明
scale	缩放：默认值是 [1, 1]。表示 [横向缩放的倍数, 纵向缩放的倍数]
origin	origin 指定了旋转和缩放的中心点，默认值是 [0, 0]
left	描述怎么根据父元素进行定位。父元素是指：如果是顶层元素，父元素就是 echarts 图表容器；如果是 group 的子元素，父元素就是 group 元素。值的类型可以是：数值，表示像素值；百分比值，如 '33%'，用父元素的高和此百分比计算出最终值；位置，如 'center'，表示自动居中。注：left 和 right 只有一个可以生效。如果指定 left 或 right，shape 里的 x、cx 等定位属性就不再生效
right	值的类型可以是：数值，表示像素值；百分比值，如 '33%'，用父元素的高和此百分比计算出最终值，位置，如 'center'，表示自动居中。注：left 和 right 只有一个可以生效。如果指定 left 或 right，shape 里的 x、cx 等定位属性就不再生效
top	配置和 left 及 right 相同，注：top 和 bottom 只有一个可以生效
bottom	配置和 left 及 right 相同，注：top 和 bottom 只有一个可以生效
bounding	决定此图形元素在定位时，对自身的包围盒计算方式。可选：'all'（默认），表示用自身以及子节点整体的经过 transform 后的包围盒进行定位，这种方式易于使整体都限制在父元素范围中；'raw'，表示仅仅用自身（不包括子节点）的没经过 transform 的包围盒进行定位，这种方式易于内容超出父元素范围的定位方式
z	z 方向的高度，决定层叠关系
z_level	决定此元素绘制在哪个 canvas 层中。注意，越多 canvas 层会占用越多资源
is_silent	是否不响应鼠标以及触摸事件
is_invisible	节点是否可见
is_ignore	节点是否完全被忽略（既不渲染，又不响应事件）
cursor	鼠标悬浮在图形元素上时，鼠标的样式是什么。同 CSS 的 cursor
is_draggable	图形元素是否可以被拖曳
is_progressive	是否渐进式渲染。当图形元素过多时才使用
width	用于描述此 group 的宽。这个宽只用于给予节点定位。即便当宽度为零的时候，子节点也可以使用 left
height	用于描述此 group 的高。这个高只用于给予节点定位。即便当高度为零的时候，子节点也可以使用 top

（3）GraphicBasicStyleOpts：原生图形基础配置项如表9-16所示。

表9-16 原生图形基础配置项

配置项	说明
fill	填充色
stroke	笔画颜色
line_width	笔画宽度
shadow_blur	阴影宽度
shadow_offset_x	阴影 X 方向偏移
shadow_offset_y	阴影 Y 方向偏移
shadow_color	阴影颜色

（4）GraphicShapeOpts：原生图形形状配置项如表9-17所示。

表9-17 原生图形形状配置项

配置项	说明
pos_x	图形元素的左上角在父节点坐标系（以父节点左上角为原点）中的横坐标值
pos_y	图形元素的左上角在父节点坐标系（以父节点左上角为原点）中的纵坐标值
width	图形元素的宽度
height	图形元素的高度
r	可以用于设置圆角矩形。r 可以缩写，例如 r 缩写为[1]相当于[1,1,1,1]，r 缩写为[1,2]相当于[1,2,1,2]

（5）GraphicImage：原生图形图片配置项如表9-18所示。

表9-18 原生图形图片配置项

配 置 项	说 明
graphic_item	图形的配置项
graphic_imagestyle_opts	图形图片样式的配置项
image	图片的内容，可以是图片的 URL
pos_x	图形元素的左上角在父节点坐标系（以父节点左上角为原点）中的横坐标值
pos_y	图形元素的左上角在父节点坐标系（以父节点左上角为原点）中的纵坐标值
width	图形元素的宽度
height	图形元素的高度
opacity	透明度为 0~1，1 即完整显示
graphic_basicstyle_opts	图形基本配置项

（6）GraphicText：原生图形文本配置项如表9-19所示。

表9-19 原生图形文本配置项

配 置 项	说 明
graphic_item	图形的配置项
graphic_textstyle_opts	图形文本样式的配置项

（7）GraphicTextStyle0pts：原生图形文本样式配置项如表9-20所示。

表9-20 原生图形文本样式配置项

配 置 项	说 明
text	文本块文字。可以使用\n 来换行
pos_x	图形元素的左上角在父节点坐标系（以父节点左上角为原点）中的横坐标值
pos_y	图形元素的左上角在父节点坐标系（以父节点左上角为原点）中的纵坐标值
font	字体大小、字体类型、粗细、字体样式。例如 //size familyfont//style weight size familyfont//weight size familyfont
text_align	水平对齐方式，取值：'left'、'center'、'right'。默认值为'left'。如果为'left'，就表示文本最左端在 x 值上；如果为'right'，就表示文本最右端在 x 值上
text_vertical_align	垂直对齐方式，取值：'top'、'middle'、'bottom'。默认值为'None'
graphic_basicstyle_opts	图形基本配置项

(8) GraphicRect: 原生图形矩形配置项如表9-21所示。

表9-21 原生图形矩形配置项

配置项	说明
graphic_item	图形的配置项
graphic_shape_opts	图形的形状配置项
graphic_basicstyle_opts	图形基本配置项

9.2 系列配置项

Pyecharts视图的系列配置项文件位于\Anaconda3\Lib\site-packages\pyecharts\options下的series_options.py文件中，可以通过set_series_options方法进行设置。

9.2.1 样式类配置项

Pyecharts的样式类配置项主要包括：ItemStyleOpts、TextStyleOpts、LabelOpts、LineStyleOpts、SplitLineOpts五项。

(1) ItemStyleOpts: 图元样式配置项如表9-22所示。

表9-22 图元样式配置项

配置项	说明
color	图形的颜色
color0	阴线图形的颜色
border_color	图形的描边颜色。支持的颜色格式同 color，不支持回调函数
border_color0	阴线图形的描边颜色
Opacity	图形透明度。支持从 0~1 的数字，为 0 时不绘制该图形

(2) TextStyleOpts: 文字样式配置项如表9-23所示。

表9-23 文字样式配置项

配 置 项	说 明
color	文字颜色
font_style	文字字体的风格可选: 'normal'、'italic'、'oblique'
font_weight	主标题文字字体的粗细, 可选: 'normal'、'bold'、'bolder'、'lighter'
font_family	文字的字体系列还可以是'serif'、'monospace'、'Arial'、'Courier New'、'Microsoft YaHei'
font_size	文字的字大小
align	文字水平对齐方式, 默认自动
vertical_align	文字垂直对齐方式, 默认自动
line_height	行高
background_color	文字块背景色。可以是直接的颜色值, 例如'#123234'、'red'、'rgba(0,23,11,0.3)'
border_color	文字块边框颜色
border_width	文字块边框宽度
border_radius	文字块的圆角
padding	文字块的内边距, 例如 padding: [3, 4, 5, 6]表示 [上, 右, 下, 左] 的边距, padding: 4 表示 padding: [4, 4, 4, 4], padding: [3, 4]表示 padding: [3, 4, 3, 4]
shadow_color	文字块的背景阴影颜色
shadow_blur	文字块的背景阴影长度
width	文字块的宽度
height	文字块的高度
rich	在 rich 里面, 可以自定义富文本样式。利用富文本样式可以在标签中做出非常丰富的效果

(3) LabelOpts: 标签配置项如表9-24所示。

表9-24 标签配置项

配置项	说明
is_show	是否显示标签
position	标签的位置, 可选'top'、'left'、'right'、'bottom'、'inside'、'insideLeft'、'insideRight'、'insideTop'、'insideBottom'、'insideTopLeft'、'insideBottomLeft'、'insideTopRight'、'insideBottomRight'
color	文字的颜色。若设置为 'auto', 则为视觉映射得到的颜色, 如系列色
font_size	文字的字体大小
font_style	文字字体的风格, 可选'normal'、'italic'、'oblique'
font_weight	文字字体的粗细, 可选'normal'、'bold'、'bolder'、'lighter'
font_family	文字的字体系列还可以是'serif'、'monospace'、'Arial'、'Courier New'、'Microsoft YaHei'
rotate	标签旋转。从-90 度到 90 度。正值是逆时针
margin	刻度标签与轴线之间的距离
interval	坐标轴刻度标签的显示间隔, 在类目轴中有效。默认会采用标签不重叠的策略间隔显示标签。可以设置成 0 强制显示所有标签。如果设置为 1, 就表示隔一个标签显示一个标签, 如果值为 2, 就表示隔两个标签显示一个标签, 以此类推。可以用数值表示间隔的数据, 也可以通过回调函数控制。回调函数格式为: (index: number, value: string) =>boolean, 其中第一个参数是类目的 index, 第二个值是类目名称, 如果跳过返回 false
horizontal_align	文字水平对齐方式, 默认自动, 可选'left'、'center'、'right'
vertical_align	文字垂直对齐方式, 默认自动, 可选'top'、'middle'、'bottom'
rich	在 rich 里面可以自定义富文本样式。利用富文本样式可以在标签中做出非常丰富的效果

(4) LineStyleOpts: 线样式配置项如表9-25所示。

表9-25 线样式配置项

配置项	说明
width	线宽
opacity	图形透明度。支持从 0~1 的数字, 为 0 时不绘制该图形
curve	线的弯曲度, 0 表示完全不弯曲
type_	线的类型, 可选'solid'、'dashed'、'dotted'
color	线的颜色

(5) SplitLineOpts: 分割线配置项如表9-26所示。

表9-26 分割线配置项

配置项	说明
is_show	是否显示分割线
linestyle_opts	线风格配置项

9.2.2 标记类配置项

Pyecharts 的标记类配置项主要包括：MarkPointItem、MarkPointOpts、MarkLineItem、MarkLineOpts、MarkAreaItem、MarkAreaOpts六项。

(1) MarkPointItem：标记点数据项如表9-27所示。

表9-27 标记点数据项

数 据 项	说 明
name	标注名称
type_	特殊的标注类型，用于标注最大值、最小值等。可选：'min'，最小值；'max'，最大值；'average'，平均值
value_index	在使用 type 时有效，用于指定在哪个维度上指定最大值和最小值，可以是 0（xAxis, radiusAxis）、1（yAxis, angleAxis），默认使用第一个数值轴所在的维度
value_dim	在使用 type 时有效，用于指定在哪个维度上指定最大值和最小值。这可以是维度的直接名称，例如折线图中可以是 x、angle 等，candlestick 图中可以是 open、close 等维度名称
coord	标注的坐标。坐标格式视系列的坐标系而定，可以是直角坐标系上的 x、y，也可以是极坐标系上的 radius、angle，例如[121, 2323]、['aa', 998]
x	相对容器的屏幕 x 坐标，单位为像素
y	相对容器的屏幕 y 坐标，单位为像素
value	标注值，可以不设置
symbol	标记的图形。ECharts 提供的标记类型包括 'circle'、'rect'、'roundRect'、'triangle'、'diamond'、'pin'、'arrow'、'none'可以通过'image://url'设置为图片，其中 URL 为图片的链接，或者 dataURI
symbol_size	标记的大小，可以设置成诸如 10 这样单一的数字，也可以用数组分开表示宽和高，例如 [20, 10] 表示标记宽为 20、高为 10
itemstyle_opts	标记点样式配置项

（2）MarkPointOpts：标记点配置项如表9-28所示。

表9-28 标记点配置项

配 置 项	说 明
data	标记点数据
symbol	标记的图形。ECharts 提供的标记类型包括 'circle'、'rect'、'roundRect'、'triangle'、'diamond'、'pin'、'arrow'、'none'可以通过 'image'
symbol_size	标记的大小，可以设置成诸如 10 这样单一的数字，也可以用数组分开表示宽和高，例如[20, 10]表示标记宽为 20、高为 10。如果需要每个数据的图形大小不一样，就可以设置回调函数的格式为：(value: Arraynumber, params: Object) =>number/Array，其中第一个参数 value 为 data 中的数据值，第二个参数 params 是其他的数据项参数
label_opts	标签配置项

(3) MarkLineItem: 标记线数据项如表9-29所示。

表9-29 标记线数据项

数 据 项	说 明
name	标注名称
type_	特殊的标注类型, 用于标注最大值和最小值等。可选: 'min', 最小值; 'max', 最大值; 'average', 平均值
x	相对容器的屏幕 x 坐标, 单位为像素
y	相对容器的屏幕 y 坐标, 单位为像素
value_index	在使用 type 时有效, 用于指定在哪个维度上指定最大值和最小值, 可以是 0 (xAxis, radiusAxis)、1 (yAxis, angleAxis), 默认使用第一个数值轴所在的维度
value_dim	在使用 type 时有效, 用于指定在哪个维度上指定最大值和最小值。这可以是维度的直接名称, 例如折线图中可以是 x、angle 等, candlestick 图中可以是 open、close 等维度名称
coord	起点或终点的坐标。坐标格式视系列的坐标系而定, 可以是直角坐标系上的 x、y, 也可以是极坐标系上的 radius、angle
symbol	终点标记的图形。ECharts 提供的标记类型包括'circle'、'rect'、'roundRect'、'triangle'、'diamond'、'pin'、'arrow'、'none'可以通过'image://url'设置为图片, 其中 URL 为图片的链接或者 dataURI
symbol_size	标记的大小, 可以设置成诸如 10 这样单一的数字, 也可以用数组分开表示宽和高, 例如 [20, 10] 表示标记宽为 20、高为 10

(4) MarkLineOpts: 标记线配置项如表9-30所示。

表9-30 标记线配置项

配 置 项	说 明
is_silent	图形是否不响应和触发鼠标事件, 默认为 false, 即响应和触发鼠标事件
data	标记线数据
symbol	标线两端的标记类型, 可以是一个数组分别指定两端, 也可以是单个统一指定, 具体格式见 data.symbol
symbol_size	标线两端的标记大小, 可以是一个数组分别指定两端, 也可以是单个统一指定
precision	标线数值的精度, 在显示平均值线的时候有用
label_opts	标签配置项
linestyle_opts	标记线样式配置项

(5) MarkAreaItem: 标记区域数据项如表9-31所示。

表9-31 标记区域数据项

配置项	说明
name	区域名称, 仅仅就是一个名称而已
type_	特殊的标注类型, 用于标注最大值和最小值等。可选: 'min', 最小值; 'max', 最大值; 'average', 平均值
value_index	在使用 type 时有效, 用于指定在哪个维度上指定最大值和最小值, 可以是 0 (xAxis, radiusAxis)、1 (yAxis, angleAxis)。默认使用第一个数值轴所在的维度
value_dim	在使用 type 时有效, 用于指定在哪个维度上指定最大值和最小值。这可以是维度的直接名称, 例如折线图中可以是 x、angle 等, candlestick 图中可以是 open、close 等维度名称
x	相对容器的屏幕 x 坐标, 单位为像素, 支持百分比形式, 例如 '20%'
y	相对容器的屏幕 y 坐标, 单位为像素, 支持百分比形式, 例如 '20%'
label_opts	标签配置项
itemstyle_opts	该数据项区域的样式, 起点和终点项的 itemStyle 会合并到一起

(6) MarkAreaOpts: 标记区域配置项如表9-32所示。

表9-32 标记区域配置项

配置项	说明
is_silent	图形是否不响应和触发鼠标事件, 默认为 False, 即响应和触发鼠标事件
label_opts	标签配置项
data	标记区域数据

9.2.3 其他类配置项

Pyecharts 的其他类配置项主要包括: EffectOpts、AreaStyleOpts、SplitAreaOpts三项。

(1) EffectOpts: 涟漪特效配置项如表9-33所示。

表9-33 涟漪特效配置项

配置项	说明
is_show	是否显示特效
brush_type	波纹的绘制方式，可选 'stroke' 和 'fill'，Scatter 类型有效
scale	动画中波纹的最大缩放比例，Scatter 类型有效
period	动画的周期，秒数，Scatter 类型有效
color	特效标记的颜色
symbol	特效图形的标记。ECharts 提供的标记类型包括 'circle'、'rect'、'roundRect'、'triangle'、'diamond'、'pin'、'arrow'、'none'可以通过 'image://url'设置为图片，其中 URL 为图片的链接，或者 dataURI
symbol_size	特效标记的大小，可以设置成诸如 10 这样单一的数字，也可以用数组分开表示高和宽，例如 [20, 10] 表示标记宽为 20、高为 10
trail_length	特效尾迹的长度。取值为 0~1，数值越大，尾迹越长。Geo 图设置 Lines 类型时有效

(2) AreaStyleOpts：区域填充样式配置项如表9-34所示。

表9-34 区域填充样式配置项

配置项	说明
opacity	图形透明度。支持 0~1 的数字，为 0 时不绘制该图形
color	填充的颜色

(3) SplitAreaOpts：分隔区域配置项如表9-35所示。

表9-35 分隔区域配置项

配置项	说明
is_show	是否显示分隔区域
areastyle_opts	分隔区域的样式配置项

9.3 运行环境

在可视化分析中，Pyecharts可以生成HTML和图片文件，还可以运行在Jupyter Notebook和Jupyter Lab环境下，每种环境下的代码存在

一定差异。下面将结合案例进行介绍。

注意

每种运行环境下生成的视图都是如图9-1所示的视图，只是视图的输出方式上存在一定的差异。

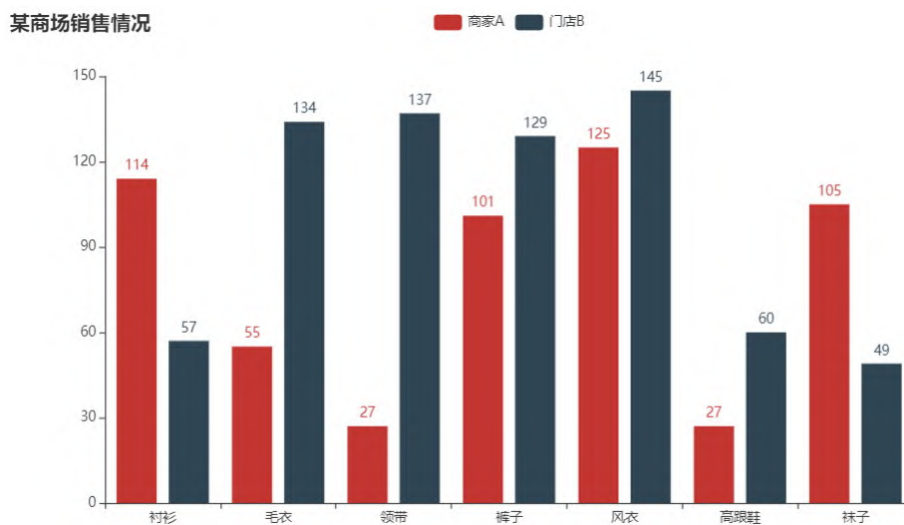


图9-1 商家A和商家B销售业绩分析

9.3.1 生成HTML

Pyecharts可以通过render函数生成HTML文件。下面使用代码绘制某商场商家A和商家B的销售情况的条形图，并将结果生成HTML文件。

```
from pyecharts.charts import Bar
from pyecharts import options as opts
bar=(
    Bar()
    .add_xaxis(["衬衫", "毛衣", "领带", "裤子", "风衣", "高跟鞋", "袜子"])
    .add_yaxis("商家A", [114, 55, 27, 101, 125, 27, 105])
```

```

        .add_yaxis("门店B", [57, 134, 137, 129, 145, 60, 49])
        .set_global_opts(title_opts=opts.TitleOpts(title="某商场销售情况"))
    )
    bar.render('mall_sales.html')

```

9.3.2 生成图片

Pyecharts可以直接将可视化视图生成图片，需要安装selenium、snapshot_selenium包，还需要下载Chromedriver，并复制到谷歌浏览器目录（…\Google\Chrome\Application）以及Python目录（…\Anaconda3\Scripts）下。下面使用代码绘制某商场商家A和商家B的销售情况的条形图，并生成图片。

```

from snapshot_selenium import snapshot as driver
from pyecharts import options as opts
from pyecharts.charts import Bar
from pyecharts.render import make_snapshot

def bar_chart() -> Bar:
    c=(
        Bar()
        .add_xaxis(["衬衫", "毛衣", "领带", "裤子", "风衣",
"高跟鞋", "袜子"])
        .add_yaxis("商家A", [114, 55, 27, 101, 125, 27,
105])
        .add_yaxis("门店B", [57, 134, 137, 129, 145, 60,
49])
        .reversal_axis()

.set_series_opts(label_opts=opts.LabelOpts(position="right"))

.set_global_opts(title_opts=opts.TitleOpts(title="Bar-测试渲染
图片"))
    )
    return c

```

```
# 需要安装 snapshot_selenium
make_snapshot(driver, bar_chart().render(), "bar.png")
```

9.3.3 Jupyter Notebook

Python代码可以在Jupyter Notebook环境中运行。下面使用代码绘制某商场商家A和商家B的销售情况的条形图。

```
from pyecharts.charts import Bar
from pyecharts import options as opts
bar=(
    Bar()
    .add_xaxis(["衬衫", "毛衣", "领带", "裤子", "风衣", "高跟鞋", "袜子"])
    .add_yaxis("商家A", [114, 55, 27, 101, 125, 27, 105])
    .add_yaxis("门店B", [57, 134, 137, 129, 145, 60, 49])
    .set_global_opts(title_opts=opts.TitleOpts(title="某商场销售情况"))
)
bar.render_notebook()
```

9.3.4 Jupyter Lab

Python代码可以在Jupyter Lab环境中运行。下面使用代码绘制某商场商家A和商家B的销售情况的条形图。

```
#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts.charts import Bar
from pyecharts import options as opts

bar=(
```

```
Bar()
    .add_xaxis(["衬衫", "毛衣", "领带", "裤子", "风衣", "高跟鞋", "袜子"])
    .add_yaxis("商家A", [114, 55, 27, 101, 125, 27, 105])
    .add_yaxis("门店B", [57, 134, 137, 129, 145, 60, 49])
    .set_global_opts(title_opts=opts.TitleOpts(title="某商场销售情况"))
)

#第一次渲染时调用load_javascript文件
bar.load_javascript()
bar.render_notebook()
```

第10章

Pyecharts基础绘图



Pyecharts可以方便地绘制一些基础视图，包括折线图、条形图、箱形图、涟漪散点图、K线图以及双坐标轴图等。本章将通过实际案例详细介绍每种视图的具体步骤。

10.1 折线图的绘制

10.1.1 折线图及其参数配置

折线图是用直线段将各个数据点连接起来而组成的图形，以折线方式显示数据的变化趋势。折线图可以显示随时间（根据常用比例设置）而变化的连续数据，因此非常适合显示相等时间间隔的数据趋势。在折线图中，类别数据沿水平轴均匀分布，值数据沿垂直轴均匀分布。例如，为了显示不同订单日期的销售额走势，可以创建不同订单日期的销售额折线图。

Pyecharts折线图的参数配置如表10-1所示。

表10-1 折线图参数配置

属 性	说 明
series_name	系列名称，用于 tooltip 的显示，legend 的图例筛选
y_axis	系列数据
is_selected	是否选中图例
is_connect_nones	是否连接空数据，空数据使用'None'填充
xaxis_index	使用的 x 轴的 index，在单个图表实例中存在多个 x 轴的时候有用
yaxis_index	使用的 y 轴的 index，在单个图表实例中存在多个 y 轴的时候有用
color	系列 label 的颜色
is_symbol_show	是否显示 symbol，如果为 false，就只有在 tooltip hover 的时候显示
symbol	标记的图形。ECharts 提供的标记类型包括 'circle'、'rect'、'roundRect'、'triangle'、'diamond'、'pin'、'arrow'、'none'可以通过'image://url'设置为图片，其中 URL 为图片的链接，或者 dataURI
symbol_size	标记的大小，可以设置成诸如 10 这样单一的数字，也可以用数组分开表示宽和高，例如[20, 10]表示标记宽为 20、高为 10
stack	数据堆叠，同个类目轴上系列配置相同的 stack 值可以堆叠放置
is_smooth	是否平滑曲线
is_step	是否显示成阶梯图
markpoint_opts	标记点配置项
markline_opts	标记线配置项
tooltip_opts	提示框组件配置项
label_opts	标签配置项
linestyle_opts	线样式配置项
areastyle_opts	填充区域配置项
itemstyle_opts	图元样式配置项

10.1.2 实例：各门店销售业绩比较分析

为了比较某企业各门店销售业绩，绘制了各门店的销售额和利润额的折线图，Python代码如下：

```

# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Line, Page
from impala.dbapi import connect

#连接Hadoop数据库
v1=[]
v2=[]
v3=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()

#读取Hadoop表数据
sql_num="SELECT
store_name,ROUND(SUM(sales/10000),2),ROUND(SUM(profit/10000),
2) FROM orders WHERE dt=2019 GROUP BY store_name"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])
    v3.append(s[2])

#画折线图
def line_toolbox() -> Line:
    c=(
        Line()
        .add_xaxis(v1)
        .add_yaxis("销售额", v2, is_smooth=True)
        .add_yaxis("利润额", v3,
is_smooth=True,is_selected=True)    #is_smooth默认是False，即折
线; is_selected默认是False，即不选中
        .set_global_opts(
            title_opts=opts.TitleOpts(title="门店销售额利润
额的比较分析", subtitle="2019年企业经营状况分析"),
            toolbox_opts=opts.ToolboxOpts(),
            legend_opts=opts.LegendOpts(is_show=True)
        )
    )

```

```

return c

#第一次渲染时调用load_javascript文件
line_toolbox().load_javascript()
#展示数据可视化图表
line_toolbox().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图10-1所示的折线图。

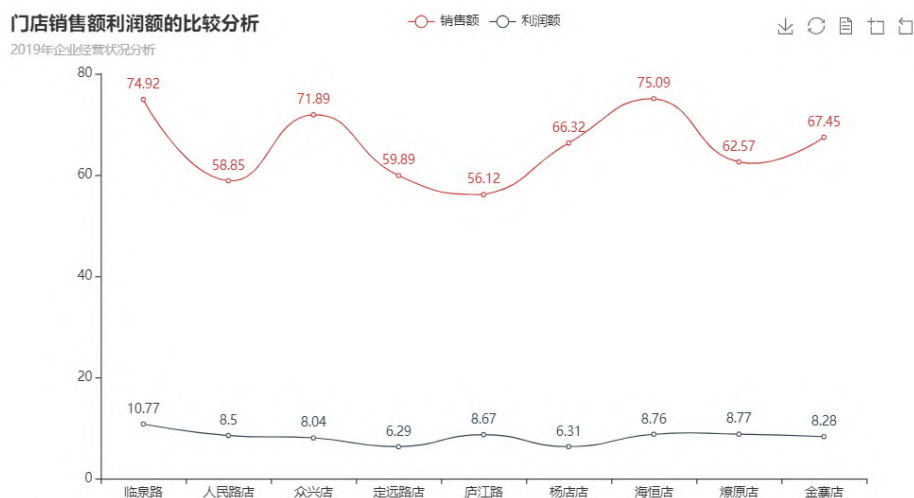


图10-1 各门店销售业绩分析

10.2 条形图的绘制

10.2.1 条形图及其参数配置

条形图是一种把连续数据画成数据条的表现形式，通过比较不同组的条形长度，从而对比不同组的数据量大小。描绘条形图的要素有3个：组数、组宽度、组限。绘制条形图时，不同组之间是有空隙的。条形图用来比较两个或以上的价值（不同时间或者不同条件），只有一个变量，通常用于较小的数据集分析。条形图亦可横向排列，或用多维方式表达。

条形图可分为垂直条和水平条。使用条形图可在各类别之间比较数据，例如客户的性别、受教育程度、购买方式等。绘制条形图时，长条柱或柱组中线需对齐项目刻度。相比之下，折线图则是将数据代表点对齐项目刻度。在数字大且接近时，两者皆可使用波浪形省略符号，以扩大表现数据间的差距，增强理解和清晰度。

Pyecharts条形图的参数配置如表10-2所示。

表10-2 条形图参数配置

属 性	说 明
series_name	系列名称，用于 tooltip 的显示，legend 的图例筛选
yaxis_data	系列数据
is_selected	是否选中图例
xaxis_index	使用的 x 轴的 index，在单个图表实例中存在多个 x 轴的时候有用
yaxis_index	使用的 y 轴的 index，在单个图表实例中存在多个 y 轴的时候有用
color	系列 label 颜色
stack	数据堆叠，同个类目轴上系列配置相同的 stack 值可以堆叠放置
category_gap	同一系列的柱间距离，默认为间距的 20%，表示柱子宽度的 20%
gap	如果想要两个系列的柱子重叠，那么可以设置 gap 为'-100%'
label_opts	标签配置项
markpoint_opts	标记点配置项
markline_opts	标记线配置项
tooltip_opts	提示框组件配置项
itemstyle_opts	图元样式配置项

add_yaxis函数的配置样例如下：

```
def add_yaxis(  
    series_name: str,  
    yaxis_data: Sequence[Numeric, opts.BarItem, dict],  
    is_selected: bool=True,  
    xaxis_index: Optional[Numeric]=None,  
    yaxis_index: Optional[Numeric]=None,
```

```

        color: Optional[str]=None,
        stack: Optional[str]=None,
        category_gap: Union[Numeric, str]="20%",
        gap: Optional[str]=None,
        label_opts: Union[opts.LabelOpts,
dict]=opts.LabelOpts(),
        markpoint_opts: Union[opts.MarkPointOpts, dict,
None]=None,
        markline_opts: Union[opts.MarkLineOpts, dict,
None]=None,
        tooltip_opts: Union[opts.TooltipOpts, dict,
None]=None,
        itemstyle_opts: Union[opts.ItemStyleOpts, dict,
None]=None,
    )

```

条形图的数据项在BarItem类中进行设置，具体如表10-3所示。

表10-3 BarItem类参数

属 性	说 明
name	数据项名称
value	单个数据项的数值
label_opts	单个柱条文本的样式设置
itemstyle_opts	图元样式配置项
tooltip_opts	提示框组件配置项

BarItem类样例如下：

```

class BarItem(
    name: Optional[str]=None,
    value: Optional[Numeric]=None,
    label_opts: Union[LabelOpts, dict, None]=None,
    itemstyle_opts: Union[ItemStyleOpts, dict, None]=None,
    tooltip_opts: Union[TooltipOpts, dict, None]=None,
)

```

在Pyecharts中有比较规范的条形图参数配置，绘制条形图时，只需要按照模板进行调用即可，基本函数形式如下：

```
def bar_base() -> Bar:
    c=(
        Bar()
        .add_xaxis(Faker.choose())
        .add_yaxis("门店A", Faker.values())
        .add_yaxis("门店B", Faker.values())

    .set_global_opts(title_opts=opts.TitleOpts(title="销售额统计",
        subtitle="2018年"))
    )
    return c
```

条形图可以默认取消显示某Series，例如取消显示门店B，将add_yaxis 修改为 add_yaxis(" 门店 B", Faker.values(), is_selected=False)。

如果要显示工具项ToolBox和图例项legend，那么可以在set_global_opts 中 添加 toolbox_opts=opts.ToolboxOpts(), legend_opts=opts.LegendOpts(is_show=False)。

10.2.2 实例：各省市商品订单数量分析

为了分析某企业在各省市的商品订单数量，绘制了各个省市商品订单量的条形图，Python代码如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
```

```

CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Bar, Page
from impala.dbapi import connect

#提取Hadoop集群数据
v1=[]
v2=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL', user='root')
cur=conn.cursor()
sql_num="select province,count(cust_id) from orders group
by province"
cur.execute(sql_num)
sh=cur.fetchall()
for s in sh:
    v2.append(s[1])
    v1.append(s[0])

#条形图参数配置
def bar_base() -> Bar:
    c=(
        Bar()
        .add_xaxis(v1,)
        .add_yaxis("客户订单量", v2)

.set_global_opts(title_opts=opts.TitleOpts(title="2019年客户订
单量区域分布"),
                    toolbox_opts=opts.ToolboxOpts(),
legend_opts=opts.LegendOpts(is_show=True))
    )
    return c

#第一次渲染时调用load_javascript文件
bar_base().load_javascript()
#展示数据可视化图表
bar_base().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图10-2所示的客户数量在各个省市的条形图。

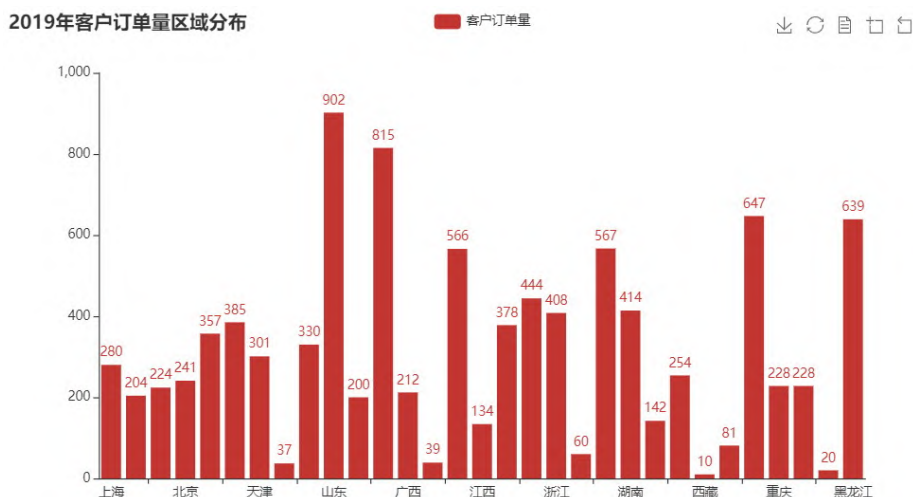


图10-2 各省市客户订单量

10.3 箱形图的绘制

10.3.1 箱形图及其参数配置

箱形图又称箱线图，是一种用作显示一组数据分散情况资料的统计图，因形状如箱子而得名。箱形图在各种领域经常被使用，常见于品质管理。

箱形图主要用于反映原始数据分布的特征，还可以进行多组数据分布特征的比较。箱形图的绘制方法是：先找出一组数据的上边缘、下边缘、中位数和两个四分位数；然后连接两个四分位数画出箱体；再将上边缘和下边缘与箱体相连接，中位数在箱体中间。

Pyecharts箱形图的参数配置如表10-4所示。

表10-4 箱形图参数配置

属 性	说 明
series_name	系列名称，用于 tooltip 的显示，legend 的图例筛选
y_axis	系列数据
is_selected	是否选中图例
xaxis_index	使用的 x 轴的 index，在单个图表实例中存在多个 x 轴的时候有用
yaxis_index	使用的 y 轴的 index，在单个图表实例中存在多个 y 轴的时候有用
label_opts	标签配置项
markpoint_opts	标记点配置项
markline_opts	标记线配置项
tooltip_opts	提示框组件配置项
itemstyle_opts	图元样式配置项

10.3.2 实例：不同类型商品的收益分析

为了分析某企业不同类型商品的收益情况，绘制了不同商品的箱形图，Python代码如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Boxplot, Page
from impala.dbapi import connect

#提取Hadoop集群数据
v1=[]
v2=[]
v3=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cur=conn.cursor()
sql_num="SELECT
subcategory,ROUND(SUM(sales/10000),2),ROUND(SUM(profit/10000)
,2) FROM orders WHERE dt=2019 GROUP BY subcategory"
cur.execute(sql_num)
```

```

sh=cur.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])
    v3.append(s[2])

def boxpolt_base() -> Boxplot:
    c=Boxplot()
    c.add_xaxis(["2019年业绩"]) \
    .add_yaxis("销售额", c.prepare_data([v2])) \
    .add_yaxis("利润额", c.prepare_data([v3])) \
    .set_global_opts(title_opts=opts.TitleOpts(title="不  
同类型商品销售收益分布分析", subtitle="2019年企业经营现  
状"), toolbox_opts=opts.ToolboxOpts())
    return c

#第一次渲染时调用load_javascript文件
boxpolt_base().load_javascript()
#展示数据可视化图表
boxpolt_base().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图10-3所示的箱形图。

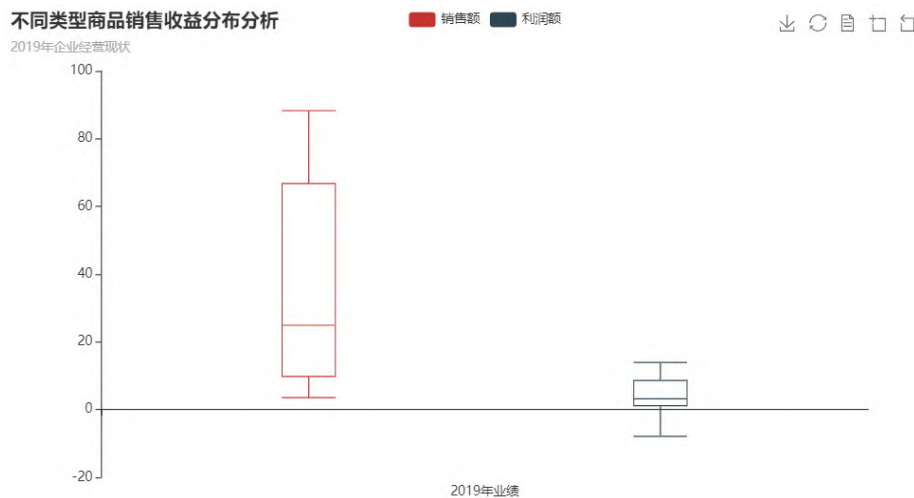


图10-3 不同类型商品业绩分析

10.4 涟漪散点图的绘制

10.4.1 涟漪散点图及其参数配置

涟漪散点图是一类特殊的散点图，只是散点图中带有涟漪特效，利用特效可以突出显示某些想要的数据。

Pyecharts涟漪散点图的参数配置如表10-5所示。

表10-5 涟漪散点图参数配置

属 性	说 明
series_name	系列名称，用于 tooltip 的显示，legend 的图例筛选
y_axis	系列数据
is_selected	是否选中图例
xaxis_index	使用的 x 轴的 index，在单个图表实例中存在多个 x 轴的时候有用
yaxis_index	使用的 y 轴的 index，在单个图表实例中存在多个 y 轴的时候有用
color	系列 label 颜色
symbol	标记的图形。ECharts 提供的标记类型包括'circle'、'rect'、'roundRect'、'triangle'、'diamond'、'pin'、'arrow'、'none'可以通过'image://url'设置为图片，其中 URL 为图片的链接，或者 dataURI
symbol_size	标记的大小，可以设置成诸如 10 这样单一的数字，也可以用数组分开表示宽和高，例如[20,10]表示标记宽为 20、高为 10
label_opts	标签配置项
markpoint_opts	标记点配置项
markline_opts	标记线配置项
tooltip_opts	提示框组件配置项
itemstyle_opts	图元样式配置项

涟漪特效散点图参数配置如图10-6所示。

表10-6 涟漪特效散点图

属 性	说 明
series_name	系列名称，用于 tooltip 的显示，legend 的图例筛选
y_axis	系列数据
is_selected	是否选中图例
xaxis_index	使用的 x 轴的 index，在单个图表实例中存在多个 x 轴的时候有用
yaxis_index	使用的 y 轴的 index，在单个图表实例中存在多个 y 轴的时候有用
color	系列 label 颜色
symbol	标记图形形状
symbol_size	标记的大小
label_opts	标签配置项
effect_opts	涟漪特效配置项
tooltip_opts	提示框组件配置项
itemstyle_opts	图元样式配置项

10.4.2 实例：不同收入等级客户价值分析

为了分析某企业不同收入等级客户的价值，绘制了不同等级客户的涟漪散点图，Python代码如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import EffectScatter, Page
from pyecharts.globals import SymbolType
from impala.dbapi import connect

#提取Hadoop集群数据
v1=[]
v2=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cur=conn.cursor()
```

```

sql_num="SELECT income,ROUND(SUM(sales/10000),2) FROM
customers,orders WHERE customers.cust_id=orders.cust_id and
dt=2019 GROUP BY income"
cur.execute(sql_num)
sh=cur.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])

def effectscatter_splitline() -> EffectScatter:
    c=(
        EffectScatter()
        .add_xaxis(v1)
        .add_yaxis("", v2, symbol=SymbolType.ARROW)
        .set_global_opts(
            title_opts=opts.TitleOpts(title="不同收入等级客
户的价值分析", subtitle="2019年企业经营现状"),

xaxis_opts=opts.AxisOpts(splitline_opts=opts.SplitLineOpts(is
_show=True)),

yaxis_opts=opts.AxisOpts(splitline_opts=opts.SplitLineOpts(is
_show=True)),

        toolbox_opts=opts.ToolboxOpts(),
        legend_opts=opts.LegendOpts(is_show=True)
    )
    return c

#第一次渲染时候调用load_javascript文件
effectscatter_splitline().load_javascript()
#展示数据可视化图表
effectscatter_splitline().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图10-4所示的涟漪散点图。

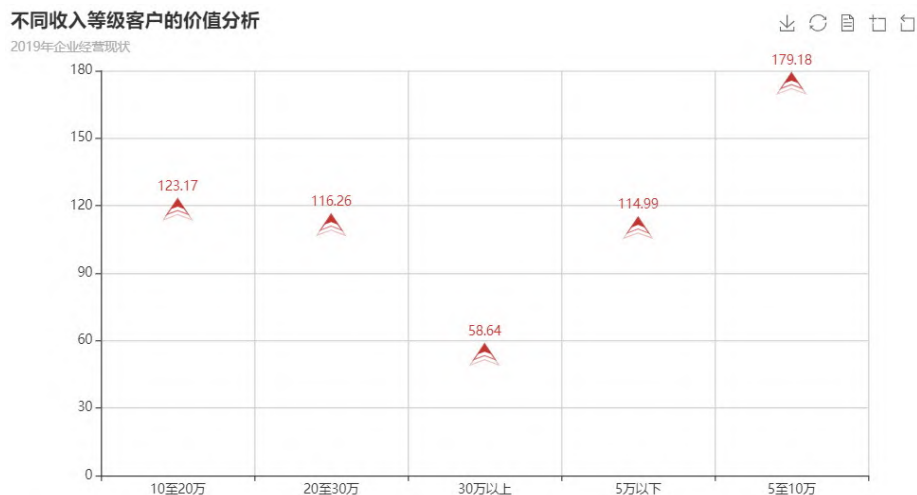


图10-4 客户价值分析

10.5 K线图的绘制

10.5.1 K线图及其参数配置

K线图又称蜡烛图，股市及期货市场中的K线图的画法包含4个数据，即开盘价、最高价、最低价、收盘价，所有的k线都是围绕这4个指标展开的，反映股票的状况。如果把每日的K线图放在一张纸上，就能得到日K线图，同样也可画出周K线图、月K线图。

Pyecharts的K线图参数配置如表10-7所示。

表10-7 K线图参数配置

属 性	说 明
series_name	系列名称，用于 tooltip 的显示，legend 的图例筛选
y_axis	系列数据
is_selected	是否选中图例
xaxis_index	使用的 x 轴的 index，在单个图表实例中存在多个 x 轴的时候有用
yaxis_index	使用的 y 轴的 index，在单个图表实例中存在多个 y 轴的时候有用
markline_opts	标记线配置项
markpoint_opts	标记点配置项
tooltip_opts	提示框组件配置项
itemstyle_opts	图元样式配置项

10.5.2 实例：企业股票价格趋势分析

为了分析某企业股票价格的区域，绘制了股票价格的K线图，具体过程如下：

- （1）导入options、Kline、Page、connect等包。
- （2）连接Hadoop集群，抽取股价表stocks数据。
- （3）配置K线图的相关参数以及全局配置项。
- （4）展示股票价格趋势。

实现上述步骤的Python代码如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Kline, Page
from impala.dbapi import connect
```

```

#连接Hadoop数据库
v1=[]
v2=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()

#读取Hadoop股价表数据
sql_num="SELECT trade_date,open,high,low,close FROM stocks
where year(trade_date)=2019 ORDER BY trade_date asc"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append([s[0]])
for s in sh:
    v2.append([s[1],s[2],s[3],s[4]])
data=v2

def kline_markline() -> Kline:
    c=(
        Kline()
        .add_xaxis(v1)
        .add_yaxis(
            "企业股票价格走势",
            data,
            markline_opts=opts.MarkLineOpts(
                data=[opts.MarkLineItem(type_="max",
value_dim="close")]
            ),
        )
        .set_global_opts(
            xaxis_opts=opts.AxisOpts(is_scale=True),
            yaxis_opts=opts.AxisOpts(
                is_scale=True,
                splitarea_opts=opts.SplitAreaOpts(
                    is_show=True,
areastyle_opts=opts.AreaStyleOpts(opacity=1)
                ),
            ),
            datazoom_opts=
[opts.DataZoomOpts(pos_bottom="-2%")],
            title_opts=opts.TitleOpts(title="企业股票价格趋
势分析", subtitle="2019年股价走势"),
            toolbox_opts=opts.ToolboxOpts(),
            legend_opts=opts.LegendOpts(is_show=True)
        )

```

```

    )
    return c

#第一次渲染时调用load_javascript文件
kline_markline().load_javascript()
#展示数据可视化图表
kline_markline().render_notebook()

```

在Jupyter Lab中运行上述代码，生成的K线图如图10-5所示。

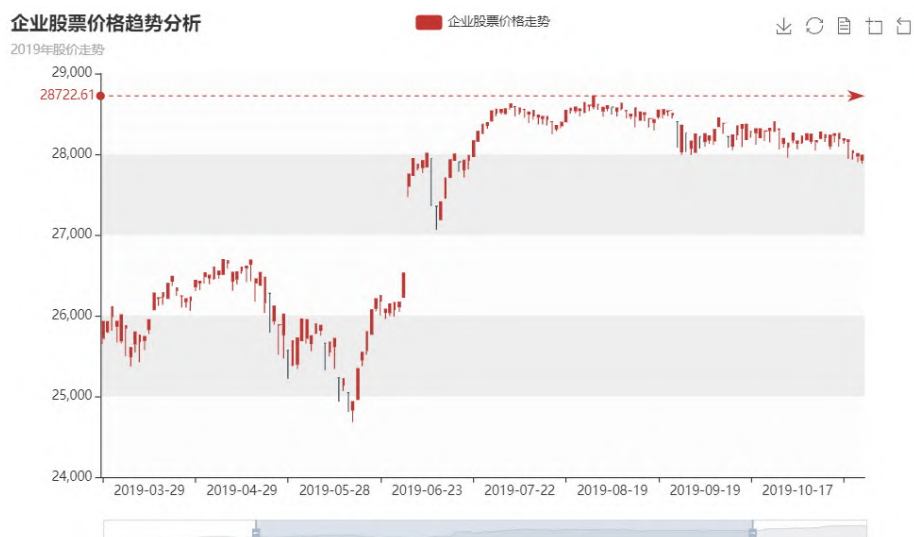


图10-5 股票价格走势

10.6 双坐标轴图的绘制

10.6.1 双坐标轴图及其参数配置

双坐标轴图是一种组合图表，一般将两种不同类型的图表组合在同一个“画布”上，如柱状图和折线图的组合；当然也可以将类型相同而数据单位不同的图表组合在一起。双坐标轴图中最难画的应该是“柱状图”与“柱状图”的组合，因为会遇到同一刻度对应“柱子”与“柱子”完全互相重叠的问题。

10.6.2 实例：区域销售业绩及数量分析

为了分析某企业在不同区域的销售业绩及数量，绘制了双坐标图，Python代码如下：

```
#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Scatter, Bar, Line
from impala.dbapi import connect

#连接Hadoop数据库
v1=[]
v2=[]
v3=[]
v4=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales', auth_mechanism='NOSASL', user='root')
cursor=conn.cursor()

#读取Hadoop表数据
sql_num="SELECT
region,ROUND(SUM(sales)/10000,2),ROUND(SUM(profit)/10000,2),R
OUND(SUM(amount),2) FROM customers,orders WHERE
customers.cust_id=orders.cust_id and dt=2019 GROUP BY region"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])
    v3.append(s[2])
    v4.append(s[3])

#柱形图与折线图组合
def overlap_bar_line() -> Bar:
    bar=(
        Bar()
        .add_xaxis(v1)
        .add_yaxis("销售额", v2)
        .add_yaxis("利润额", v3)
        .extend_axis(
```

```

        yaxis=opts.AxisOpts(
            axislabel_opts=opts.LabelOpts(formatter="{value} 件"), interval=500
        )

    .set_series_opts(label_opts=opts.LabelOpts(is_show=False))
    .set_global_opts(
        title_opts=opts.TitleOpts(title="区域销售业绩比较分析", subtitle="2019年企业经营状况分析"),
        toolbox_opts=opts.ToolboxOpts(),
        yaxis_opts=opts.AxisOpts(
            axislabel_opts=opts.LabelOpts(formatter="{value} 万元"), interval=20
        ),
    )

    line=Line().add_xaxis(v1).add_yaxis("销售数量", v4,
yaxis_index=1)
    bar.overlap(line)
    return bar

#第一次渲染时调用load_javascript文件
overlap_bar_line().load_javascript()
#展示数据可视化图表
overlap_bar_line().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图10-6所示的双坐标轴图。

区域销售业绩比较分析

2019年企业经营状况分析

■ 销售额 ■ 利润额 ○ 销售数量

📄 🔄 📊 📏 🏠

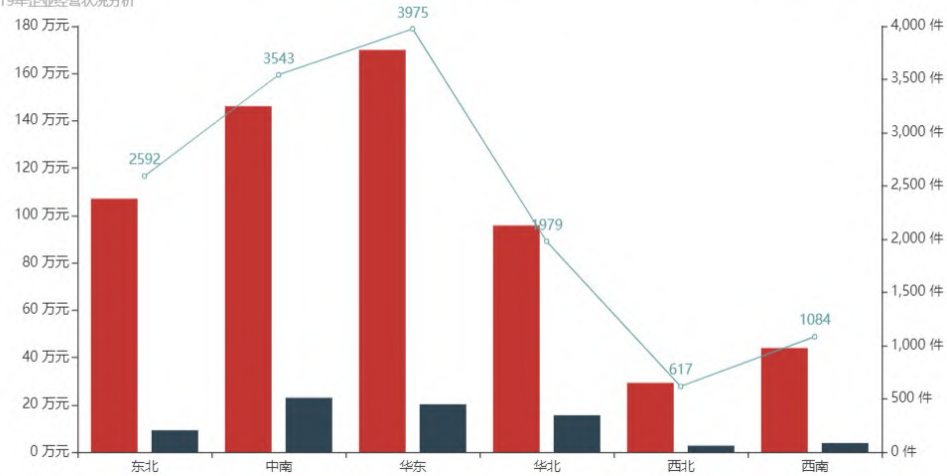


图10-6 双坐标轴图

第11章

Pyecharts高级绘图



Pyecharts可以生成一些比较复杂的视图，包括日历图、漏斗图、仪表盘、环形图、雷达图、旭日图、主题河流图、词云、玫瑰图等。本章将通过实际案例详细介绍每种视图的具体步骤。

11.1 日历图的绘制

11.1.1 日历图及其参数配置

日历图是一个日历数据视图，用于提供一段时间的日历布局，使我们可以更好地查看所选日期每一天的数据。

Pyecharts日历图的参数配置如表11-1所示。

表11-1 日历图参数配置

属 性	说 明
series_name	系列名称，用于 tooltip 的显示，legend 的图例筛选
yaxis_data	系列数据，格式为 [(date1, value1), (date2, value2), ...]
is_selected	是否选中图例
label_opts	标签配置项
calendar_opts	日历坐标系组件配置项
tooltip_opts	提示框组件配置项
itemstyle_opts	图元样式配置项

日历图坐标系组件的配置项如表11-2所示。

表11-2 日历图坐标系组件配置项

属 性	说 明
pos_left	calendar 组件离容器左侧的距离。left 的值可以是像 20 这样的具体像素值，可以是像'20%'这样相对于容器高宽的百分比，也可以是'left'、'center'、'right'。如果 left 的值为'left'、'center'、'right'，组件就会根据相应的位置自动对齐
pos_top	calendar 组件离容器上侧的距离。top 的值可以是像 20 这样的具体像素值，可以是像'20%'这样相对于容器高宽的百分比，也可以是'top'、'middle'、'bottom'。如果 top 的值为'top'、'middle'、'bottom'，组件就会根据相应的位置自动对齐
pos_right	calendar 组件离容器右侧的距离。right 的值可以是像 20 这样的具体像素值，也可以是像'20%'这样相对于容器高宽的百分比。默认自适应
pos_bottom	calendar 组件离容器下侧的距离。bottom 的值可以是像 20 这样的具体像素值，也可以是像'20%'这样相对于容器高宽的百分比。默认自适应
orient	日历坐标的布局朝向。可选：'horizontal'、'vertical'
range_	必填，日历坐标的范围支持多种格式，使用示例：某一年 range:2017，某个月 range:'2017-02'，某个区间 range:['2017-01-02','2017-02-23']
daylabel_opts	星期轴的样式
monthlabel_opts	月份轴的样式
yearlabel_opts	年份的样式

11.1.2 实例：企业股票每日交易量分析

为了分析某企业股票的成交量，绘制了股票每日交易量的日历图，Python代码如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Calendar, Page
from impala.dbapi import connect

#连接Hadoop数据库
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()

#读取Hadoop表数据
sql_num="SELECT trade_date,volume FROM stocks WHERE
year(trade_date)=2018"
cursor.execute(sql_num)
sh=cursor.fetchall()
v1=[]
for s in sh:
    v1.append([s[0],s[1]])
data=v1

#画日历图
def calendar_base() -> Calendar:

    c=(
        Calendar()
        .add("", data,
calendar_opts=opts.CalendarOpts(range_="2018"))
        .set_global_opts(
            title_opts=opts.TitleOpts(title="2018年股票交易
量分析"),
            visualmap_opts=opts.VisualMapOpts(
                max_=1000000000,
                min_=40000000,
                orient="horizontal",      #vertical表示垂直
的, horizontal表示水平的
                is_pieewise=True,
```

```
        pos_top="200px",
        pos_left="10px"
    ),
    toolbox_opts=opts.ToolboxOpts(),
    legend_opts=opts.LegendOpts(is_show=True)
)
)
return c

#第一次渲染时调用load_javascript文件
calendar_base().load_javascript()
#展示数据可视化图表
calendar_base().render_notebook()
```

在Jupyter Lab中运行上述代码，生成如图11-1所示的日历图。

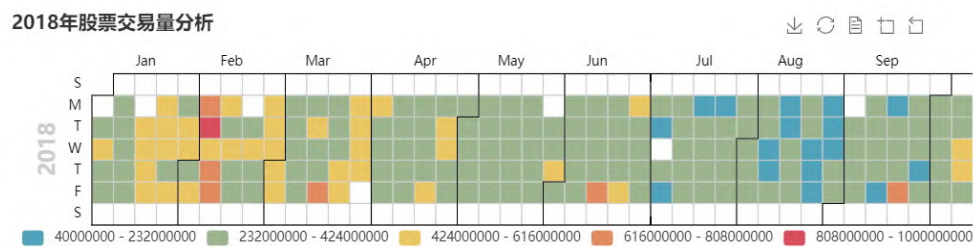


图11-1 日历图

11.2 漏斗图的绘制

11.2.1 漏斗图及其参数配置

漏斗图又叫倒三角图，适用于业务流程比较规范、周期长、环节多的流程分析，通过漏斗各环节业务数据的比较，能够直观地发现和说明问题所在，还可以应用于对数据从某个维度上进行比较。

Pyecharts漏斗图的参数配置如表11-3所示。

表11-3 漏斗图参数配置

属 性	说 明
series_name	系列名称，用于 tooltip 的显示，legend 的图例筛选
data_pair	系列数据项，格式为 [(key1, value1), (key2, value2)]
is_selected	是否选中图例
color	系列 label 颜色
sort_	数据排序，可以取 'ascending'、'descending'、'none'（表示按 data 顺序）
gap	数据图形间距
label_opts	标签配置项
tooltip_opts	提示框组件配置项
itemstyle_opts	图元样式配置项

11.2.2 实例：华东地区各省市利润额分析

为了分析某企业的商品在华东地区各省市的利润额情况，绘制了利润额的漏斗图，Python代码如下：

```
#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Funnel, Page
from impala.dbapi import connect

#连接Hadoop数据库
v1=[]
v2=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism=
'NOSASL',user='root')
cursor=conn.cursor()
#读取Hadoop数据
sql_num="SELECT province,ROUND(SUM(profit),2) FROM orders
WHERE dt=2019 and region='华东' GROUP BY province"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
```

```

v1.append(s[0])
v2.append(s[1])

#画漏斗图
def funnel_label() -> Funnel:
    c=(
        Funnel()
        .add("利润额",
            [list(z) for z in zip(v1, v2)],
            sort_="descending", #默认是
sort_="descending",即从大到小,也可以设置为ascending,即反向漏斗
            label_opts=opts.LabelOpts(position="inside"),
        )

    .set_global_opts(title_opts=opts.TitleOpts(title="区域利润额比
较分析", subtitle="2019年企业经营状况"),
                    toolbox_opts=opts.ToolboxOpts(),

legend_opts=opts.LegendOpts(is_show=True)
                    )
    )
    return c

#第一次渲染时调用load_javascript文件
funnel_label().load_javascript()
#展示数据可视化图表
funnel_label().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图11-2所示的漏斗图。

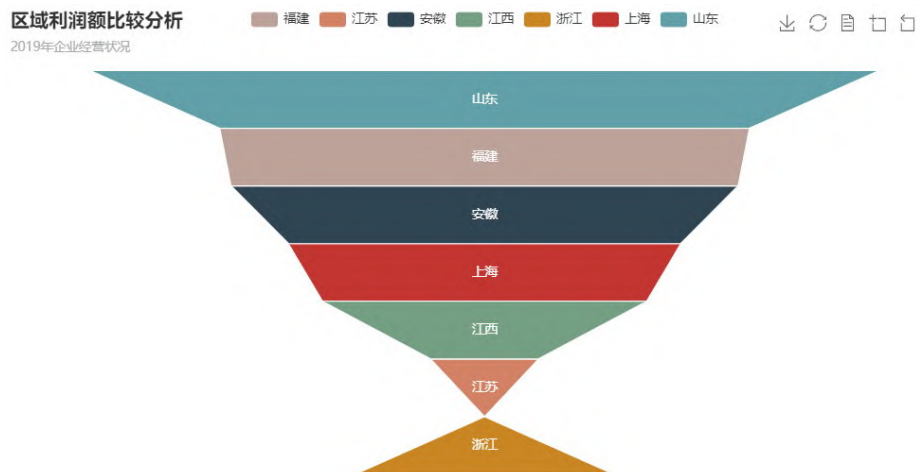


图11-2 漏斗图

11.3 仪表盘的绘制

11.3.1 仪表盘及其参数配置

仪表盘也被称为拨号图表或速度表图，其显示类似于拨号/速度计上读数的数据，是一种拟物化的展示形式。仪表盘的颜色可以用来划分指示值的类别，使用刻度标示数据，指针指示维度，指针角度表示数值。

仪表盘只需分配最小值和最大值，并定义一个颜色范围，指针（指数）将显示出关键指标的数据或当前进度。仪表盘可用于速度、体积、温度、进度、完成率、满意度等。

Pyecharts仪表盘的参数配置如表11-4所示。

表11-4 仪表盘参数配置

属 性	说 明
series_name	系列名称，用于 tooltip 的显示，legend 的图例筛选
data_pair	系列数据项，格式为 [(key1, value1), (key2, value2)]
is_selected	是否选中图例
min_	最小的数据值
max_	最大的数据值
split_number	仪表盘平均分割段数
start_angle	仪表盘起始角度。圆心正右手侧为 0 度，正上方为 90 度，正左手侧为 180 度
end_angle	仪表盘结束角度
label_opts	标签配置项
tooltip_opts	提示框组件配置项
itemstyle_opts	图元样式配置项

11.3.2 实例：企业2019年销售业绩完成率

为了分析某企业在2019年的销售业绩完成情况，绘制了销售额的仪表盘，Python代码如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Gauge, Page

def gauge_color() -> Gauge:
    c=(
        Gauge()
        .add("2019年公司销售指标完成率",
            [("完成率", 95.5)],
            axisline_opts=opts.AxisLineOpts(
                linestyle_opts=opts.LineStyleOpts(
                    color=[(0.3, "#67e0e3"), (0.7,
"#37a2da"), (1, "#fd666d")], width=30
                )
            ),
        )
        .set_global_opts(
            title_opts=opts.TitleOpts(title="公司销售指标分
析", subtitle="2019年企业经营状况"),
            toolbox_opts=opts.ToolboxOpts(),
            legend_opts=opts.LegendOpts(is_show=True),
        )
    )
    return c

#第一次渲染时调用load_javascript文件
gauge_color().load_javascript()
#展示数据可视化图表
gauge_color().render_notebook()
```

在Jupyter Lab中运行上述代码，生成如图11-3所示的仪表盘。



图11-3 仪表盘

11.4 环形图的绘制

11.4.1 环形图及其参数配置

环形图是由两个及两个以上大小不一的饼图叠在一起的，挖去中间的部分所构成的图形。环形图与饼图类似，但是又有区别。环形图中间有一个“空洞”，每个样本用一个环来表示，样本中的每一部分数据用环中的一段表示。因此，环形图可显示多个样本各部分所占的相应比例，从而有利于构成的比较研究。

Pyecharts环形图的参数配置如表11-5所示。

表11-5 环形图参数配置

属 性	说 明
series_name	系列名称，用于 tooltip 的显示，legend 的图例筛选
data_pair	系列数据项，格式为 [(key1, value1), (key2, value2)]
color	系列 label 颜色
radius	饼图的半径，数组的第一项是内半径，第二项是外半径，默认设置成百分比，相对于容器高宽中较小的一项的一半
center	饼图的中心（圆心）坐标，数组的第一项是横坐标，第二项是纵坐标，默认设置成百分比。设置成百分比时，第一项是相对于容器宽度的，第二项是相对于容器高度的
rosetype	是否展示成南丁格尔图，通过半径区分数据大小，有'radius'和'area'两种模式。 radius : 扇区圆心角展现数据的百分比，半径展现数据的大小； area : 所有扇区圆心角相同，仅通过半径展现数据大小
is_clockwise	饼图的扇区是否是顺时针排布的
label_opts	标签配置项
tooltip_opts	提示框组件配置项
itemstyle_opts	图元样式配置项

11.4.2 实例：不同教育群体的购买力分析

为了分析某企业客户群中不同教育群体的购买力情况，绘制了销售额的环形图，Python代码如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Page, Pie
from impala.dbapi import connect

#连接Hadoop数据库
v1=[]
v2=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()
```

```

#读取Hadoop表数据
sql_num="SELECT education,ROUND(SUM(sales/10000),2) FROM
customers,orders WHERE customers.cust_id=orders.cust_id and
dt=2019 GROUP BY education"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])

#画环形图
def pie_radius() -> Pie:
    c=(
        Pie()
        .add("",[list(z) for z in zip(v1, v2)],radius=
["40%", "75%"],)
        .set_colors(["blue", "green", "purple", "red",
"silver"])    #设置颜色
        .set_global_opts(
            title_opts=opts.TitleOpts(title="2019年不同教育
群体的购买力分析", subtitle="2019年销售经营状况分析"),
            toolbox_opts=opts.ToolboxOpts(),
            legend_opts=opts.LegendOpts(orient="vertical",
pos_top="35%", pos_left="2%"
            ),
        )

    .set_series_opts(label_opts=opts.LabelOpts(formatter="{b}:
{c}"))
    )
    return c

#第一次渲染时调用load_javascript文件
pie_radius().load_javascript()
#展示数据可视化图表
pie_radius().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图11-4所示的环形图。

2019年不同教育群体的购买力分析
2019年销售经营状况分析

导出 刷新 全屏 打印

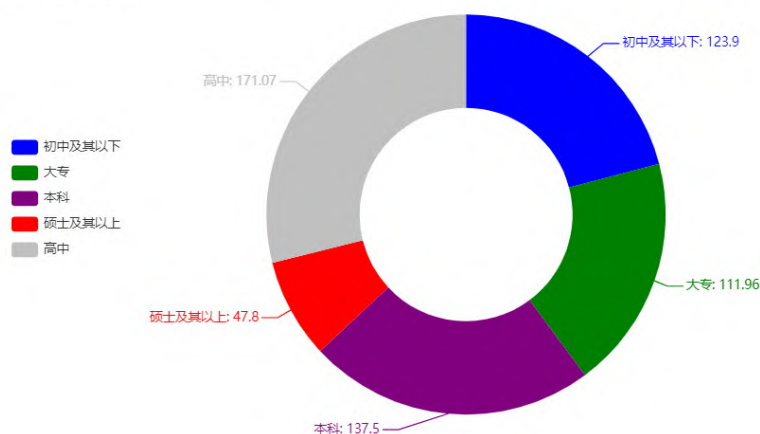


图11-4 环形图

11.5 雷达图的绘制

11.5.1 雷达图及其参数配置

雷达图又叫作蜘蛛网图，适用于显示3个或更多维度的变量。雷达图是以在同一点开始的轴上显示的3个或更多个变量的二维图表的形式来显示多元数据的，其中轴的相对位置和角度通常是无意义的。

雷达图的每个变量都有一个从中心向外发射的轴线，所有的轴之间的夹角相等，同时每个轴有相同的刻度，将轴到轴的刻度用网格线链接作为辅助元素，将每个变量在其各自的轴线的的数据点连接成一条多边形。

Pyecharts雷达图的参数配置如表11-6所示。

表11-6 雷达图参数配置

属 性	说 明
schema	雷达指示器配置项列表
shape	雷达图绘制类型，可选 'polygon' 和 'circle'
textstyle_opts	文字样式配置项
splitline_opt	分割线配置项
splitarea_opt	分隔区域配置项
axisline_opt	坐标轴轴线配置项

雷达图的数据项配置如表11-7所示。

表11-7 雷达图数据项配置

属 性	说 明
series_name	系列名称，用于 tooltip 的显示，legend 的图例筛选
data	系列数据项
is_selected	是否选中图例
symbol	ECharts 提供的标记类型包括 'circle'、'rect'、'roundRect'、'triangle'、'diamond'、'pin'、'arrow'、'none'可以通过 'image://url' 设置为图片，其中 URL 为图片的链接，或者 dataURI
color	系列 label 颜色
label_opts	标签配置项
linestyle_opts	线样式配置项
areastyle_opts	区域填充样式配置项
tooltip_opts	提示框组件配置项

雷达图的指示器配置如表11-8所示。

表11-8 雷达图指示器配置

属 性	说 明
name	指示器名称
min_	指示器的最小值，可选，建议设置
max_	指示器的最大值，可选，默认为 0
color	标签特定的颜色

11.5.2 实例：不同区域销售业绩的比较

为了分析某企业的商品在不同区域的销售业绩情况，绘制了销售额的雷达图，Python代码如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB
from pyecharts import options as opts
from pyecharts.charts import Page, Radar
from impala.dbapi import connect

#连接Hadoop数据库
v1=[]
v2=[]
v3=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()

#读取Hadoop表数据
sql_num="SELECT
region,ROUND(SUM(sales)/10000,2),ROUND(SUM(profit)/10000,2)
FROM orders WHERE dt=2019 GROUP BY region"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])
    v3.append(s[2])

#画雷达图
def radar_base() -> Radar:
    c=(
        Radar()
        .add_schema(
            schema=[
                opts.RadarIndicatorItem(name="华东",
max_=200),
                opts.RadarIndicatorItem(name="华南",
max_=200),
```

```

max_=200),
    opts.RadarIndicatorItem(name="东北",
max_=200),
    opts.RadarIndicatorItem(name="中南",
max_=200),
    opts.RadarIndicatorItem(name="西南",
max_=200),
    opts.RadarIndicatorItem(name="西北",
    ]
)
.add("销售额", [v2])
.add("利润额", [v3])
.set_global_opts(
    title_opts=opts.TitleOpts(title="区域销售额与利
润额的比较分析", subtitle="2019年企业经营状况"),

legend_opts=opts.LegendOpts(selected_mode="single"),
    toolbox_opts=opts.ToolboxOpts()
)

.set_series_opts(label_opts=opts.LabelOpts(is_show=False))
)
return c

#第一次渲染时调用load_javascript文件
radar_base().load_javascript()
#展示数据可视化图表
radar_base().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图11-5所示的雷达图。



图11-5 雷达图

11.6 旭日图的绘制

11.6.1 旭日图及其参数配置

旭日图可以展示多级数据，具有独特的外观。旭日图是一种现代饼图，它超越传统的饼图和环形图，能够清晰地表达层级和归属关系，以父子层次结构来显示数据构成情况。

Pyecharts旭日图的参数配置如表11-9所示。

表11-9 旭日图参数配置

属 性	说 明
series_name	系列名称，用于 tooltip 的显示，legend 的图例筛选
data_pair	数据项
center	旭日图的中心（圆心）坐标，数组的第一项是横坐标，第二项是纵坐标。支持设置成百分比，设置成百分比时，第一项是相对于容器宽度的，第二项是相对于容器高度的
radius	旭日图的半径。可以为如下类型：Sequence.<int str>，数组的第一项是内半径，第二项是外半径
highlight_policy	当鼠标移动到一个扇形块时，可以高亮显示相关的扇形块。'descendant'：高亮显示该扇形块和后代元素，其他元素将被淡化；'ancestor'：高亮显示该扇形块和祖先元素；'self'：只高亮显示自身；'none'：不会淡化其他元素
node_click	单击节点后的行为。可取值为：false，单击节点后无反应；'rootToNode'，单击节点后以该节点为根节点；'link'：如果节点数据中有 link，单击节点后就会进行超链接跳转
sort_	扇形块根据数据 value 的排序方式，如果未指定 value，其值就为子元素 value 之和。'desc'：降序排序；'asc'：升序排序；'null'：表示不排序，使用原始数据的顺序。使用 JavaScript 回调函数进行排列
levels	旭日图多层级配置
label_opts	标签配置项
itemstyle_opts	数据项的配置

旭日图的数据项配置如表11-10所示。

表11-10 旭日图数据项配置

属 性	说 明
value	数据值，如果包含 children，就可以不写 value 值。这时，将使用子元素的 value 之和作为父元素的 value。如果 value 大于子元素之和，就可以用来表示还有其他子元素未显示
name	显示在扇形块中的描述文字
link	单击此节点可跳转的超链接。Sunburst.add.node_click 值为 'link' 时才生效
target	意义同 HTML <a>标签中的 target，但跳转方式不同，blank 是在新窗口或者新的标签页中打开，self 则在当前页面或者当前标签页打开
label_opts	标签配置项
itemstyle_opts	数据项配置项
children	子节点数据项配置（与 SunburstItem 一致，递归下去）

11.6.2 实例：绘制我的家庭树旭日图

本例绘制的我的家庭树旭日图主要用于分析家庭人员的相互关系，Python代码如下：

[illegible]

```

value=1),
        ],
    ),
    ],
),
opts.SunburstItem(
    name="三爷爷",
    children=[
        opts.SunburstItem(
            name="李叔叔",
            children=[
                opts.SunburstItem(name="表哥李靖",
value=1),
                opts.SunburstItem(name="表妹李静",
value=2),
            ],
        ),
    ],
),
]

c=(
    Sunburst()
        .add(series_name="我的家庭树旭日图", data_pair=data,
radius=[0, "90%"])

.set_global_opts(title_opts=opts.TitleOpts(title="我的家庭树旭日图"),
                  toolbox_opts=opts.ToolboxOpts())

.set_series_opts(label_opts=opts.LabelOpts(formatter="{b}"))
)
return c

#第一次渲染时调用load_javascript文件
sunburst().load_javascript()
#展示数据可视化图表
sunburst().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图11-6所示的旭日图。

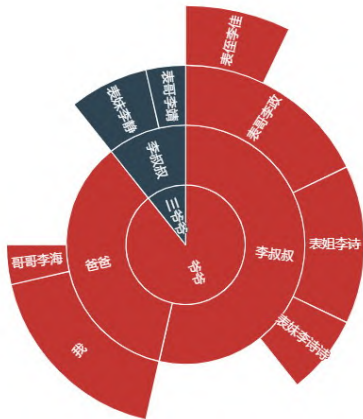


图11-6 旭日图

11.7 主题河流图的绘制

11.7.1 主题河流图及其参数配置

主题河流图是一种特殊的流图，主要用来表示事件或主题等在一段时间内的变化。这是一种围绕中心轴线移位的堆积面积图，导致流动的有机形状显示了不同类别的数据随时间的变化，通过使用流动的有机形状，有点类似河流的水流。

在主题河流图中，每个流的形状大小与每个类别中的值成比例，平行流动的轴变量用于时间，是显示大量数据集的理想选择，以便随时发现各种类别的趋势和模式。

Pyecharts主题河流图的参数配置如表11-11所示。

表11-11 主题河流图参数配置

属 性	说 明
series_name	系列名称，用于 tooltip 的显示，legend 的图例筛选
data	系列数据项
is_selected	是否选中图例
label_opts	标签配置项
tooltip_opts	提示框组件配置项
singleaxis_opts	单轴组件配置项

11.7.2 实例：不同类型商品销售情况分析

为了分析某企业不同类型商品的销售额情况，绘制了不同商品销售额的主题河流图，Python代码如下：

```
# -*- coding: utf-8 -*-
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB
from pyecharts import options as opts
from pyecharts.charts import Page, ThemeRiver
from impala.dbapi import connect

#连接Hadoop数据库
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()

#读取Hadoop表数据
sql_num="SELECT order_date,ROUND(SUM(sales),2),category
FROM orders WHERE order_date>='2019-10-01' and
order_date<='2019-10-31' GROUP BY category,order_date"
cursor.execute(sql_num)
sh=cursor.fetchall()
v1=[]
v2=[]
for s in sh:
    v1.append([s[0],s[1],s[2]])
```

```

#画主题河流图
def themeriver() -> ThemeRiver:
    c=(
        ThemeRiver()
        .add(
            ["办公用品","家具","技术"],
            v1,

singleaxis_opts=opts.SingleAxisOpts(type_="time",
pos_bottom="10%"),
        )

    .set_global_opts(title_opts=opts.TitleOpts(title="不同类型商品
销售额比较分析", subtitle="2019年企业经营状况"),
                    toolbox_opts=opts.ToolboxOpts(),

legend_opts=opts.LegendOpts(is_show=True)
                    )
    )
    return c

#第一次渲染时调用load_javascript文件
themeriver().load_javascript()
#展示数据可视化图表
themeriver().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图11-7所示的主题河流图。

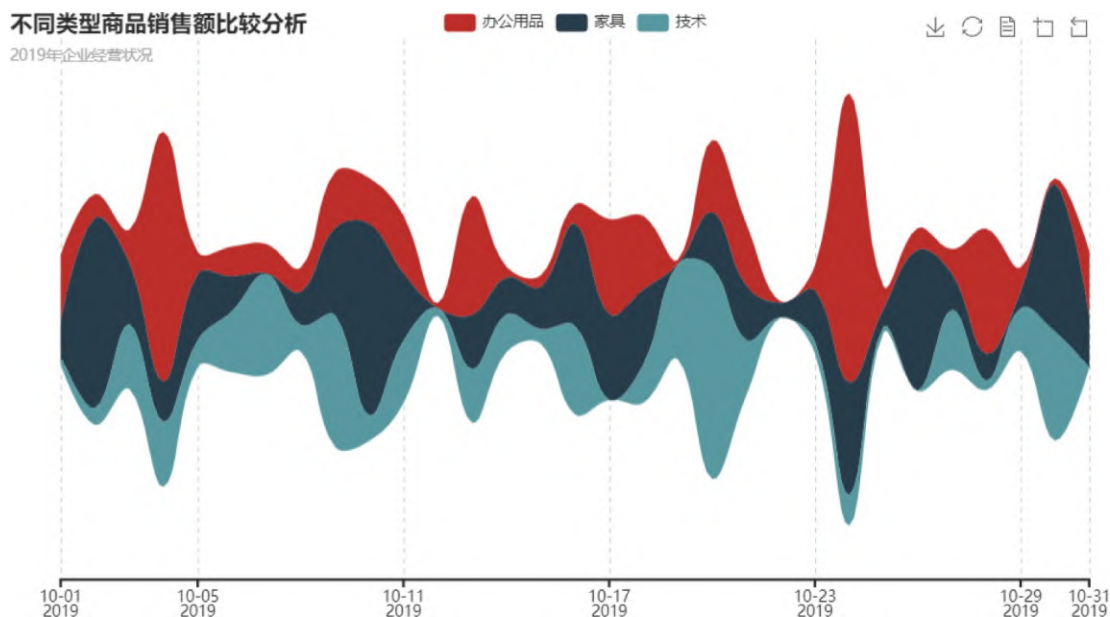


图11-7 主题河流图

11.8 词云的绘制

11.8.1 词云及其参数配置

词云就是对文本中出现频率较高的关键词予以视觉上的突出，形成“关键词云层”或“关键词渲染”，从而过滤掉大量的文本信息，使用户只要一眼扫过文本就可以领略文本的主旨。

Pyecharts词云的参数配置如表11-12所示。

表11-12 词云参数配置

属 性	说 明
series_name	系列名称，用于 tooltip 的显示，legend 的图例筛选
data_pair	系列数据项，[(word1, count1), (word2, count2)]，其中 word1 是指关键词 1，count1 是关键词 1 的数量，word2 是指关键词 2，count2 是关键词 2 的数量
shape	词云图轮廓，有'circle'、'cardioid'、'diamond'、'triangle-forward'、'triangle'、'pentagon'、'star'可选
word_gap	单词间隔
word_size_range	单词字体大小范围
rotate_step	旋转单词角度
tooltip_opts	提示框组件配置项

11.8.2 实例：商品类型关键词词云

为了分析某企业商品类型的构成情况，绘制了商品类型的关键词词云，Python代码如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Page, WordCloud
from pyecharts.globals import SymbolType
from impala.dbapi import connect

#读取Hadoop表数据
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()
sql_num="SELECT subcategory,count(subcategory) FROM orders
where dt=2019 GROUP BY subcategory"
cursor.execute(sql_num)
sh=cursor.fetchall()
v1=[]
for s in sh:
    v1.append((s[0],s[1]))
```

```

#画词云图
def wordcloud() -> WordCloud:
    c=(
        WordCloud()
        .add("", v1, word_size_range=[20,
160],shape=SymbolType.DIAMOND)

    .set_global_opts(title_opts=opts.TitleOpts(title="2019年销售商
品类型关键词词云"),toolbox_opts=opts.ToolboxOpts())
    )
    return c

#第一次渲染时调用load_javascript文件
wordcloud().load_javascript()
#展示数据可视化图表
wordcloud().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图11-8所示的词云。



图11-8 词云

11.9 玫瑰图的绘制

11.9.1 玫瑰图及其参数配置

玫瑰图是弗罗伦斯·南丁格尔发明的，又名鸡冠花图、极坐标区域图，是南丁格尔在克里米亚战争期间提交关于士兵死伤的报告时发明的一种图表。玫瑰图是在极坐标下绘制的柱状图，使用圆弧的半径长短表示数据的大小（数量的多少）。

由于半径和面积的关系是平方的关系，因此玫瑰图会将数据的比例大小夸大，尤其适合对比大小相近的数值。由于圆形有周期的特性，因此玫瑰图也适用于表示一个周期内的时间概念，比如星期、月份。

11.9.2 实例：不同职业群体的购买力分析

为了分析某企业不同职业群体的购买力情况，绘制了不同群体销售额的玫瑰图，Python代码如下：

```
# -*- coding: utf-8 -*-

#声明Notebook类型，必须在引入pyecharts.charts等模块前声明
from pyecharts.globals import CurrentConfig, NotebookType
CurrentConfig.NOTEBOOK_TYPE=NotebookType.JUPYTER_LAB

from pyecharts import options as opts
from pyecharts.charts import Page, Pie
from impala.dbapi import connect

#连接Hadoop数据库
v1=[]
v2=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()

#读取Hadoop表数据
sql_num="SELECT occupation,ROUND(SUM(sales/10000),2) FROM
customers,orders WHERE customers.cust_id=orders.cust_id and
dt=2019 GROUP BY occupation"
cursor.execute(sql_num)
```

```

sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])

#画玫瑰图
def rosetype() -> Pie:
    c=(
        Pie()
        .add(
            "",
            [list(z) for z in zip(v1, v2)],
            radius=["30%", "75%"],
            center=["50%", "55%"],
            rosetype="radius",
            label_opts=opts.LabelOpts(is_show=False),
        )
        .set_colors(["blue", "green", "purple", "red",
"silver"]) #设置颜色

    .set_global_opts(title_opts=opts.TitleOpts(title="2019年不同职业群体的购买力分析", subtitle="2019年销售经营状况分析"),

    legend_opts=opts.LegendOpts(orient="horizontal",
pos_top="5%", pos_left="30%"),
                        toolbox_opts=opts.ToolboxOpts())

    .set_series_opts(label_opts=opts.LabelOpts(formatter="{b}:{c}"))
    )
    return c

#第一次渲染时调用load_javascript文件
rosetype().load_javascript()
#展示数据可视化图表
rosetype().render_notebook()

```

在Jupyter Lab中运行上述代码，生成如图11-9所示的玫瑰图。

2019年不同职业群体的购买力分析

2019年销售经营状况分析

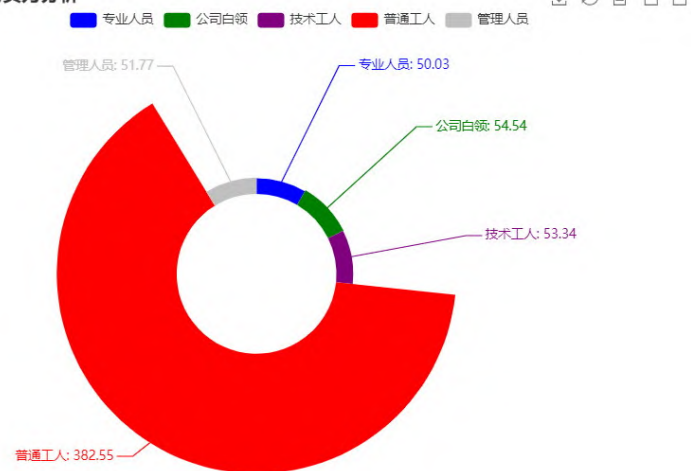


图11-9 玫瑰图

第12章

Web端的数据可视化



Web端的数据可视化可通过Pyecharts与Django框架结合来实现。

本章通过实际案例介绍Pyecharts与Django的集成，包括Django框架等。基于对于可视化的巨大需求以及成本因素，利用Pyecharts + Django的可视化方式显然是一种比较好的选择。

12.1 搭建Django开发环境

12.1.1 Django框架简介

Django是一个开放源代码的Web应用框架，由Python写成。Django遵守BSD版权，初次发布于2005年7月，并于2008年9月发布了第一个正式版本1.0。Django采用MVC的软件设计模式，即模型（Model）、视图（View）和控制器（Controller）。Django版本与Python版本的对应关系如表12-1所示。

表12-1 Django版本与Python版本的对应关系

Django 版本	Python 版本
1.8	2.7、3.2、3.3、3.4、3.5
1.9, 1.10	2.7、3.4、3.5
1.11	2.7、3.4、3.5、3.6
2.0	3.4、3.5、3.6、3.7
2.1, 2.2	3.5、3.6、3.7、3.8

Python下有许多款不同的Web框架。Django是重量级选手中很有代表性的一位。许多成功的网站和App都基于Django。Django最初被设计用于具有快速开发需求的新闻类站点，目的是实现简单快捷的网站开发。

Django是一个基于MVC构造的框架。但是在Django中，控制器接收用户输入的部分由框架自行处理，所以Django更关注模型（Model）、模板（Template）和视图（Views），称为MTV模式，具体如表12-2所示。

表12-2 MVC构造的框架

层 次	职 责
模型，即数据存取层	处理与数据相关的所有事务：如何存取、如何验证有效性、包含哪些行为以及数据之间的关系等
模板，即表现层	处理与表现相关的决定：如何在页面或其他类型的文档中进行显示
视图，即业务逻辑层	存取模型及调取恰当模板的相关逻辑。模型与模板的桥梁

（1）模型层

Django提供了一个抽象的模型层，为了构建和操纵Web应用的数据。模型是数据唯一而且准确的信息来源，它包含正在存储的数据的重要字段和行为。一般来说，每个模型都映射一个数据库表。每个模型都是一个Python的类，这些类继承django.db.models.Model，模型类的每个属性都相当于一个数据库的字段。

（2）模板层

模板层提供了一个对设计者友好的语法用于渲染向用户呈现的信息。作为一个Web框架，Django需要一种动态生成HTML的便捷方法。模板包含所需HTML输出的静态部分，以及一些特殊的模板语法，用于将动态内容插入静态部分。Django项目可以配置一个或多个模板引擎。Django定义了一个标准的API，用于加载和渲染模板，而不用考虑后端的模板系统。

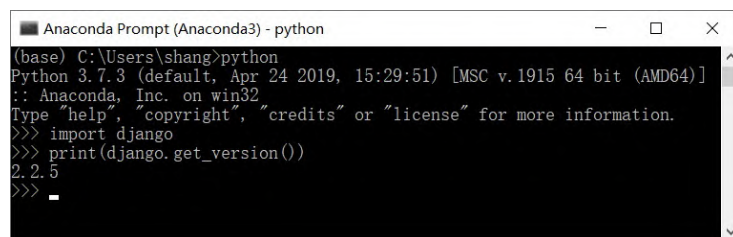
（3）视图层

Django具有“视图”的概念，负责处理用户的请求并返回响应。为了给一个应用设计URL，需要创建一个Python模块，通常被称为URLconf（URL Configuration）。这个模块是纯粹的Python代码模块，包含URL模式到Python函数（视图）的简单映射，映射可短可长，它可以引用其他的映射。而且，因为这个模块是纯粹的Python代码，所以可以动态构造。

12.1.2 Django开发环境

如果需要把大量的数据在Web端进行可视化，就需要绘制各类图表，数据保存在服务器中，使用D3.js这个数据可视化前端库来画图，但是需要编写大量的JS代码。但是Django框架只需要在后端编写少量的Python代码，再传送到前端让浏览器解析，从而大大地减少了工作量。

下面介绍Django开发环境。在Anaconda开发环境下，可以通过pip install django 命令安装Django。此外，我们还需要安装djanoestframework、sqlparse等包。这里安装的是Django 2.2.5版本，如图12-1所示。



```

Anaconda Prompt (Anaconda3) - python
(base) C:\Users\shang>python
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
:: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> print(django.get_version())
2.2.5
>>> _

```

图12-1 安装Django

Django工作机制（见图12-2）如下：

（1）用`manage .py runserver`启动Django服务器时就载入了在同一目录下的`settings .py`。该文件包含项目中的配置信息，其中重要的配置是`ROOT_URLCONF`，它告诉Django哪个Python模块应该用作本站的URLConf，默认的是`urls .py`。

（2）当访问URL的时候，Django会根据`ROOT_URLCONF`的设置来装载URLConf。

（3）然后按顺序逐个匹配URLConf里的URLpatterns。如果找到，就会调用相关联的视图函数，并把HttpRequest对象作为第一个参数（通常是request）。

（4）最后该view函数负责返回一个HttpResponse对象。

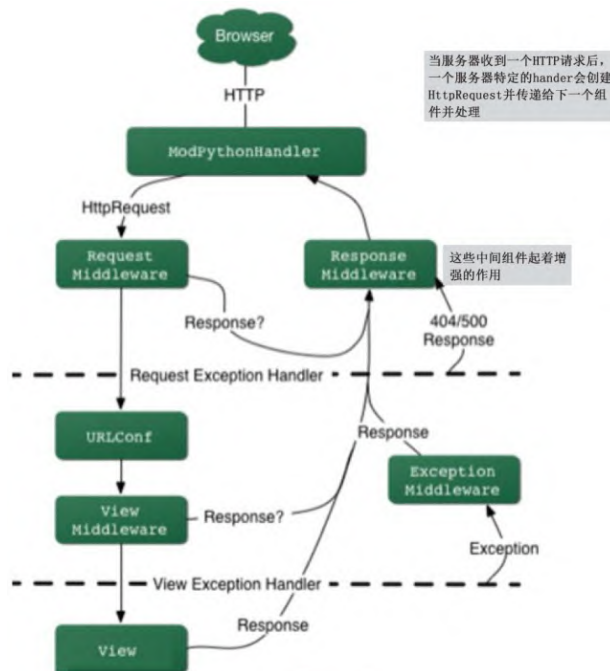


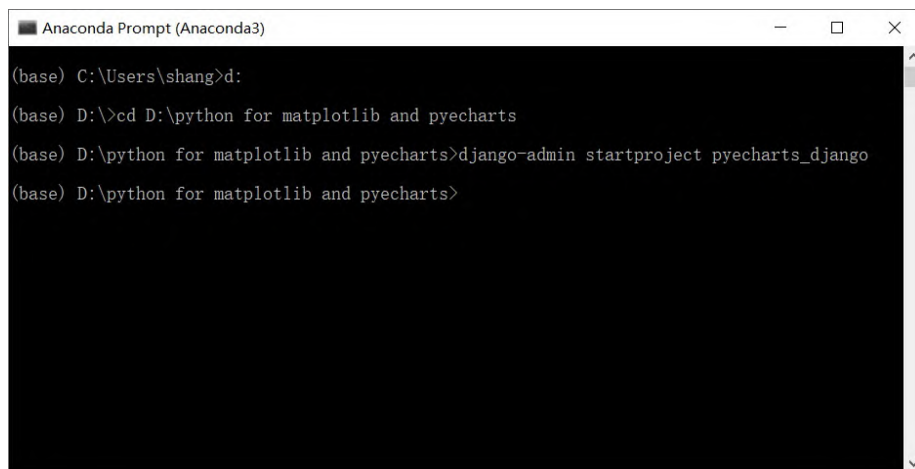
图12-2 Django的工作机制

12.2 Pyecharts与Django集成案例

为了研究某企业2019年在全国各个省市的有效订单与有效客户的数量，数据需要在Web端进行可视化，并绘制相关图表。可视化的数据存放在我们搭建的Hadoop集群中，通过Pyecharts与Django的集成，我们只需要在后端编写少量的Python代码，传送到前端让浏览器解析，就可以生成比较美观的可视化视图页面。

12.2.1 创建项目运行环境

在Anaconda Prompt中，切换到D盘的Python for Matplotlib and pyecharts 文件夹下，通过 `django-admin startproject pyecharts_django` 来创建pyecharts_django项目，如图12-3所示。

A screenshot of the Anaconda Prompt (Anaconda3) window. The window has a title bar with the text "Anaconda Prompt (Anaconda3)" and standard Windows window controls (minimize, maximize, close). The command prompt shows the following sequence of commands and their outputs:

```
(base) C:\Users\shang>d:
(base) D:\>cd D:\python for matplotlib and pyecharts
(base) D:\python for matplotlib and pyecharts>django-admin startproject pyecharts_django
(base) D:\python for matplotlib and pyecharts>
```

图12-3 切换工作路径

通过上面的操作，在D盘的Python for Matplotlib and pyecharts文件夹下将会生成pyecharts_django文件夹，其中包含manage.py和pyecharts_django两个文件：

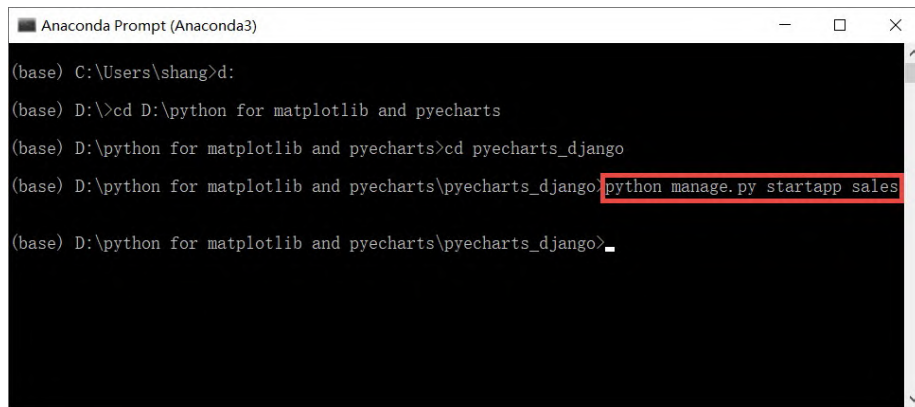
- manage.py：一个实用的命令行工具，可以以各种方式与该Django项目进行交互。
- pyecharts_django：项目的容器，存放项目相关的文件。

项目容器pyecharts_django文件夹下，初始默认包含以下文件：

- __init__.py：一个空文件，告诉Python该目录是一个Python包。
- settings.py：该Django项目的设置/配置。
- urls.py：该Django项目的URL声明，一份由Django驱动的网站“目录”。
- wsgi.py：一个WSGI兼容的Web服务器的入口，以便运行项目。

在D盘Python for Matplotlib and pyecharts文件夹下的pyecharts_django文件夹中，创建sales应用，运行Python manage.py

startapp sales命令，如图12-4所示，在pyecharts_django文件夹下将会新建一个sales项目文件。



```
Anaconda Prompt (Anaconda3)
(base) C:\Users\shang>d:
(base) D:\>cd D:\python for matplotlib and pyecharts
(base) D:\python for matplotlib and pyecharts>cd pyecharts_django
(base) D:\python for matplotlib and pyecharts\pyecharts_django>python manage.py startapp sales
(base) D:\python for matplotlib and pyecharts\pyecharts_django>_
```

图12-4 创建新的应用

12.2.2 配置项目参数文件

首先需要为应用添加应用名称，需要修改setting.py文件，它位于 D:\Python for Matplotlib and pyecharts\pyecharts_django\pyecharts_django 文件夹下，在 setting.py中添加注册新建的sales应用程序，修改的内容如下：

```
INSTALLED_APPS=[
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'sales',          #添加的APP名称
    'rest_framework',
]
```

然后打开生成的sales文件夹，再打开views.py文件，通过代码生成一个名叫index的视图，具体代码如下：

```

#导入相关的包
import json
from random import randrange
from django.http import HttpResponse
from rest_framework.views import APIView
from pyecharts import options as opts
from pyecharts.charts import Bar, Page
from impala.dbapi import connect

#连接Hadoop数据库
v1=[]
v2=[]
v3=[]
conn=connect(host='192.168.1.7', port=10000,
database='sales',auth_mechanism='NOSASL',user='root')
cursor=conn.cursor()

#读取Hadoop表数据
sql_num="SELECT province,count(distinct
order_id),count(distinct cust_id) FROM orders WHERE dt=2019
and return=0 GROUP BY province"
cursor.execute(sql_num)
sh=cursor.fetchall()
for s in sh:
    v1.append(s[0])
    v2.append(s[1])
    v3.append(s[2])

# Create your views here.
def response_as_json(data):
    json_str=json.dumps(data)
    response=HttpResponse(
        json_str,
        content_type="application/json",
    )
    response["Access-Control-Allow-Origin"]="*"
    return response

def json_response(data, code=200):
    data={
        "code": code,
        "msg": "success",
        "data": data,
    }
    return response_as_json(data)

```

```

def json_error(error_string="error", code=500, **kwargs):
    data={
        "code": code,
        "msg": error_string,
        "data": {}
    }
    data.update(kwargs)
    return response_as_json(data)

JsonResponse=json_response
JsonError=json_error

#画直方图
def bar_base() -> Bar:
    c=(
        Bar()
        .add_xaxis(v1)
        .add_yaxis("有效订单数", v2)
        .add_yaxis("有效客户数", v3, is_selected=True)
#is_selected默认是False, 即不选中
        .set_global_opts(
            title_opts=opts.TitleOpts(title="有效订单与有效
客户分析", subtitle="2019年企业经营状况分析"),
            toolbox_opts=opts.ToolboxOpts(),
            datazoom_opts=opts.DataZoomOpts(),
            legend_opts=opts.LegendOpts(is_show=True))
        .dump_options()
    )
    return c

class ChartView(APIView):
    def get(self, request, *args, **kwargs):
        return JsonResponse(json.loads(bar_base()))

class IndexView(APIView):
    def get(self, request, *args, **kwargs):
        return
HttpResponse(content=open("../templates/index.html").read())

```

为了使得编写的index视图有一个URL映射，在sales文件夹下新建一个urls.py文件，在其中输入如下代码：

```

from django.conf.urls import url
from . import views

urlpatterns=[
    url(r'^bar/$', views.ChartView.as_view(),
name='sales'),
    url(r'^index/$', views.IndexView.as_view(),
name='sales'),
]

```

在 D 盘 Python for Matplotlib and pyecharts\pyecharts_django\pyecharts_django 文件夹下创建 sales 的 urls 模块，打开 urls.py 文件，并在 urlpatterns 模块中插入一个 include() 函数，在其中输入如下代码：

```

from django.contrib import admin
from django.urls import path
from django.conf.urls import url, include

urlpatterns=[
    path('admin/', admin.site.urls),
    url(r'^sales/', include('sales.urls'))
]

```

为了简化可视化页面的配置，可以复制 Anaconda3\Lib\site-packages\pyecharts\render 文件夹下的 templates 到 D:\Python for Matplotlib and pyecharts\pyecharts_django 文件夹下，然后在 templates 中新建一个 index.html 文件，代码如下：

```

<!DOCTYPE html>
<html>
<h1 align="center" style="color:red;">2019年企业经营状况分析
</h1>
<title>客户分析</title>

```

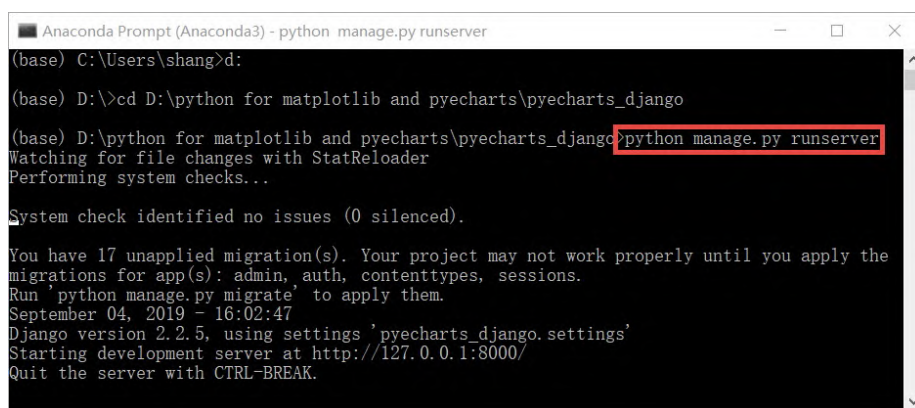
`<p class="parallax-alt" style="text-indent: 2em;">`2019年，该电商企业在全国各个省市的有效订单数和有效客户数存在明显的差异，排名靠前的是山东省、广东省和黑龙江，其中，山东省的有效订单数为274单，有效客户数为145个，其次是广东省，依次分别为251单和124个，以及黑龙江，依次分别为238单和113个，如图所示。`</p>`

```
<head>
  <meta charset="UTF-8">
  <script
src="https://cdn.bootcss.com/jquery/3.0.0/jquery.min.js">
</script>
  <script type="text/javascript"
src="https://assets.pyecharts.org/assets/echarts.min.js">
</script>
</head>
<style>
  div {
    margin: 0 auto;
    background-color: #cccccc;
  }
</style>
<body>
  <div id="bar" style="width:1450px; height:550px;">
</div>
  <script>
    var
chart=echarts.init(document.getElementById('bar'), 'white',
{renderer: 'canvas'});
    $(
      function () {
        getData(chart);
        setInterval(fetchData, 2000);
      }
    );
    function getData() {
      $.ajax({
        type: "GET",
        url: "http://127.0.0.1:8000/sales/bar",
        dataType: 'json',
        success: function (result) {
          chart.setOption(result.data);
        }
      });
    }
  </script>
</body>
  <p class="parallax-alt" style="text-indent: 2em;"><a
href="https://blog.csdn.net/shanghaiwren/">王老师的博客</a> : 在
```


我的个人博客上，有本人已经出版的所有专著的介绍，以及在学习工作中遇到的各类问题的解决方法及心得。</p>
</html>

12.2.3 测试项目运行效果

待上述项目的参数文件都配置完毕后，就可以测试一下实际效果。运行Python manage.py runserver命令，出现如图12-5所示的界面，说明应用正常启动。



```
Anaconda Prompt (Anaconda3) - python manage.py runserver
(base) C:\Users\shang>d:
(base) D:\>cd D:\python for matplotlib and pyecharts\pyecharts_django
(base) D:\python for matplotlib and pyecharts\pyecharts_django>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 17 unapplied migration(s). Your project may not work properly until you apply the
migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
September 04, 2019 - 16:02:47
Django version 2.2.5, using settings 'pyecharts_django.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

图12-5 测试项目参数

在浏览器中输入http://127.0.0.1:8000/sales/index/，就可以看到我们设计的页面信息，由于省份较多，可以通过拖曳下方的滚动条的方式查看所有省份的数据，如图12-6所示。

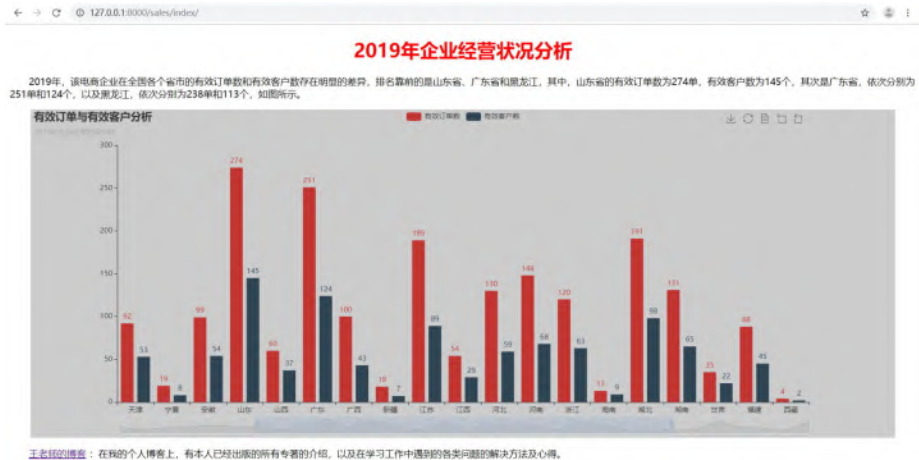


图12-6 浏览可视化页面

通过图12-6可以看出：在2019年，该电商企业在全国各个省市的有效订单数和有效客户数存在明显的差异，排名靠前的是山东省、广东省和黑龙江省，其中，山东省的有效订单数为274单，有效客户数为145个，其次是广东省，有效订单数为251单，有效客户数为124个，接着是黑龙江省，有效订单数为238单，有效客户数为113个。

附录

集群各节点的参数配置

Hadoop的参数配置

集群的Hadoop版本是2.5.2，可以到其官方网站下载，需要配置的文件为core-site.xml、hdfs-site.xml、mapred-site.xml、yarn-site.xml、slaves五个，都在Hadoop的/etc/hadoop文件夹下，配置完成后需要向集群其他机器节点分发，具体配置参数如下：

(1) core-site.xml

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://master:9000</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/home/dong/hadoopdata</value>
  </property>
  <property>
    <name>hadoop.proxyuser.root.hosts</name>
    <value>*</value>
  </property>
  <property>
    <name>hadoop.proxyuser.root.groups</name>
    <value>*</value>
  </property>
</configuration>
```

(2) hdfs-site.xml

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.permissions</name>
    <value>>false</value>
  </property>
</configuration>
```

(3) mapred-site.xml

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>mapreduce.map.memory.mb</name>
    <value>2048</value>
  </property>
  <property>
    <name>mapreduce.map.java.opts</name>
    <value>-Xmx2048M</value>
  </property>
  <property>
    <name>mapreduce.reduce.memory.mb</name>
    <value>4096</value>
  </property>
  <property>
    <name>mapreduce.reduce.java.opts</name>
    <value>-Xmx4096M</value>
  </property>
</configuration>
```

(4) yarn-site.xml

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.resourcemanager.address</name>
    <value>master:18040</value>
  </property>
  <property>
    <name>yarn.resourcemanager.scheduler.address</name>
    <value>master:18030</value>
  </property>
  <property>
    <name>yarn.resourcemanager.resource-
tracker.address</name>
    <value>master:18025</value>
  </property>
  <property>
    <name>yarn.resourcemanager.admin.address</name>
    <value>master:18141</value>
  </property>
  <property>
    <name>yarn.resourcemanager.webapp.address</name>
    <value>master:18088</value>
  </property>
</configuration>
```

(5) slaves

```
slave1
slave2
```

我们还需要将配置好的Hadoop文件复制到其他节点，注意此步骤的操作仍然是在master节点上，复制到slave1和slave2的语句如下：

```
scp -r /home/dong/hadoop-2.5.2 root@slave1:/home/dong/
scp -r /home/dong/hadoop-2.5.2 root@slave2:/home/dong/
```

Hive的参数配置

Hive将元数据存储在RDBMS中，一般常用MySQL和Derby。默认情况下，Hive元数据保存在内嵌的Derby数据库中，只能允许一个会话连接，仅仅适合简单的测试，实际生产环境中不适用。为了支持多用户会话，需要一个独立的元数据库，一般使用MySQL作为元数据库，Hive内部对MySQL也提供了很好的支持，因此在安装Hive之前需要安装MySQL数据库。

配置Hive时一定要记得加入MySQL的驱动包（mysql-connector-java-5.1.26-bin.jar），该JAR包放置在Hive根路径下的lib目录下。Hive是运行在Hadoop环境之上的，因此需要安装Hadoop环境，这里我们将Hive安装在Hadoop完全分布式模式的master节点上，需要配置hive-site.xml和hive-env.sh两个文件，具体配置参数如下：

(1) hive-env.sh

在hive-env.sh文件的最后添加以下内容：

```
export JAVA_HOME=/usr/java/jdk1.7.0_71/
export HADOOP_HOME=/home/dong/hadoop-2.5.2
export HIVE_HOME=/home/dong/apache-hive-1.2.2-bin
export HIVE_CONF_DIR=/home/dong/apache-hive-1.2.2-
bin/conf
```

(2) hive-site.xml

```
<configuration>
  <property>
    <name>hive.metastore.warehouse.dir</name>
    <value>/user/hive/warehouse</value>
```

```
</property>
<property>
  <name>hive.execution.engine</name>
  <value>mr</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:mysql://192.168.1.7:3306/hive?
createDatabaseIfNotExist=true&useSSL=false</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionDriverName</name>
  <value>com.mysql.jdbc.Driver</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionUserName</name>
  <value>root</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionPassword</name>
  <value>root</value>
</property>
<property>
  <name>hive.metastore.uris</name>
  <value>thrift://192.168.1.7:9083</value>
  <description></description>
</property>
<property>
  <name>hive.server2.authentication</name>
  <value>NOSASL</value>
</property>
<property>
  <name>hive.cli.print.header</name>
  <value>true</value>
</property>
<property>
  <name>hive.cli.print.current.db</name>
  <value>true</value>
</property>
<property>
  <name>hive.server2.thrift.port</name>
  <value>10000</value>
</property>
<property>
  <name>hive.server2.thrift.bind.host</name>
  <value>192.168.1.7</value>
</property>
</configuration>
```

Spark的参数配置

我们可以到 Spark 官网（<http://spark.apache.org/downloads.html>）下载 spark-1.4.0-bin-hadoop2.4，同时还需要下载scala-2.10.4.tgz，需要配置spark-defaults.conf、spark-env.sh、slaves共3个文件，配置完成后需要向集群其他机器节点分发。

（1）spark-defaults.conf

在spark-defaults.conf的最后添加以下内容：

```
spark.master=spark://master:7077
```

（2）spark-env.sh

在spark-env.sh的最后添加以下内容：

```
export HADOOP_CONF_DIR=/home/dong/hadoop-2.5.2/
export JAVA_HOME=/usr/java/jdk1.7.0_71/
export SCALA_HOME=/home/dong/scala-2.10.4
export SPARK_MASTER_IP=192.168.1.7
export SPARK_MASTER_PORT=7077
export SPARK_MASTER_WEBUT_PORT=8080
export SPARK_WORKER_PORT=7078
export SPARK_WORKER_WEBUT_PORT=8081
export SPARK_WORKER_CORES=1
export SPARK_WORKER_INSTANCES=1
export SPARK_WORKER_MEMORY=2g
export SPARK_JAR=/home/dong/spark-1.4.0-bin-hadoop2.4/lib/spark-assembly-1.4.0-hadoop2.4.0.jar
```


(3) slaves

```
slave1  
slave2
```

此外，还需要将配置好的hive-site.xml复制到Spark的配置文件下，最后将配置好的Spark和Scala复制到slave1和slave2两个从节点，注意此步骤的所有操作仍然是在master节点上，具体语句如下：

```
scp -r /home/dong/scala-2.10.4 root@slave1:/home/dong/  
scp -r /home/dong/spark-1.4.0-bin-hadoop2.4  
root@slave1:/home/dong/  
scp -r /home/dong/scala-2.10.4 root@slave2:/home/dong/  
scp -r /home/dong/spark-1.4.0-bin-hadoop2.4  
root@slave2:/home/dong/
```

Zeppelin的参数配置

Zeppelin是运行在Hive环境之上的，因此需要先安装和启动Hive，这里我们将其安装在Hadoop集群的master节点上，需要配置zeppelin-env.sh、zeppelin-site.xml和shiro.ini三个文件，具体配置参数如下：

(1) zeppelin-env.sh

在zeppelin-env.sh的最后添加以下内容：

```
export JAVA_HOME=/usr/java/jdk1.7.0_71/  
export MASTER=spark://192.168.1.7:7077  
export SPARK_HOME=/home/dong/spark-1.4.0-bin-hadoop2.4  
export HADOOP_CONF_DIR=/home/dong/hadoop-2.5.2/etc/hadoop
```

(2) zeppelin-site.xml

修改以下配置的参数值，其他参数值不用修改：

```
<property>
  <name>zeppelin.server.addr</name>
  <value>192.168.1.7</value>
  <description>Server address</description>
</property>
<property>
  <name>zeppelin.server.port</name>
  <value>7080</value>
  <description>Server port.</description>
</property>
<property>
  <name>zeppelin.anonymous.allowed</name>
  <value>false</value>
  <description>Anonymous user allowed by
default</description>
</property>
```

(3) shiro.ini

修改账号并添加root登录账号，密码是root，后期登录需要账号和密码，结果如下：

```
root=root, admin
#admin=password1, admin
#user1=password2, role1, role2
#user2=password3, role3
#user3=password4, role2
```

此外，还需要安装jdk-7u71-linux-x64.tar.gz，这个比较简单，只需要先解压文件，然后配置etc下的profile文件即可，可以参考网络上的相关资料，这里不做详细介绍。

集群的启动与关闭

由于Hadoop集群上的软件较多，因此集群的启动程序命令相对比较复杂，为了防止启动错误，我们这里使用绝对路径，具体启动命令如下：

(1) Hadoop的启动和关闭：

启动： /home/dong/hadoop-2.5.2/sbin/start-all.sh。

关闭： /home/dong/hadoop-2.5.2/sbin/stop-all.sh。

(2) Hive的启动：

```
nohup hive --service metastore > metastore.log 2>&1 &  
hive --service hiveserver2 &
```

Hive的关闭一般是通过kill命令实现的，即kill加进程编号。

(3) Spark的启动和关闭：

启动： /home/dong/spark-1.4.0-bin-hadoop2.4/sbin/start-all.sh。

关闭： /home/dong/spark-1.4.0-bin-hadoop2.4/sbin/stop-all.sh。

(4) Zeppelin的启动和关闭：

启动： /home/dong/zeppelin-0.7.3-bin-all/bin/zeppelin-daemon.sh start。

关 闭 : /home/dong/zeppelin-0.7.3-bin-all/bin/zeppelin-daemon.sh stop。

参考文献

- [1] 刘宝华, 牛婷婷, 秦洲, 张立东. 基于Tableau大数据的隧道技术状况分析[J]. 公路, 2019, 03:342-346.
- [2] 白玲. Tableau在医疗卫生数据可视化分析中的应用[J]. 中国数字医学, 2018, 1310:72-74+77.
- [3] 白玲. 基于Tableau工具的医疗数据可视化分析[J]. 中国医院统计, 2018, 2505:399-401.
- [4] 古锐昌, 丁钰琳. Tableau在气象大数据可视化分析中的应用[J]. 广东气象, 2017, 3906:40-42.
- [5] 陈佳艳. 基于Tableau实现在线教育大数据的可视化分析[J]. 江苏商论, 2018, 02:123-125.
- [6] 黄亮, 戴小鹏, 王奕. 基于Tableau的商业数据可视化分析[J]. 电脑知识与技术, 2018, 1429:14-15+17.
- [7] 王露, 杨晶晶, 黄铭. 基于R语言和Tableau的气象数据可视化分析[J]. 计算机与网络, 2017, 4324:69-71.
- [8] 李良才, 张家铭, 崔昌宇, 邓文佩, 叶玮. 基于Tableau实现MOOC学习行为数据可视化分析[J]. 电脑编程技巧与维护, 2016, 22:47+75.
- [9] 赵三珊, 沈豪栋, 许唐云, 王华, 李莉华. 基于Tableau技术的电网企业综合计划监测体系研究[J]. 电力与能源, 2018, 3903:339-343.

[10] 张蕾, 李昂, 向翰丞. 基于Tableau的大电量客户用电量异常分析[J]. 电工技术, 2018, 13:76-77.

[11] 杨月. Tableau在航运企业航线营收数据分析中的应用[J]. 集装箱化, 2018, 2908:8-9.

[12] 郭二强, 李博. 基于Excel和Tableau实现企业业务数据化管理[J]. 电子技术与软件工程, 2018, 20:1611.

[13] 杨小军, 张雪超, 李安琪. 利用Excel和Tableau实现业务工作数据化管理[J]. 电脑编程技巧与维护, 2017, 12:66-611.

[14] 马佳琪, 滕国文. 基于Matplotlib的大数据可视化应用研究[J]. 电脑知识与技术, 2019, 15(17):18-19.

[15] 刘雨珂, 王平. 基于Python+Pandas+Matplotlib的学生成绩数据统计与图形输出实现[J]. 福建电脑, 2017, 33(11):104-106+142.

[16] 李保源. Matplotlib在计算结果可视化中的应用[J]. 现代计算机, 2007(01):81-84.

[17] 贾利娟, 刘娟, 王健, 周国民. 基于PyEcharts的全球玉米贸易数据可视化系统建设及应用展望[J]. 农业展望, 2019, 15(03):46-54.

[18] 蔡敏. Python语言的Web开发应用分析[J]. 无线互联科技, 2019, 16(04):27-28.

[19] 杨菲菲. 基于Hadoop的面向信管专业的数据分析与数据挖掘课程群的构建研究[J]. 电脑知识与技术, 2018, 14(28):95-97.

[20] 廖先富, 刘俊男. 基于Django与HDFS的分布式三维模型文件数据库构建[J]. 电子技术与软件工程, 2018(18):189-191.

[21] 赖伟. 基于Hadoop和Django的电商用户画像系统[D]. 首都经济贸易大学, 2018.

[22] 陈豪, 吴健. 基于Hadoop和Python的多角度电影数据可视化分析[J]. 现代信息科技, 2017, 1(05):11-13.

[23] 吴义. 基于Hadoop和Django的大数据可视化分析Web系统[D]. 东华大学, 2016.

推荐 大数据分析 & 挖掘从业者 | 阅读
AI 技术人员



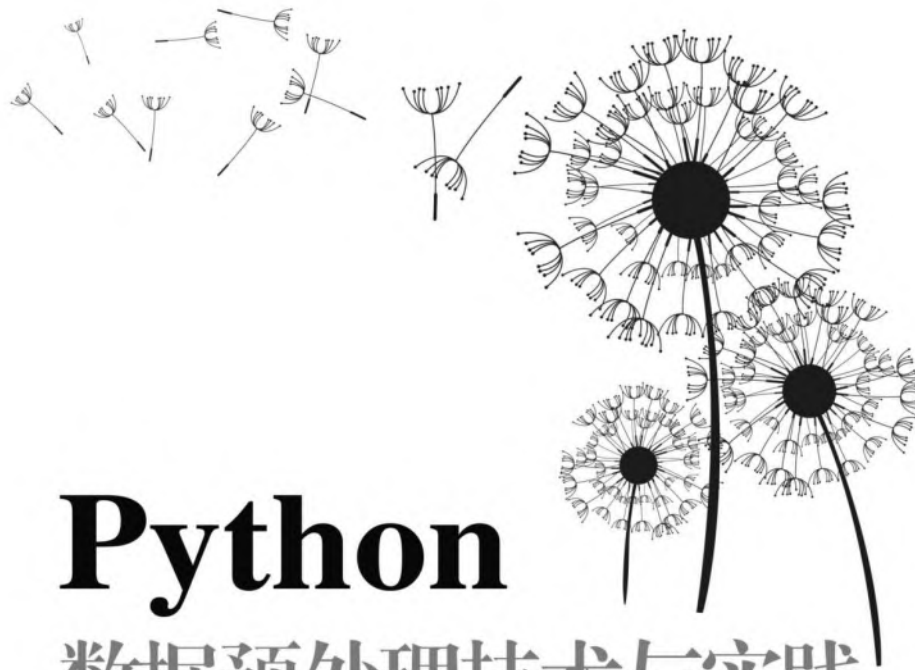
Python

数据预处理技术与实践

涉及Python数据科学工具库、数据采集与存储、数据清洗、中文分词、PCA降维、特征向量化、数据可视化、XGBoost文本分类等内容

白宁超 / 唐 聃 / 文 俊 著

清华大学出版社



Python

数据预处理技术与实践

白宁超 / 唐 聃 / 文 俊 著

清华大学出版社
北 京

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

图书在版编目（CIP）数据

Python数据预处理技术与实践 / 白宁超，唐聃，文俊著．—北京：清华大学出版社，2019.10

ISBN 978-7-302-53971-1

I. ①P… II. ①白…②唐…③文… III. ①软件工具—程序设计 IV. ①TP311.561

中国版本图书馆CIP数据核字（2019）第224198号

责任编辑：王金柱

封面设计：王翔

责任校对：闫秀华

责任印制：沈露

出版发行：清华大学出版社

网 址：<http://www.tup.com.cn>，<http://www.wqbook.com>

地 址：北京清华大学学研大厦A座

邮 编：100084

社总机：010-62770175

邮 购： 010-62786544

投稿与读者服务： 010-62776969, c-service@tup.tsinghua.edu.cn

质 量 反 馈： 010-62772015, zhiliang@tup.tsinghua.edu.cn

印 装 者： 三河市国英印务有限公司

经 销： 全国新华书店

开 本： 170mm×240mm

印 张： 16.75

字 数： 375千字

版 次： 2019年12月第1版

印 次： 2019年12月第1次印刷

产品编号： 084072-01

作者简介

白宁超

大数据工程师，现任职于四川省计算机研究院，研究方向包括数据分析、自然语言处理和深度学习。主持和参与国家自然科学基金项目和四川省科技支撑计划项目多项，出版专著1部。

唐聃

教授，硕士生导师，成都信息工程大学软件工程学院院长，四川省学术和技术带头人后备人选。研究方向包括编码理论与人工智能，《自然语言处理理论与实战》一书作者。

文俊

硕士，大数据算法工程师，现任职于成都广播电视台橙视传媒大数据中心。曾以技术总监身份主持研发多个商业项目，负责公司核心算法模型构建。主要研究方向包括数据挖掘、自然语言处理、深度学习及云计算。

内容简介

本书基础理论和工程应用相结合，循序渐进地介绍了数据预处理的基本概念、基础知识、工具应用和相关案例，包括网络爬虫、数据抽取、数据清洗、数据集成、数据变换、数据向量化、数据规约等知识，书中针对每个知识点，都给出了丰富的教学实例和实现代码，最后，通过一个新闻文本分类的实际项目讲解了数据预处理技术在实际中的应用。

本书的特点是几乎涵盖了数据预处理的各种常用技术及主流工具应用，示例代码很丰富，适合于大数据从业者、AI技术人员以及高校大数据专业的学生使用。

前言

当前，大数据与人工智能技术炙手可热，其对应的工作岗位也逐年增加，薪资也较为诱人。我们在做大数据与人工智能处理时，不可避免地会遇到数据的问题。现实中的数据是不完整的，比如存在缺失值、干扰值等脏数据，这样就没有办法直接挖掘数据的价值，也不能将其直接应用于人工智能设备。为了提高数据的质量产生了数据预处理技术。数据预处理主要是指对原始数据进行文本抽取、数据清理、数据集成、数据变换、数据降维等处理，其目的是提高数据质量，以更好地提升算法模型的性能，其在数据挖掘、自然语言处理、机器学习、深度学习算法中应用广泛。数据预处理是一项很庞杂的工程，当你面对一堆数据手足无措的时候，当你面对数据预处理背后的坑一筹莫展的时候，当你的算法性能迟迟不能提升的时候，本书可以帮助你解决以上问题。本书从什么是数据预处理及其相关基础知识入手，分别介绍了网络爬虫、数据抽取、数据清洗、数据集成、数据变换、数据向量化、数据规约等技术，并结合实际项目和案例帮助读者将所学知识快速应用于工程实践，相信会对数据预处理感兴趣的读者和大数据从业者有所帮助。

本书的主要内容

本书从逻辑上可分为三部分，共12章内容，各部分说明如下：

第一部分（第1~3章），主要介绍数据预处理的基础知识，包括数据预处理的基本概念、工作流程、应用场景、开发环境、入门演练和Python科学计算工具包Numpy、SciPy、Pandas及其实际应用。如果读

者已具备一定的数据预处理基础，可以跳过此部分，从第三章开始学习。

第二部分（第3~10章），是数据预处理的实战进阶部分，共计8章。第3章介绍数据采集与存储，主要涉及数据类型和采集方式，其中着重介绍了爬虫技术；第4章介绍不同格式的文本信息抽取和文件读取；第5章介绍了高效读取文件、正则清洗文本信息、网页数据清洗和文本批量清洗工作；第6章介绍了中文分词、封装分词工具包、NLTK词频处理、命名实体抽取和批量分词处理工作；第7章介绍了特征向量化处理，其中涉及数据解析、缺失值处理、归一化处理、特征词文本向量化、词频-逆词频、词集模型、词袋模型和批量文本特征向量化工作；第8章介绍基于Gensim文本特征向量化，涉及构建语料词典、词频统计、词频-逆词频计算、主题模型和特征降维等。第9章介绍了主成分分析PCA降维技术的原理和实际案例；第10章介绍了Matplotlib数据可视化及案例。

第三部分（包括第11章和第12章），是数据预处理的实际应用部分，主要介绍竞赛神器XGBoost的算法原理、应用、优化调参以及数据预处理在文本分类中的实际应用。

本书的主要特色

本书主要包括以下特色：

- 本书理论与应用相结合，循序渐进地介绍了数据预处理的相关概念、基础知识、常用工具及应用案例，书中实战案例均来自于笔者的实际项目，具有较强的实用性。
- 本书涵盖了数据预处理实际开发中绝大部分重要的知识点，介绍了当今数据预处理涉及的各种技术和热门工具，技术先进，内容详尽，代码可读性及可操作性强。

- 本书针对每一个知识点，尽可能地通过示例来讲解，每一个示例都给出了源码和说明，这些源码本身具备复用的价值，可以直接用于工程项目。另外，笔者还在GitHub上开辟了专门的讨论区，便于读者进行技术交流。

本书面向的读者

本书主要面向以下读者：

- 大数据技术从业者
- AI技术人员
- 准备上手数据采集、数据挖掘与数据分析的初学者□
- 大数据及相关专业的学生
- 培训机构的学员

源码下载和说明

本书的源码支持GitHub下载，下载地址：

<https://github.com/bainingchao/PyDataPreprocessing>

关于源码的说明：

- PyDataPreprocessing：本书源代码的根目标。
- Chapter+数字：分别代表对应章节的源码。
- Corpus：本书所有的训练语料。
- Files：所有文件文档。
- Packages：本书所需要下载的工具包。

本书作者介绍

本书主要由白宁超、唐聃、文俊编写，参与编写的还有田霖、黄河、于小明。

- **白宁超** 大数据工程师，现任职于四川省计算机研究院，研究方向包括数据分析、自然语言处理和深度学习。近3年，主持和参与国家自然科学基金项目和四川省科技支撑计划项目多项，出版专著一部。
- **唐聃** 教授，硕士生导师，成都信息工程大学软件工程学院院长，四川省学术和技术带头人后备人选。研究方向包括编码理论与人工智能，《自然语言处理理论与实战》作者。
- **文俊** 硕士，大数据算法工程师，现任职于成都广播电视台橙视传媒大数据中心。曾以技术总监身份主持研发多个商业项目，负责公司核心算法模型构建。主要研究方向包括数据挖掘、自然语言处理、深度学习以及云计算。
- **田霖** 成都东软学院计算机科学与工程系教师，研究方向包括数据挖掘和强化学习，曾参与四川省智慧环保、四川省涉税信息等多个省级项目。
- **黄河** 博士，重庆大学语言认知及语言应用研究基地研究员，研究方向为计算语言学、语料库处理技术、深度学习和数据挖掘。
- **于小明** 讲师，主治医师，现任职于河南省中医院（河南中医药大学第二附属医院），主要从事医疗领域大数据分析，临床泌尿外科小领域本体构建等方面的研究工作。

在本书编写过程中，参考了很多相关资料，在此对他们的贡献表示感谢，虽笔者已尽心竭力，但限于水平和时间仓促，书中难免存在错漏，欢迎读者批评指正。

联系邮箱: booksaga@163.com

作者
2019年5月

目 录

[前言](#)

[第1章 概述](#)

[1.1 Python数据预处理](#)

[1.1.1 什么是数据预处理](#)

[1.1.2 为什么要做数据预处理](#)

[1.1.3 数据预处理的工作流程](#)

[1.1.4 数据预处理的应用场景](#)

[1.2 开发工具与环境](#)

[1.2.1 Anaconda介绍与安装](#)

[1.2.2 Sublime Text](#)

[1.3 实战案例：第一个中文分词程序](#)

[1.3.1 中文分词](#)

[1.3.2 实例介绍](#)

[1.3.3 结巴实现中文分词](#)

[1.4 本章小结](#)

[第2章 Python科学计算工具](#)

[2.1 NumPy](#)

[2.1.1 NumPy的安装和特点](#)

[2.1.2 NumPy数组](#)

[2.1.3 Numpy的数学函数](#)

[2.1.4 NumPy线性代数运算](#)

[2.1.5 NumPy IO操作](#)

[2.2 SciPy](#)

[2.2.1 SciPy的安装和特点](#)

[2.2.2 SciPy Linalg](#)

[2.2.3 SciPy文件操作](#)

[2.2.4 SciPy插值](#)

[2.2.5 SciPy Ndimage](#)

[2.2.6 SciPy优化算法](#)

[2.3 Pandas](#)

[2.3.1 Pandas的安装和特点](#)

[2.3.2 Pandas的数据结构](#)

[2.3.3 Pandas的数据统计](#)

[2.3.4 Pandas处理丢失值](#)

[2.3.5 Pandas处理稀疏数据](#)

[2.3.6 Pandas的文件操作](#)

[2.3.7 Pandas可视化](#)

[2.4 本章小结](#)

[第3章 数据采集与存储](#)

[3.1 数据与数据采集](#)

[3.2 数据类型与采集方法](#)

[3.2.1 结构化数据](#)

[3.2.2 半结构化数据](#)

[3.2.3 非结构化数据](#)

[3.3 网络爬虫技术](#)

[3.3.1 前置条件](#)

[3.3.2 Scrapy技术原理](#)

[3.3.3 Scrapy新建爬虫项目](#)

[3.3.4 爬取网站内容](#)

[3.4 爬取数据以JSON格式进行存储](#)

[3.5 爬取数据的MySQL存储](#)

[3.5.1 MySQL与Navicat部署](#)

[3.5.2 MySQL存储爬虫数据](#)

[3.6 网络爬虫技术扩展](#)

[3.7 本章小结](#)

[第4章 文本信息抽取](#)

[4.1 文本抽取概述](#)

[4.2 文本抽取问题](#)

[4.3 Pywin32抽取文本信息](#)

[4.3.1 Pywin32介绍](#)

[4.3.2 抽取Word文档文本信息](#)

[4.3.3 抽取PDF文档文本信息](#)

[4.3.4 打造灵活的文本抽取工具](#)

[4.4 文本批量编码](#)

[4.5 实战案例：遍历文件批量抽取新闻文本内容](#)

[4.5.1 递归读取文件](#)

[4.5.2 遍历抽取新闻文本](#)

[4.6 本章小结](#)

[第5章 文本数据清洗](#)

[5.1 新闻语料的准备](#)

[5.2 高效读取文件](#)

[5.2.1 递归遍历读取新闻](#)

[5.2.2 yield生成器](#)

[5.2.3 高效遍历读取新闻](#)

[5.3 通过正则表达式来清洗文本数据](#)

[5.3.1 正则表达式](#)

[5.3.2 清洗文本数据](#)

[5.4 清洗HTML网页数据](#)

[5.5 简繁体转换](#)

[5.6 实战案例：批量新闻文本数据清洗](#)

[5.6.1 高效读取文件内容](#)

[5.6.2 抽样处理文件](#)

[5.6.3 通过正则表达式批量清洗文件](#)

[5.7 本章小结](#)

[第6章 中文分词技术](#)

[6.1 中文分词简介](#)

[6.1.1 中文分词概述](#)

[6.1.2 常见中文分词方法](#)

[6.2 结巴分词精讲](#)

- [6.2.1 结巴分词的特点](#)
- [6.2.2 结巴分词的安裝](#)
- [6.2.3 结巴分词核心方法](#)
- [6.2.4 结巴中文分词的基本操作](#)
- [6.2.5 自定义分词词典](#)
- [6.2.6 关键词提取](#)
- [6.2.7 词性标注](#)

[6.3 HanLP分词精讲](#)

- [6.3.1 JPython的安装](#)
- [6.3.2 调用HanLP的Java包](#)
- [6.3.3 HanLP分词](#)
- [6.3.4 HanLP实现自定义分词](#)
- [6.3.5 命名实体识别与词性标注](#)
- [6.3.6 HanLP实现关键词抽取](#)
- [6.3.7 HanLP实现自动摘要](#)

[6.4 自定义去除停用词](#)

- [6.4.1 以正则表达式对文本信息进行清洗](#)
- [6.4.2 结巴中文分词词性解读](#)
- [6.4.3 根据词性规则构建自定义停用词](#)

[6.5 词频统计](#)

- [6.5.1 NLTK介绍与安装](#)
- [6.5.2 统计新闻文本词频](#)
- [6.5.3 统计特定词频和次数](#)
- [6.5.4 特征词的频率分布表](#)
- [6.5.5 频率分布图与频率累计分布图](#)
- [6.5.6 基于Counter的词频统计](#)

[6.6 自定义去高低词频](#)

[6.7 自定义规则提取特征词](#)

[6.8 实战案例：新闻文本分词处理](#)

[6.9 本章小结](#)

[第7章 文本特征向量化](#)

7.1 解析数据文件

7.2 处理缺失值

7.2.1 什么是数据缺失值

7.2.2 均值法处理数据缺失值

7.2.3 Pandas处理缺失值

7.3 数据的归一化处理

7.3.1 不均衡数据分析

7.3.2 归一化的原理

7.3.3 归一化的优点

7.4 特征词转文本向量

7.5 词频-逆词频 (TF-IDF)

7.6 词集模型与词袋模型

7.7 实战案例：新闻文本特征向量化

7.8 本章小结

第8章 Gensim文本向量化

8.1 Gensim的特性和核心概念

8.2 Gensim构建语料词典

8.3 Gensim统计词频特征

8.4 Gensim计算TF-IDF

8.5 Gensim实现主题模型

8.5.1 主题模型

8.5.2 潜在语义分析 (LSA)

8.5.3 隐含狄利克雷分布 (LDA)

8.5.4 LDA的模型实现

8.5.5 随机映射 (RP)

8.6 实战案例：Gensim实现新闻文本特征向量化

8.6.1 参数设置

8.6.2 生成词典模型

8.6.3 生成TF-IDF模型

8.7 本章小结

第9章 PCA降维技术

[9.1 什么是降维](#)

[9.2 PCA概述](#)

[9.3 PCA应用场景](#)

[9.4 PCA的算法实现](#)

[9.4.1 准备数据](#)

[9.4.2 PCA数据降维](#)

[9.4.3 高维向低维数据映射](#)

[9.5 实战案例：PCA技术实现新闻文本特征降维](#)

[9.5.1 加载新闻数据](#)

[9.5.2 前N个主成分特征](#)

[9.5.3 PCA新闻特征降维可视化](#)

[9.6 本章小结](#)

[第10章 数据可视化](#)

[10.1 Matplotlib概述](#)

[10.1.1 认识Matplotlib](#)

[10.1.2 Matplotlib的架构](#)

[10.2 Matplotlib绘制折线图](#)

[10.2.1 折线图的应用场景](#)

[10.2.2 折线图的绘制示例](#)

[10.3 Matplotlib绘制散点图](#)

[10.3.1 散点图的应用场景](#)

[10.3.2 散点图的绘制示例](#)

[10.4 Matplotlib绘制直方图](#)

[10.4.1 直方图的应用场景](#)

[10.4.2 直方图的绘制示例](#)

[10.5 练习：Matplotlib绘制气温图](#)

[10.6 练习：Matplotlib绘制三维图](#)

[10.6.1 练习1：绘制三维梯度下降图](#)

[10.6.2 练习2：绘制三维散点图](#)

[10.7 本章小结](#)

[第11章 竞赛神器XGBoost](#)

11.1 XGBoost概述

11.1.1 认识XGBoost

11.1.2 XGBoost的应用场景

11.2 XGBoost的优点

11.3 使用XGBoost预测毒蘑菇

11.3.1 XGBoost的开发环境及安装

11.3.2 数据准备

11.3.3 参数设置

11.3.4 模型训练

11.3.5 可视化特征排名

11.4 XGBoost优化调参

11.4.1 参数解读

11.4.2 调参原则

11.4.3 调参技巧

11.5 预测糖尿病患者

11.5.1 数据准备

11.5.2 预测器模型构建

11.5.3 调参提高预测器的性能

11.6 本章小结

第12章 XGBoost实现新闻文本分类

12.1 文本分类概述

12.2 文本分类的原理

12.2.1 文本分类的数学描述

12.2.2 文本分类的形式化描述

12.3 分类模型评估

12.4 数据预处理

12.4.1 通用的类库

12.4.2 阶段1：生成词典

12.4.3 阶段2：词典向量化TF-IDF

12.4.4 阶段3：生成主题模型

12.5 XGBoost分类器

[12.6 新闻文本分类应用](#)

[12.7 本章小结](#)

[参考文献](#)

[返回总目录](#)

第1 章 概 述

日常生活中每天都在产生数据，这些原始数据存在数据不完整、数据不一致、数据异常等情况，严重影响数据的质量，甚至可能导致利用上的偏差，因此数据预处理技术应运而生。本章首先概述数据预处理，让读者对其有一个整体的认识；然后介绍Python数据预处理的开发工具与运行环境；最后结合中文分词的实战案例，让读者了解数据预处理的工作流程。

1.1 Python数据预处理

1.1.1 什么是数据预处理

大数据与人工智能时代离不开海量的原始数据做支撑，这些原始数据存在数据不完整、数据不一致、数据异常等问题，很难得到高质量的数据用于数据建模，甚至可能导致工程应用的偏差。因此，要对原始数据做一定的处理。这种从原始数据到挖掘数据，对数据进行的操作叫作数据预处理。数据预处理通常包括数据清理、数据集成、数据归约、数据变换和数据降维，目的是挖掘数据背后的应用价值和社会价值。

数据预处理通俗地理解就是将原始数据转化为机器可以认知的数据形式，以适应相关技术或者算法模型。比如新闻分类案例中，原始数据是一篇篇的新闻文本，分类器并不能直接处理，需要对新闻文本

分词、去除干扰词、提取词特征、词特征转化、词特征降维等操作，分类器才能对数据进行学习优化，实现工程应用。

总而言之，原始数据可能存在数据不完整、数据偏态、数据噪声、数据特征维度高、数据缺失值、数据错误值等一系列问题，经过数据预处理后的数据能够达到数据完整干净、数据特征比重合适、数据特征维度合理、数据无缺失值等优点，使数据利用更加准确、高质。

1.1.2 为什么要做数据预处理

早期互联网时代数据量较少，主要存储在数据库、文件系统等介质中，其数据分析主要靠人工统计完成。随着网络的普及，海量数据应运而生，依旧采用人工统计方法对数据处理已不合时宜。伴随着计算能力和硬件设施的提升，先前的算法理论（如神经网络等）有了用武之地，使得计算机处理海量数据成为当今数据分析人员的主要工作。

在大数据与人工智能的时代，甚至未来的一段时间，不管是无人驾驶还是智能机器人，或是其他应用，主要还是在监督式学习下进行的，这里的监督学习即需要有参考意义的历史数据做基础。这些数据不仅仅是数据库文件、文本文件，还包括视频、语音、网页等各种介质的数据。数据的存在形式呈现多样化，我们将其称之为异源数据，顾名思义指的是来自不同数据源的数据。

异源数据也是最原始的数据，包括人们在网上任何行为记录。这些行为绝大多数是正确的，但是也可能存在错误。比如，有时候收集数据的设备可能出故障；或者是人为输入错误；数据传输中的错误；命名约定或所用的数据代码不一致导致的错误，等等。如何对这

些原始数据进行预处理来提高数据质量？如何通过高质量的数据来挖掘数据背后的价值？这就是为什么要做数据预处理的直观原因之一。

数据价值挖掘的研究工作大多都集中在算法的探讨，而忽视对数据本身的研究。事实上，数据预处理对挖掘数据价值十分重要，一些成熟的算法对其处理的数据集合都有一定的要求：比如数据的完整性好、冗余性小、属性的相关性小等。数据预处理是数据建模的重要一环，且必不可少，要挖掘出有效的知识，必须为其提供干净、准确、简洁的数据。实际应用系统中收集的数据通常是“脏”数据。没有高质量的数据，就没有高质量的挖掘结果。

1.1.3 数据预处理的工作流程

构建新闻分类器时，如何正确有效地将不同数据源中的信息整合到一起，直接影响到分类器的最终结果，数据预处理正是解决这一问题的有力方案。数据预处理包含以下几个方面：

- 数据采集。指的是从网页、文件库、数据库等多渠道采集数据，这些数据主要以结构化、半结构化和非结构化的形式存在。
- 数据集成。指的是将从多个数据源中获取到的数据结合起来并统一存储。
- 文本提取。指的是将不同格式存储的文本信息统一处理，转化为文本格式。
- 数据清理。指的是通过填写缺失的值、光滑噪声数据、识别或删除离群点并解决不一致性来清理数据。
- 数据转换。指的是按照预先设计好的规则对抽取的数据进行转换，如把数据压缩到0.0~1.0区间。

- 数据归约。数据归约技术可以用来得到数据集的归约表示，它虽然小得多，但仍然接近于保持原数据的完整性。

1.1.4 数据预处理的应用场景

大数据和人工智能技术能够将隐藏于海量数据中的信息和知识挖掘出来，为人类的社会经济活动提供依据，从而提高各个领域的运行效率。其应用领域较为广泛，包括以下领域：

- 商业智能技术。
- 政府决策技术。
- 电信数据信息处理与挖掘技术。
- 电网数据信息处理与挖掘技术。
- 气象信息分析技术。
- 环境监测技术。
- 警务云应用系统（视频 / 网络监控、智能交通、反电信诈骗、指挥调度等公安信息系统）。
- 大规模基因序列分析比对技术。
- Web信息挖掘技术。
- 多媒体数据并行化处理技术。
- 影视制作渲染技术。
- 其他各种行业的云计算和海量数据处理应用技术等。

1.2 开发工具与环境

1.2.1 Anaconda介绍与安装

Anaconda是一种Python语言的免费增值开源发行版，用于进行大规模数据处理、预测分析和科学计算，致力于简化包的管理和部署。

Anaconda包含了Conda、Python在内的超过180个科学包及其依赖项。Anaconda使用软件包管理系统Conda进行包管理。

1. Anaconda的优点

Anaconda是一个用于科学计算的Python发行版，支持Linux、Mac、Windows系统，提供了包管理与环境管理的功能，可以很方便地解决多版本Python并存、切换以及各种第三方包的安装问题。Anaconda利用工具 / 命令Conda来进行Package和Environment的管理，并且已经包含了Python和相关的配套工具。这里先解释一下Conda、Anaconda这些概念的差别。Conda可以理解为一个工具，也是一个可执行命令，其核心功能是包管理与环境管理。包管理与pip的使用类似，环境管理则允许用户方便地安装不同版本的Python并可以快速切换。Anaconda则是一个打包的集合，里面预装好了Conda、某个版本的Python、众多包（Package）、科学计算工具等，所以也称为Python的一种发行版。其实还有Miniconda，顾名思义，它只包含最基本的内容——Python与Conda，以及相关的必须依赖项，对于存储空间要求严格的用户Miniconda是一个不错的选择。其有以下优点：

- 开源。
- 安装过程简单。
- 高性能使用Python和R语言。
- 免费的社区支持。
- Conda包管理。
- 1,000+开源库

2. 安装Anaconda

Anaconda 安装包下载地址为：
<https://www.anaconda.com/download/>，进入下载页面会显示出

Python 3.0以上版本和Python 2.0两种版本，如图1-1所示。关于Python 3 和 Python 2 的区别请查看网址：<http://www.runoob.com/python/python-2x-3x.html>）。推荐下载Python 3.0以上版本，请读者根据自己的操作系统是32位还是64位选择对应的版本下载。本书是基于Windows 10操作系统安装的。

（1）安装Anaconda集成环境，双击下载后的Anaconda安装文件，如图1-2所示。

（2）然后一直单击“Next”按钮，直到完成配置（环境变量自动配置），配置完成后，查看是否安装成功。打开主菜单->所有应用查看安装，出现如图1-3所示的菜单命令则表示安装成功。

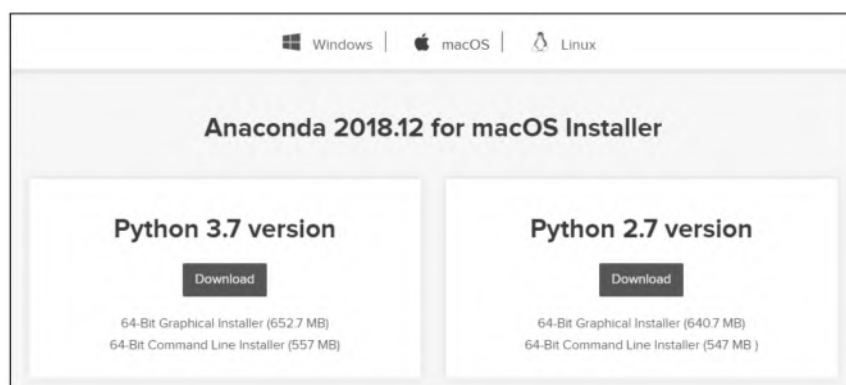


图1-1 不同Anaconda版本



图1-2 Anaconda安装

(3) 按WIN+R组合键输入“cmd”命令以启动命令行环境，然后输入“conda -V”查看版本号，如图1-4所示。



图1-3 安装后的Anaconda



图1-4 查看Anaconda版本

(4) 可以看到本机Anaconda版本是4.5.12，上文介绍了其集成了诸多科学包及其依赖项，看看具体包括哪些？在“命令提示符”环境下，输入“conda list”，运行结果如图1-5所示（截取部分包）。

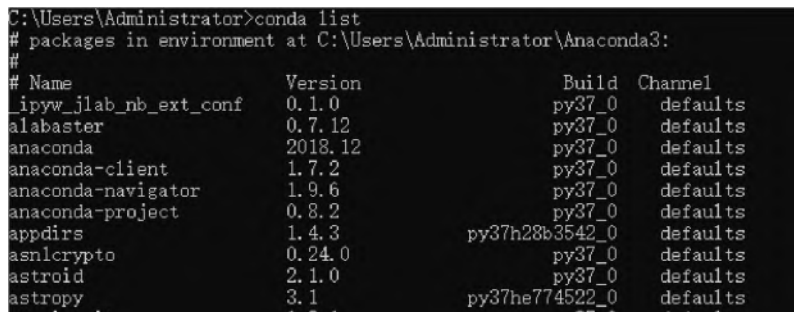


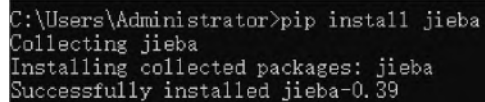
图1-5 查看Anaconda预安装包

(5) Anaconda里面有一个包名为“pip”，这个包和Linux的包安装命令是一样的，当需要安装第三方包的时候，直接使用这个命令即

可，比如，需要进行中文分词，会依赖Python的分词包jieba，可以执行如下命令：

```
pip install jieba
```

如图1-6所示。



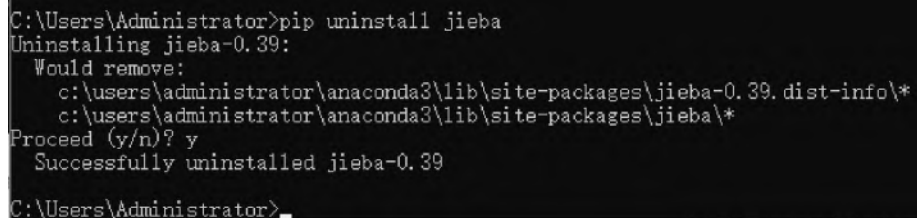
```
C:\Users\Administrator>pip install jieba
Collecting jieba
Installing collected packages: jieba
Successfully installed jieba-0.39
```

图1-6 pip安装结巴分词包

(6) 如果不想使用这个包了，也可以直接卸载掉，命令如下：

```
pip uninstall jieba
```

如图1-7所示。



```
C:\Users\Administrator>pip uninstall jieba
Uninstalling jieba-0.39:
Would remove:
  c:\users\administrator\anaconda3\lib\site-packages\jieba-0.39.dist-info\*
  c:\users\administrator\anaconda3\lib\site-packages\jieba\*
Proceed (y/n)? y
Successfully uninstalled jieba-0.39
C:\Users\Administrator>
```

图1-7 pip卸载结巴分词包

至此，完成了Anaconda的安装配置以及包文件的自定义下载。需要注意的是，Anaconda自身集成了Python、pip、nltk、NumPy、Matplotlib等一系列常用包。现在，已经可以使用Python了，考虑到熟悉Python开发的人员，常用Pycharm开发工具，熟悉Java的开发人员常用Eclipse开发工具，熟悉C#的开发人员常用VS开发工具，因此只要将Anaconda集成到PyDev、Pycharm、Eclipse、VS等编译环境中即可。总之，Anaconda是一款极为简便的集成软件包，可以将其与Sublime、

PyCharm、MyEclipse、Visual Studio等编译环境很巧妙地融合起来（本书采用Anaconda+Sublime Text），也可以方便地导入第三方工具包，从而极大地简化软件开发工作的流程。

1.2.2 Sublime Text

1. Sublime Text简介

Sublime Text是一款跨平台的文本编辑器，同时支持Windows、Linux、Mac OS X等操作系统和基于Python的插件。Sublime Text是专有软件，可通过包（Package）扩充本身的功能。大多数的包使用自由软件授权发布，并由社区构建和维护。Sublime Text是由程序员Jon Skinner于2008年1月份开发出来的，它最初被设计为一个具有丰富扩展功能的Vim。它具有漂亮的用户界面和强大的功能，例如代码缩略图、Python的插件、代码段等。还可自定义键绑定、菜单和工具栏。Sublime Text的主要功能包括：拼写检查、书签、完整的Python API、Goto功能、即时项目切换、多选择、多窗口，等等。

Sublime Text支持众多编程语言，并支持语法上色。内置支持的编程语言包含：ActionScript、AppleScript、ASP、batch files、C、C++、C#、Clojure、CSS、D、Diff、Erlang、Go、Graphviz (DOT)、Groovy、Haskell、HTML、Java、JSP、JavaScript、JSON、LaTeX、Lisp、Lua、Makefiles、Markdown、MATLAB、Objective-C、OCaml、Perl、PHP、Python、R、Rails、Regular Expressions、reStructuredText、Ruby、Scala、shell scripts (Bash)、SQL、Tcl、Textile、XML、XSL和YAML。用户可通过下载外挂支持更多的编程语言。

2. Sublime Text的优点

Sublime Text主要有如下优点：

- 主流前端开发编辑器。
- 体积较小，运行速度快。
- 文本功能强大。
- 支持编译功能且可在控制台看到输出。
- 内嵌Python解释器支持插件开发以达到可扩展目的。
- Package Control（包控制）：Sublime Text支持的大量插件可通过其进行管理。

3. Sublime Text的安装

本文介绍Sublime Windows 10系统下的安装配置，关于Linux和Mac OS下的安装基本一致，读者可自行尝试。Sublime Text 3安装包下载地址是：<http://www.sublimetext.com/3>。单击该网址进入Sublime Text主页，选择对应的操作系统与版本，如图1-8所示。

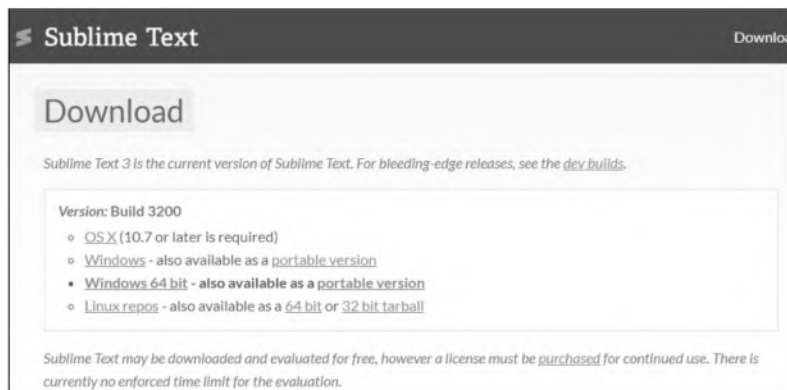


图1-8 下载Sublime Text 3

（1）双击下载好的Sublime Text 3工具包，出现如图1-9所示的界面。

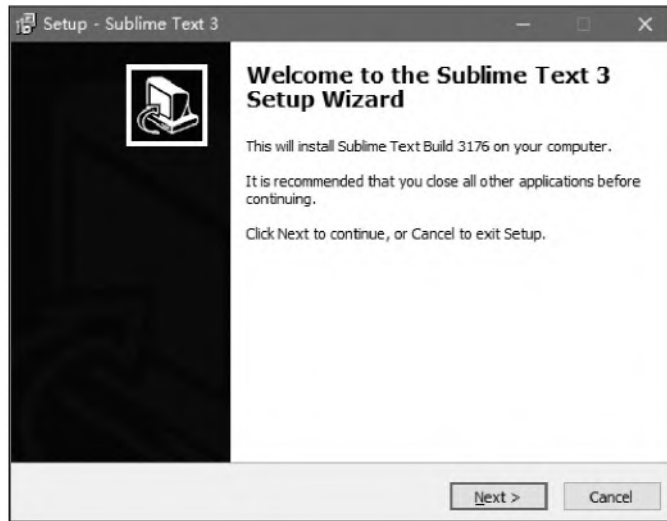


图1-9 安装Sublime

(2) 一直单击Next按钮安装即可，中间保存路径可以自定义。安装成功后的结果如图1-10所示。

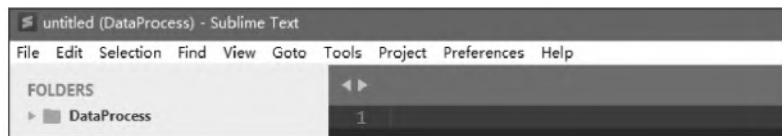


图1-10 安装完成Sublime

(3) 安装插件Package Control。

① 自动安装 Package Control 。 打开 <https://packagecontrol.io/installation>，复制Sublime Text 3中的代码，如图1-11所示。

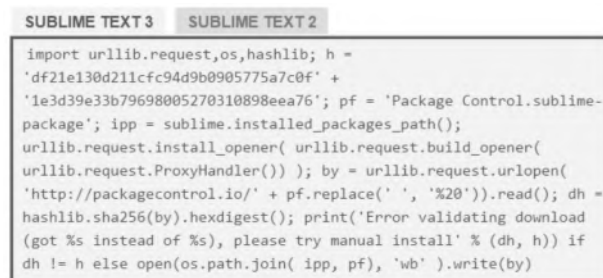


图1-11 安装插件Package Control

按“Ctrl+`”组合键，将上述文本代码内容复制粘贴到文本框中，按Enter即可。如图1-12所示。

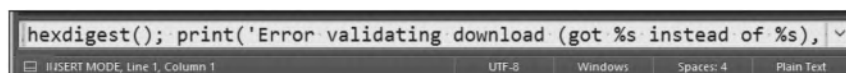


图1-12 执行Package Control代码

②如果Package Control官网无法打开，也可以手动安装Package Control。

在百度网盘下载Package Control的安装包（链接：https://pan.baidu.com/s/14hs2-0F5L_18UHKUkPGayQ）提取码：m7a9。下载完成后里面包含两个文件分别是：

文件1： Package Control.sublime-package

文件2： channel_v3.json

然后，打开Sublime存放插件的目录：在Sublime Text → Preference → Browse Packages...找到“Installed Packages”文件夹，并将以上两个文件复制进去，然后重启，如图1-13所示。



图1-13 存放Sublime插件

最后，在Sublime下打开Preference → Package Settings → Package Control → Setting-User，添加如下代码，如图1-14所示。

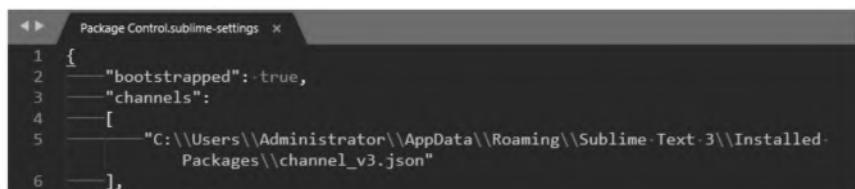


图1-14 配置Sublime插件

(4) 成功安装后，在Sublime Text 3中同时按住Ctrl+Shift+P组合键。最终安装成功，如图1-15所示。

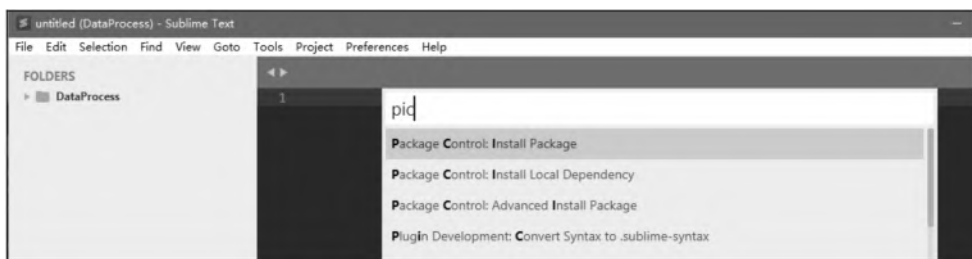


图1-15 安装所需插件

(5) 单击“Package Control:Install Package”进入查找Python环境配置插件“SublimeREPL”，下载安装完成后，单击“Preferences->Browse Package...”查看安装的包，如图1-16所示。

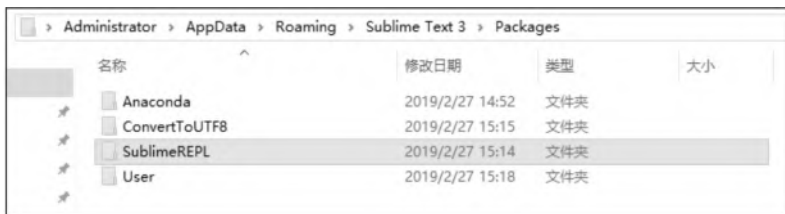


图1-16 安装运行环境

(6) 单击“Package Control:Install Package”查找Python环境配置插件Anaconda，如图1-17所示。

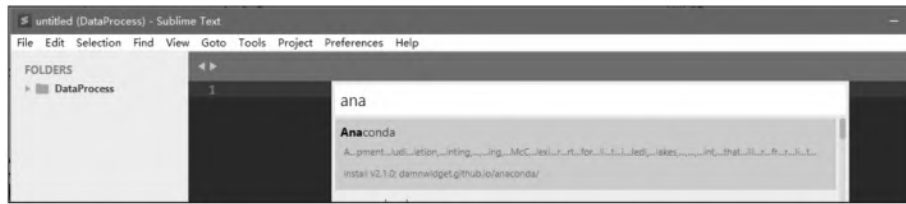


图1-17 集成Anaconda

(7) 自定义快捷键配置。打开Preferences → Key Bindings输入如下代码，其中，F5运行程序，F6切换IDEL工具，Ctrl+D自定义删除行，其他快捷键是通用的，网上有很多快捷键的资料，这里不再赘述。

```

1  [
2      {
3          "keys": ["F5"],
4          "caption": "SublimeREPL: Python - RUN current file",
5          "command": "run_existing_window_command",
6          "args": {
7              "id": "repl_python_run",
8              "file": "config/Python/Main.sublime-menu"
9          }
10 }, {
11     "keys": ["F6"],
12     "caption": "SublimeREPL: Python",
13     "command": "run_existing_window_command",
14     "args": {
15         "id": "repl_python",
16         "file": "config/Python/Main.sublime-menu"
17     }
18 }, {
19     "keys": ["Ctrl+D"],
20     "command": "run_macro_file",
21     "args": {"file": "res://Packages/Default/
                Delete Line.sublime-macro"}
22 }
23 ]

```


(8) 激活版本: Help >Enter LICENSE。

```
1  ----- BEGIN LICENSE -----
2  sgbteam
3  Single User License
4  EA7E-1153259
5  8891CBB9 F1513E4F 1A3405C1 A865D53F
6  115F202E 7B91AB2D 0D2A40ED 352B269B
7  76E84F0B CD69BFC7 59F2DFEF E267328F
8  215652A3 E88F9D8F 4C38E3BA 5B2DAAE4
9  969624E7 DC9CD4D5 717FB40C 1B9738CF
10 20B3C4F1 E917B5B3 87C38D9C ACCE7DD8
11 5F7EF854 86B9743C FADC04AA FB0DA5C0
12 F913BE58 42FEA319 F954EFDD AE881E0B
13 ----- END LICENSE -----
```

至此，完成了Sublime Text 3的安装配置工作，有关详细插件安装的说明，请参考网址：<http://www.open-open.com/news/view/26d731>，有关快捷键的使用，请查看网址：<https://segmentfault.com/a/1190000004463984>。

注 意

读者也可以使用PyCharm、Eclipse等常用的Python开发工具。

1.3 实战案例：第一个中文分词程序

在数据处理工作中，分词是一项必不可少的工作，本节使用Sublime Text完成第一个分词案例。下面介绍什么是中文分词及实现方法。

1.3.1 中文分词

中文分词是指将一个汉字序列切分成一个一个单独的词。分词就是将连续的字序列按照一定的规范重新组合成词序列的过程，在英文行文中，单词间以空格作为自然分界符，中文没有一个形式上的分界符，虽然英文也同样存在短语的划分问题，不过在词这一层上，中文比英文要复杂得多，也困难得多。

例如：

英文句子：I am a student.

中文意思：我是一名学生。

由于英文的语言使用习惯，通过空格很容易拆分出单词，而中文字词界限模糊，往往不容易区别哪些是“字”、哪些是“词”，这也是为什么要把中文词语进行切分的原因。

1. 中文分词的发展

与英文为代表的印欧语系语言相比，中文由于继承自古代汉语的传统，词语之间常没有分隔。古代汉语中除了联绵词和人名、地名等，词通常是单个汉字，所以当时没有分词书写的必要。而现代汉语中双字或多字词逐渐增多，一个字已经不再等同于一个词了。

在中文里，“词”和“词组”边界模糊，现代汉语的基本表达单元虽然为“词”，且以双字或者多字词居多，但由于人们认识水平的不同，对词和短语的边界还很难去区分。

例如：“对随地吐痰者给予处罚”，“随地吐痰者”本身是一个词还是一个短语，不同的人会有不同的标准，同样的，“海上”“酒厂”等即使是同一个人也可能做出不同的判断。如果汉语真的要分词书写，必然会出现混乱，难度也很大。中文分词的方法其实不局限于中文应用，也被应用于英文处理，例如帮助判别英文单词的边界等。

2. 中文分词的用途

中文分词是文本处理的基础，对于输入的一段中文，成功地进行中文分词，可以达到电脑自动识别语句含义的效果。中文分词技术属于自然语言处理技术的范畴，目前在自然语言处理技术中，中文处理技术比西文处理技术要落后很大一截，而许多西文的处理方法中文却不能直接采用，就是因为中文必须有分词这道工序。中文分词是中文信息处理的基础，搜索引擎就是中文分词的一个应用，其他的比如机器翻译（Machine Translation）、语音合成、自动分类、自动摘要、自动校对等，都需要用到分词。因为中文需要分词，可能会影响一些研究，但同时也为一些企业带来机会，因为国外的计算机处理技术要想进入中国市场，首先也是要解决中文分词问题。

中文分词对于搜索引擎来说，最重要的并不是找到所有的结果，因为在上百亿的网页中找到所有结果没有太多的意义，也没有人能看得完；相反，最重要的是把最相关的结果排在最前面，这也称为相关度排序。中文分词的准确与否，常常直接影响到对搜索结果的相关度排序。从定性分析的角度来看，搜索引擎的分词算法不同、词库的不同都会影响页面的返回结果。

1.3.2 实例介绍

本节实现一个有趣的应用。将电影《流浪地球》中的经典句子“道路千万条，安全第一条；行车不规范，亲人两行泪。”进行中文分词。这里需要使用第三方工具包结巴（jieba）来实现。

1.3.3 结巴实现中文分词

打开 Sublime Text 并在根目录 PyDataPreprocessing 下创建 Chapter1 文件夹，然后在 Chapter1 下面创建 CutWords.py 文件并打开。

在编辑代码之前，先查看一下 jieba 包能否正常导入。按住 Alt+Shift+2 组合键进行分屏，然后按 F6 键进入 Python IDE 环境下，成功导入后如图 1-18 所示。

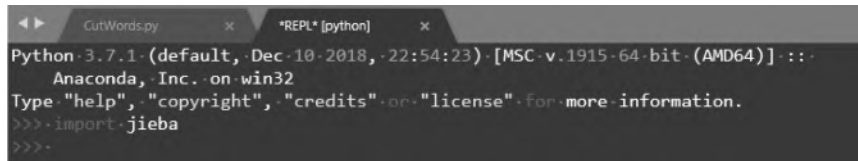


图 1-18 Sublime Text 下运行 Python IDE

图 1-18 说明 jieba 包已经成功导入，编写如下代码（源代码见：Chapter1/CutWords.py）：

```
1  # coding:utf8
2
3  """
4  DESC: Python 数据预处理之第一个分词程序范例
5  Author: 伏草惟存
6  Prompt: code in Python3 env
7  """
8
9  import jieba
10
11  str = "道路千万条,安全第一条;行车不规范,亲人两行泪。"
12  print("原句: \n" + str)
13
14  seg_list = jieba.cut(str)
15  print("分词: \n" + "/".join(seg_list))
```

代码说明：

其中第 1 行是对中文编码进行设置；第 3-7 行是注释信息；第 9 行导入 jieba 分词模块；第 14 行的调用 jieba 模块中的 cut 方法对字符串分词，数据以列表（List）形式返回；第 15 行是格式化分词结果，将

List数据转化为String数据打印出来。运行代码查看结果，如图1-19所示。



```
1 #coding:utf8
2
3 """
4 DESC: Python数据预处理之第一个分词程序范例
5 Author: 伏草惟存
6 Prompt: .code-in-Python3-env
7 """
8
9 import jieba
10
11 str = "道路千万条,安全第一条;行车不规范,亲人两行泪."
12 print("原句: {}".format(str))
13
14 seg_list = jieba.cut(str)
15 print("分词: {}".format("/".join(seg_list)))
```

```
原句:
道路千万条,安全第一条;行车不规范,亲人两行泪.
Building prefix dict from the default dictionary
Dumping model to file cache C:\Users\ADMINI~1\AppData\Local\Temp\jieba.cache
Loading model cost 0.978 seconds.
Prefix dict has been built successfully.
分词:
道路/千万条/,安全/第一条;/行车/不规范/,亲人/两行泪.
***Repl Closed***
```

图1-19 中文分词的结果

1.4 本章小结

本章主要介绍了什么是数据预处理，为什么要做数据预处理，之后介绍了数据预处理的工作流程和应用场景，使读者对数据预处理有一个整体认识。工欲善其事必先利其器，选择合适的开发工具有助于更高效的工作，Anaconda+Sublime Text就是一组不错的搭档。当然，读者也可以使用PyCharm、MyEclipse、Visual Studio等开发工具。下一章将介绍Python数据预处理的几个科学库。

第2 章

Python科学计算工具

Python语言及其应用可谓如火如荼，读者应该对它们并不陌生。本章主要针对科学计算工具包NumPy、SciPy、Pandas分别从包的简介、安装、特点和常见方法几个方面进行介绍。科学技术工具不仅在数据预处理中使用，在机器学习和深度学习等领域都有着广泛的应用。

2.1 NumPy

NumPy是Python语言常用的科学计算工具包，本节主要介绍NumPy的安装和特点以及NumPy的数组、NumPy的数学函数、NumPy的线性代数和NumPy IO操作等内置模块（源代码见：Chapter2/NumpyDome.py）。

2.1.1 NumPy的安装和特点

NumPy（官网：<http://www.numpy.org/>，如图2-1所示），是Python语言的一个扩展程序库它支持大量的维度数组与矩阵运算，此外也提供大量的数学函数库。NumPy的前身Numeric最早是由Jim Hugunin与其他协作者共同开发，2005年，Travis Oliphant在Numeric中结合了另一个同性质的程序库Numarray的特色，并加入了其他扩展而开发了NumPy。

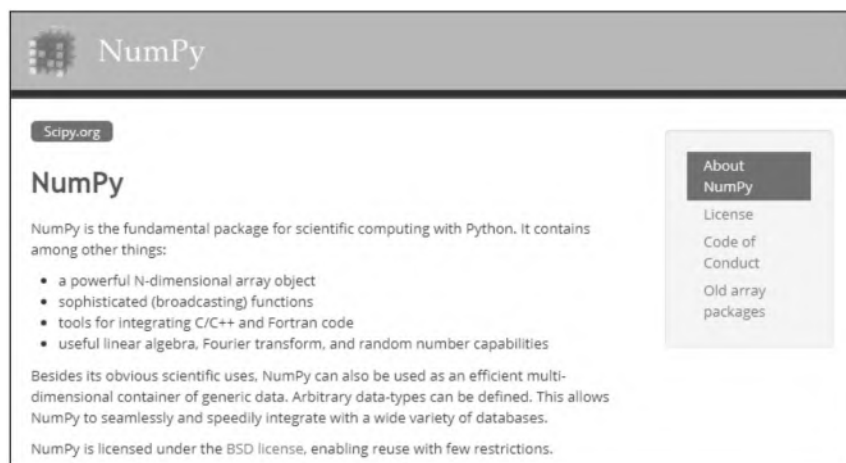


图2-1 NumPy官网

1. NumPy的安装

方法一 Anaconda已经集成了NumPy，读者在DOS环境下查看：
`conda list`

方法二 读者可以通过pip自动安装，执行如下命令：`pip install numpy`

方法三 读者在GitHub上下载NumPy源码（<https://github.com/numpy/numpy>），然后在源码根目录下执行如下命令：`python setup.py install`

安装完成后，检测是否成功。首先启动Sublime，然后按F6键进入Python环境，最后输入`import numpy`，得到以下状态即表示安装成功。如图2-2所示。

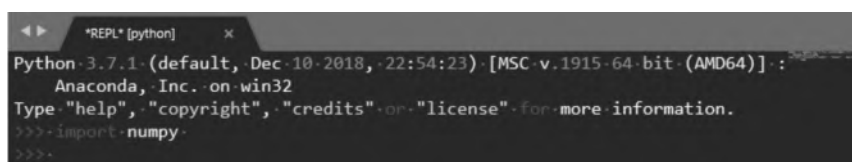


图2-2 验证NumPy是否安装成功

2. NumPy的特点

NumPy参考了CPython（一个使用字节码的解释器），但在CPython解释器上所写的数学算法代码通常远比编译过的相同代码要慢。为了解决这个难题，NumPy引入了多维数组以及可以直接有效率地操作多维数组的函数与运算符。在NumPy上只要能被表示为针对数组或矩阵运算的算法，其运行效率几乎都可以与编译过的等效C语言代码一样快。

NumPy通常与SciPy（Scientific Python）和Matplotlib（绘图库）一起使用，这种组合广泛用于替代MatLab，是一个强大的科学计算环境，有助于我们通过Python学习数据科学或者机器学习。

2.1.2 NumPy数组

数组操作是NumPy的主要模块之一，这里主要介绍一维数组、多维数组、数组基本运算、常数数组、数值范围创建数组、切片和索引以及数组的其他相关操作的实现。

1. 创建一维数组

```
1 import numpy as np
2 # 创建一维数组
3 arr1 = np.array([1, 2, 3])
4 print(arr1)
```

运行结果：

```
[1 2 3]
```

2. 创建多维数组

```
1 import numpy as np
2 arr2 = np.array([[1, 2], [3, 4], [5, 6]])
3 print(arr2)
```


运行结果：

```
[[1 2]  [3 4]  [5 6]]
```

3. 数组基本运算

```
1 import numpy as np
2
3 arr2 = np.array([[1, 2], [3, 4], [5, 6]])
4 print("数组的维数:", arr2.ndim)
5 print("数组元素总个数", arr2.size)
6 print("元素类型", arr2.dtype)
```

运行结果：

```
数组的维数: 2
数组元素总个数: 6
元素类型: int32
```

4. 创建常数数组

```
1 import numpy as np
2
3 arr3 = np.zeros((2, 3), dtype = float, order = 'C')
4 print("创建 0 数组:", arr3)
5
6 arr4 = np.ones([2, 3], dtype = None, order = 'C')
7 print("创建 1 数组:", arr4)
```

代码说明：

- dtype: 表示数据类型，省略时默认为浮点数据类型。
- order: 有C和F两个选项，分别代表行优先和列优先。

运行结果：

创建0数组: `[[0. 0. 0.] [0. 0. 0.]]`
创建1数组: `[[1. 1. 1.] [1. 1. 1.]]`

5. 使用数值范围创建数组

创建从1开始到10终止，步长为2的浮点型数组。

```
1 import numpy as np
2
3 arr5 = np.arange(1,10,2,dtype=float)
4 print(arr5)
```

运行结果:

```
[1. 3. 5. 7. 9.]
```

6. 切片和索引

```
1 import numpy as np
2
3 arr6 = np.arange(20)
4 s = slice(1,20,3)  # 从索引 1 开始到索引 20 停止，步长为 3
5 print (arr6[s])
6
7 b = arr6[2:14:2]   # 从索引 2 开始到索引 14 停止，步长为 2
8 print(b)
```

运行结果:

```
[ 1  4  7 10 13 16 19] [ 2  4  6  8 10 12]
```

7. 数组操作的其他方法

NumPy数组的其他操作方法包括数组排序、数组逆序、最大最小值差、矩阵百分比、统计中位数、算术平均值、加权平均值、标准差、

方差等，其操作也比较简单，这里不再逐一示例，请参考表2-1自行实现。

表2-1 NumPy数组的其他方法

方法描述	函 数	方法描述	函 数
数组排序	numpy.sort()	统计中位数	numpy.median()
数组逆序	numpy.argsort()	算术平均值	numpy.mean()
指定轴的最小值	numpy.amin()	加权平均值	numpy.average()
指定轴的最大值	numpy.amax()	标准差	numpy.std()
最大最小值差	numpy.ptp()	方差	numpy.var()
矩阵百分比	numpy.percentile()		

2.1.3 Numpy的数学函数

1. 三角函数

```
1 import numpy as np
2
3 # 不同角度的正弦值
4 arr7 = np.array([0,45,90])
5 # 通过乘 pi/180 转化为弧度
6 print (np.sin(arr7*np.pi/180))
```

运行结果：

[0. 0.70710678 1.]

2. 三角函数操作的其他方法

三角函数的操作方法包括正弦、余弦、正切、反正弦、反余弦、反正切等，实现方法与上述代码类同，请参考表2-2自行使用。

表2-2 三角函数的操作方法

方法描述	函 数	方法描述	函 数
正弦	numpy.sin()	反正弦	numpy.arcsin()
余弦	numpy.cos()	反余弦	numpy.arccos()
正切	numpy.tan()	反正切	numpy.arctan()

3. 矩阵运算

```

1  import numpy as np
2
3  arr8 = np.arange(9, dtype = np.float_).reshape(3,3)
4  print ('第一个数组: \n',arr8)
5  arr9 = np.array([10,10,10])
6  print ('第二个数组: \n',arr9)
7  print ('两个数组之和: \n',np.add(arr8,arr9))

```

运行结果：

```

第一个数组: [[0.  1.  2.]  [3.  4.  5.]  [6.  7.  8.]]
第二个数组: [10 10 10]
两个数组之和: [[10. 11. 12.]  [13. 14. 15.]  [16. 17.
18.]]

```

4. 矩阵运算的其他方法

矩阵运算在数据预处理中的使用频率非常高，涉及的方法有相加、相减、相乘等，限于篇幅不能逐一举例，读者可以参加表2-3自行使用。

表2-3 矩阵运算的其他方法

方法描述	函 数	方法描述	函 数
矩阵相加	numpy.add()	矩阵倒数	numpy.reciprocal() 如：1/2 的倒数为 2/1
矩阵相减	numpy.subtract()	矩阵求幂	numpy.power()
矩阵相乘	numpy.multiply()	矩阵取余	numpy.mod()
矩阵相除	numpy.divide()		

2.1.4 NumPy线性代数运算

1. 点积

```
1 import numpy as np
2
3 arr10 = np.array([[1,2],[3,4]])
4 arr11 = np.array([[5,6],[7,8]])
5 print("数组的点积:\n",np.dot(arr10,arr11))
6 print("向量的点积:\n",np.vdot(arr10,arr11))
7 print ("向量的内积:\n",np.inner(arr10,arr11))
```

运行结果:

数组的点积:[[19 22] [43 50]]
向量的点积:70
向量的内积:[[17 23] [39 53]]
内积计算: 15+26=17, 17+28=23 35+46=39, 37+48=53

2. 线性代数操作的其他方法

线性代数运算也是比较重要的知识，NumPy使用频率也较高，具体函数如表2-4所示，读者可参照上述代码自行使用。

表2-4 线性代数操作的其他方法

方法描述	函 数	方法描述	函 数
矩阵乘积	numpy.matmul()	矩阵线性方程解	numpy.linalg.solve()
矩阵的行列式	numpy.linalg.det()	逆矩阵	numpy.linalg.inv()

2.1.5 NumPy IO操作

NumPy为ndarray对象引入了一个简单的文件格式npz，npz文件用于存储重建ndarray所需的数据、图形、dtype和其他信息。

1. 数组保存为numpy文件

```
1 import numpy as np
2
3 arr12 = np.array([[17,23],[39,53]])
4 # 保存到 outfile.npy 文件中
5 np.save('outfile.npy',arr12)
6
7 #加载 npy 文件
8 arr13 = np.load('outfile.npy')
9 print (arr13)
```

运行结果：

```
[[17. 23.] [39. 53.]]
```

2. 数组保存为txt文件

```
1 import numpy as np
2
3 np.savetxt('out.txt',arr12)
4 # 加载 txt 文件
5 arr14 = np.loadtxt('out.txt')
6 print(arr14)
```

运行结果：

```
[[17. 23.] [39. 53.]]
```

2.2 SciPy

SciPy是一个Python的常用科学计算工具包，本节主要介绍SciPy的安装和特点以及SciPy Linalg、SciPy文件操作、SciPy插值、SciPy Ndimage 和 SciPy 算法优化等常用的内置模块（源代码见：Chapter2/ScipyDome.py）。

2.2.1 SciPy的安装和特点

SciPy官网（网址：<https://www.scipy.org/>）如图2-3所示，包含最优化、线性代数、积分、插值、特殊函数、快速傅里叶变换、信号处理和图像处理、常微分方程求解等模块。

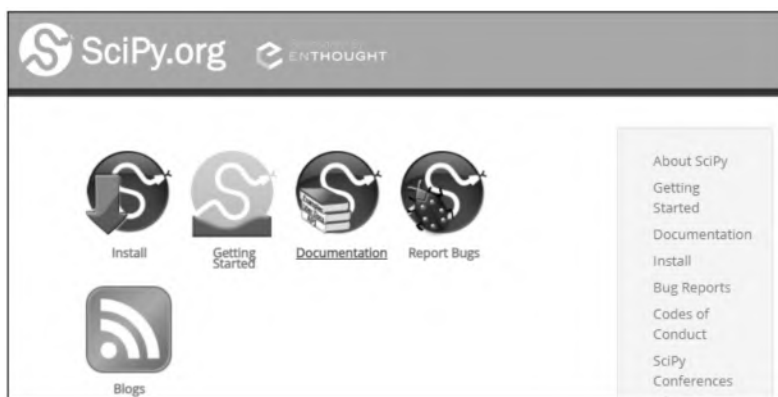


图2-3 SciPy官网

1. SciPy的安装

可使用以下3种方法安装SciPy：

方法一 Anaconda已经集成了SciPy，读者可使用命令`conda list`在DOS环境下查看。

方法二 可以通过pip命令自动安装，请执行如下命令：`pip install scipy`。

方法三 在GitHub上下载SciPy源码（网址：<https://github.com/numpy/numpy>），然后在源码根目录下执行如下命令：`python setup.py install`。

安装完成后，检测是否成功。首先打开Sublime，然后按F6键进入Python环境，最后输入`import sicpy`，得到如图2-4所示的状态即表示

安装成功。

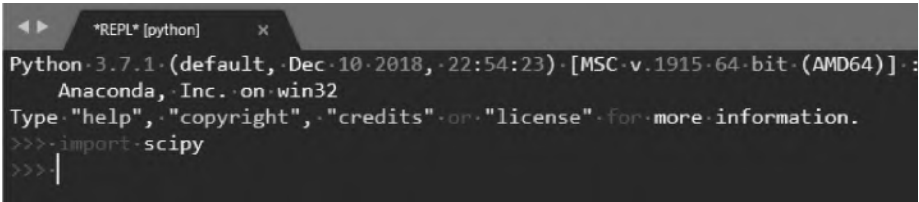
A screenshot of a Python REPL window titled "*REPL* [python]". The window shows the following text: "Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64-bit (AMD64)] on win32", "Anaconda, Inc. on win32", "Type 'help', 'copyright', 'credits' or 'license' for more information.", and the command prompt ">>> import scipy" followed by a cursor on the next line ">>>".

图2-4 验证SciPy是否成功安装

2. SciPy的特点

SciPy库依赖于NumPy，它提供了便捷且快速的N维数组操作。SciPy库与NumPy数组一起工作，并提供了许多用户友好和高效的实现方法，它们一起运行在所有流行的操作系统上，安装快速且免费。

SciPy提供了丰富的子模块，具体参见表2-5所示。

表2-5 SciPy的子模块

模块描述	子 模 块	模块描述	子 模 块
线性代数	scipy.linalg	向量量化	scipy.cluster
优化算法	scipy.optimize	数学常量	scipy.constants
矩阵线性方程解	scipy.sparse	数据输入输出	scipy.io
统计函数	scipy.stats	信号处理	scipy.signal
快速傅里叶变换	scipy.fftpack	特殊数学函数	scipy.special
积分	scipy.integrate	空间数据结构和算法	scipy.spatial
N 维图像	scipy.ndimage		

2.2.2 SciPy Linalg

SciPy Linalg模块包含线性代数函数，使用该模块可以计算逆矩阵、求特征值、解线性方程组以及求行列式等。

1. 求解线性方程组

我们使用Linalg求解如下方程组：

```
1  3x+2y=2
2  x-y=4
3  5y+z=-2
```

完整的代码实现如下：

```
1  import numpy as np
2  from scipy import linalg
3
4  a = np.array([[3, 2, 0], [1, -1, 0], [0, 5, 1]])
5  b = np.array([2, 4, -2])
6  res1 = linalg.solve(a, b)
7  print ('线性方程组的解是: ',res1)
```

代码说明：

数组a表示未知数的系数矩阵，数组b表示等号右边方程组值的矩阵。

运行结果：

线性方程组的解是： [2. -2. 8.]

2. 求解行列式

```
1  import numpy as np
2  from scipy import linalg
3
4  A = np.array([[1,2],[3,4]])
5  res2 = linalg.det(A)
6  print ('行列式的解是: ',res2)
```

运行结果：

行列式的解是：14-23=-2.0

3. 求解特征值和特征向量

```
1 <pre>
2 import numpy as np
3 from scipy import linalg
4
5 A = np.array([[1,2],[3,4]])
6 λ, v = linalg.eig(A)
7 print ("特征值:\n",λ,"\n 特征向量:\n", v)
```

代码说明：

- A表示方阵。
- 特征值 λ 和相应的特征向量v的关系是： $Av = \lambda v$ 。

运行结果：

特征值:[-0.37228132+0.j 5.37228132+0.j]
特征向量:[[-0.82456484 -0.41597356] [0.56576746
-0.90937671]]

4. 奇异值分解 (SVD)

```
1 import numpy as np
2 from scipy import linalg
3
4 a = np.random.randn(2, 3) + 1.j*np.random.randn(2, 3)
5 U, s, Vh = linalg.svd(a)
6 print ('酉矩阵 1:\n',U,'\n 酉矩阵 2:\n', Vh,'\n 奇异值\n',s)
```

代码说明：

- 矩阵a分解为两个酉矩阵（U和Vh）和一个奇异值。
- $a=U*s*Vh$ 。

运行结果：

```
酉矩阵 1: [[-0.55728327+6.19565045e-18j  0.83032244+1.53440325e-16j]
[-0.73499882+3.86279815e-01j -0.4933054 +2.59257451e-01j]]
酉矩阵 2: [[-0.01859058+0.2815052j  0.48875549-0.51402169j
-0.59175181-0.25911153j]  [ 0.05125373-0.51573816j
-0.43163592-0.32192764j -0.03326072-0.66357664j]  [-0.6951096
+0.41063251j -0.24183519-0.38531617j  0.37284682-0.04728662j]]
奇异值:  [4.13890535  0.88993684]
```

2.2.3 SciPy文件操作

1. 保存MATLAB文件

```
1 import numpy as np
2 import scipy.io as sio
3
4 #保存一个matlab 文件
5 vect = np.arange(10)
6 sio.savemat('array.mat', {'vect':vect})
7 加载matlab 文件
8 import numpy as np
9 import scipy.io as sio
10
11 #加载一个matlab 文件
12 mat_file_content = sio.loadmat('array.mat')
13 print (mat_file_content)
```

运行结果：

```
{ '__header__': b'MATLAB 5.0  MAT-file Platform: nt,  Created on: Tue
Feb 26 16:32:00 2019',  '__version__': '1.0',  '__globals__': [],
'vect': array([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]]) }
```

2. 查看MATLAB文件变量

```
1 import numpy as np
2 import scipy.io as sio
3
4 # 列出 MATLAB 文件中的变量
5 mat_file_content = sio.whosmat('array.mat')
6 print (mat_file_content)
```

运行结果：

```
[('vect', (1, 10), 'int32')]
```

2.2.4 SciPy插值

插值是在直线或曲线上的两点之间找到值的过程。插值不仅适用于统计学，而且在科学、商业和需要预测两个现有数据点之间的值时也很有用。

1. 绘制二维空间图

```
1 import numpy as np
2 from scipy.interpolate import *
3 import matplotlib.pyplot as plt
4
5 # 两个维度空间点绘图
6 x = np.linspace(0, 4, 12)
7 y = np.cos(x**2/3+4)
8 print (x,y)
9 plt.plot(x, y, 'o')
10 plt.show()
```

二维空间的余弦散列点运行结果如图2-5所示。

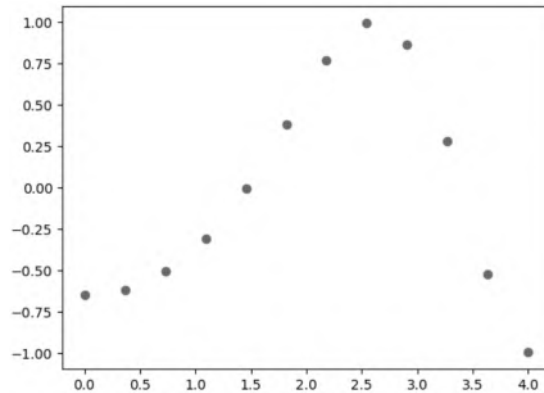


图2-5 绘制二维空间图

2. 一维插值

一种基于固定数据点来推导出函数的便捷方法，可以使用线性插值在给定数据定义域内的任意位置推导出该函数。

```
1 import numpy as np
2 from scipy.interpolate import *
3 import matplotlib.pyplot as plt
4
5 f1 = interp1d(x, y, kind = 'linear')
6 f2 = interp1d(x, y, kind = 'cubic')
7 xnew = np.linspace(0, 4, 30)
8 plt.plot(x, y, 'o', xnew, f1(xnew), '-', xnew, f2(xnew), '--')
9 plt.legend(['data', 'linear', 'cubic', 'nearest'], loc = 'best')
10 plt.show()
```

运行结果如图2-6所示。

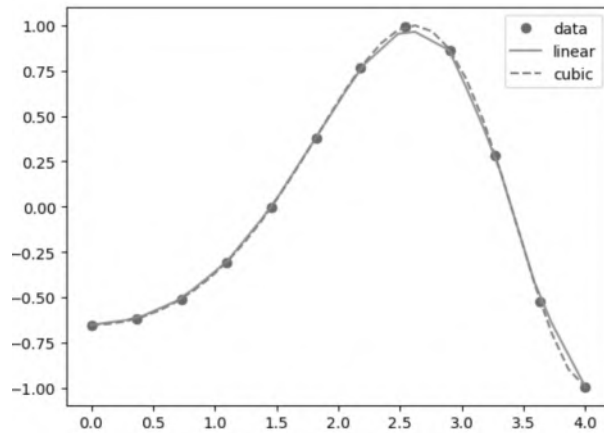


图2-6 一维插值

3. 样条曲线

```
1 import numpy as np
2 from scipy.interpolate import *
3 import matplotlib.pyplot as plt
4
5 # 默认平滑参数
6 x = np.linspace(-3, 3, 50)
7 y = np.exp(-x**2) + 0.1 * np.random.randn(50)
8 plt.plot(x, y, 'ro', ms = 5)
9
10 # 手动更改平滑量
11 spl = UnivariateSpline(x, y)
12 xs = np.linspace(-3, 3, 1000)
13 spl.set_smoothing_factor(0.5)
14 plt.plot(xs, spl(xs), 'b', lw = 3)
15 plt.show()
```

运行结果如图2-7所示。

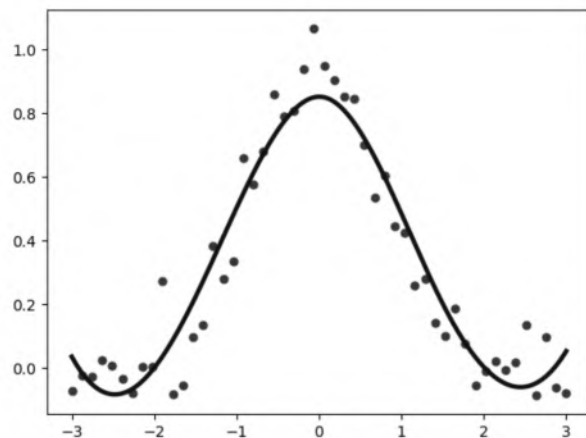


图2-7 样条曲线插值

2.2.5 SciPy Ndimimage

SciPy的ndimage子模块专用于图像处理，ndimage表示一个n维图像。

图像处理中一些最常见的任务如下：

- 图像的输入、输出和显示。
- 基本操作，如裁剪、翻转、旋转等。
- 图像过滤，如去噪、锐化等。
- 图像分割，如标记对应于不同对象的像素。
- 分类。
- 特征提取。
- 注册。

下面使用SciPy实现其中的一些功能。

1. 打开图像

```
1 from scipy import misc,ndimage
2 import scipy.ndimage as nd
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 f = misc.face()
7 misc.imsave('face.png', f)
8 plt.imshow(f)
9 plt.show()
```

运行结果如图2-8所示。



图2-8 打开图像

2. 图像倒置

```
1 from scipy import misc,ndimage
2 import scipy.ndimage as nd
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 face = misc.face()
7 flip_ud_face = np.flipud(face)
8 plt.imshow(flip_ud_face)
9 plt.show()
```


运行结果如图2-9所示。



图2-9 图像倒置

3. 按指定的角度旋转图像

```
1 from scipy import misc, ndimage
2 import scipy.ndimage as nd
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 face = misc.face()
7 rotate_face = ndimage.rotate(face, 45)
8 plt.imshow(rotate_face)
9 plt.show()
```

运行结果如图2-10所示。

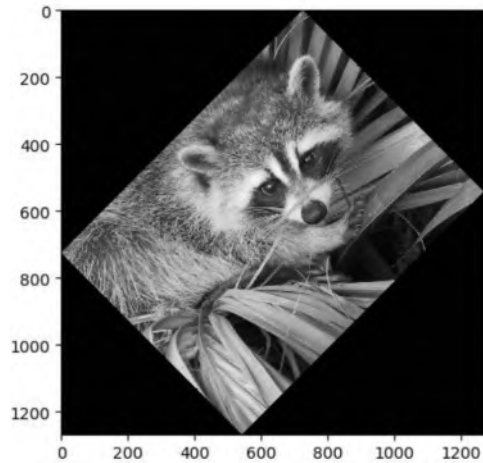


图2-10 旋转图像

4. 模糊图像

```
1 from scipy import misc, ndimage
2 import scipy.ndimage as nd
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 face = misc.face()
7 blurred_face = ndimage.gaussian_filter(face, sigma=5)
8 plt.imshow(blurred_face)
9 plt.show()
```

代码说明：

sigma值表示5级模糊程度。通过调整sigma值，可以看到图像质量的变化。

运行结果如图2-11所示。

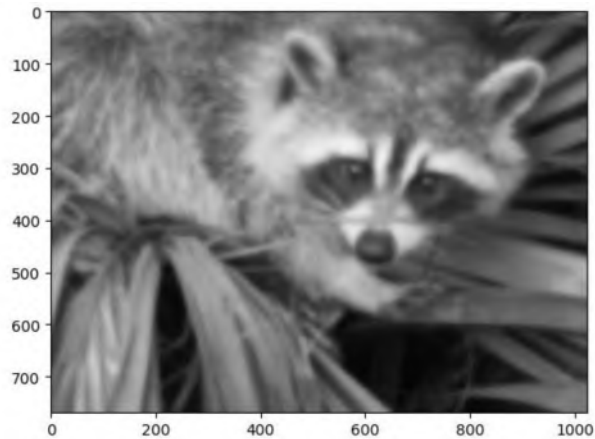


图2-11 模糊图像

2.2.6 SciPy优化算法

1. 梯度下降优化算法

求解方程： $x^2 - 2x$ 的最小值，求解代码如下：

```
1 def f(x):  
2     return x**2-2*x  
3  
4 x = np.arange(-10, 10, 0.1)  
5 plt.plot(x, f(x))  
6 plt.show()
```

运行结果如图2-12所示。

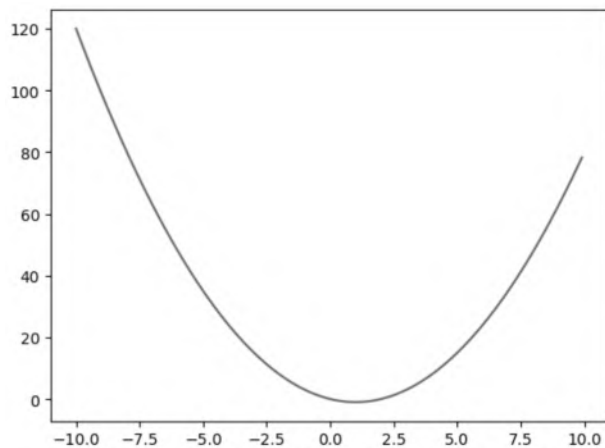


图2-12 函数最小值

通过可视化结果可以发现，最小值比0略小。下面是梯度下降的实现代码：

```
1 from scipy import optimize
2
3 # 梯度下降优化算法
4 def f(x):
5     return x**2-2*x
6 initial_x = 0
7 optimize.fmin_bfgs(f,initial_x)
```

运行结果：

```
Optimization terminated successfully.  Current function value:
-1.000000  Iterations: 2  Function evaluations: 9
Gradient evaluations: 3
```

结果显示最优值为-1，也满足可视化的判断。关于梯度下降算法的理论知识，读者可参考相关资料。

2. 最小二乘法优化算法

```

1 # 最小二乘法
2 from scipy.optimize import least_squares
3 def fun_rosenbrock(x):
4     return np.array([10 * (x[1] - x[0]**2), (1 - x[0])])
5 input = np.array([2, 2])
6 res = least_squares(fun_rosenbrock, input)
7 print ('最小值是: ',res)

```

运行结果:

最小值是: = [1.0,1.0]

本节展示了SciPy在优化算法等领域的应用，有关具体理论知识，读者可以自行查找资料学习。

2.3 Pandas

Pandas也是一个常用的Python科学计算工具包，被广泛用于包括金融、经济、统计、分析等学术和商业领域的数据分析、数据清洗和准备等工作，是数据预处理的核心模块。使用Pandas可以完成数据处理和分析的5个典型步骤即数据加载、数据准备、操作、模型和分析。本节主要介绍Pandas的安装和特点以及Pandas的数据结构、数据统计、处理缺失值、稀疏数据、文件操作和可视化技术。

Pandas的官网：<https://pandas.pydata.org/>，如图2-13所示。

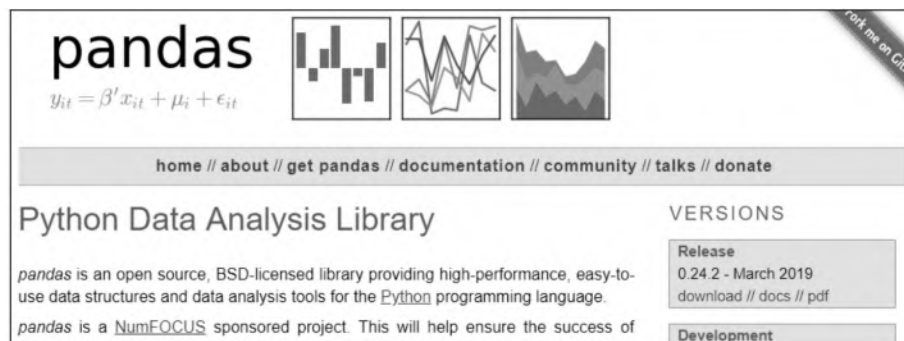


图2-13 Pandas官网

(源代码见: Chapter2/PandasDome.py)

2.3.1 Pandas的安装和特点

1. Pandas的安装

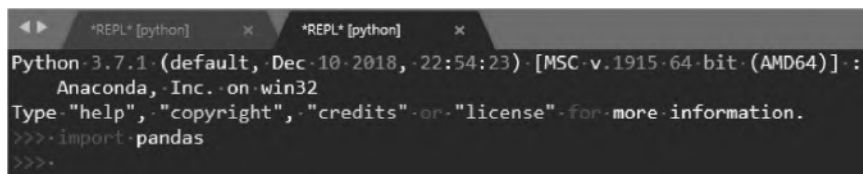
可以通过以下3种方法安装Pandas:

方法一 Anaconda已经集成了Pandas, 读者在命令提示符环境下查看: `conda list`。

方法二 可以通过pip自动安装, 执行如下命令: `pip install pandas`。

方法三 在GitHub上下载Pandas源码(网址: <https://github.com/pandas-dev/pandas>), 然后在源码根目录下执行如下命令: `python setup.py install`。

安装完成后, 检测是否成功。首先启动Sublime, 然后按F6键进入Python环境, 输入`import pandas`, 得到以下状态即安装成功。如图2-14所示。



```
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64-bit (AMD64)] :
Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import pandas
>>>
```

图2-14 验证Pandas是否安装成功

2. Pandas的特点

Pandas快速高效、对缺失值自动处理及支持异构数据类型等诸多特点深受Python爱好者的喜欢，其主要特点包括：

- 快速高效的DataFrame对象，具有默认和自定义的索引。
- 将数据从不同文件格式加载到内存中的数据对象工具。
- 丢失数据的数据对齐和综合处理。
- 重组和摆动日期集。
- 基于标签的切片、索引和大数据集的子集。
- 可以删除或插入来自数据结构的列。
- 按数据分组进行聚合和转换。
- 高性能合并和数据加入。
- 时间序列功能。
- 具有异构类型列的表格数据，例如SQL表格或Excel数据。

2.3.2 Pandas的数据结构

Pandas可处理以下3种数据结构：

1. 系列（Series）

系列是具有均匀数据的一维数组结构。表2-6中的系列是整数1，3，5，6…的集合。

表2-6 一维数组

A	B	C	D	E	F
1	3	5	6	8	10

代码实现：

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 data = np.array(['a','b','c','d'])
6 index=[100,101,102,103]
7 obj = pd.Series(data,index)
8 print(obj)

```

代码说明：

代码中使用了pandas.Series方法处理一位数组，该方法共有4个参数，即pandas.Series(data, index, dtype, copy)，各参数的说明如下：

- data：传入的数据参数，数据类型支持数组、列表等。
- index：索引，与数据的长度相同。
- dtype：用于表示数据类型，省略是默认为浮点数据类型。
- copy：复制一份数据，默认为false。

运行结果：

```
100 a  101 b  102 c  103 d  dtype: object
```

2. 数据帧（DataFrame）

数据帧是一个具有异构数据的二维数组，如表2-7所示。

表2-7 二维数组

姓 名	年 龄	性 别	籍 贯
张三	25	男	四川-成都
李四	22	男	河南-郑州
王五	25	女	陕西-西安

代码实现：

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 data = {'Name': ['Tom', 'Jack', 'Steve', 'Ricky'], 'Age': [28, 34, 29, 42]}
6 df = pd.DataFrame(data, index=['rank1', 'rank2', 'rank3', 'rank4'],
7 dtype=float)
7 print(df[:2])
```

代码说明：

代码中使用了pandas.DataFrame方法处理二维数据，该方法共有5个参数，即pandas.DataFrame(data, index, columns, dtype, copy)，各参数说明如下：

- data：传入的数据参数，数据类型支持数组、列表、字典等。
- index：索引，与数据的长度相同。
- columns：列参数。
- dtype：用于表示数据类型，省略是默认为浮点型。
- copy：复制数据，默认为false（不复制）。

运行结果：

```
      Name  Age
rank1  Tom  28.0
rank2  Jack  34.0
```

3. 面板 (Panel)

面板是具有异构数据的三维数据结构，在图形表示中很难表示出面板，但是一个面板可以说明为DataFrame的容器。

代码实现：

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
6         'Item2' : pd.DataFrame(np.random.randn(4, 2))}
7 p = pd.Panel(data)
8 print( p['Item1'])
```

代码说明：

代码中使用了pandas.Panel方法，该方法表示创建一个面板。该方法共有6个参数，即pandas.Panel (data, items, majoraxis, minoraxis, dtype, copy)，各参数的含义如下：

- data: 传入的数据参数，数据类型支持数组、列表、字典等。
- items: axis 0，每个项目对应于内部包含的数据帧（DataFrame）。
- major_axis: axis 1，表示每个数据帧（DataFrame）的行。
- minor_axis: axis 2，表示每个数据帧（DataFrame）的列。
- dtype: 用于表示数据类型，省略是默认为浮点数据类型。
- copy: 复制数据，默认为false（不复制）。

运行结果：

	0	1	2
0	-1.056665	-2.042454	0.761872
1	-0.973774	-1.263141	0.586314
2	0.463656	-0.786289	-0.791540
3	-1.533797	-0.891689	0.035241

2.3.3 Pandas的数据统计

1. 利用DataFrame创建二维数组

代码实现：

```
1 import pandas as pd
2 import numpy as np
3
4 d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve',
5     'Minsu','Jack','Lee','David','Gasper','Betina','Andres']),
6     'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
7     'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,
8     2.98, 4.80,4.10,3.65])}
9 df = pd.DataFrame(d)
10 print(df)
```

运行结果：

	Name	Age	Rating
0	Tom	25	4.23
1	James	26	3.24
2	Ricky	25	3.98
3	Vin	23	2.56
4	Steve	30	3.20
5	Minsu	29	4.60
6	Jack	23	3.80
7	Lee	34	3.78
8	David	40	2.98
9	Gasper	30	4.80
10	Betina	51	4.10
11	Andres	46	3.65

2. 数据统计

代码实现：

```

1 # 统计数据之和
2 print('数据之和:\n',df.sum())
3 # 统计数据的均值
4 print('数据的均值:\n',df.mean())
5 # 统计数据的标准偏差
6 print('数据的标准偏差:\n',df.std())

```

运行结果:

数据之和:

```

Name      TomJamesRickyVin...
Age       382
Rating    44.92
dtype: object

```

数据的均值:

```

Age       31.833333
Rating    3.743333
dtype: float64

```

数据的标准偏差:

```

Age       9.232682
Rating    0.661628
dtype: float64

```

Pandas的其他统计方法，如表2-8所示。

表2-8 Pandas的其他统计方法

方法描述	函 数	方法描述	函 数
非空观测数量	pandas.count()	所有值中的最小值	pandas.min()
所有值之和	pandas.sum()	所有值中的最大值	pandas.max()
所有值的平均值	pandas.mean()	绝对值	pandas.abs()
所有值的中位数	pandas.median()	数组元素的乘积	pandas.prod()
值的模值	pandas.mode()	数组元素的累加和	pandas.cumsum()
值的标准偏差	pandas.std()	累计乘积	pandas.cumprod()

2.3.4 Pandas处理丢失值

机器学习和数据挖掘等领域由于数据缺失导致的数据质量差，在模型预测的准确性上面临着严重的问题。为了提高模型效果可以使用Pandas对数据缺失值进行处理。

1. 构造一个含有缺失值的数据集

使用重构索引（Reindexing），创建一个缺少值的DataFrame。
代码实现：

```
1 import pandas as pd
2 import numpy as np
3
4 df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',
5 'h'], columns=['one', 'two', 'three'])
6 df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
7 print (df)
```

运行结果：

	one	two	three
a	0.749993	-0.524868	0.996772
b	NaN	NaN	NaN
c	-2.129368	-1.000072	1.515642
d	NaN	NaN	NaN
e	1.617229	-0.046227	-1.003343
f	0.763989	-0.283858	-0.009689
g	NaN	NaN	NaN
h	1.246244	0.236078	-2.142169

在输出结果中，NaN表示不是数字的值。

2. 检查缺失值

Pandas 提供了 `isnull()` 和 `notnull()` 函数，它们也是 Series 和 DataFrame 对象的方法，可用于检查缺失值，代码如下：

```
1 import pandas as pd
2 import numpy as np
3
4 df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',
5 'h'], columns=['one', 'two', 'three'])
6 df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
7 print (df['one'].isnull())
```

运行结果：

	one	two	three
a	False	False	False
b	True	True	True
c	False	False	False
d	True	True	True
e	False	False	False
f	False	False	False
g	True	True	True
h	False	False	False

3. 清理 / 填充缺少数据

(1) 用标量值 0 替换缺失值，其中 g 行表示缺少的数据。

```
1 import pandas as pd
2 import numpy as np
3
4 df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',
5 'h'], columns=['one', 'two', 'three'])
6 df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
7 print ("NaN replaced with '0':")
8 print (df.fillna(0))
```

运行结果：

	one	two	three
a	-0.514238	2.088999	0.432063
b	0.000000	0.000000	0.000000
c	-1.252190	1.228565	0.593918
d	0.000000	0.000000	0.000000
e	-0.356422	-0.316063	2.038978
f	-1.124368	1.445957	0.080762
g	0.000000	0.000000	0.000000
h	0.492614	-0.259457	-0.935228

(2) 用缺失值的前一行替换缺失值，例如a，b行，其中b为缺少数据。

实现代码如下：

```
1 import pandas as pd
2 import numpy as np
3
4 df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',
5 'h'], columns=['one', 'two', 'three'])
6 df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
7 print (df.fillna(method='pad'))
```

运行结果：

	one	two	three
a	-1.643899	0.189767	-0.164862
b	-1.643899	0.189767	-0.164862
c	1.016934	0.283606	0.906205
d	1.016934	0.283606	0.906205
e	-0.046524	-0.060849	-0.046224
f	-0.631722	-1.034078	0.416126
g	-0.631722	-1.034078	0.416126
h	0.485579	-1.348188	-0.099870

(3) 剔除缺失值。

```
1 import pandas as pd
2 import numpy as np
3
4 df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',
5 'h'], columns=['one', 'two', 'three'])
6 df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
7 print (df.dropna(axis=0))
```

代码说明：

- axis=0: 表示作用在行上，如果axis=1，则表示作用在列上。

运行结果：

	one	two	three
a	0.617105	1.021967	-0.991709
c	0.238544	-0.100752	-0.131364
e	1.213844	0.091464	1.189781
f	-0.061523	-0.229137	-0.032633
h	-0.455861	-1.027256	-0.236553

(4) 设置任意值替换缺失值，如one列的2000被60替换，two列的1000被10替换。

代码如下：

```
1 import pandas as pd
2 import numpy as np
3
4 df = pd.DataFrame({'one': [10, 20, 30, 40, 50, 2000],
5 'two': [1000, 0, 30, 40, 50, 60]})
6 print (df.replace({1000: 10, 2000: 60}))
```

运行结果：

	one	two
0	10	10
1	20	0
2	30	30
3	40	40
4	50	50
5	60	60

(5) 忽略缺失值。

```
1 import pandas as pd
2 import numpy as np
3
4 df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',
5 'h'], columns=['one', 'two', 'three'])
6 df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
7 print("df.dropna():\n{}\n".format(df.dropna()))
```

运行结果：

	one	two	three
a	-1.183113	0.669814	-0.416021
c	0.478092	-0.065262	0.145993
e	0.006571	-1.060615	-1.099261
f	-0.245773	0.596491	-0.213138
h	-0.794740	0.252769	0.108073

2.3.5 Pandas处理稀疏数据

当任何匹配特定值的数据（NaN/缺失值或任何值）被省略时，稀疏对象被“压缩”。

稀疏数据处理

```

1  import pandas as pd
2  import numpy as np
3
4  df = pd.DataFrame(np.random.randn(10000, 4))
5  df.ix[:9998] = np.nan
6  sdf = df.to_sparse()
7  # 调用 to_dense 标准密集进行稀疏数据的处理
8  print ('稀疏数据集: \n',sdf.to_dense())
9  # 稀疏率
10 print ('稀疏率: \n',sdf.density)

```

运行结果:

稀疏数据集:

	0	1	2	3
0	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN
...	...	...	...	...
9996	NaN	NaN	NaN	NaN
9997	NaN	NaN	NaN	NaN
9998	NaN	NaN	NaN	NaN
9999	-1.104343	-0.516193	1.156061	-0.328376

[10000 rows x 4 columns]

稀疏率:

0.0001

2.3.6 Pandas的文件操作

Pandas库提供了一系列的read_函数来读取各种格式的文件，这些read_函数如下所示:

read_csv	read_table	read_fwf
read_clipboard	read_excel	read_hdf
read_html	read_json	read_msgpack
read_pickle	read_sas	read_sql
read_stata	read_feather	

下面通过示例来说明部分函数的使用。

1. 操作Excel文件

```
1 import pandas as pd
2 import numpy as np
3
4 df1 = pd.read_excel("data/test1.xlsx")
5 print("df1:\n{}\n".format(df1))
```

代码说明：

本例表示使用Pandas的read_excel方法来处理Excel格式文件并打印输出Excel文件的内容。

运行结果：

```
df1:
   学号  姓名  性别  年龄
0  212001  张三  男    23
1  212002  李四  男    22
2  212003  王五  男    23
3  212004  赵六  男    21
4  212005  钱七  男    24
```

2. 操作CSV文件

```
1 import pandas as pd
2 import numpy as np
3
4 df2 = pd.read_csv("data/test2.csv", sep=",")
5 print("df2:\n{}\n".format(df2))
```

代码说明：

本例使用Pandas的read_csv方法来处理CSV格式文件并打印出文件的内容。

运行结果：

```
df2:
      学号  姓名  性别  年龄
0  212001  张三  男    23
1  212002  李四  男    22
2  212003  王五  男    23
3  212004  赵六  男    21
4  212005  钱七  男    24
```

3. 操作JSON文件

```
1 import pandas as pd
2 import numpy as np
3
4 # 转存 JSON 文件
5 df = pd.DataFrame([['a', 'b'], ['c', 'd']], index=['row 1', 'row 2'], columns=['col 1', 'col 2'])
6 data_json=df.to_json(orient='columns')
7 print(data_json)
8
9 # 读取 json 文件
10 df3 = pd.read_json("data/test3.json")
11 print("df3:\n{}\n".format(df2))
```

代码说明：

本例首先将列表数据转化为JSON格式数据，然后调用Pandas的read_json方法处理JSON格式文件并打印输出内容。

运行结果：

JSON 文件

```
{"col 1":{"row 1":"a","row 2":"c"},"col 2":{"row 1":"b","row 2":"d"}}
```

df3:

	学号	姓名	性别	年龄
0	212001	张三	男	23
1	212002	李四	男	22
2	212003	王五	男	23
3	212004	赵六	男	21
4	212005	钱七	男	24

2.3.7 Pandas可视化

1. 绘制折线图

```
1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import numpy as np
4
5 # 折线图
6 df = pd.DataFrame(np.random.randn(10,4), index=
7     pd.date_range('2019/2/27',periods=10), columns=list('ABCD'))
8 df.plot()
9 plt.show()
```

运行结果如图2-15所示。

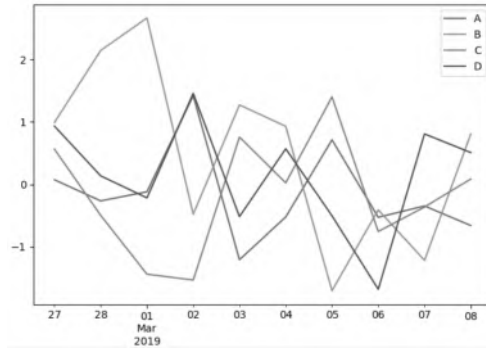


图2-15 折线图

2. 绘制条形图

```
1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import numpy as np
4
5 df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d'])
6 df.plot.bar()
7 plt.show()
```

运行结果如图2-16所示。

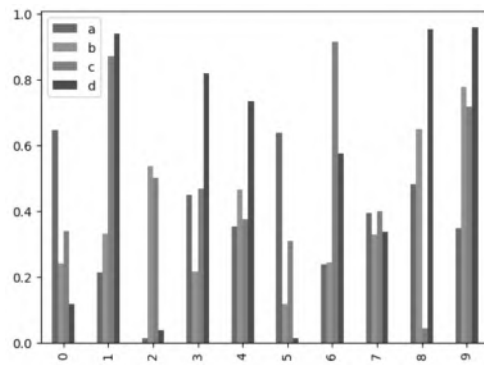


图2-16 条形图

3. 绘制堆积条形图

```
1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import numpy as np
4
5 # 堆积条形图
6 df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d'])
7 df.plot.bar(stacked=True)
8 plt.show()
```

运行结果如图2-17所示。

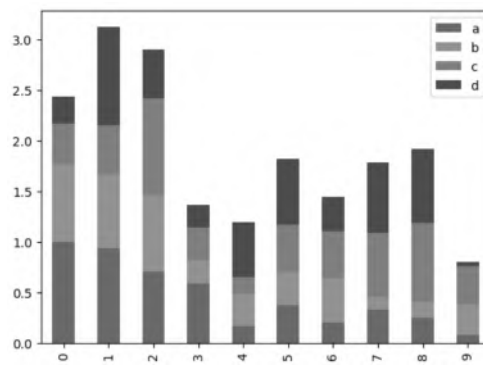


图2-17 堆积条形图

4. 绘制水平条形图

```
1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import numpy as np
4
5 # 水平条形图
6 df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d'])
7 df.plot.barh(stacked=True)
8 plt.show()
```

运行结果如图2-18所示。

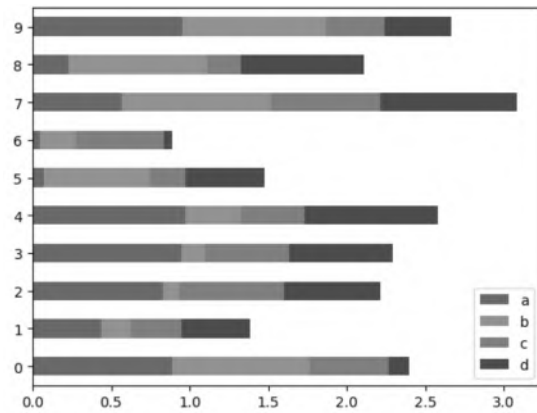


图2-18 水平条形图

5. 绘制直方图

```

1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import numpy as np
4
5 # 直方图
6 df = pd.DataFrame({'a':np.random.randn(1000)+1,
7                    'b':np.random.randn(1000), 'c':
8                    np.random.randn(1000) - 1}, columns=['a', 'b', 'c'])
9 df.plot.hist(bins=20)
10 plt.show()

```

运行结果如图2-19所示。

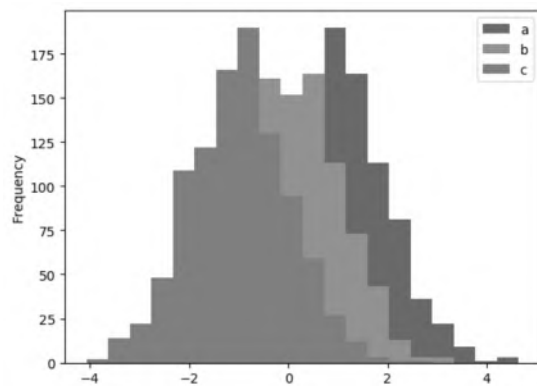


图2-19 直方图

6. 绘制多个直方图

```
1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import numpy as np
4
5 # 多个直方图
6 df=pd.DataFrame({'a':np.random.randn(1000)+1,'b':np.random.randn
7   (1000),'c':
8   np.random.randn(1000) - 1}, columns=['a', 'b', 'c'])
9 df.hist(bins=20)
```

运行结果如图2-20所示。

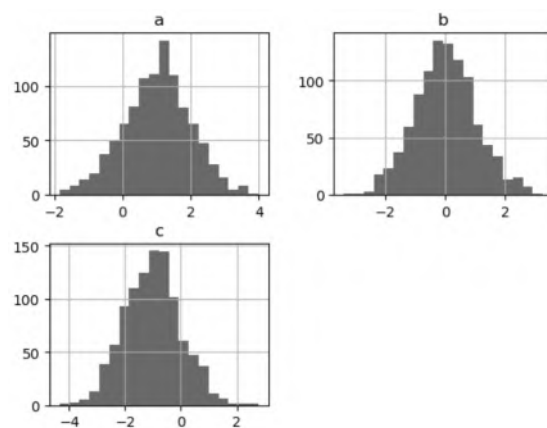


图2-20 多个直方图

7. 绘制箱形图

```

1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import numpy as np
4
5 # 箱形图
6 df = pd.DataFrame(np.random.rand(10, 5), columns=['A', 'B', 'C', 'D',
    'E'])
7 df.plot.box()
8 plt.show()

```

运行结果如图2-21所示。

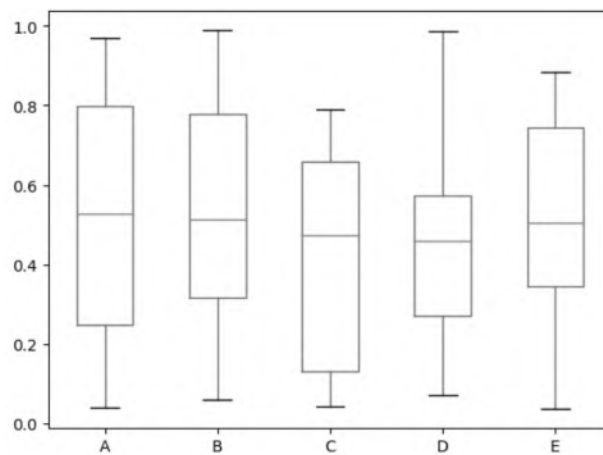


图2-21 箱形图

8. 绘制区域块图形

```

1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import numpy as np
4
5 # 区域块图形
6 df = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c',
    'd'])
7 df.plot.area()
8 plt.show()

```

运行结果如图2-22所示。

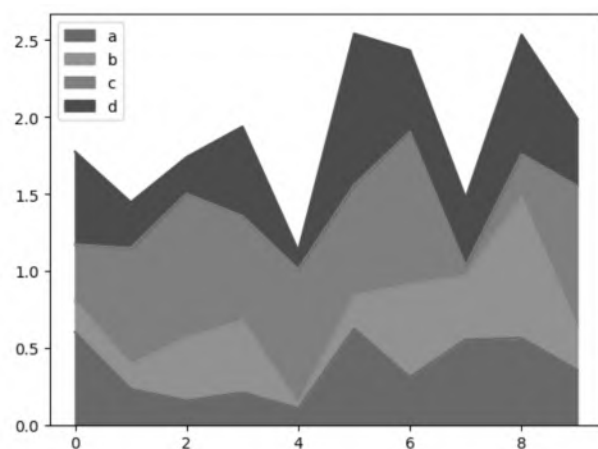


图2-22 区域块图形

9. 绘制散点图形

```
1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import numpy as np
4
5 # 散点图形
6 df = pd.DataFrame(np.random.rand(50, 4), columns=['a', 'b', 'c',
7 'd'])
8 df.plot.scatter(x='a', y='b')
9 plt.show()
```

运行结果如图2-23所示。

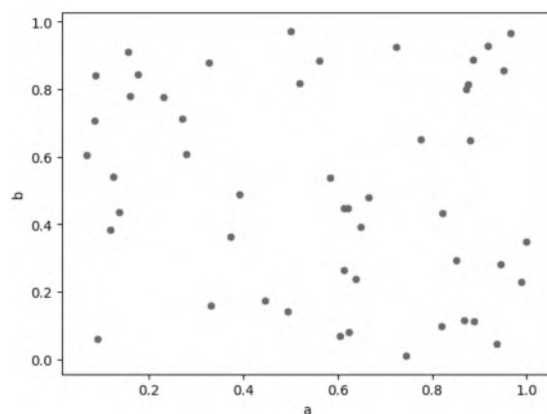


图2-23 散点图形

10. 绘制饼状图

```
1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import numpy as np
4
5 # 饼状图
6 df = pd.DataFrame(3 * np.random.rand(4), index=['A', 'B', 'C', 'D'],
7                   columns=['x'])
8 df.plot.pie(subplots=True)
```

运行结果如图2-24所示。

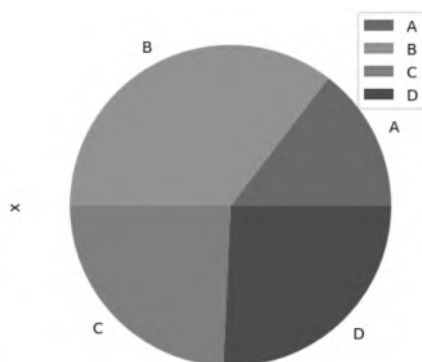


图2-24 饼状图

2.4 本章小结

本章介绍了数据预处理中常用的Python科学工具包NumPy、SciPy和Pandas，系统地学习了这些包的概念和使用。NumPy工具包中的数学函数和线性代数知识，对于科研和阅读文献非常有帮助，IO和数组操作也是在文本处理中常见的。SciPy工具包常用于算法优化和实验结果分析等方面。Pandas工具包对缺失值、稀疏值处理方面较为擅长。下

一章将介绍数据采集与存储以及使用Scrapy实现爬取数据并存储在MySQL数据库中等知识。

第3 章

数据采集与存储

随着网络和信息技术的不断普及，人类产生的数据量正在呈指数级增长，数据的形式也更加丰富，主要有结构化数据、半结构化数据、非结构化数据。面对各种形式的数据应当采用什么样的数据采集策略，如何实现网络爬虫爬取网页信息，如何对抓取到的网页信息进行本地化存储，都是数据预处理过程中经常会遇到的问题。本章从数据的分类入手，分别介绍数据采集和存储的常用方法与技术。

3.1 数据与数据采集

数据是指未经过处理的原始记录，如一堆杂志、一叠报纸、开会记录或整本病人的病历记录等，数据因缺乏组织和分类，是无法明确地表达事物代表的意义的。人工智能领域中的数据主要有3类：结构化数据、半结构化数据和非结构化数据，其表现形式不仅仅指文字，也包括图片、音频、视频等一系列可以存储知识的原始资料。

世界上每时每刻都在产生大量的数据，包括物联网传感器数据、社交网络数据、商品交易数据等。为了挖掘这些数据背后的价值，首先需要采集数据，因面对的场景的不同，采集数据的策略也会有差异。比如，针对关系型数据库中的数据、本地存储的文件、图片、音视频数据，直接拷贝即可；如果面对的是庞杂无序的网络数据，则需要采用网络爬虫技术进行处理了。接下来我们将从不同层面对数据的采集和存储方法进行介绍。

3.2 数据类型与采集方法

本节介绍大数据领域中三种主流的数据形式，即结构化数据、半结构化数据和非结构化数据，然后介绍常用的数据采集方法——爬虫技术。

3.2.1 结构化数据

结构化数据是指可以使用关系型数据库表示和存储，表现为二维形式的数据。一般特点是，数据以行为单位，一行数据表示一个实体的信息，每一行数据的属性是相同的。如表3-1所示。

表3-1 二维数据表

Id	Name	Age	Gender
1	张三	21	男
2	李花	24	女
3	王五	22	男

- 数据特点：关系模型数据，关系数据库表示。
- 常见格式：如MySQL、Oracle、SQL Server格式等。
- 应用场合：数据库、系统网站、数据备份、ERP等。
- 数据采集：DB导出、SQL等方式。

结构化数据的存储和排列是很有规律的，这对查询和修改等操作很有帮助，但其扩展性较差。

3.2.2 半结构化数据

半结构化数据是结构化数据的一种形式，它并不符合关系型数据库或其他以数据表的形式关联起来的数据模型结构，其通过相关标记

用来分隔语义元素以及对记录和字段进行分层。因此，它也被称为自描述的结构。半结构化数据同一类实体可以有不同的属性，但这些属性的顺序并不重要，即使它们被组合在一起。如XML格式文件的数据就是半结构数据的一种，以下是一个简单的示例：

```
1 <person>
2   <name>李花</name>
3   <age>13</age>
4   <gender>女</gender>
5 </person>
```

- 数据特点：非关系模型数据，有一定的格式。
- 常见格式：比如Email、HTML、XML、JSON格式等。
- 应用场合：邮件系统、档案系统、新闻网站等。
- 数据采集：网络爬虫、数据解析等方式。

不同的半结构化数据的属性的个数是不定的。有人说半结构化数据是以树或者图的数据结构存储的数据，如上面的例子中，<person>标签是树的根节点，<name>标签是子节点。这样的数据格式可以自由地表达很多有用的信息，如个人描述信息（元数据）等。可见，半结构化数据的扩展性是很好的。

3.2.3 非结构化数据

指没有固定结构的数据，各种文档、图片、视频 / 音频等都属于非结构化数据。对于这类数据可整体进行存储，一般存储为二进制的格式。如图3-1所示。

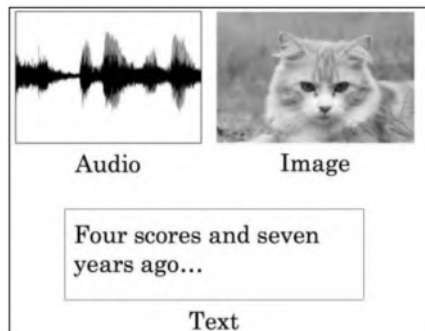


图3-1 非结构化数据

- 数据特点：没有固定格式的数据。
- 常见格式：Word、PDF、PPT、图片、音视频等。
- 应用场合：图片识别、人脸识别、医疗影像、文本分析等。
- 数据采集：网络爬虫、数据存档等方式。

3.3 网络爬虫技术

网络数据采集是指通过网络爬虫或网站公开API等方式从网站上获取数据信息，该方法可以将半结构化数据、非结构化数据从网页中抽取出来，将其存储为统一的本地数据，支持图片、音频、视频等数据采集。

3.3.1 前置条件

可以搭建以下环境，进行网络爬虫的开发：

- 软件系统：Windows 10/Linux。
- 开发环境：Sublime Text3/PyCharm。
- 数据库：MySQL 5.0+Navicat 10.0.11。
- 编程语言：Python 3.7+Anaconda 4.4。
- 爬虫框架：Scrapy。

- 目标网站: <http://blog.jobbole.com/all-posts/>。

3.3.2 Scrapy技术原理

Scrapy是一个为爬取网站数据、提取结构化数据而设计的应用程序框架，通常我们可以很简单地通过Scrapy框架实现一个爬虫，抓取指定网站的内容或图片。

Scrapy爬虫完整架构如图3-2所示（此图来源于网络），其中箭头线表示数据流向。

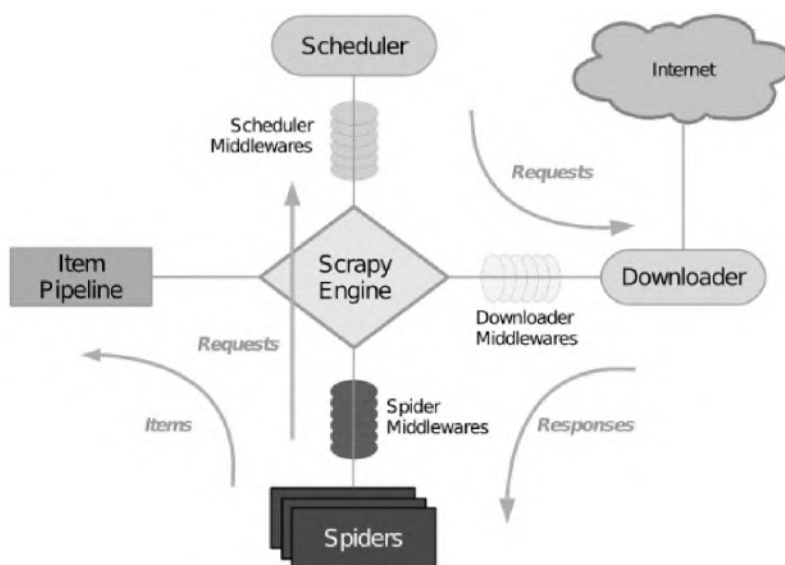


图3-2 Scrapy架构图

- Scrapy Engine（引擎）：负责Spider、ItemPipeline、Downloader、Scheduler之间的通信，信号和数据传递等。
- Scheduler（调度器）：它负责接受引擎发送过来的Request请求，并按照一定的方式进行整理排列和入队，当引擎需要时交还给引擎。
- Downloader（下载器）：负责下载Scrapy Engine（引擎）发送的所有Requests请求，并将其获取到的Responses（响应）交还

给Scrapy Engine（引擎），由引擎交给Spider来处理。

- Spider（爬虫）：它负责处理所有的Responses，从中分析提取数据，获取Item字段需要的数据，并将需要跟进的URL提交给引擎，再次进入Scheduler（调度器）。
- Item Pipeline（管道）：它负责处理Spider中获取到的Item，并进行后期处理（详细分析、过滤、存储等）的地方。
- Downloader Middlewares（下载中间件）：一个可以自定义扩展下载功能的组件。
- Spider Middlewares（Spider中间件）：一个可以自定义扩展的操作引擎和Spider中间通信的功能组件（比如进入Spider的Responses和从Spider出去的Requests）。

3.3.3 Scrapy新建爬虫项目

（1）安装Scrapy爬虫框架，按WIN+R组合键调出命令行环境，执行如下命令：

```
1 pip install scrapy
```

（2）按WIN+R组合键调出命令行环境进入根目录Chapter3文件夹下，创建爬虫项目：

```
1 scrapy startproject 项目名称
```

如图3-3所示。

```
C:\Users\Administrator\Desktop>cd Chapter3
C:\Users\Administrator\Desktop\Chapter3>scrapy startproject BoLeSpider
New Scrapy project 'BoLeSpider', using template directory 'C:\Users\Administrator\Desktop\Chapter3\BoLeSpider', created in:
C:\Users\Administrator\Desktop\Chapter3\BoLeSpider

You can start your first spider with:
cd BoLeSpider
scrapy genspider example example.com

C:\Users\Administrator\Desktop\Chapter3>
```

图3-3 创建BoLeSpider项目

(3) BoLeSpider为项目名称，可以看到将会创建一个名为BoLeSpider的目录，其目录结构大致如下：

```
1  BoLeSpider/
2      scrapy.cfg
3      BoLeSpider/
4          __init__.py
5          items.py
6          pipelines.py
7          settings.py
8          spiders/
9              __init__.py
10             ...
```

下面简单介绍一下主要文件的作用，这些文件分别是：

- scrapy.cfg：项目的配置文件。
- BoLeSpider/：项目的Python模块，将会从这里引用代码。
- BoLeSpider/items.py：项目的目标文件。
- BoLeSpider/pipelines.py：项目的管道文件。
- BoLeSpider/settings.py：项目的设置文件。
- BoLeSpider/spiders/：存储爬虫代码的目录。

(4) 在BoLeSpider项目下创建爬虫目录：

```
1 >> cd BoLeSpider
2 >> Scrapy genspider jobbole http://www.jobbole.com/
```

如图3-4所示。

```
C:\Users\Administrator\Desktop\Chapter3>cd BoLeSpider
C:\Users\Administrator\Desktop\Chapter3\BoLeSpider>Scrapy genspider jobbole http://www.jobbole.com/
Created spider 'jobbole' using template 'basic' in module:
  BoLeSpider.spiders.jobbole
C:\Users\Administrator\Desktop\Chapter3\BoLeSpider>_
```

图3-4 爬虫目标网站

(5) 在同级目录下，执行如下命令，调用爬虫主程序。

```
1 scrapy crawl jobbole
```

如图3-5所示。

```
C:\Users\Administrator\Desktop\Chapter3\BoLeSpider>scrapy crawl jobbole
2019-03-25 10:27:01 [scrapy.utils.log] INFO: Scrapy 1.5.1 started (bot: BoLeSpider)
2019-03-25 10:27:01 [scrapy.utils.log] INFO: Versions: lxml 4.2.5.0, libxml2 2.9.8, cssselect 1.0.3,
parsel 1.5.1, w3lib 1.20.0, Twisted 18.9.0, Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915
64 bit (AMD64)], pyOpenSSL 18.0.0 (OpenSSL 1.1.1a 20 Nov 2018), cryptography 2.4.2, Platform Window
s-10-10.0.17134-SP0
2019-03-25 10:27:01 [scrapy.crawler] INFO: Overridden settings: {'BOT_NAME': 'BoLeSpider', 'NEWSPIDER
_MODULE': 'BoLeSpider.spiders', 'ROBOTSTXT_OBEY': True, 'SPIDER_MODULES': ['BoLeSpider.spiders']}
2019-03-25 10:27:02 [scrapy.middleware] INFO: Enabled extensions:
['scrapy.extensions.corestats.CoreStats',
'scrapy.extensions.telnet.TelnetConsole',
'scrapy.extensions.logstats.LogStats']
```

图3-5 运行爬虫项目

(6) 在BoLeSpider目录下创建main.py:

```
1 # -*- coding: utf-8 -*-
2
3 import sys,os
4 from scrapy.cmdline import execute
5
6 sys.path.append(os.path.dirname(os.path.abspath(__file__)))
7 execute(["scrapy", "crawl", "jobbole"]) # scrapy crawl jobbole
```

执行main.py，效果如图3-6所示。

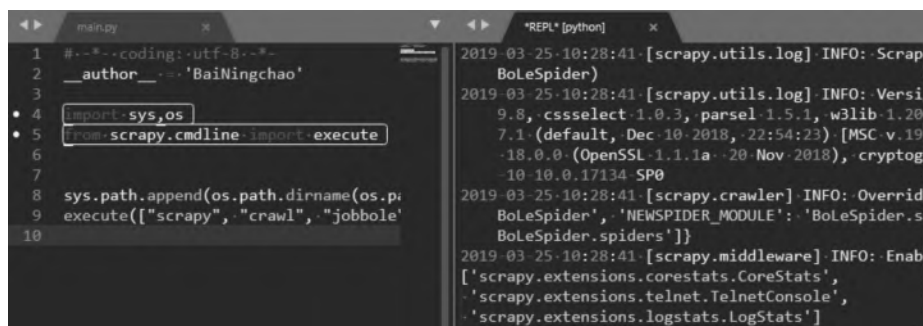


图3-6 封装main函数运行爬虫项目

main.py中的方法与在命令行环境下scrapy crawl jobbole的执行效果是一致的，之所以单独封装，是为了调试和运行的便利。

3.3.4 爬取网站内容

3.3.3节完成了爬虫项目的构建，接下来主要做4个方面的工作：一是对项目进行相关配置；二是对目标爬取内容分析及提取文本特征信息；三是完成数据爬取工作；四是将数据进行本地化存储。

1. 爬虫项目配置

在做数据爬取工作时，如果不符合爬虫协议爬取工作将会中断。比如遇到404错误页面等情况，爬虫会自动退出。显然，这不符合对爬虫结果的期望，我们希望跳过错误页面继续爬取。因此，为了实现对不符合协议的网页继续爬取，需要打开Scrapy爬虫架构中的setting.py文件进行修改，具体修改如下：

```
1  ROBOTSTXT_OBEY = False。 # 此处的 True 改为 False
2  ITEM_PIPELINES = {
3      'BoLeSpider.pipelines.BolespiderPipeline': 1,
4  }
```

2. 分析爬取的内容

在文章列表<http://blog.jobbole.com/all-posts/>（实验时可正常访问，目前该官网处于关闭状态，读者重点明白原理及技术细节）中随机打开一篇文章，对单篇文章信息进行分析，并根据需求获取目标数据。假设需要获取文章的【新闻题目、创建时间、URL、点赞数、收藏数、评论数】，如图3-7所示。



图3-7 分析特征数据

3. 爬取文章数据

（1）获取单篇文章的数据有两种方式，分别是基于xpath和CSS的方法。基于 xpath 的方法操作过程是，使用<http://blog.jobbole.com/114638/>网址打开单篇文章，按F12键查看源代码，比如想获取文章，可以用光标选中题目的区域，并在右侧用鼠标右键单击源码处，再选中Copy，继续单击“Copy Xpath”。此时的路径为：`//*[@id="post-114638"]/div[1]/h1`，如图3-8所示。



图3-8 xpath方法获取数据

（2）在命令行环境下执行以下shell命令：

```
1 cd BoLeSpider
2 scrapy shell http://blog.jobbole.com/114638/
```

(3) 开始对每个特征进行测试，具体特征数据的测试代码如下：

```
1 title = response.xpath('复制选中的 xpath 路径/text()').extract()
```

(4) 完整的特征数据的提取结果如图3-9所示。

```
In [2]: title = response.xpath('//*[@id="post-114638"]/div[1]/h1/text()').extract()
In [3]: title
Out[3]: ['能从远程获得乐趣的 Linux 命令']
In [4]:
```

图3-9 测试特征数据提取文本信息

注 意

有时候按照以上方法操作，但却没有提取到文本信息，则有可能是代码错误，请仔细检查代码。也有可能是反爬虫技术的作用，此时，需要登录后再进行数据爬取，这属于更深层次的爬虫技术，本文不再涉及。

(5) 使用xpath方法获取数据。

以上逐个特征测试无误后，将代码放在Chapter3/BoLeSpider/BoLeSpider/spiders/jobbole.py文件中的parse方法下：


```

1  # 获得单页的信息
2  def parse(self, response):
3      # xpath 获取内容
4      title = response.xpath('//*[@id="post-114638"]/div[1]/h1/
      text()').extract() # 新闻题目
5      create_date = response.xpath('//*[@id="post-114638"]/div[2]/p/
      text()').extract()[0].strip().replace('·', '') # 创建时间
6      url = response.url # url
7      dianzan = self.re_match(response.xpath('//*[@id="post-114638"]/
      div[3]/ div[5]/span[1]/text()').extract()[1]) # 点赞数
8      soucang = self.re_match(response.xpath('//*[@id="post-114638"]/
      div[3]/ div[5]/span[2]/text()').extract()[0]) # 收藏数
9      comment = self.re_match(response.xpath('//*[@id="post-114638"]/
      div[3]/ div[5]/a/span/text()').extract()[0]) # 评论数
10 print('标题:', title, '\n', '发布时间:', create_date, '\n', '文章地址:',
      url, '\n', '点赞数: ', dianzan, '\n', '收藏数', soucang, '\n', '评论数',
      comment)

```

(6) 使用CSS方法获取数据。

```

1  # 获得单页的信息
2  def parse(self, response):
3      # css 获取内容
4      title = response.css('.entry-header h1::text').extract()
      # 新闻题目
5      create_date = response.css ('p.entry-meta-hide-on-
      mobile::text').extract()[0].strip().replace('·', '') # 创建时间
6      url = response.url # url
7      dianzan = self.re_match(response.css('.vote-post-up
      h10::text').extract()[0]) # 点赞数
8      soucang = self.re_match(response.css
      ('.bookmark-btn::text').extract()[0])
      # 收藏数
9      comment = self.re_match(response.css ('a[href=
      "#article-comment"] span::text').extract()[0]) # 评论数
10      print(title, '\n', create_date, '\n', url, '\n', dianzan, '\n',
      soucang, '\n', comment)

```

（ 7 ） 获 取 页 面 信 息 （ 源 代 码 见 ：
Chapter3/BoLeSpider/BoLeSpider/spiders/jobbole.py ） 。

```

1  # -*- coding: utf-8 -*-
2  import scrapy,re
3
4  # 获取单页信息
5  class JobboleSpider(scrapy.Spider):
6      name = 'jobbole'
7      allowed_domains = ['http://www.jobbole.com/']
8      start_urls = ['http://blog.jobbole.com/114638']
9
10     # 获得单页的信息
11     def parse(self, response):
12         # css 获取内容
13         title = response.css('.entry-header h1::text').extract()
14             # 新闻题目
15         create_date = response.css ('p.entry-meta-hide-on-
mobile::text').extract()[0].strip().replace('·','') # 创建时间
16         url = response.url # url
17         dianzan = self.re_match(response.css('.vote-post-up
h10::text').extract()[0]) # 点赞数
18         soucang = self.re_match(response.css
('.bookmark-btn::text').extract()[0]) # 收藏数
19         comment = self.re_match(response.css
('a[href="#article-comment"] span::text').extract()[0]) # 评论数
20
21         print('标题:',title,'\n','发布时间:',create_date,'\n',
22             '文章地址:',url,'\n','点赞数: ',dianzan,'\n','收藏数',
23             soucang,'\n','评论数',comment)
24
25     # 对点赞数、收藏数、评论数等进行正则数字提取
26     def re_match(self,value):
27         match_value = re.match('.*?(\d+).*',value)
28         if match_value:
29             value = int(match_value.group(1))
30         else:
31             value = 0
32         return value

```

(8) 运行main.py函数，获取到所有的信息，如图3-10所示。

```

2019-03-25 11:06:37 [scrapy.extensions.telnet] DEBUG: Telnet console listening on 127.0.0.1:6024
2019-03-25 11:06:37 [scrapy.downloadermiddlewares.redirect] DEBUG: Redirecting (301) to <GET http://blog.jobbole.com/114638/> from <GET http://blog.jobbole.com/114638>
2019-03-25 11:06:37 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://blog.jobbole.com/114638/> (referer: None)
标题: ['能从远程获得乐趣的Linux命令']
发布时间: 2019/01/13
文章地址: http://blog.jobbole.com/114638/
点赞数: 1
收藏数: 3
评论数: 1

```

图3-10 获取单篇文章特征数据

4. 爬取列表页所有文章

(1) 实现列表页所有文章信息的爬取工作

按F12键分析网页，找到下一页并获取所有列表页的文章链接。

执行以下代码获取所有列表页的文章链接，如图3-11所示。

```

1 scrapy shell http://blog.jobbole.com/all-posts/
2 response.css("#archive .floated-thumb .post-thumb a::attr(href)").extract()

```

```

In [1]: response.css("#archive .floated-thumb .post-thumb a::attr(href)").extract()
Out[1]:
['http://blog.jobbole.com/114666/',
'http://blog.jobbole.com/114663/',
'http://blog.jobbole.com/114655/',
'http://blog.jobbole.com/114648/',
'http://blog.jobbole.com/114641/',
'http://blog.jobbole.com/114638/',
'http://blog.jobbole.com/114636/',
'http://blog.jobbole.com/114633/',
'http://blog.jobbole.com/114605/',
'http://blog.jobbole.com/114630/',
'http://blog.jobbole.com/114628/',
'http://blog.jobbole.com/114610/',
'http://blog.jobbole.com/114591/',
'http://blog.jobbole.com/114589/',
'http://blog.jobbole.com/114573/',
'http://blog.jobbole.com/114570/',
'http://blog.jobbole.com/114561/',
'http://blog.jobbole.com/114551/',
'http://blog.jobbole.com/114549/']
In [2]:

```

图3-11 获取列表页所有文章的链接

(2) 设置目标特征的实体类

打开Scrapy框架内置的Chapter3/BoLeSpider/items.py文件，设计爬虫目标特征的实体类（这里可以将爬虫目标特征的实体类作为数据库操作中实体类来理解）。代码如下：

```
1  # -*- coding: utf-8 -*-
2  # Define here the models for your scraped items
3  # See documentation in:
4  # https://doc.scrapy.org/en/latest/topics/items.html
5  import scrapy
6  from scrapy.loader.processors import MapCompose
7
8  class BolespiderItem(scrapy.Item):
9      # define the fields for your item here like:
10     # name = scrapy.Field()
11     pass
12
13 # 设置提取字段的实体类
14 class JobBoleItem(scrapy.Item):
15     title = scrapy.Field()          # 文章题目
16     create_date = scrapy.Field()    # 发布时间
17     url = scrapy.Field()            # 当前文章 url 路径
18     dianzan = scrapy.Field()        # 点赞数
19     soucang = scrapy.Field()        # 收藏数
20     comment = scrapy.Field()        # 评论数
```

（3）修改代码文件jobbole.py

首先将BoLeSpider/spiders/jobbole.py文件的starturls修改为列表页的路径，然后在parse方法中解析文章，并提取下一页交给Scrapy提供下载。parsesdetail方法负责每一篇文章的下载，re_match方法是使用正则表达式对文本信息数据进行处理。完整的代码如下：

```
1 import datetime
2 from scrapy.http import Request
3 from urllib import parse
4 from BoLeSpider.items import JobBoleItem
5
6 class JobboleSpider(scrapy.Spider):
7     name = 'jobbole'
8     allowed_domains = ['http://www.jobbole.com/']
9     # start_urls = ['http://blog.jobbole.com/114638']
10    start_urls = ['http://blog.jobbole.com/all-posts/'] # 所有页信息
11
12    # 获取列表下所有页信息
13    def parse(self, response):
14        # 1. 获取文章列表中具体文章的 url 并交给解析函数进行字段的解析
15        post_urls =
16        response.css("#archive .floated-thumb .post-thumb
17        a::attr(href)").extract()
18
19        for post_url in post_urls:
20            yield Request(url=parse.urljoin(response.url, post_url),
21                callback=self.parse_detail, dont_filter=True) # scrapy 下载
22
23    # 2. 提取下一页并交给 scrapy 下载
```

```

20         next_url = response.css(".next.page-numbers::
attr(href)").extract_first("")
21         if next_url:
22             yield Request(url=parse.urljoin(response.url, post_url),
callback=self.parse, dont_filter=True)
23
24         # scrapy shell http://blog.jobbole.com/114638/
25         def parses_detail(self, response):
26             article_item = JobBoleItem()
27             article_item['title'] = response.css('.entry-header
h1::text').extract()
28             article_item['create_date'] = date_convert(response.css
("p.entry-meta-hide-on-mobile::text").extract()[0].strip().replac
e(".", "").strip())
29             article_item['url'] = response.url
30             article_item['dianzan'] = re_match(response.css
('.vote-post-up h10::text').extract()[0])
31             article_item['soucang'] = re_match(response.css
('.bookmark-btn::text').extract()[0])
32             article_item['comment'] = re_match(response.css
('a[href="#article-comment"] span::text').extract()[0])
33             yield article_item
34
35
36         # *****使用正则表达式对字段进行格式化处理*****
37
38         # 对点赞数、收藏数、评论数等用正则表达式进行数字的提取
39         def re_match(value):
40             match_value = re.match('.*?(\d+).*', value)
41             if match_value:
42                 nums = int(match_value.group(1))
43             else:
44                 nums = 0
45             return nums
46
47
48         # 对时间进行格式化处理
49         def date_convert(value):
50             try:
51                 create_date = datetime.datetime.strptime(value,
"%Y/%m/%d").date()
52             except Exception as e:
53                 create_date = datetime.datetime.now().date()
54             return create_date

```

(4) 运行main.py

提取列表页数据，如图3-12所示。

```
{
  'comment': 0,
  'create_date': datetime.date(2018, 12, 16),
  'dianzan': 1,
  'soucang': 1,
  'title': ['神奇的Linux·命令行字符形状工具·boxes'],
  'url': 'http://blog.jobbole.com/114549/'
}
2019-03-25 11:11:33 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://b
jobbole.com/114591/>
{
  'comment': 0,
  'create_date': datetime.date(2019, 1, 3),
  'dianzan': 1,
  'soucang': 0,
  'title': ['cat·命令的源码进化史'],
  'url': 'http://blog.jobbole.com/114591/'
}
2019-03-25 11:11:33 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://b
jobbole.com/114589/> (referer: http://blog.jobbole.com/all-posts/)
2019-03-25 11:11:33 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://b
jobbole.com/114589/>
{
  'comment': 0,
  'create_date': datetime.date(2018, 12, 29),
  'dianzan': 1,
  'soucang': 1,
  'title': ['救命！我的电子邮件发不到500英里以外！'],
  'url': 'http://blog.jobbole.com/114589/'
}
```

图3-12 获取列表页所有文章的特征数据

3.4 爬取数据以JSON格式进行存储

上一节介绍了如何分析网页数据并爬取数据，得到数据以后如何进行存储呢？本节主要介绍JSON格式数据的本地化存储，具体操作步骤如下。

1. 修改管道文件

使用pipeline.py文件作为管道文件，负责处理Spider中获取到的实体特征信息，并进行存储。这里需要导入JsonItemExporter模块进行JSON操作，然后在JsonExporterPipeline方法中执行具体的写操作，完整的代码如下：


```

1  from scrapy.exporters import JsonItemExporter
2  import codecs
3
4  class BolespiderPipeline(object):
5      def process_item(self, item, spider):
6          return item
7
8
9  # 调用 scrapy 提供的 json export 导出 json 文件
10 class JsonExporterPipleline(object):
11     def __init__(self):
12         self.file = open('articleexport.json', 'wb')
13         self.exporter = JsonItemExporter(self.file,
14 encoding="utf-8", ensure_ascii=False)
15         self.exporter.start_exporting()
16
17     def close_spider(self, spider):
18         self.exporter.finish_exporting()
19         self.file.close()
20
21     def process_item(self, item, spider):
22         self.exporter.export_item(item)
23         return item

```

2. 修改设置文件

在setting.py设置文件中修改方法的执行优先级。数字从小到大，数字越小，优先级就越高，具体设置如下：

```

1  ITEM_PIPELINES = {
2      # 'BoLeSpider.pipelines.BolespiderPipeline': 1,
3      'BoLeSpider.pipelines.JsonExporterPipleline': 1,
4  }

```

3. JSON格式数据的本地化存储

运行main.py文件，实现本地JSON文件存储。执行完成后，打开articleexport.json文件查看结果，如图3-13所示。



```
1 [{"title": ["Linux 搜索文件和文件夹的 4 种简单方法"], "create_date": "2018-12-16", "url": "http://blog.jobbole.com/114561/", "dianzan": 1, "soucang": 0, "comment": 0}, {"title": ["神奇的 Linux 命令行字符形状工具 boxes"], "create_date": "2018-12-16", "url": "http://blog.jobbole.com/114549/", "dianzan": 1, "soucang": 1, "comment": 0}, {"title": ["cat 命令的源码进化史"], "create_date": "2019-01-03", "url": "http://blog.jobbole.com/114591/", "dianzan": 0, "soucang": 0, "comment": 0}, {"title": ["微软变了！招程序员的流程完全改变"], "create_date": "2019-01-05", "url": "http://blog.jobbole.com/114610/", "dianzan": 2, "soucang": 6, "comment": 1}, {"title": ["学会这两件事，Git 老司机"], "create_date": "2018-12-18", "url": "http://blog.jobbole.com/114551/", "dianzan": 1, "soucang": 0, "comment": 0}, {"title": ["计算机科学自学指南"], "create_date": "2018-12-22", "url": "http://blog.jobbole.com/114573/", "dianzan": 2, "soucang": 18, "comment": 0}, {"title": ["救命！我的电子邮件发不到 500 英里以外！"], "create_date": "2018-12-29", "url": "http://blog.jobbole.com/114589/", "dianzan": 1, "soucang": 1, "comment": 0}, {"title": ["在 Linux 上使用 tarball"], "create_date": "2019-01-07", "url": "http://blog.jobbole.com/114628/", "dianzan": 1, "soucang": 1, "comment": 0}, {"title": ["追思杰出的 Lin"]
```

图3-13 JSON格式数据的本地化存储

3.5 爬取数据的MySQL存储

相对于JSON存储，大家更为熟悉数据库存储，本节介绍爬取数据的本地数据库存储方法。本书选用的是通用的MySQL数据库，当然也可以选择Oracle、SQL Server等数据库。

3.5.1 MySQL与Navicat部署

MySQL原本是一个开放源码的关系数据库管理系统，原开发者为瑞典的MySQL AB公司，该公司于2008年被Sun公司收购。2009年，Oracle公司（甲骨文公司）收购了Sun公司，于是MySQL成为Oracle旗下的产品。

MySQL由于性能高、成本低、可靠性好，已经成为最流行的开源数据库，被广泛地应用在Internet上的中小型网站中。随着MySQL的不断成熟，它也逐渐用于更多大规模网站和应用，比如维基百科、Google

和Facebook等网站。非常流行的开源软件组合LAMP中的“M”指的就是MySQL。

Navicat是香港卓软数码科技有限公司生产的一系列MySQL、MariaDB、MongoDB、Oracle、SQLite、PostgreSQL及Microsoft SQL Server的图形化数据库管理及开发软件。它有一个类似浏览器的图形用户界面，支持本地和远端数据库的多重连接。它的设计合乎各种用户的需求，这些用户包括数据库管理员，程序员，各种类型的个人客户，以及与合作伙伴共享信息的不同企业或公司。

由于MySQL与Navicat为大家所熟悉，基本安装和部署有关的资料较多，本书不再赘述。

3.5.2 MySQL存储爬虫数据

1. 数据库表的设计

打开本地MySQL数据库，其中用户名为root，密码为root。成功登录后新建数据库名为test，并在test数据库下新建myarticles数据表。具体设计如表3-2所示。

表3-2 数据表设计

字 段 名	数据类型	是否主键	备 注
Id	int	是	新闻编号
Title	varchar	否	新闻题目
Createdata	datetime	否	创建时间
url	varchar	否	新闻网址
Dianzan	int	否	点赞数
Soucang	int	否	收藏数
Comment	int	否	评论数

数据表的设计如图3-14所示。

栏位	索引	外键	触发器	选项	注释	SQL 预览
名						
id						
title						
createdate						
url						
dianzan						
soucang						
comment						
img_url						
img_path						

图3-14 MySQL数据库表的设计

2. 数据库存储的同步机制

打开pipeline.py修改代码如下，本方法采用的是同步读写机制，即提取文件的同时执行写操作。首先建立本地数据库的连接并把中文编码格式设置为UTF-8，之后执行一般的SQL插入操作。

```

1  # -*- coding: utf-8 -*-
2  from scrapy.exporters import JsonItemExporter
3
4  class BolespiderPipeline(object):
5      def process_item(self, item, spider):
6          return item
7
8  # 将爬取的数据字段存储在 MySQL 数据库中
9  import MySQLdb
10 import MySQLdb.cursors
11
12 class MysqlPipeline(object):
13     #采用同步的机制写入 MySQL
14     def __init__(self):
15         self.conn = MySQLdb.connect('127.0.0.1', 'root', 'root',
16         'test', charset="utf8", use_unicode=True)
17         self.cursor = self.conn.cursor()
18
19     def process_item(self, item, spider):
20         insert_sql = """
21             insert into myarticles(title,
22             createdate,url,dianzan,soucang,comment) VALUES(%s,%s,%s,%s,%s,%s)
23             """
24         self.cursor.execute(insert_sql, (item["title"],
25         item["create_date"], item["url"],
26         item["dianzan"],item["soucang"],item["comment"]))
27         self.conn.commit()

```

3. 数据库存储的异步机制

当爬取海量网络数据的时候，爬取速度与存储速度往往会产生冲突，采用前文介绍的数据库存储技术很可能会造成数据阻塞。基于此，需要改进数据存储方式，即采用异步存储机制。下面来介绍异步存储机制的代码实现。

首先在pipeline.py文件中修改代码如下：

```

1  from twisted.enterprise import adbapi
2
3  class MysqlTwistedPipeline(object):
4      def __init__(self, dbpool):
5          self.dbpool = dbpool
6
7      @classmethod
8      def from_settings(cls, settings): # cls 即 MysqlTwistedPipeline
9          dbparms = dict(
10             host = settings["MYSQL_HOST"],
11             db = settings["MYSQL_DBNAME"],
12             user = settings["MYSQL_USER"],
13             passwd = settings["MYSQL_PASSWORD"],
14             charset='utf8',
15             cursorclass=MySQLdb.cursors.DictCursor,
16             use_unicode=True
17         )
18         dbpool = adbapi.ConnectionPool("MySQLdb", **dbparms)
19         return cls(dbpool)
20
21     def process_item(self, item, spider):
22         #使用 twisted 将 MySQL 插入变成异步执行
23         query = self.dbpool.runInteraction(self.do_insert, item)
24         query.addErrback(self.handle_error, item, spider) #处理异常
25
26     def handle_error(self, failure, item, spider):
27         #处理异步插入的异常
28         print (failure)
29
30     def do_insert(self, cursor, item):
31         insert_sql = """
32             insert into myarticles(title, createdate,url,dianzan,
33             soucang,comment,img_url,img_path) VALUES(%s,%s,%s,%s,%s,%s,%s,%s,%s)
34         """
35         cursor.execute(insert_sql, (item["title"], item["create_date"],
36             item["url"], item["dianzan"], item["soucang"], item["comment"],
37             item["front_image_url"], item["front_image_path"]))

```

其次，在setting.py文件末行添加如下代码，用于实现全局变量的设置。

```
1  # 数据库设置
2  MYSQL_HOST = "127.0.0.1"
3  MYSQL_DBNAME = "test"
4  MYSQL_USER = "root"
5  MYSQL_PASSWORD = "admin"
```

4. 修改设置文件

在setting.py设置文件中修改方法的执行优先级。数字从小到大，数字越小，优先级就越高，具体设置如下：

```
1  ITEM_PIPELINES = {
2      # 'BoLeSpider.pipelines.BolespiderPipeline': 1,
3      # 'BoLeSpider.pipelines.JsonExporterPipeleline': 1,
4      'BoLeSpider.pipelines.MysqlPipeline': 1,
5  }
```

5. 本地化MySQL存储

运行main.py文件，实现本地MySQL数据存储。执行完成后，打开myarticles数据表查看，结果如图3-15所示。

id	title	createdate	url	dianzan	soucang	comment
81	在 Linux 上使用 tarball	2019-01-07	http://blog.jobbole.com/114628/	1	1	0
82	计算机科学自学指南	2018-12-22	http://blog.jobbole.com/114573/	2	16	1
83	微软变了! 招程序员的流	2019-01-05	http://blog.jobbole.com/114610/	2	5	1
84	5 款 Linux 街机游戏	2019-01-11	http://blog.jobbole.com/114636/	1	0	0
85	Linux 搜索文件和文件夹	2018-12-19	http://blog.jobbole.com/114561/	1	0	0
86	救命! 我的电子邮件发不	2018-12-29	http://blog.jobbole.com/114589/	1	1	0
87	追思杰出的 Linux 内核	2019-01-08	http://blog.jobbole.com/114630/	2	1	1
88	cat 命令的源码进化史	2019-01-03	http://blog.jobbole.com/114591/	1	0	0
89	Vim 命令合集	2019-01-18	http://blog.jobbole.com/114641/	1	2	2
90	关于 top 工具的 6 个替	2018-12-11	http://blog.jobbole.com/114546/	1	1	0
91	Python 中星号的本质及	2019-02-16	http://blog.jobbole.com/114655/	1	2	1
92	神奇的 Linux 命令行字	2018-12-16	http://blog.jobbole.com/114549/	1	1	0
93	能从远程获得乐趣的 Linu	2019-01-13	http://blog.jobbole.com/114638/	1	3	1
94	克劳德·香农 (信息论之	2019-02-16	http://blog.jobbole.com/114648/	1	2	1
95	学会这两件事, 让你成为	2018-12-18	http://blog.jobbole.com/114551/	1	0	2
96	5 个好用的开发者 Vim 插	2019-02-24	http://blog.jobbole.com/114666/	1	0	0
97	14 个依然很棒的 Linux A	2019-02-19	http://blog.jobbole.com/114663/	1	1	1
98	在 Linux 命令行上拥有一	2018-12-21	http://blog.jobbole.com/114570/	1	0	0
99	从软件工程的角度解读任	2019-01-08	http://blog.jobbole.com/114605/	3	4	0

图3-15 MySQL存储文章数据

3.6 网络爬虫技术扩展

数据采集是一项庞杂的工作，倘若是文档文件或者数据库文件，采用拷贝和文件导出的方法即可完成。面对海量的非结构化文件，尤其是网络数据不可避免地会选择网络爬虫技术。网络爬虫作为一门单独的学科领域，其涉及的知识非常深，仅仅这一项技术足够一本书去阐述，故而本书只是管中窥豹地介绍了爬虫技术，更多的网络爬虫技术难点包括：

- 实现网站虚拟登录并爬取数据。
- 网站反爬策略。
- 网站模板定期变动。
- 网站URL抓取失败。
- 网站频繁抓取IP被封。

3.7 本章小结

本章介绍了结构化、半结构化和非结构化数据及其数据的采集策略。面对非结构化网页信息，带领读者实现了页面分析和数据爬取，并把抓取的数据进行本地化存储。由于网络爬虫技术内容较多，本书篇幅有限，只是管中窥豹地介绍了网络爬虫技术与方法。下一章介绍文本信息抽取，即对采集的数据（包括DOC、PDF、HTML、Excel等）抽取文本信息。

第4 章 文本信息抽取

通过数据采集获取的数据信息往往五花八门、杂乱无章，因此需要对这些不同类型的数据进行集成，并将集成数据传入到电脑中，然后通过算法模型挖掘其潜在的价值，作为智能应用的支撑。本章介绍文本信息抽取技术，以帮助读者更好地利用爬取的数据。

4.1 文本抽取概述

数据是智能时代的根基，但无论是以数据库文件为代表的结构化数据、网页数据为代表的半结构化数据，还是以图片、音视频为代表的非结构化数据，往往都是五花八门、杂乱无章的，那么，如何对不同类型的数据进行数据集成，将其处理成统一的文档格式输入到算法模型之中，成为数据处理的首要任务。

处理不同格式的文档，通常会采取不同的策略。在处理结构化和半结构化数据时，可以直接提取文本信息，做进一步的数据预处理；而处理非结构化的数据（诸如图片、音视频）时，可采用一定的技术手段，获取其对应的数据特征矩阵。这一点不太容易理解，比如说想解析一张图片的数据，图片是有长、宽组成的，还包括红、蓝、绿三种基本色，假设只关注这5个基本特征，那么就可以采用特征数据点来表示，如表4-1所示。

表4-1 模拟图片特征数据

图片名	长(bit)	宽(bit)	红	绿	蓝
猫 1	12	100	0	0	1
狗 2	101	234	1	1	1
猪 3	202	24	0	1	0

表4-1的数据表示猫1这张图片，长宽位点是（12，100），由纯蓝色构成；狗2这张图片长宽位点是（101，234），由红绿蓝3色构成；猪3这张图片长宽位点是（202，24），由纯绿色构成。这就是非结构数据图片转化为数值型数据的原理。

一般文本抽取形式是，将采集到的不同类型文档统一处理成文本信息或者特征矩阵，得到高质量的数据集，然后将数据集放进算法模型中来挖掘其背后的价值。总之，最终的目的是将数据传入到电脑中，通过算法模型挖掘其潜在的价值，作为智能应用的支撑，这个过程被称为数据挖掘。如图4-1所示给出了一个数据挖掘的流程。



图4-1 无序数据处理示意图

4.2 文本抽取问题

采集的原数据存在数据质量差、文档格式杂、数据表示形式多样化、数据信息错误等诸多问题，单纯考虑文本信息处理工作，就文本信息而言，采集到的文档数据可能是网页、SQL文件、PDF文档、DOC文档等，对这些文本数据集成的思路就是文本信息的提取，然后进行格式化处理，常见的文本抽取方式包括：

- 使用在线格式转换工具转换。
- 使用Office内置格式进行转换。
- 自己开发文本抽取工具进行文本抽取。

如图4-2所示。

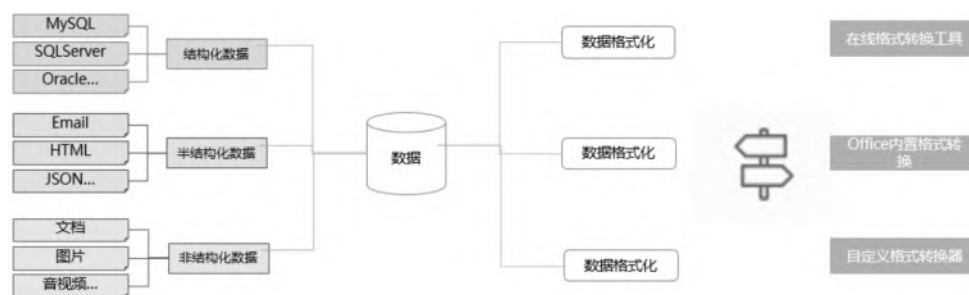


图4-2 常见的文本抽取方式

其中，前两种方法存在以下不足：

- 格式转换后，识别乱码较多。
- 不支持或者限制支持批量处理。
- 批量转化收费问题。
- 格式转换后的txt文件存在编码问题。
- 生成的文件名存在数字乱码。
- 操作不够灵活便捷。

实际工程中期待的目标是：

- 支持PDF/Word等文档自动化文本抽取。
- 提取高质量的文本内容。
- 自动过滤不符合指定格式的文件。
- 生成的目标文件与原文件文件名保持一致。
- 生成文档采用统一的编码格式保存（如：UTF-8）。
- 支持默认保存路径和自定义保存路径。

基于以上目标，下面来介绍文本抽取的具体方法。

4.3 Pywin32抽取文本信息

4.3.1 Pywin32介绍

Pywin32是Python的第三方库文件，它提供了从Python访问Windows API的功能。Windows Pywin32允许开发者像VC一样来使用Python开发win32应用和对Windows系统实现自动化操作，Pywin32核心模块是win32.com。

读者可通过以下3种方法安装Pywin32。

方法一 通过shell命令执行`pip install pypiwin32`。

方法二 下载后缀为exe的文件，然后手动安装。值得注意的是，下载Pywin32版本号要与Python版本号保持一致，其官网下载地址为：<https://github.com/mhammond/pywin32/releases>。对应各版本的插件如图4-3所示。



pywin32-224.win-amd64-py2.7.exe	7.28 MB
pywin32-224.win-amd64-py3.5.exe	9.19 MB
pywin32-224.win-amd64-py3.6.exe	9.19 MB
pywin32-224.win-amd64-py3.7.exe	9.19 MB
pywin32-224.win32-py2.7.exe	6.7 MB
pywin32-224.win32-py3.5.exe	8.4 MB
pywin32-224.win32-py3.6.exe	8.4 MB
pywin32-224.win32-py3.7.exe	8.39 MB
Source code (zip)	
Source code (tar.gz)	

图4-3 不同版本的Pywin32

方法三 在GitHub网站上下载源代码，找到软件下载包Packages，选择对应的版本并下载。

4.3.2 抽取Word文档文本信息

这里我们以抽取Word文档中的文本为例，来介绍抽取文件信息的方法。工作环境为Win10-64bit和Python 3.5，且已成功预安装Pywin32插件。实例文档为本书第1章内容，如图4-4所示。



图4-4 Word文档内容节选

1. 算法思路

本案例的算法思路如下：

- (1) 切分文件上级目录和文件名。
- (2) 修改转化后的文件名。
- (3) 设置保存路径。
- (4) 加载处理应用，将Word转为txt文本。
- (5) 本地化保存文本。

首先输入待处理文档的文件路径，并将文件完整目录切分为上级目录和文件名；根据文件名的后缀判断是否符合要求，符合处理要求后，修改新的文档后缀和保持完整路径。最后，执行应用程序保存抽取的文本信息。具体流程如图4-5所示。

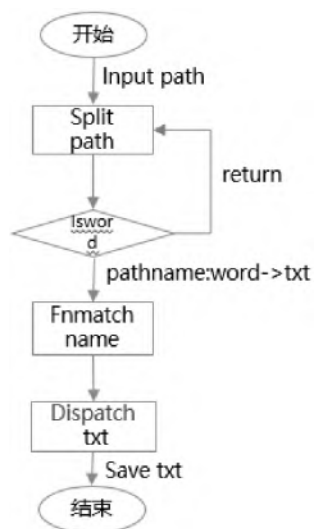


图4-5 抽取Word文档的流程图

2. 代码实现

代码采用utf8编码格式，在导入的模块中使用fnmatch方法检查文档后缀，调用Dispatch方法加载com程序。具体的代码如下（源代码见：Chapter4/word2txt.py）：

```

1  import os,fnmatch
2  from win32com import client as wc
3  from win32com.client import Dispatch
4  def Word2Txt(filePath,savePath=''):
5      # 1. 切分文件上级目录和文件名
6      dirs,filename = os.path.split(filePath)
7      # 2. 修改转化后的文件名
8      new_name = ''
9      if fnmatch.fnmatch(filename,'*.doc'):
10         new_name = filename[:-4]+'.txt'
11     elif fnmatch.fnmatch(filename,'*.docx'):
12         new_name = filename[:-5]+'.txt'
13     else: return
14     print('->',new_name)
15     # 3. 文件转化后的保存路径
16     if savePath=='': savePath = dirs
17     else: savePath = savePath
18     word_to_txt = os.path.join(savePath,new_name)
19     print('->',word_to_txt)
20     # 4. 加载处理应用, word 转为 txt
21     wordapp = wc.Dispatch('Word.Application') # 打开 word 应用程序
22     mytxt = wordapp.Documents.Open(filePath)
23     mytxt.SaveAs(word_to_txt,FileFormat = 4) # 4 表示提取文本
24     mytxt.Close()
25 if __name__=='__main__':
26     filepath = os.path.abspath(r'../Files/wordtotxt/Python 数据预处理.docx')
27     # savepath = '' # 自定义保存路径
28     Word2Txt(filepath)

```

(1) 代码功能描述： 将Word文件转存为txt文件，默认存储在当前路径下，可以指定存储文件的路径。

(2) 参数描述： 其中filePath表示文件路径， savePath表示指定的保存路径。

(3) 结果分析： 首先对文件路径进行处理，即分割成根目录和文件名，其目的是为了实现文件后缀的修改，并设置新的保存路径。然后加载win32.com内置方法Dispatch对文本信息进行提取。

执行以上程序，得到抽取的文本内容，如图4-6所示。

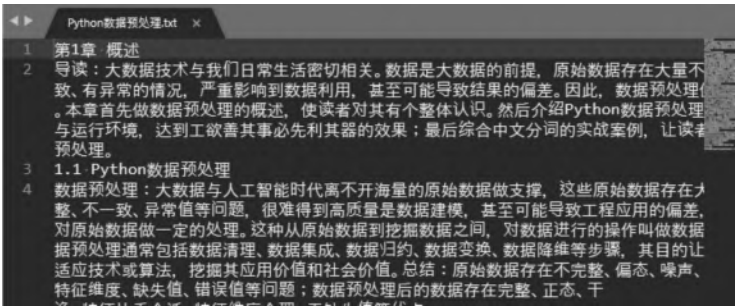


图4-6 Word转为txt文档

4.3.3 抽取PDF文档文本信息

1. 算法思路

本案例的算法思路如下：

- (1) 切分文件上级目录和文件名。
- (2) 修改转化后的文件名。
- (3) 设置保存路径。
- (4) 加载处理应用，将PDF转为txt文本。
- (5) 本地化保存文本。

2. 代码实现

代码实现抽取PDF文件的文本信息，其中参数filePath是文件路径，参数savePath是可选参数，指定保存路径。实现的代码如下（源

代码见：Chapter4/pdf2txt.py）：

```
1  import os,fnmatch
2  from win32com import client as wc
3  from win32com.client import Dispatch
4
5  def Pdf2Txt(filePath,savePath=''):
6      # 1. 切分文件上级目录和文件名
7      dirs,filename = os.path.split(filePath)
8      # print('目录: ',dirs,'\n 文件名: ',filename)
9
10     # 2. 修改转化后的文件名
11     new_name = ""
12     if fnmatch.fnmatch(filename,'*.pdf') or fnmatch.fnmatch
13         (filename,'*.PDF'):
14         new_name = filename[:-4]+'.txt' # 截取".pdf"之前的文件名
15     else: return
16     print('新的文件名: ',new_name)
17
18     # 3. 文件转化后的保存路径
19     if savePath=="": savePath = dirs
20     else: savePath = savePath
21     pdf_to_txt = os.path.join(savePath,new_name)
22     print('保存路径: ',pdf_to_txt)
23
24     # 4. 加载处理应用, pdf 转为 txt
25     wordapp = wc.Dispatch('Word.Application')
26     mytxt = wordapp.Documents.Open(filePath)
27     mytxt.SaveAs(pdf_to_txt,4) # 4 表示提取文本
28     mytxt.Close()
```

4.3.4 打造灵活的文本抽取工具

前面的两个案例是通过文件后缀判断文件类型，进而选择不同的策略进行文本内容抽取。本小节将以上方法封装起来，做一个多文档自适应的文本抽取工具。该工具只需要传入文件路径，就可以自动判

断文档类型并抽取文本信息，最终可以实现任意文档内容的批量抽取。本节只是封装上述两个功能并留下接口，读者可以自动扩展实现Excel、PPT等不同文档内容的抽取。

首先来看看如何自动识别文档类型。

前文讲过，进行文本信息抽取的核心步骤是切分目录、修改后缀名、设置新的保存路径、抽取文本并保存。其中文档识别主要集中在后缀名的判定上，那么，如何实现程序自动识别不同类型的文档呢？实际上只需要将文档识别函数封装即可，具体实现代码如下（完整的代码见Chapter4/ExtractTxt.py）：

```
1 def TranType(filename,typename):
2     # 新的文件名称
3     new_name = ""
4     if typename == '.pdf' : # pdf->txt
5         if fnmatch.fnmatch(filename,'*.pdf') :
6             new_name = filename[:-4]+'.txt' # 截取".pdf"之前的文件名
7         else: return
8     elif typename == '.doc' or typename == '.docx' : # word->txt
9         if fnmatch.fnmatch(filename, '*.doc') :
10             new_name = filename[:-4]+'.txt'
11         elif fnmatch.fnmatch(filename, '*.docx'):
12             new_name = filename[:-5]+'.txt'
13         else: return
14     else:
15         print('警告: \n 您输入[' ,typename,']不合法，本工具支持 pdf/doc/docx 格式,请输入正确格式。')
16         return
17     return new_name
```

(1) 代码功能描述： 根据文件后缀修改文件名。

(2) 参数描述： 其中filePath表示文件路径，typename表示文件后缀。

(3) 返回数据: new_name用于返回修改后的文件名。

```
1 def Files2Txt(filePath,savePath=''):
2     ...
3     # 获取后缀
4     typename = os.path.splitext(filename)[-1].lower()
5     new_name = TranType(filename,typename)
6     print('新的文件名: ',new_name)
7     ...
```

这样,在主函数中只需要输入文件名,抽取工具就会实现自动识别、文本抽取、自定义保存等一系列工作。具体代码如下:

```
1 if __name__ == '__main__':
2     filePath1 = os.path.abspath(r'../Files/wordtotxt/Python 数据预处
    理.docx')
3     filePath2 = os.path.abspath(r'../Files/pdftotxt/Python 数据预处
    理.pdf')
4     Files2Txt(filePath1)
```

注 意

本节对文本信息抽取是在Windows环境下,在Linux和Mac系统下并不适,读者在Linux环境下可以使用Apache Tika toolkit工具。

(参考: <https://www.cnblogs.com/baiboy/p/tika.html>)

4.4 文本批量编码

文本抽取大多数时候是按照指定UTF-8编码格式,但有时候网络下载的文档存在编码不一致的问题,甚至不知道哪些是GB2312,哪些是UTF-8,倘若文档较少还能逐一筛选,如果是百万级甚至更多的文本数量通过人工处理显然是不现实的。本节介绍一款UTF-8批量转换器,利

用该转换器可批量转换文件的编码，读者可以下载源码并在Packages文件夹下找到该转换器。具体操作如下：

- (1) 找到需要进行编码转换的文件夹，如图4-7所示。
- (2) 在右侧区域选中需要转换的文件，然后单击鼠标右键或者按F9键，保存到指定文件夹下即可。

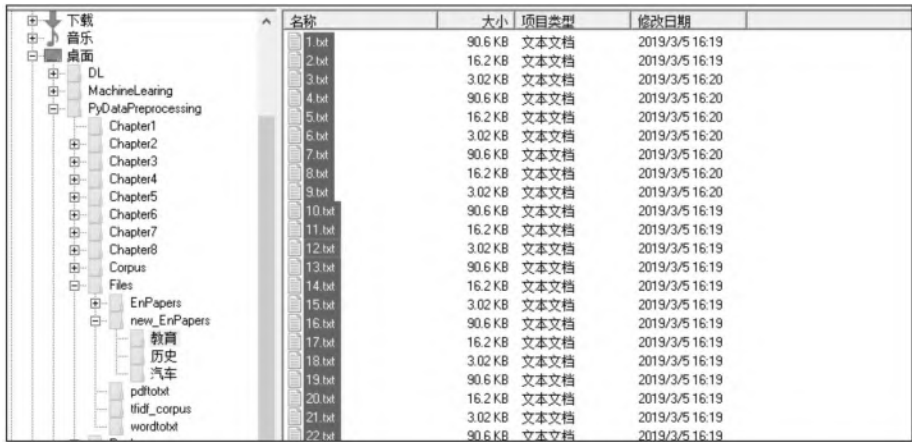


图4-7 UTF-8批量转换器

4.5 实战案例：遍历文件批量抽取新闻文本内容

本节实现新闻文本的批量抽取——对历史、教育、汽车各30篇的PDF文档进行文本内容的抽取（本语料来源于复旦大学新闻语料库）。结合4.3节介绍的文本抽取方法，实现根目录下文档的批量抽取，并自动保存到指定位置。

4.5.1 递归读取文件

文件的读写操作对于处理文本来说是基础工作，如何对根目录下的所有文件读取呢？递归遍历是一种策略，首先通过传入根目录来判断是否是文件夹，然后判断文件夹下是否有二级目录。如果不存在二

级目录，就打印出文件名；如果存在二级目录，继续做递归处理，直到遍历所有文件为止。实现代码如下（源代码见：Chapter4/TraverFiles.py）：

```
1  '''
2  功能描述：遍历目录处理子文件
3  参数描述： 1 rootDir 目标文件的根目录
4  '''
5  class TraversalFun():
6      # 1. 初始化
7      def __init__(self, rootDir):
8          self.rootDir = rootDir # 目录路径
9
10     # 2. 遍历目录文件
11     def TraversalDir(self):
12         TraversalFun.AllFiles(self, self.rootDir)
13
14     # 3. 递归遍历所有文件，并提供具体文件操作功能
15     def AllFiles(self, rootDir):
16         # 返回指定目录包含的文件或文件夹的名字列表
17         for lists in os.listdir(rootDir):
18             # 待处理文件夹名字集合
19             path = os.path.join(rootDir, lists)
20             # 核心算法，对文件具体操作
21             if os.path.isfile(path):
22                 print(os.path.abspath(path))
23             # 递归遍历文件目录
24             elif os.path.isdir(path):
25                 TraversalFun.AllFiles(self, path)
```

代码说明：

该代码文件主要遍历目录处理子文件，其中参数rootDir是目标文件的根目录。

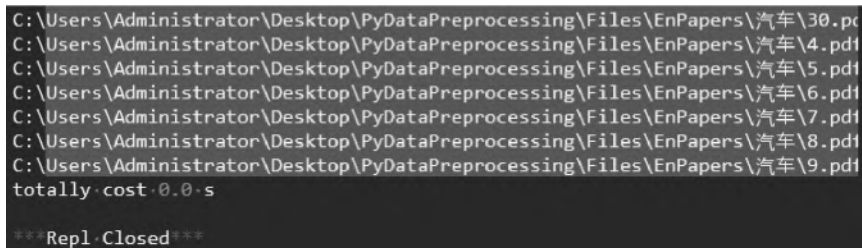
在main函数里面调用方法，代码如下：

```

1  if __name__ == '__main__':
2      time_start=time.time()
3
4      # 根目录文件路径
5      rootDir = r"../Files/EnPapers"
6      tra=TraversalFun(rootDir) # 默认方法参数打印所有文件路径
7      tra.TraversalDir()        # 遍历文件并进行相关操作
8
9      time_end=time.time()
10     print('totally cost',time_end-time_start,'s')

```

运行结果如图4-8所示。



```

C:\Users\Administrator\Desktop\PyDataPreprocessing\Files\EnPapers\汽车\30.pdf
C:\Users\Administrator\Desktop\PyDataPreprocessing\Files\EnPapers\汽车\4.pdf
C:\Users\Administrator\Desktop\PyDataPreprocessing\Files\EnPapers\汽车\5.pdf
C:\Users\Administrator\Desktop\PyDataPreprocessing\Files\EnPapers\汽车\6.pdf
C:\Users\Administrator\Desktop\PyDataPreprocessing\Files\EnPapers\汽车\7.pdf
C:\Users\Administrator\Desktop\PyDataPreprocessing\Files\EnPapers\汽车\8.pdf
C:\Users\Administrator\Desktop\PyDataPreprocessing\Files\EnPapers\汽车\9.pdf
totally cost 0.0 s
***Repl Closed***

```

图4-8 遍历所有文件名

注 意

以上实现的递归方法是一种常规的文件遍历方法，本书后续会介绍一种更高效的迭代方法。

4.5.2 遍历抽取新闻文本

1. 初始化方法参数

本节直接调用4.3.4节介绍的Files2Txt方法。执行以下命令调用自己封装的模块（源代码见：Chapter4/ConvFormat.py）：

```

1  import ExtractTxt as ET

```

其中ExtractTxt就是前面的ExtractTxt.py文件的名称，ET是模块的简写名，这样就可以调用其下所有的方法了。

接下来对文本批量抽取类进行初始化修改，在4.3.4节的基础上对代码作如下修改（添加一个保存目录的参数saveDir）：

```
1  # 1. 初始化
2  def __init__(self, rootDir, func=None, saveDir=""):
3      self.rootDir = rootDir # 目录路径
4      self.func = func      # 参数方法
5      self.saveDir = saveDir # 保存路径
```

代码中有3个参数，第一个参数rootDir，必选参数，是待处理文件的根目录；第二个参数func是可选参数，也是方法参数，这里即为传入的ET.Files2Txt方法（一个用来抽取文本内容的核心方法）；第三个参数saveDir也是可选参数，文档抽取后的保存根目录，默认为空即保存在当前目录下，支持用户自定义路径。

2. 遍历目录文件

在遍历文件的时候，上一节传入了一个参数即根目录，这里需要传入两个参数，其中多出来的一个参数用来保存根目录：

- rootDir：原始文本的根目录。
- save_dir：遍历后文本保存的根目录。

接下来根据前面的知识，对代码做如下修改：


```
1  # 2. 遍历目录文件
2  def TraversalDir(self):
3      # 切分文件上级目录和文件名
4      dirs,latername = os.path.split(self.rootDir)
5      # print(rootDir,'\n',dirs,'\n',latername)
6
7      # 保存目录
8      save_dir = ""
9      if self.saveDir=="": # 默认文件保存路径
10         save_dir = os.path.abspath(os.path.join(dirs, 'new_'+latername))
11     else: save_dir = self.saveDir
12
13     # 创建目录文件
14     if not os.path.exists(save_dir): os.makedirs(save_dir)
15     print("保存目录: \n"+save_dir)
16
17     # 遍历文件并将其转化txt文件
18     TraversalFun.AllFiles(self,self.rootDir,save_dir)
```

3. 遍历目录子文件

在子文件操作过程中有两点变化，第一，判断是否为目标文件，如果是目标文件，则传入文本抽取方法func进行处理；第二，如果是文件夹，则对保存子目录进行修改。具体代码修改如下：

```

1  # 3. 递归遍历所有文件，并提供具体文件的操作功能
2  def AllFiles(self,rootDir,save_dir=''):
3      # 返回指定目录包含的文件或文件夹的名字列表
4      for lists in os.listdir(rootDir):
5          # 待处理文件夹名字的集合
6          path = os.path.join(rootDir, lists)
7
8          # 核心算法，对文件具体操作
9          if os.path.isfile(path):
10             self.func(os.path.abspath(path),os.path.abspath(save_dir))
11
12         # 递归遍历文件目录
13         if os.path.isdir(path):
14             newpath = os.path.join(save_dir, lists)
15             if not os.path.exists(newpath):
16                 os.mkdir(newpath)
17             TraversalFun.AllFiles(self,path,newpath)

```

4. 批量抽取文本信息

最后，在main函数里面运行根目录并抽取所有文本信息，代码如下：

```

1  if __name__ == '__main__':
2      time_start=time.time()
3      # 根目录文件路径
4      rootDir = r"../Files/EnPapers"
5      # saveDir = r"../Corpus/TxtEnPapers"
6      tra=TraversalFun(rootDir,ET.Files2Txt)
7                                     # 默认方法参数打印所有文件路径
8      tra.TraversalDir()
9      time_end=time.time()
10     print('totally cost',time_end-time_start,'s')

```

批量文档信息抽取的运行结果，如图4-9所示。

至此，自定义文本信息批量抽取工作全部完成，90篇文档抽取用时275秒左右。其时间分布主要为两点，一是调用API打开提取文档文本信息所用的时间；二是遍历读取文件所用的时间。针对遍历文件读取，递归方法随着数据量增加将极大地影响运行时间，后面章节会基于此改进一种迭代方法的文件遍历算法。

```
保存路径： C:\Users\Administrator\Desktop\PyDataPreprocessing\Files\new_EnPapers
\汽车\6.txt
保存路径： C:\Users\Administrator\Desktop\PyDataPreprocessing\Files\new_EnPapers
\汽车\7.txt
保存路径： C:\Users\Administrator\Desktop\PyDataPreprocessing\Files\new_EnPapers
\汽车\8.txt
保存路径： C:\Users\Administrator\Desktop\PyDataPreprocessing\Files\new_EnPapers
\汽车\9.txt
totally cost 6250.105193853378 s

***Repl Closed***
```

图4-9 批量抽取所有文本信息

4.6 本章小结

本章介绍了数据采集和存储及对不同形式数据特征的处理策略。针对文本信息抽取问题，实战打造了一款适合多格式批量处理的文本抽取工具；然后通过一个综合实战案例来遍历文件并批量抽取新闻文本内容，该遍历方法封装完成后可以作为工具来使用。下一章介绍在完成数据集成和文本信息的抽取以后，如何高效遍历文件，如何进行数据清洗以及数据清洗包括哪些步骤等内容。

第5 章

文本数据清洗

数据清洗是指删除、更正错误数据，处理不完整、格式有误或多余的数据，解决来自各个信息系统不同数据间一致性的问题。本章介绍以正则方法来清洗文本数据、处理网页数据和处理简繁中文问题，利用高效读取文件的方法来完成批量文本的清洗工作。

5.1 新闻语料的准备

语料可以理解为语言材料，包括口语材料和书面材料。语料的定义较为广泛，其来源可能是教材、报纸、综合刊物、新闻材料、图书等，语料所涉及的学科门类也较为复杂。本章所介绍的新闻语料，狭义上来讲，就是为实验或工程应用所准备的相对规范的数据集。其目的是通过监督式学习方法，训练算法模型以达到工程应用的目的。本书新闻语料来源于复旦大学新闻语料摘选，原始语料有近千万条，为了适应讲解，笔者选择了平衡语料30余万条，具体语料信息如表5-1所示。

表5-1 新闻语料

编 号	新闻类别	新闻数量	语料大小
1	财经	37098	117MB
2	教育	41936	141MB
3	科技	65534	122MB
4	时政	63086	118MB
5	体育	65534	191MB
6	娱乐	65534	136MB

为什么不是自己构建语料而采用开源数据集？3.3节介绍的数据采集和爬虫技术，可以完成对某特定领域数据的爬取和整理，结合数据预处理技术，最终整理成相对规范的文本信息，理论上讲完全是可行的。但是，就实际情况而言，语料库构建需要遵循以下几个原则：

- 代表性。在应用领域中，不是根据量来划分是否是语料库，而是语料库需在一定的抽样框架范围内采集而来，并且在特定的抽样框架内做到代表性和普遍性。
- 结构性。有目的地收集语料的集合，必须以电子形式存在，计算机可读的语料集合结构性体现在语料库中语料记录的代码，元数据项、数据类型、数据宽度、取值范围、完整性约束。
- 平衡性。主要体现在平衡因子，包括学科、年代、文体、地域、登载语料的媒体、使用者的年龄、性别、文化背景、阅历、预料用途（私信 / 广告等），根据实际情况选择其中一个或者几个重要的指标作为平衡因子，最常见的平衡因子有学科、年代、文体、地域等。
- 规模性。大规模的语料对语言研究特别是对自然语言研究处理很有用，但是随着语料库的增大，垃圾语料越来越多。但语料达到一定规模以后，语料库的功能并不能随之增长，因此，语料库规模应根据实际情况而定。
- 元数据。元数据对于研究语料库有着重要的意义，我们可以通过元数据了解语料的时间、地域、作者、文本信息等，还可以构建不同的子语料库。除此之外，还可以对不同的子语料对比，另外还可以记录语料知识版权、加工信息、管理信息等。

此外，一个高效的数据语料本身就是有经济价值的，且构造烦琐、费时费力，这也是本文选择如上开源语料库的原因所在。当然，针对一些特定的商业需求，并没有那么幸运能够找到开源语料，这个时候就需要自己构建，可采用前面章节介绍的方法去实现。

5.2 高效读取文件

在4.5.1节介绍过一种递归的方法——遍历读取文件，该方法效率较低。本节介绍一种基于生成器方法的文件遍历。开始本节之前，先回顾一下4.5.1节的递归遍历方法，然后介绍yield生成器读取文件。

5.2.1 递归遍历读取新闻

递归在计算机科学中是指一种通过重复将问题分解为同类的子问题而解决问题的方法。递归式方法可以被用于解决很多的计算机科学问题，因此它是计算机科学中一个十分重要的概念。计算机科学家尼克劳斯·维尔特如此描述递归：递归的强大之处在于它允许用户用有限的语句描述无限的对象。因此，在计算机科学中，递归可以被用来描述无限步的运算，尽管描述运算的程序是有限的。

事实上，递归算法的核心思想就是分而治之，在一些适用场景下可以让人惊讶不已。这里通过一个实验即递归算法遍历读取CSCMNews文件夹下的30余万新闻语料，来介绍递归在遍历文件中的应用，该例每读取5000条信息在屏幕上打印一条读取完成的信息。代码实现如下（源代码见：Chapter5/FileRead.py）：

```

1  # 遍历 CSCMNews 目录文件
2  def TraversalDir(rootDir):
3      # 返回指定目录包含的文件或文件夹的名字列表
4      for i,lists in enumerate(os.listdir(rootDir)):
5          # 待处理文件夹名字的集合
6          path = os.path.join(rootDir, lists)
7          # 核心算法, 对文件具体操作
8          if os.path.isfile(path):
9              if i%5000 == 0:
10                 print('{t} *** {i} \t docs has been read'.format(i=i,
11                                                                    t=time.strftime('%Y-%m-%d %H:%M:%S',
12                                                                    time.localtime()))
11             # 递归遍历文件目录
12             if os.path.isdir(path):
13                 TraversalDir(path)

```

运行main主函数来读取目录文件并验证所花费的时间。

```

1  if __name__ == '__main__':
2      t1=time.time()
3      # 根目录文件路径
4      rootDir = r"../Corpus/CSCMNews"
5      TraversalDir(rootDir)
6
7      t2=time.time()
8      print('totally cost %.2f' % (t2-t1)+' s')

```

实验证明, 完成约30万新闻文本读取需花费65.28秒(这里还没有对文件执行任何操作), 但随着语料数量的增加, 执行速度会越来越慢, 因此递归并不适合遍历读取大量文件的应用场景。递归遍历读取新闻语料的结果如图5-1所示。

```
2019-03-25-14:10:42-***-5000- docs has been read
2019-03-25-14:10:43-***-10000- docs has been read
2019-03-25-14:10:44-***-15000- docs has been read
2019-03-25-14:10:45-***-20000- docs has been read
2019-03-25-14:10:46-***-25000- docs has been read
2019-03-25-14:10:47-***-30000- docs has been read
2019-03-25-14:10:47-***-35000- docs has been read
totally cost 62.07 s
***Repl Closed***
```

图5-1 递归遍历文件

5.2.2 yield生成器

1. 生成器

生成器是计算机科学中特殊的子程序。实际上，所有生成器都是迭代器。生成器非常类似于返回数组的函数，都具有参数、可被调用、产生一系列的值。但是，生成器不是构造出数组包含所有的值并一次性返回，而是每次产生一个值，因此生成器看起来像函数，但行为像迭代器。

2. yield生成器

yield是一个类似return的关键字，只是这个函数返回的是个生成器。可能读者还是不太理解，没有关系，通过生成斐波那契数列就会理解了。斐波那契（Fibonacci）数列是一个非常简单的递归数列，这个数列除了第一项和第二项外，数列中的任意一项都可由前面两项相加而来，示例如下：

- 形式化表示：1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144……
- 数学化表示： $F(0)=1$, $F(1)=1$, $F(n)=F(n-1)+F(n-2)$ ($n \geq 2$, $n \in \mathbb{N}^*$)

实现斐波那契数列的前10万项，代码的实现如下（源代码见：Chapter5/genyield.py）：

```
1  # 普通斐波那契数列
2  def fab1(max):
3      n, a, b = 0, 0, 1
4      while n < max:
5          a, b = b, a + b
6          n = n + 1
7  # 最大迭代次数
8  maxnum = 100000
9  t1 = time.time()
10 fab1(maxnum)
11 t2 = time.time()
12 print('fab1 total tims %.2f ' % (1000*(t2-t1)) + ' ms')
```

执行程序结果如下：

```
fab1 total tims 135.92  ms
```

接下来，使用yield生成器实现斐波那契数列的前10万项：

```
1  # 生成器：斐波那契数列
2  def fab2(max):
3      n, a, b = 0, 0, 1
4      while n < max:
5          yield b      # 使用 yield
6          a, b = b, a + b
7          n = n + 1
8
9  # 最大迭代次数
10 maxnum = 100000
11 t1 = time.time()
12 fab2(maxnum)
13 t2 = time.time()
14 print('fab2 total tims %.2f ' % (1000*(t2-t1)) + ' ms')
```

执行程序结果如下：

```
fab2 total tims 1.00 ms
```

通过上面两个实验对比发现，fab2方法更加高效，代码简洁明了。可见，把一个函数改为生成器函数就获得了迭代能力，只需要添加关键字yield即可。另外yield还有以下几个特性：

- yield数据存储在非内存中，而数组、列表、字符串、文件等数据的缺点是所有数据都在内存中，海量的数据会消耗大量内存。
- yield生成器是可以迭代的，工作原理就是重复调用next()方法，直到捕获一个异常。
- 有yield的函数不再是一个普通的函数，而是一个生成器（Generator），可用于迭代。
- yield是一个类似return的关键字。

5.2.3 高效遍历读取新闻

yield生成器可以大大提升执行效率，如果是读取文件的操作，只需要构建一个类文件就可以，实现的代码如下（源代码见：Chapter5/EfficRead.py）：

```

1  # 加载目录文件
2  class loadFiles(object):
3      def __init__(self, par_path):
4          self.par_path = par_path
5      def __iter__(self):
6          folders = loadFolders(self.par_path)
7          # level directory
8          for folder in folders:
9              catg = folder.split(os.sep)[-1]
10             #secondary directory
11             for file in os.listdir(folder):
12                 yield catg, file
13
14  # 加载目录下的子文件
15  class loadFolders(object): # 迭代器
16      def __init__(self, par_path):
17          self.par_path = par_path
18      def __iter__(self):
19          for file in os.listdir(self.par_path):
20              file_abspath = os.path.join(self.par_path, file)
21              # if file is a folder
22              if os.path.isdir(file_abspath):
23                  yield file_abspath

```

其中，loadFiles类负责加载目录文件，而loadFolders类负责加载文件夹下的子文件。

上述代码最大的变化就是return关键字改为yield，如此便成了生成器函数，通过yield生成器函数遍历30余万新闻文件，每完成5000个文件读取便打印一条信息，调用main函数如下：

```

1  if __name__=='__main__':
2      start = time.time()
3      filepath = os.path.abspath(r'../Corpus/CSCMNews')
4      files = loadFiles(filepath)
5      for i, msg in enumerate(files):
6          if i%5000 == 0:
7              print('{t} *** {i} \t docs has been Read'.format(i=i,
              t=time.strftime('%Y-%m-%d %H:%M:%S',time.localtime()))))
8          end = time.time()
9      print('total spent times:%.2f' % (end-start)+ ' s')

```

执行以上函数，运行结果如图5-2所示。

```

2019-03-25-14:22:12-***-300000--docs has been Read
2019-03-25-14:22:12-***-305000--docs has been Read
2019-03-25-14:22:12-***-310000--docs has been Read
2019-03-25-14:22:12-***-315000--docs has been Read
2019-03-25-14:22:12-***-320000--docs has been Read
2019-03-25-14:22:12-***-325000--docs has been Read
2019-03-25-14:22:12-***-330000--docs has been Read
2019-03-25-14:22:12-***-335000--docs has been Read
total spent times:0.71 s
***Repl Closed***

```

图5-2 yield生成器遍历文件的结果

递归遍历读取30万新闻文件耗时65.28秒，而yield生成器仅仅耗时0.71秒，可见，yield生成器优势巨大，随着对文件操作和数据量的增加，这种区别甚至可以达到指数级。读者可以将本节封装的yield生成器类文件保留下来，应用到其他文件操作之中。

5.3 通过正则表达式来清洗文本数据

5.3.1 正则表达式

在很多文本编辑器里，正则表达式（代码中常简写为regex、regexp或RE）通常被用来检索、替换那些匹配某个模式的文本。许多程序设计语言都支持利用正则表达式进行字符串操作。

正则表达式是一种用来匹配字符串的强有力的武器，它的设计思想是用一种描述性的语言来给字符串定义一个规则，凡是符合规则的字符串，就认为它“匹配”了，否则，该字符串就是“不匹配”的。

正则表达式的基本语法如表5-2所示。

表5-2 正则语法

单 字 符	匹配数量	匹配位置
\d 匹配数字	*: 0 个或者更多	^: 一行的开头
\w 匹配(数字、字母)	+: 1 个或更多，至少 1 个	\$: 一行的结尾
\W 匹配非(数字、字母)	?: 0 个或 1 个	\b: 单词结尾
\s 匹配(空格、tab 等)	{min, max}: 出现次数在一个范围内	
\S 匹配非(空格、tab 等)	{n}: 匹配出现 n 次的	
. 匹配任何的字符	[]: 满足任意一个都可以	

下面看几个关于正则表达式的应用示例（源代码见：Chapter5/regular.py）。

1. 提取“0”结束的字符串

```
1 line = 'this is a dome about this scrapy2.0'
2 regex_str='^t.*0$'
3 match_obj = re.match(regex_str,line)
4 if match_obj:
5     print(match_obj.group(1))
```

2. 提取指定字符“t”与“t”之间的子串

```
1 line = 'this is a dome about this scrapy2.0'
2 regex_str=".*?(t.*?t).*" # 提取 tt 之间的子串
3 match_obj = re.match(regex_str,line)
4 if match_obj:
5     print(match_obj.group(0))
```

3. 提取“课程”前面的内容

```
1 line = '这是 Scrapy 学习课程,这次课程很好'
2 regex_str=".*?([\u4E00-\u9FA5]+课程)"
3 match_obj = re.match(regex_str,line)
4 if match_obj:
5     print(match_obj.group(0))
```

4. 提取日期内容

```
1 line = 'xxx 出生于 1989 年'
2 regex_str=".*?(\d+)年"
3 match_obj = re.match(regex_str,line)
4 if match_obj:
5     print(match_obj.group(0))
```

5. 提取不同格式的出生日期

```
1 line = '张三出生于 1990 年 10 月 1 日'
2 line = '李四出生于 1990-10-1'
3 line = '王五出生于 1990-10-01'
4 line = '孙六出生于 1990/10/1'
5 line = '张七出生于 1990-10'
6
7 regex_str='.*出生于(\d{4}[年/-]\d{1,2}([月/-]\d{1,2}|[月/-]$$))'
8 match_obj = re.match(regex_str,line)
9 if match_obj:
10     print(match_obj.group(1))
```

5.3.2 清洗文本数据

使用正则表达式处理文本数据，可以剔除“脏”数据和对满足指定条件的数据进行筛选。这里选用一篇体育新闻对其文本信息进行清洗，原始新闻文本节选如下：

马晓旭意外受伤让国奥警惕 无奈大雨格外青睐殷家军 记者傅亚雨沈阳报道来到沈阳，国奥队依然没有摆脱雨水的困扰。7月31日下午6点，国奥队的日常训练再度受到大雨的干扰，无奈之下队员们只慢跑了25分钟就草草收场。31日上午10点，国奥队在奥体中心外场训练的时候，天就是阴沉沉的，气象预报显示当天下午沈阳就有大雨，但幸好队伍上午的训练并没有受到任何干扰。下午6点，当球队抵达训练场时，大雨已经下了几个小时，而且丝毫没有停下来的意思。抱着试一试的态度，球队开始了当天下午的例行训练，25分钟过去了，天气没有任何转好的迹象，为了保护球员们，国奥队决定中止当天的训练，全队立即返回酒店。

首先读取文本信息，编写如下代码（源代码见：Chapter5/REdealText.py）：

```
1  # 读取文本信息
2  def readFile(path):
3      str_doc = ""
4      with open(path, 'r', encoding='utf-8') as f:
5          str_doc = f.read()
6      return str_doc
7
8  # 1 读取文本
9  path= r'../Corpus/CSCMNews/体育/0.txt'
10 str_doc = readFile(path)
11 print(str_doc)
```

假设需要清除文本中的特殊符号、标点、英文、数字等，仅只保留汉字信息，同时去除换行符，并将多个空格转变成一个空格，具体代码实现如下：

```

1 def textParse(str_doc):
2     # 通过正则表达式过滤掉特殊符号、标点、英文、数字等。
3     r1 = '[a-zA-Z0-9\'!\"#$%&\'()*+,-./:~;|<=>?@, -.?★、...【】《》?'
4     # 去除换行符
5     str_doc=re.sub(r1, ' ', str_doc)
6     # 多个空格成1个
7     str_doc=re.sub(r2, ' ', str_doc)
8     return str_doc
9
10 # 正则清洗字符串
11 word_list=textParse(str_doc)
12 print(word_list)

```

执行上述代码，得到文本信息清洗后的结果，如图5-3所示。

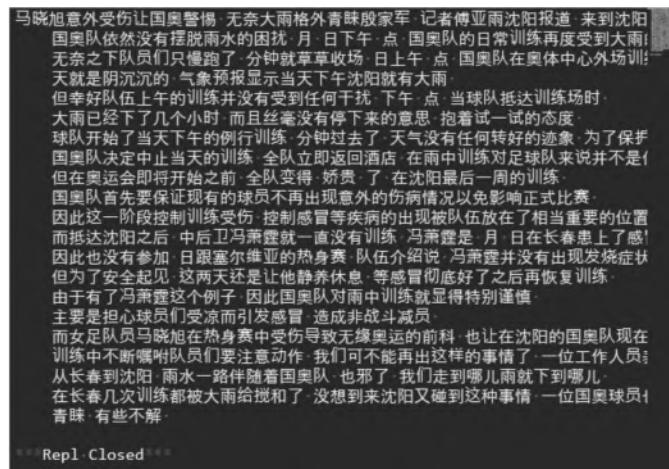


图5-3 通过正则表达式来清洗文本信息

以上实验，只是简单地使用了正则表达式方法来处理文本信息，正则表达式的具体使用情况也要视需要处理的文本信息来定。

5.4 清洗HTML网页数据

有时候面对的不一定是纯文本信息，也有可能是网页数据，或者是微博数据，此时，如何清洗这些半结构化的数据呢？

设想现在有这样一个需求，任务是做信息抽取，然后构建足球球员技能数据库。我们首先想到的是一些足球网站，然后编写爬虫代码去爬取与足球相关的新闻，并对这些网页信息本地化存储，如图5-4所示是与足球相关的新闻网页。

```
<!DOCTYPE html>
<html>
<head>
.....<meta charset="utf-8">
.....<meta http-equiv="X-UA-Compatible" content="IE=edge">
.....<meta name="baidu_union_verify" content="
e375dfeaf5937c9114d7fea78c91675">
.....<meta name="keywords" content="
懂球帝|足球新闻|足球App|足球资讯|足球直播|足球视频|足球比分|中国足球"/>
.....<title>上港5-4恒大5分领跑剑指冠军，下轮打平便可夺冠，武
磊平纪录-广州恒大淘宝|上海上港|蔡慧康|武磊|胡尔克|张成
林|阿兰|保利尼奥|王荣超|吕文君|懂球帝</title>
.....<script src="https://static1.dongqiudi.com/web-new/static/js/
jQuery.1.11.min.js"></script>
.....<link rel="shortcut icon" href="https://static1.dongqiudi.com/web-new/
web/images/fav.ico?20170706" type="image/x-icon">
.....<link rel="stylesheet" href="https://static1.dongqiudi.com/
web-new/web/css/style.css?2018102201">
.....<style>
</head>
```

图5-4 爬取足球相关网页新闻

乍一看非常头疼，如何抽取这里的文本信息呢？一篇一篇地手工处理显然不现实，如果采用上面正则表达式的方法又会出现各种形式的干扰数据。这里介绍一种网页数据通用的正则处理方法。以下是具体的实现代码（源代码见：Chapter5/DealHtml.py）：

```

1  # 清洗 HTML 标签文本
2  # @param htmlstr HTML 字符串.
3  def filter_tags(htmlstr):
4      # 过滤 DOCTYPE
5      htmlstr = ' '.join(htmlstr.split()) # 去掉多余的空格
6      re_doctype = re.compile(r'<!DOCTYPE .*?> ', re.S)
7      s = re_doctype.sub('',htmlstr)
8      # 过滤 CDATA
9      re_cdata = re.compile('<![CDATA\[ [ ]*> //\] > ', re.I)
10     s = re_cdata.sub('', s)
11     # Script
12     re_script = re.compile('<\s*script[>]*>[<]*<\s*/\s*script\s*>',
re.I)
13     s = re_script.sub('', s) # 去掉 SCRIPT
14     # style
15     re_style = re.compile('<\s*style[>]*>[<]*<\s*/\s*style\s*>',
re.I)
16     s = re_style.sub('', s) # 去掉 style
17     # 处理换行
18     re_br = re.compile('<br\s*?/?>')
19     s = re_br.sub('', s) # 将 br 转换为换行
20     # HTML 标签
21     re_h = re.compile('</?\w+[>]*>')
22     s = re_h.sub('', s) # 去掉 HTML 标签
23     # HTML 注释
24     re_comment = re.compile('<!--[>]*-->')
25     s = re_comment.sub('', s)
26     # 多余的空行
27     blank_line = re.compile('\n+')
28     s = blank_line.sub('', s)
29     # 剔除超链接
30     http_link = re.compile(r'(http://.+\.html)')
31     s = http_link.sub('', s)
32     return s
33
34 # 通过正则表达式来处理 HTML 网页数据
35 s=filter_tags(str_doc)
36 print(s)

```

执行上述代码，得到如图5-5所示的结果。

```
上港5-4恒大5分领跑剑指冠军，下轮打平便可夺冠，武磊平纪录-广州恒大淘宝 | 上海  
武磊 胡尔克 张成林 阿兰 保利尼奥 王燊超 吕文君 恒大帝北京时间11月3日19  
中超第28轮迎来天王山之战，广州恒大淘宝坐镇主场迎战上海上港。  
上半场吕文君和蔡慧康先后进球两度为上港取得领先，保利尼奥和阿兰两度为恒  
，补时阶段保利尼奥进球反超比分；下半场武磊进球追平李金羽单赛季进球纪录，  
王燊超造成张成林乌龙，胡尔克点射破门，阿兰补时打进点球。最终，上海上港客  
4战胜广州恒大淘宝，赛季双杀恒大同时也将积分榜上的领先优势扩大到五分，  
上港下轮只要战平就将夺得冠军。非常抱歉！  
***Repl Closed***
```

图5-5 通过正则表达式来清洗网页信息

5.5 简繁体转换

做中文信息处理，尤其是用网络爬虫爬取网络文本信息时，很可能一些深度遍历的网站来源于香港地区和其他国外网站。众所周知，香港地区由于历史原因仍旧广泛使用繁体字，采集的数据中不免存在简繁体共存，这对中文分析是非常不便的，必须统一成一种字体格式，才方便后续的工作。

幸运的是不用自己编写转换程序，只需要去官网下载一个简繁体工具包zhtools即可。读者可以从本书源码中直接获取，也可以去GitHub上下载（<https://github.com/skydark/nstools/tree/master/zhtools>）。下载完成之后打开zh_wiki.py文件可以看到如下内容：

```
1 # -*- coding: utf-8 -*-
2 # copy from Wikipedia
3
4 zh2Hant = {
5     '呆': '𡇗',
6     "打印机": "印表機",
7     '帮助文件': '說明檔案',
8     "画": "畫",
9     "板": "板",
10    "表": "表",
11    "才": "才",
12    "丑": "醜",
13    "出": "出",
14    "淀": "澱",
15    ...
16    "希特拉": "希特勒",
17    "黛安娜": "戴安娜",
18    "希拉": "赫拉",
19 }
```

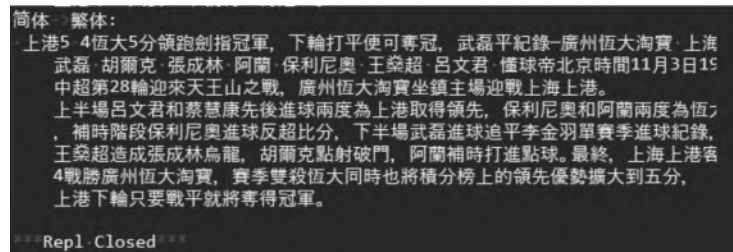
上述工具包字典包括字词约8260个，几乎涵盖了常用简繁体及其字词。当然，读者也可以自定义添加新的字词。通过这个字典我们很快就会发现，这是一种基于规则方法实现的中文简繁体转换。接下来，尝试将上一节网页中提取的简体中文进行繁体化，编写代码如下（源代码见：Chapter5/zhline.py）：

```

1 from zhtools.langconv import *
2
3 # 转换简体到繁体
4 str2 = r'上港 5-4 恒大 5 分领跑剑指冠军，下轮打平便可夺冠，武磊平纪录-广州恒大
   淘宝 上海上港 蔡慧康 武磊 胡尔克 张成林 阿兰 保利尼奥 王燦超 吕文君 懂球帝北京
   时间 11 月 3 日 19:35，中超第 28 轮迎来天王山之战，广州恒大淘宝坐镇主场迎战上海
   上港。上半场吕文君和蔡慧康先后进球两度为上港取得领先，保利尼奥和阿兰两度为恒大
   将比分扳平，补时阶段保利尼奥进球反超比分，下半场武磊进球追平李金羽单赛季进球纪
   录，王燦超造成张成林乌龙，胡尔克点射破门，阿兰补时打进点球。最终，上海上港客场
   5-4 战胜广州恒大淘宝，赛季双杀恒大同时也将积分榜上的领先优势扩大到五分，上港下
   轮只要战平就将夺得冠军。'
5 line2 = Converter('zh-hant').convert(str2)
6 print('简体->繁体:\n',line2)

```

执行上述代码，运行结果如图5-6所示。



```

简体 → 繁体:
上港 5-4 恒大 5 分领跑剑指冠军，下轮打平便可夺冠，武磊平纪录-广州恒大淘宝 上海
武磊 胡尔克 张成林 阿兰 保利尼奥 王燦超 吕文君 懂球帝北京时间11月3日19
中超第28轮迎来天王山之战，广州恒大淘宝坐镇主场迎战上海上港。
上半场吕文君和蔡慧康先后进球两度为上港取得领先，保利尼奥和阿兰两度为恒大
，补时阶段保利尼奥进球反超比分，下半场武磊进球追平李金羽单赛季进球纪录，
王燦超造成张成林乌龙，胡尔克点射破门，阿兰补时打进点球。最终，上海上港客
4战胜广州恒大淘宝，赛季双杀恒大同时也将积分榜上的领先优势扩大到五分，
上港下轮只要战平就将夺得冠军。
*** Repl Closed ***

```

图5-6 简体字转化为繁体字

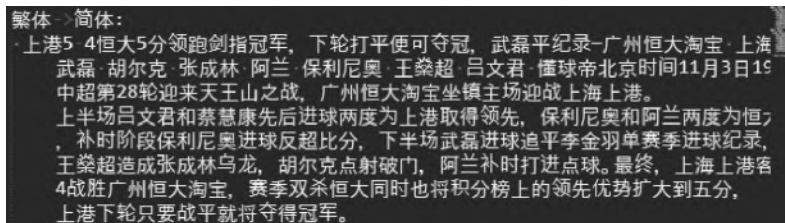
繁体字再转回简体字也比较简单，实现代码如下：

```

1 from zhtools.langconv import *
2 # 转换繁体到简体
3 str1 = '上港 5-4 恒大 5 分领跑剑指冠军，下轮打平便可夺冠，武磊平纪录—广州恒大淘宝 上海
上海港 蔡慧康 武磊 胡尔克 张成林 阿兰 保利尼奥 王燊超 吕文君 懂球帝北京时间 11
月 3 日 19:35，中超第 28 轮迎来天王山之战，广州恒大淘宝坐镇主场迎战上海上港。上半
场吕文君和蔡慧康先后进球两度为上港取得领先，保利尼奥和阿兰两度为恒大将比分扳
平，补时阶段保利尼奥进球反超比分，下半场武磊进球追平李金羽单赛季进球纪录，王燊
超造成张成林乌龙，胡尔克点射破门，阿兰补时打进点球。最终，上海上港客场 5-4 战胜
广州恒大淘宝，赛季双杀恒大同时也将积分榜上的领先优势扩大到五分，上港下轮只要战
平就将夺得冠军。'
4 line1 = Converter('zh-hans').convert(str1)
5 print('繁体->简体:\n',line1)

```

执行上述代码，运行结果如图5-7所示。



繁体 -> 简体:
上港 5-4 恒大 5 分领跑剑指冠军，下轮打平便可夺冠，武磊平纪录—广州恒大淘宝 上海
武磊 胡尔克 张成林 阿兰 保利尼奥 王燊超 吕文君 懂球帝北京时间 11 月 3 日 19
中超第 28 轮迎来天王山之战，广州恒大淘宝坐镇主场迎战上海上港。
上半场吕文君和蔡慧康先后进球两度为上港取得领先，保利尼奥和阿兰两度为恒
，补时阶段保利尼奥进球反超比分，下半场武磊进球追平李金羽单赛季进球纪录，
王燊超造成张成林乌龙，胡尔克点射破门，阿兰补时打进点球。最终，上海上港客
4 战胜广州恒大淘宝，赛季双杀恒大同时也将积分榜上的领先优势扩大到五分，
上港下轮只要战平就将夺得冠军。

图5-7 繁体字转化为简体字

5.6 实战案例：批量新闻文本数据清洗

5.6.1 高效读取文件内容

在5.2节中，详细地介绍了迭代遍历与yield生成器遍历的两个小实验，通过实验对比发现，高效文件读取方式效果显著。上节只是读取文件名，那么如何修改文件内容呢？这是本节侧重的知识点。在代码中，loadFolders方法保持不变，主要对loadFiles方法进行修改即可。

第5.2节中的loadFiles方法代码如下：

```

1 class loadFiles(object):
2     def __init__(self, par_path):
3         self.par_path = par_path
4     def __iter__(self):
5         folders = loadFolders(self.par_path)
6         for folder in folders:          # level directory
7             catg = folder.split(os.sep)[-1]
8             for file in os.listdir(folder):    # secondary directory
9                 yield catg, file

```

把上述代码修改如下：

```

1 class loadFiles(object):
2     def __init__(self, par_path):
3         self.par_path = par_path
4     def __iter__(self):
5         folders = loadFolders(self.par_path)
6         for folder in folders:          # level directory
7             catg = folder.split(os.sep)[-1]
8             for file in os.listdir(folder):    # secondary directory
9                 file_path = os.path.join(folder, file)
10                if os.path.isfile(file_path):
11                    this_file = open(file_path, 'rb') #rb 读取方式更快
12                    content = this_file.read().decode('utf8')
13                    yield catg, content
14                    this_file.close()

```

在第5.2节中，其中for file in os.listdir(folder)后直接返回类名和文件名，本节在此循环后主要体现了两个方面的变化，一是对子路径是否为文件的判断；二是对子文件读取操作，并返回文本信息内容。这样修改后封装的文件读取更加高效、便捷、通用。

5.6.2 抽样处理文件

在统计学中，抽样是一种推论统计方法，它是指从目标总体中抽取一部分个体作为样本，通过观察样本的某一或某些属性，依据所获得的数据对总体的数量特征得出具有一定可靠性的估计判断，从而达到对总体的认识。抽样方法诸多，常见的有以下几种：

- 简单随机抽样。也叫纯随机抽样，指从总体 N 个单位中随机地抽取 n 个单位作为样本，使得每一个容量为样本都有相同的概率被抽中。其特点是，每个样本单位被抽中的概率相等，样本的每个单位完全独立，彼此间无一定的关联性和排斥性。简单随机抽样是其他各种抽样形式的基础，通常只是在总体单位之间差异程度较小和数目较少时，才采用这种方法。
- 系统抽样。也称等距抽样，指将总体中的所有单位按一定顺序排列，在规定的范围内随机地抽取一个单位作为初始单位，然后按事先规定好的规则确定其他样本单位。先从数字 $1 \sim k$ 之间随机抽取一个数字 r 作为初始单位，以后依次取 $r+k$ 、 $r+2k \cdots \cdots$ 单位。这种方法操作简便，可提高估计的精度。
- 分层抽样。指将抽样单位按某种特征或某种规则划分为不同的层，然后从不同的层中独立、随机地抽取样本。从而保证样本的结构与总体的结构比较相近，以提高估计的精度。
- 整群抽样，将总体中若干个单位合并为组，抽样时直接抽取群，然后对选中群内的所有单位全部实施调查。抽样时只需群的抽样框，可简化工作量，缺点是估计的精度较差。

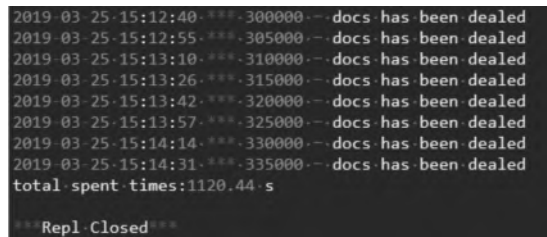
接下来，实现新闻文本的抽样读取。假设抽样率为5即每隔5条信息处理一篇文章，然后每处理5000篇文章在屏幕上打印一条信息，其执行代码如下：


```

1  if __name__=='__main__':
2      start = time.time()
3      filepath = os.path.abspath(r'../Corpus/CSCMNews')
4      files = loadFiles(filepath)
5      n = 5 # n 表示抽样率
6      for i, msg in enumerate(files):
7          if i % n == 0:
8              if int(i/n) % 1000 == 0:
9                  print('{t} *** {i} \t docs has been dealt'.format(i=i,
10                             t=time.strftime('%Y-%m-%d %H:%M:%S',
11                             time.localtime()))))
10     end = time.time()
11     print('total spent times:%.2f' % (end-start)+ ' s')

```

使用简单抽样的方法得到的信息处理结果如图5-8所示，该过程总耗时为1120.44秒。



```

2019-03-25-15:12:40.***.300000--docs has been dealt
2019-03-25-15:12:55.***.305000--docs has been dealt
2019-03-25-15:13:10.***.310000--docs has been dealt
2019-03-25-15:13:26.***.315000--docs has been dealt
2019-03-25-15:13:42.***.320000--docs has been dealt
2019-03-25-15:13:57.***.325000--docs has been dealt
2019-03-25-15:14:14.***.330000--docs has been dealt
2019-03-25-15:14:31.***.335000--docs has been dealt
total spent times:1120.44 s
***Repl Closed***

```

图5-8 抽样处理文本信息

5.6.3 通过正则表达式批量清洗文件

我们的最终目的是通过批量操作清洗文本信息，到目前为止，已经实现了文件的遍历和文本提取。其实知道了如何使用简单抽样，距离最终目标也只差一步之遥了。本节使用正则表达式处理文本的textParse方法，直接导入即可，导入方法如下：

```

1  from REdealText import textParse

```

接下来，提取文章类别、文章内容、通过正则表达式进行清洗，实现的代码如下（源代码见：Chapter5/30wClear.py）：

```
1 for i, msg in enumerate(files):
2     if i % n == 0:
3         catg = msg[0]      # 文章类别
4         content = msg[1]   # 文章内容
5         content = textParse(content) # 通过正则表达式进行清洗
```

打印结果，依旧保持每5000条新闻打印一条信息，注意观察执行时间。完整的代码如下：

```
1 if __name__ == '__main__':
2     start = time.time()
3     filepath = os.path.abspath(r'../Corpus/CSCMNews')
4     files = loadFiles(filepath)
5     n = 5 # n 表示抽样率， n 抽 1
6     for i, msg in enumerate(files):
7         if i % n == 0:
8             catg = msg[0]    # 文章类别
9             content = msg[1] # 文章内容
10            content = textParse(content) # 正则清洗
11            if int(i/n) % 1000 == 0:
12                print('{t} *** {i} \t docs has been dealt'
13                      .format(i=i, t=time.strftime('%Y-%m-%d %H:%M:%S',
14              time.localtime())) , '\n', catg, ':\t', content[:20])
15            end = time.time()
16            print('total spent times:%.2f' % (end-start)+ ' s')
```

本例最后一步，运行main函数，得到最终的结果，如图5-9所示。

```
财经 : -- 广发基金一半专户跌破面净 招商瑞泰 元最
2019-03-25-14:51:12-***-325000---docs has been dealt
财经 : -- 中证期货 商品窄幅波动依旧 美联储声明平
2019-03-25-14:51:33-***-330000---docs has been dealt
财经 : -- 收评 农产品多数走强 乙烯豆油领涨 新浪
2019-03-25-14:51:53-***-335000---docs has been dealt
财经 : -- 国内钢价罕见暴跌背后 楼市调控逼退钢贸投
total spent times:1210.27 s

*** Repl Closed ***
```

图5-9 新闻文本数据的批量清洗

5.7 本章小结

本章介绍了新闻语料的基本情况 & 语料构建的相关原则，对比了递归遍历与生成器遍历，并打造了一款高效的文件读取工具。最后，结合正则表达式的数据清洗方法完成了新闻语料的批量处理。下一章继续学习中文分词的相关技术。

第6 章

中文分词技术

中文分词技术属于自然语言处理技术的范畴，中文分词是其他中文信息处理的基础。本章首先介绍中文分词的相关概念和应用，接着对停用词和词性进行剖析，最后完成30万新闻文本的批量分词处理工作。

6.1 中文分词简介

6.1.1 中文分词概述

中文分词指的是将一个汉字序列切分成一个一个单独的词。分词就是将连续的字序列按照一定的规范重新组合成词序列的过程。在英文的行文中，单词之间是以空格作为自然分界符的，而中文只是句和段能通过明显的分界符来简单划界，而词是没有任何一个形式上的分界符的。虽然英文也同样存在短语的划分问题，不过在词这一层上，中文比之英文要复杂得多、困难得多。例如，英文句子：I am a student. 中文含义：我是一名学生。由于英文的语言使用习惯，通过空格很容易拆分出单词；而中文字词界限模糊往往不容易区别哪些是“字”，哪些是“词”。这也是为什么要把中文的词语进行切分的原因。

中文分词的方法其实不局限于中文应用，也被应用到英文处理。例如手写识别，英文单词之间的空格就不很清楚，中文分词方法可以反过来帮助判别英文单词的边界。

6.1.2 常见中文分词方法

早在20世纪80年代就有中文分词的研究工作，曾有人提出“正向最大匹配法”“逆向最大匹配法”“双向扫描匹配法”“逐词遍历法”等方法，共计多达16种之多。由于这些分词方法多是基于规则和词表的方法，随着统计方法的发展，不少学者提出了很多关于统计模型的中文分词方法，主要有以下几种：

（1）基于字符串匹配的分词方法

基本思想是基于词典匹配，将待分词的中文文本根据一定的规则切分和调整，然后跟词典中的词语进行匹配，匹配成功则按照词典的词进行分词，匹配失败则调整或者重新选择，如此反复循环即可。代表方法有基于正向最大匹配和基于逆向最大匹配及双向匹配法。

（2）基于理解的分词方法

基本思想是通过专家系统或者机器学习神经网络方法模拟人的理解能力。前者是通过专家对分词规则的逻辑推理并总结形成特征规则，不断迭代完善规则，其受到资源消耗大和算法复杂度高的制约。后者通过机器模拟人类理解的方式，虽可以取得不错的效果，但是依旧受训练时间长和过度拟合（Overfitting，也称为过拟合）等因素困扰。

（3）基于统计的分词方法

关于统计的中文分词方法的基本思想本文整理如下：

- 基于隐马尔可夫模型的中文分词方法。基本思想是通过文本作为观测序列去确定隐藏序列的过程。该方法采用Viterbi算法对

新词的识别效果不错，但是具有生成式模型的缺点，需要计算联合概率，因此随着文本的增大存在计算量大的问题。

- 基于最大熵模型的中文分词方法。基本思想是学习概率模型时，在可能的概率分布模型中，以熵最大的进行切分。该法可以避免生成模型的不足，但是存在偏移量问题。
- 基于条件随机场模型的中文分词方法。基本思想主要来源于最大熵马尔可夫模型，主要关注的字跟上下文标记位置有关，进而通过解码找到词边界。因此需要大量训练语料，而训练和解码又非常耗时。

综上所述，关于词典和规则的方法其分词速度较快，但是在不同领域取得的效果差异很大，还存在构造费时费力、算法复杂度高、移植性差等缺点。基于统计的中文分词，虽然其相较于规则的方法能取得不错的效果，但也依然存在模型训练时间长、分词速度慢等问题。针对这些问题，本文提出基于隐马尔可夫统计模型和自定义词典结合的方法，其在分词速度、歧义分析、新词发现和准确率方面都具有良好的效果。

6.2 结巴分词精讲

6.2.1 结巴分词的特点

结巴分词（jieba分词）是基于Python的中文分词工具，其分词功能强大且安装方便，该分词工具具备以下特征：

- 支持三种分词模式，即：
 - ◆ 全模式分词，把句子中所有的可以成词的词语都扫描出来，速度非常快，但是不能解决歧义。

- ◆ 精确模式分词，试图将句子最精确地切开，适合文本分析。
- ◆ 搜索引擎模式分词，在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎分词。
- 支持繁体分词。
- 支持自定义词典。
- MIT授权协议。

6.2.2 结巴分词的安装

结巴分词安装方法包括：全自动安装、半自动安装和手工安装。其中全自动安装也是最为简便和常用的安装方式，具体安装方式如下：

- 全自动安装：`pip install jieba`。
- 半自动安装：先下载jieba分词包文件 <http://pypi.python.org/pypi/jieba/>，解压后在根目录文件下执行`python setup.py install`命令即可。
- 手动安装：将jieba目录放置于当前目录或者`site-packages`目录。

6.2.3 结巴分词核心方法

- `jieba.cut`方法，可接受三个输入参数，分别是分词的字符串、`cut_all`参数（用来控制是否采用全模式）和HMM参数（用来控制是否使用HMM模型）。
- `jieba.cutforsearch`方法，可以接受两个参数，一是需要分词的字符串；二是是否使用HMM模型（该方法适合用于搜索引擎构

建倒排索引的分词，粒度比较细）。

- 待处理的字符串可以是unicode、UTF-8、GBK编码格式。注意：不建议直接输入GBK字符串，如果这样做，可能会出现无法预料的错误——解码成UTF-8。
- jieba.cut方法和jieba.cutforsearch方法返回的结构都是一个可迭代的generator，可以使用for循环来获得分词后得到的每一个词语（Unicode编码）。
- jieba.lcut方法和jieba.lcutforsearch方法直接返回列表（List）。
- 可以使用jieba.Tokenizer(dictionary=DEFAULT_DICT)新建自定义分词器，用于同时使用不同的词典。jieba.dt为默认分词器，所有全局分词相关的函数都是该分词器的映射。

6.2.4 结巴中文分词的基本操作

jieba分词包括三种分词模型，即全模式分词、精确分词和搜索引擎分词，可以实现自定义调整词典、关键词提取、词性标注和句法分析等功能（源代码见：Chapter6/jiebaCut.py）。

1. 全模式分词

示例代码如下：

```
1 import jieba
2 # 1 全模式，扫描所有可以成词的词语，速度非常快，不能解决歧义
3 seg_list = jieba.cut("我来到成都四川大学", cut_all=True)
4 print("\nFull Mode: " + "/ ".join(seg_list))
```

代码运行结果：Full Mode: 我 / 来到 / 成都 / 四川 / 四川大学 / 大学

2. 精确模式

示例代码如下：

```
1 # 2 默认是精确模式，适合文本分析
2 seg_list = jieba.cut("我来到成都四川大学", cut_all=False)
3 print("\nDefault Mode: " + "/ ".join(seg_list)) # 默认模式
```

代码运行结果：Default Mode: 我 / 来到 / 成都 / 四川大学

3. 搜索引擎分词

示例代码如下：

```
1 # 3 搜索引擎模式，对长词再次切分，提高召回率，适合用于搜索引擎分词
2 # jieba.cut_for_search 该方法适合用于搜索引擎构建倒排索引的分词，粒度比较细
3 seg_list = jieba.cut_for_search("小明硕士毕业于中国科学院计算所，后在日本京都大学深造", HMM=False)
4 print('\n 搜索引擎模式: '+' ', ".join(seg_list))
```

代码运行结果：

搜索引擎模式：小，明，硕士，毕业，于，中国，科学，学院，科学院，中国科学院，计算，计算所，，，后，在，日本，京都，大学，日本京都大学，深造

6.2.5 自定义分词词典

1. 自定义调整词典

无论哪种分词都不能做到100%的准确，尤其是对一些歧义词的处理，比如“如果放到数据库中将会出错”，其中“数据库中，将会出

错”和“数据库，中将”有不同的分词方式，为保证准确分词，有时候需要进行自定义词典的调整。下面代码是其中的一些代表示例：

```
1 print('='*40)
2 print('2. 添加自定义词典/调整词典')
3 print('-'*40)
4 print('原文档: \t'+'/'.join(jieba.cut('如果放到数据库中将出错.',
    HMM=False)))
5 # 如果/放到/数据库/中将/出错/。
6 print(jieba.suggest_freq(('中', '将'), True))
7 print('改进文档: \t'+'/'.join(jieba.cut('如果放到数据库中将出错.',
    HMM=False)))
8 # 如果/放到/数据库/中/将/出错/。
9 print('\n 原文档: \t'+'/'.join(jieba.cut('「台中」正确应该不会被切开',
    HMM=False)))
10 # 「/台/中/」/正确/应该/不会/被/切开
11 print(jieba.suggest_freq('台中', True))
12 #69
13 print('改进文档: \t'+'/'.join(jieba.cut('「台中」正确应该不会被切开',
    HMM=False)))
14 # 「/台中/」/正确/应该/不会/被/切开
```

运行结果如图6-1所示。（注：图中的“将”字是由系统默认打出的，是系统存在的问题。）

```
2. 添加自定义词典/调整词典
-----
原文档: 如果/放到/数据库/中将/出错/。
494
改进文档: ——如果/放到/数据库/中/将/出错/。

原文档: 「/台/中/」/正确/应该/不会/被/切开
69
改进文档: ——「/台中/」/正确/应该/不会/被/切开
```

图6-1 自定义调整词典

运行上述代码，图6-1结果显示“如果放到数据库中将出错。”在默认情况下分词结果：“如果 / 放到 / 数据库 / 中将 / 出错 /”，这显然是错误的，该语境下“中将”应分开处理，正确的分词为：“如

果 / 放到 / 数据库 / 中 / 将 / 出错 / ”。这里使用add_word(word, freq=None, tag=None)在程序中动态修改词典。

注 意

自动计算的词频在使用HMM新词发现功能时可能无效。

2. 自定义调整词典解决歧义分词问题

上述方法手动定义词典显得很不方便。还有更好的方法——可以将自定义词典放在一个文本文件中来解决歧义分词问题，还可以实现批量处理。这里使用源代码中Files/user_dict.txt文件作为自定义的分词词典，比如加载“很高兴”这个词，代码如下：

```
1 import sys
2 sys.path.append("../")
3 jieba.load_userdict("../Files/user_dict.txt") # 加载自定义分词词典
4
5 seg_list1 = jieba.cut("今天很高兴在学习网和大家交流学习")
6 print('\n\n加载自定义分词词典: \n'+"/ ".join(seg_list1))
```

运行结果：

加载自定义分词词典：今天 / 很高兴 / 在 / 学习网 / 和 / 大家 / 交流 学习

6.2.6 关键词提取

关键词提取是指从一段文本内容中提取重要的信息，调用jieba分词中的extract_tags方法可以实现该功能，该方法有三个参数，分别是待处理的字符串、提取前n个词和是否根据权重设置。以下是一个提取一段文本关键词的代码示例：

```

1 s = "此外，公司拟对全资子公司吉林欧亚置业有限公司增资 4.3 亿元，增资后，吉林欧亚置业注册资本由 7000 万元增加到 5 亿元。吉林欧亚置业主要经营范围为房地产开发及百货零售等业务。目前在建吉林欧亚城市商业综合体项目。2013 年，实现营业收入 0 万元，实现净利润-139.13 万元。"
2 for x, w in jieba.analyse.extract_tags(s,10, withWeight=True):
3     print('%s %s' % (x, w))

```

运行结果如图6-2所示。

```

3. 关键词提取
-----
TF IDF
-----
欧亚 0.7300142700289363
吉林 0.659038184373617
置业 0.4887134522112766
万元 0.3392722481859574
增资 0.33582401985234045
4.3 0.25435675538085106
7000 0.25435675538085106
2013 0.25435675538085106
139.13 0.25435675538085106
实现 0.19900979900382978

```

图6-2 根据词频概率提取关键词

6.2.7 词性标注

词性标注就是对分词结果根据词性进行标记。后续章节会专门介绍词性的用法，另外词性标注在过滤特征词或者实体处理中都是有用的。以下是词性标注的用法示例代码：

```

1 words = jieba.posseg.cut("我爱北京天安门")
2 for word, flag in words:
3     print('%s %s' % (word, flag))

```

运行结果如图6-3所示。

```

4. 词性标注
-----
我 r
爱 v
北京 ns
天安门 ns

```

图6-3 词性标注

图 6-3 是根据分词工具构建的词性表（源代码见：PyDataPreprocessing/Files/词性.txt），其中r为代词，v为动词，ns为地名。

6.3 HanLP分词精讲

HanLP是一个由一系列模型与算法组成的Java工具包，其目标是普及自然语言处理在生产环境中的应用。HanLP具备功能完善、性能高效、架构清晰、语料时新、可自定义等特点，在提供丰富功能的同时，其内部模块坚持低耦合、模型坚持惰性加载、服务坚持静态提供、词典坚持明文发布，使用起来非常方便，同时自带一些语料处理工具，以帮助用户训练自己的语料。（源代码见Chapter6/HanLPCut.py）

6.3.1 JPyPe1的安装

HanLP是由Java语言开发的分词工具，Python想调用该分词工具，需要借助JPyPe1包。JPyPe1是Python调用Java库文件的模块。其安装方法有如下两种方式：

- `pip install JPyPe1`
- `pip install JPyPe1-0.6.3-cp37-cp37m-winamd64.whl`

6.3.2 调用HanLP的Java包

要使用HanLP需要调用Java包文件，具体实现代码如下：

```

1 from jpyype import *
2 startJVM(getDefaultJVMPath(),
   "-Djava.class.path=C:\\hanlp\\hanlp-1.3.2.jar;
3     C:\\hanlp", "-Xms1g", "-Xmx1g") # 启动 JVM, Linux 需替换分号;为
   冒号:
4 ...
5 shutdownJVM()

```

代码说明:

- HanLP 下载地址 (链接 : <https://pan.baidu.com/s/13TB1jHX0FeLTjLBC844sbw>提取码: 2ax7), 读者将其解压到C盘下即可。
- 安装配置jre1.7+, 本文省略具体安装步骤。

6.3.3 HanLP分词

1. 第一个Demo

```

1 paraStr1='中国科学院计算技术研究所的宗成庆教授正在教授自然语言处理课程'
2 print("="*30+"HanLP 分词"+"="*30)
3 HanLP = JClass('com.hankcs.hanlp.HanLP')
4 print(HanLP.segment(paraStr1))

```

本段代码实现对字符串进行中文分词, 运行结果如图6-4所示。

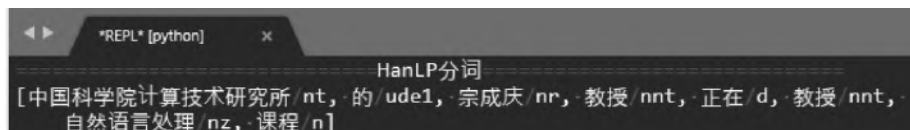


图6-4 HanLP默认分词

2. 标准分词

```

1 print("="*30+"标准分词"+"="*30)
2 StandardTokenizer =
    JClass('com.hankcs.hanlp.tokenizer.StandardTokenizer')
3 print(StandardTokenizer.segment(paraStr1))

```

一个 Dome 中的 HanLP.segment 其实是对 StandardTokenizer.segment 的包装，该分词算法是基于词图生成的。运行结果如图6-5所示。

```

标准分词
[中国科学院计算技术研究所 /nt, 的 /ude1, 宗成庆 /nr, 教授 /nnt, 正在 /d, 教授 /nnt,
自然语言处理 /nz, 课程 /n]

```

图6-5 HanLP标准分词

3. NLP分词

```

1 # NLP 分词 NLPTokenizer 会执行全部命名实体识别和词性标注
2 print("="*30+"NLP 分词"+"="*30)
3 NLPTokenizer = JClass('com.hankcs.hanlp.tokenizer.NLPTokenizer')
4 print(NLPTokenizer.segment(paraStr1))

```

NLP分词NLPTokenizer会执行词性标注和命名实体识别，默认模型训练自9970万字的大型综合语料库，是已知范围内全世界最大的中文分词语料库。语料库规模决定实际效果，面向生产环境的语料库应当在千万字量级。运行结果如图6-6所示。

```

NLP分词
[中国科学院计算技术研究所 /nt, 的 /ude1, 宗成庆 /nr, 教授 /nnt, 正在 /d, 教授 /v,
自然语言处理 /nz, 课程 /n]

```

图6-6 HanLP NLP分词

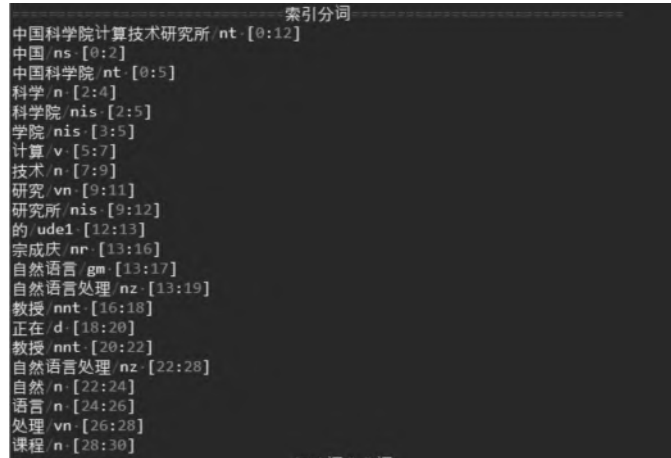
4. 索引分词

```

1 print("="*30+"索引分词"+"="*30)
2 IndexTokenizer =
  JClass('com.hankcs.hanlp.tokenizer.IndexTokenizer')
3 termList= IndexTokenizer.segment(paraStr1);
4 for term in termList :
5     print(str(term) + " [" + str(term.offset) + ":" + str(term.offset
      + len(term.word)) + "]"")

```

索引分词IndexTokenizer是面向搜索引擎的分词器，能够对长词全切分，另外通过term.offset可以获取词在文本中的偏移量。运行结果如图6-7所示。



```

索引分词
中国科学院计算技术研究所 nt [0:12]
中国 ns [0:2]
中国科学院 nt [0:5]
科学 n [2:4]
学院 nis [2:5]
学院 nis [3:5]
计算 v [5:7]
技术 n [7:9]
研究 vn [9:11]
研究所 nis [9:12]
的 ude1 [12:13]
宗成庆 nr [13:16]
自然语言 gm [13:17]
自然语言处理 nz [13:19]
教授 nnt [16:18]
正在 d [18:20]
教授 nnt [20:22]
自然语言处理 nz [22:28]
自然 n [22:24]
语言 n [24:26]
处理 vn [26:28]
课程 n [28:30]

```

图6-7 HanLP索引分词

5. 极速词典分词

```

1 print("="*30+" 极速词典分词"+"="*30)
2 SpeedTokenizer =
  JClass('com.hankcs.hanlp.tokenizer.SpeedTokenizer')
3 print(NLPTokenizer.segment(paraStr1))

```

极速分词是词典最长分词，速度极其快，精度一般。其算法原理是基于多模式匹配，运行结果如图6-8所示。


```
===== 极速词典分词 =====  
[中国科学院计算技术研究所/nt, 的/ude1, 宗成庆/nr, 教授/nnt, 正在/d, 教授/v, 自然语言处理/nz, 课程/n]
```

图6-8 HanLP极速分词

6.3.4 HanLP实现自定义分词

自定义分词是一份全局的用户自定义词典，可以随时增删，影响全部分词器。另外可以在任何分词器中关闭它。通过代码动态增删不会保存到词典文件。示例代码如下：

```
1 paraStr2 = '攻城狮逆袭单身狗，迎娶白富美，走上人生巅峰'  
2 print("="*30+" 自定义分词"+"="*30)  
3 CustomDictionary =  
4   JClass('com.hankcs.hanlp.dictionary.CustomDictionary')  
5 CustomDictionary.add('攻城狮')  
6 CustomDictionary.add('单身狗')  
7 HanLP = JClass('com.hankcs.hanlp.HanLP')  
8 print(HanLP.segment(paraStr2))
```

该分词算法基于Trie树和多模式匹配来实现的，运行结果如图6-9所示。

```
===== 自定义分词 =====  
[攻城狮/nz, 逆袭/nz, 单身狗/nz, , /w, 迎娶/v, 白富美/nr, , /w, 走上/v, 人生/n, 巅峰/n]
```

图6-9 HanLP自定义分词

6.3.5 命名实体识别与词性标注

命名实体识别就是对语料中的人名、地名、机构名、时间、地点等识别。诸如下面对机构名“中国科学院计算技术研究所”，人名“宗成庆”，课程名“自然语言处理”识别示例代码如下：

```

1 paraStr1='中国科学院计算技术研究所的宗成庆教授正在教授自然语言处理课程'
2 print("="*20+"命名实体识别与词性标注"+"="*30)
3 NLPTokenizer = JClass('com.hankcs.hanlp.tokenizer.NLPTokenizer')
4 print(NLPTokenizer.segment(paraStr1))

```

运行结果如图6-10所示。

```

=====命名实体识别与词性标注=====
[中国科学院计算技术研究所/nt, 的/ude1, 宗成庆/nr, 教授/nnt, 正在/d, 教授/v,
自然语言处理/nz, 课程/n]

```

图6-10 HanLP命名实体识别

6.3.6 HanLP实现关键词抽取

对于一段文字，其关键词并非简单的词频统计，而是对信息描述的重要体现。HanLP实现关键词抽取是基于TextRank算法来实现的，示例代码如下：

```

1 paraStr3="水利部水资源司司长陈明忠 9月29日在国务院新闻办举行的新闻发布会上
  透露，根据刚刚完成了水资源管理制度的考核，有部分省接近了红线的指标,有部分省超
  过红线的指标。对一些超过红线的地方，陈明忠表示，对一些取用水项目进行区域的限
  批,严格地进行水资源论证和取水许可的批准。"
2 print("="*30+"关键词提取"+"="*30)
3 print(HanLP.extractKeyword(paraStr3, 8))

```

运行结果如图6-11所示。

```

=====关键词提取=====
[水资源, 陈明忠, 进行, 红线, 用水, 国务院新闻办, 表示, 新闻]

```

图6-11 HanLP抽取关键词

6.3.7 HanLP实现自动摘要

HanLP自动摘要内部采用TextRank算法实现，用户可以直接调用。下面示例是文章摘要自动实现的代码：

```
1 paraStr0 ="据报道，2017 年 5 月，为了承接一项工程，来自河南一家建筑公司的柳先生和十几名同事被派到南京做前期筹备工作。为了节约开支，大伙商量决定采取 A A 制的方式搭伙做饭。不料，2017 年 11 月建邺区市场监管局(下称市监局)上门进行检查，要求柳先生等人不得在出租房内做饭，并且开出了一份《行政处罚听证告知书》...作为食品药品安全卫生监督管理部门，市监局严格认真执法本无可厚非。但此次重罚搭伙吃饭的做法除了反映执法能力问题外，滥用执法权力的冲动亦不可忽视。每一个执法者都须切记，执法的核心宗旨是保护公民的合法权益，人民授予的权力更不可被滥用。"
2 print("="*30+"自动摘要"+"="*30)
3 print(HanLP.extractSummary(paraStr0, 5))
```

运行结果如图6-12所示。

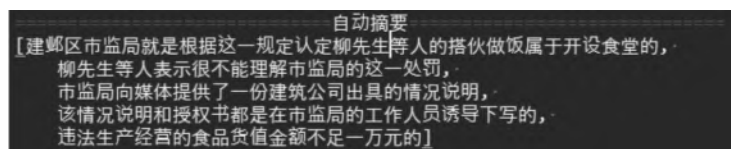


图6-12 HanLP实现自动文摘

6.4 自定义去除停用词

6.4.1 以正则表达式对文本信息进行清洗

文本分词中的去除停用词在数据预处理中是不可或缺的内容，这里将分词处理操作进行封装，主要涉及文本内容读取和正则清洗，接着进行分词和停用词处理等。以下是给出各个功能的实现代码（源代码见：Chapter6/StopWords.py）

1. 读取文本信息

代码示例如下：

```
1 def readFile(path):
2     str_doc = ""
3     with open(path, 'r', encoding='utf-8') as f:
4         str_doc = f.read()
5     return str_doc
6
7 # 1 读取文本
8 path= r'../Corpus/CSCMNews/体育/0.txt'
9 str_doc = readFile(path)
```

2. 分词去停用词

代码示例如下：

```
1 # 利用 jieba 对文本进行分词，返回切词后的 list（列表）
2 def seg_doc(str_doc):
3     # 1 正则处理原文本
4     sent_list = str_doc.split('\n')
5     # map 内置高阶函数：一个函数 f 和 list，函数 f 依次作用在 list。
6     sent_list = map(textParse, sent_list) # 通过正则表达式进行处理，去
    掉一些字符，例如\u3000
7     # 2 获取停用词
8     stwlist = get_stop_words()
9     # 3 分词并去除停用词
10    word_2dlist = [rm_tokens(jieba.cut(part,
    cut_all=False), stwlist) for part in sent_list]
11    # 4 合并列表
12    word_list = sum(word_2dlist, [])
13    return word_list
```

3. 以正则表达式清洗字符串

代码示例如下：

```

1  # 以正则表达式对字符串进行清洗
2  def textParse(str_doc):
3      # 通过正则表达式过滤掉特殊符号、标点、英文、数字等
4      r1 = '[a-zA-Z0-9'!'#$%&\'()*+,-./:; |<=>?@, -。?★、...【】《》?'
        '\\"'!'[\\"^_`{|}~]+'
5      str_doc=re.sub(r1, ' ', str_doc)
6      # 去掉字符
7      str_doc = re.sub('\u3000', '', str_doc)
8      # 去除空格
9      # str_doc=re.sub('\s+', ' ', str_doc)
10     # 去除换行符
11     # str_doc = str_doc.replace('\n', ' ')
12     return str_doc

```

4. 创建停用词列表

代码示例如下：

```

1  # 创建停用词列表
2  def get_stop_words(path=r'../Files/NLPIR_stopwords.txt'):
3      file = open(path, 'r',encoding='utf-8').read().split('\n')
4      return set(file)
5
6  # 去掉一些停用词和数字
7  def rm_tokens(words, stwlist):
8      words_list = list(words)
9      stop_words = stwlist
10     for i in range(words_list. len  ())[::-1]:
11         if words_list[i] in stop_words:      # 去除停用词
12             words_list.pop(i)
13         elif words_list[i].isdigit():        # 去除数字
14             words_list.pop(i)
15         elif len(words_list[i]) == 1:        # 去除单个字符
16             words_list.pop(i)
17         elif words_list[i] == " ":          # 去除空字符
18             words_list.pop(i)
19     return words_list

```

5. 执行分词后的方法

代码示例如下：

```
1  if __name__=='__main__':
2      # 1 读取文本
3      path= r'../Corpus/CSCMNews/体育/0.txt'
4      str_doc = readFile(path)
5      # print(str_doc)
6
7  # 2 分词去停用词
8  word_list = seg_doc(str_doc)
9  print(word_list)
```

运行结果如图6-13所示。



```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\ADMINI~1\AppData\Local\Temp\jieba.cache
Loading model cost 0.832 seconds.
Prefix dict has been built successfully.
['马晓旭', '意外', '受伤', '国奥', '警惕', '无奈', '大雨', '青峰', '股家', '记者',
'傅亚雨', '沈阳', '报道', '来到', '沈阳', '国奥队', '依然', '摆脱', '雨水',
'困扰', '下午', '国奥队', '日常', '训练', '再度', '大雨', '干扰', '无奈', '之下',
'队员', '慢跑', '分钟', '草草收场', '上午', '国奥队', '奥体中心', '外场',
'训练', '阴沉沉', '气象预报', '显示', '当天', '下午', '沈阳', '大雨', '幸好',
'队伍', '上午', '训练', '干扰', '下午', '球队', '抵达', '训练场', '大雨', '几个',
'小时', '丝毫', '停下来', '试一试', '态度', '球队', '当天', '下午', '例行',
'训练', '分钟', '天气', '转好', '迹象', '保护', '球员', '国奥队', '中止', '当天',
'训练', '全队', '返回', '酒店', '训练', '足球队', '稀罕', '奥运会', '全队',
'变得', '矫贵', '沈阳', '一周', '训练', '国奥队', '保证', '现有', '球员',
'出现意外', '伤病', '情况', '影响', '正式', '比赛', '这一', '阶段', '控制',
'训练', '受伤', '控制', '感冒', '疾病', '队伍', '放在', '位置', '抵达', '沈阳',
'后卫', '马萧霆', '训练', '马萧霆', '长春', '患上', '感冒', '参加', '塞尔维亚']
```

图6-13 分词处理的结果

6.4.2 结巴中文分词词性解读

在结巴分词中，其给出的中文分词都有其特定的词性，并给出了字符标记说明，以下列出了中文词性的说明及其字符说明。

1	Ag	形语素	形容词性语素。形容词代码为 a, 语素代码 g 前面置以 A。
2	a	形容词	取英语形容词 adjective 的第 1 个字母。
3	ad	副形词	直接作状语的形容词。形容词代码 a 和副词代码 d 并在一起。
4	an	名形词	具有名词功能的形容词。形容词代码 a 和名词代码 n 并在一起。
5	b	区别词	取汉字“别”的声母。
6	c	连词	取英语连词 conjunction 的第 1 个字母。
7	Dg	副语素	副词性语素。副词代码为 d, 语素代码 g 前面置以 D。
8	d	副词	取 adverb 的第 2 个字母, 因其第 1 个字母已用于形容词。
9	e	叹词	取英语叹词 exclamation 的第 1 个字母。
10	f	方位词	取汉字“方”。
11	g	语素	绝大多数语素都能作为合成词的“词根”, 取汉字“根”的声母。
12	h	前接成分	取英语 head 的第 1 个字母。
13	i	成语	取英语成语 idiom 的第 1 个字母。
14	j	简称略语	取汉字“简”的声母。
15	k	后接成分	
16	l	习用语	习用语尚未成为成语, 有点“临时性”, 取“临”的声母。
17	m	数词	取英语 numeral 的第 3 个字母, n, u 已有他用。
18	Ng	名语素	名词性语素。名词代码为 n, 语素代码 g 前面置以 N。
19	n	名词	取英语名词 noun 的第 1 个字母。
20	nr	人名	名词代码 n 和“人(ren)”的声母并在一起。
21	ns	地名	名词代码 n 和处所词代码 s 并在一起。
22	nt	机构团体	“团”的声母为 t, 名词代码 n 和 t 并在一起。
23	nz	其他专名	“专”的声母的第 1 个字母为 z, 名词代码 n 和 z 并在一起。
24	o	拟声词	取英语拟声词 onomatopoeia 的第 1 个字母。
25	p	介词	取英语介词 prepositional 的第 1 个字母。
26	q	量词	取英语 quantity 的第 1 个字母。
27	r	代词	取英语代词 pronoun 的第 2 个字母, 因 p 已用于介词。
28	s	处所词	取英语 space 的第 1 个字母。
29	Tg	时语素	时间词性语素。时间词代码为 t, 在语素的代码 g 前面置以 T。
30	t	时间词	取英语 time 的第 1 个字母。
31	u	助词	取英语助词 auxiliary
32	Vg	动语素	动词性语素。动词代码为 v。在语素的代码 g 前面置以 V。
33	v	动词	取英语动词 verb 的第一个字母。
34	vd	副动词	直接作状语的动词。动词和副词的代码并在一起。
35	vn	名动词	指具有名词功能的动词。动词和名词的代码并在一起。
36	w	标点符号	
37	x	非语素字	非语素字只是一个符号, 字母 x 通常用于代表未知数、符号。
38	y	语气词	取汉字“语”的声母。
39	z	状态词	取汉字“状”的声母的前一个字母。
40	un	未知词	不可识别词及用户自定义词组。取英文 unknow 首两个字母。

6.4.3 根据词性规则构建自定义停用词

打开user_dict.txt文件构建自定义停用词表如下：

```
1 <!-- 词语、词频（可省略）、词性（可省略） -->
2 很高兴 vd
3
```

6.5 词频统计

词频统计是一种用于情报检索与文本挖掘的常用加权技术，用以评估一个词对于一个文件或者一个语料库中的一个领域文件集的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。NLTK自然语言工具包大大方便了词频的处理，此外，它还包含分词、词性标注、命名实体识别、句法分析等功能。

6.5.1 NLTK介绍与安装

1. NLTK介绍

NLTK（Natural Language Toolkit，自然语言工具包）是Python编程语言实现的自然语言处理工具，它是由宾夕法尼亚大学计算机和信息科学的史蒂芬·伯德和爱德华·洛珀编写的。NLTK支持NLP研究和教学相关的领域，它收集的大量公开数据集、模型上提供了全面易用的接口，涵盖了分词、词性标注（Part-Of-Speech Tag, POS-Tag）、命名实体识别（Named Entity Recognition, NER）、句法分析（Syntactic Parse）等功能，广泛应用于经验语言学、认知科学、人

工智能、信息检索和机器学习领域。目前已有25个国家的32所大学将NLTK作为教学工具。

NLTK的模块及功能如表6-1所示。

表6-1 NLTK的模块及功能

任 务	模 块	描 述
获取语料库	nltk.corpus	语料库和词典的标准化接口
字符串处理	nltk.tokenize, nltk.stem	分词、句子分解和提取主干（不支持中文）
搭配研究	nltk.collocations	t 检验、卡方检验和互信息
词性标注	nltk.tag	n-gram、backoff 和 HMM
分类/聚类	nltk.classify、nltk.cluster	决策树、最大熵、朴素贝叶斯、EM 和 K-means
分块	nltk.chunk	正则表达式、n-gram 和命名实体
解析	nltk.parse	图标、基于特征、一致性和概率性
语义解释	nltk.sem、nltk.inference	演算、模型检验
指标评测	nltk.metrics	准确率、召回率和协议系数
概率与估计	nltk.probability	频率分布和平滑概率分布
应用	nltk.app、nltk.chat	图形化关键字排序、分析器，wordNet 查看器
语言学领域工作	nltk.toolbox	处理 SIL 工具箱格式的数据

2. NLTK的安装

（1）查看Python版本，如图6-14所示。

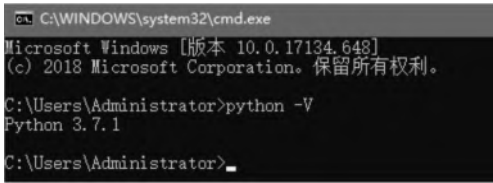


图6-14 Python版本

（2）对于 Windows 系统，下载 NLTK 的安装文件 nltk-3.2.1.win32.exe (<https://pan.baidu.com/s/1qYzXFPy>)，并执行这个exe文件，这个安装文件会自动匹配到Python安装路径，如果没有找到路径说明 NLTK 版本不正确，到官网（网址：

<https://pypi.python.org/pypi/nltk/3.2.1>) 选择正确的版本下载，如图6-15所示。

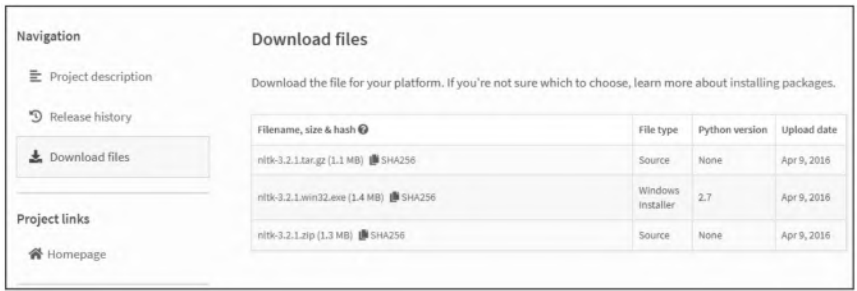


图6-15 Python安装的根本路径

(3) 安装成功后，打开Python编辑器，输入以下命令下载NLTK数据包：

```
1 >>> import nltk
2 >>> nltk.download()
```

如图 6-16 所示，选中 book，修改下载路径“D:\Users\Administrator\Anaconda3\nltk_data”。（book包含了数据案例和内置函数。）

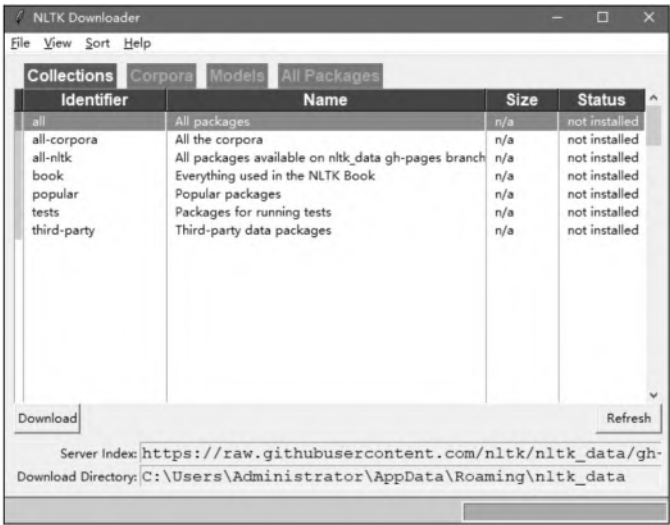


图6-16 下载NLTK的book数据包

(4) 配置环境变量。计算机->属性->高级系统设置->高级->环境变量 - 系统变量 ->path , 输入如下路径 :
C:\Users\Administrator\AppData\Roaming\nltk_data。

(5) 打开Python解释器输入如下代码:

```
>>> from nltk.book import *
```

出现如图6-17所示, 表示安装成功。

```
>>> from nltk.book import *
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
>>>
```

图6-17 成功安装NLTK数据包

6.5.2 统计新闻文本词频

安装完成NLTK以后, 使用NLTK的方法统计中文分词的词频信息, 编写如下代码(源代码见: Chapter6/FreqWord.py):

```
1 from nltk import *
2 fdist=FreqDist(word_list)
3 print(fdist.keys(),fdist.values())
```

代码运行结果如下:

```
dict_keys(['马晓旭', '意外', '受伤',
... '没想到', '碰到', '不解'])
dict_values([2, 1, 3, ... 1, 1, 1])
```

6.5.3 统计特定词频和次数

还可以统计指定词语的出现频率及其次数，编写如下代码：

```
1 print('='*3,'指定词语词频统计','='*3)
2 w='训练'
3 print(w,'出现频率: ',fdist.freq(w))      # 给定样本的频率
4 print(w,'出现次数: ',fdist[w])           # 出现次数
```

代码运行结果如下：

```
=== 指定词语词频统计 ===
训练 出现频率:  0.06878306878306878
训练 出现次数:  13
```

6.5.4 特征词的频率分布表

由于散落的列表形式不容易观察词频分布情况，因此使用特征词频分布表来显示，具体代码如下：

```
1 fdist=FreqDist(word_list)
2 print('='*3,'频率分布表','='*3)
3 fdist.tabulate(10) # 频率分布表
```

代码说明：tabulate方法中的参数表示显示前N个特征词。

代码运行结果：

训练	沈阳	国奥队	大雨	下午	球员	感冒	冯萧霆	受伤	雨水
13	8	8	5	4	4	4	4	3	3

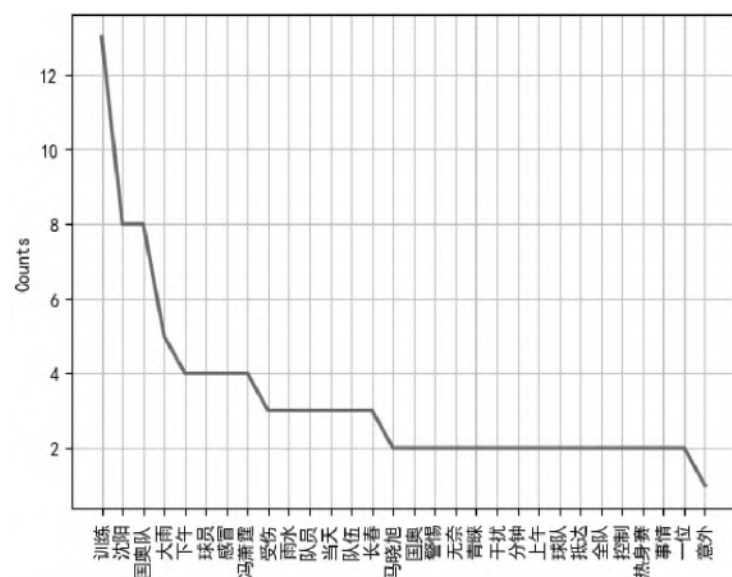
6.5.5 频率分布图与频率累计分布图

1. 频率分布图

分布表的形式显然优于列表的展现形式，但是还有更好的分析方法——图形化分析，所谓“一图胜千言”就是突出可视化的强大之处，比如用频率分布图（其中中文字体解决方法下文有介绍）来分析，编写如下代码：

```
1 from nltk import *
2 fdist=FreqDist(word_list)
3 fdist.plot(30) # 频率分布图
```

运行结果如图6-18所示。



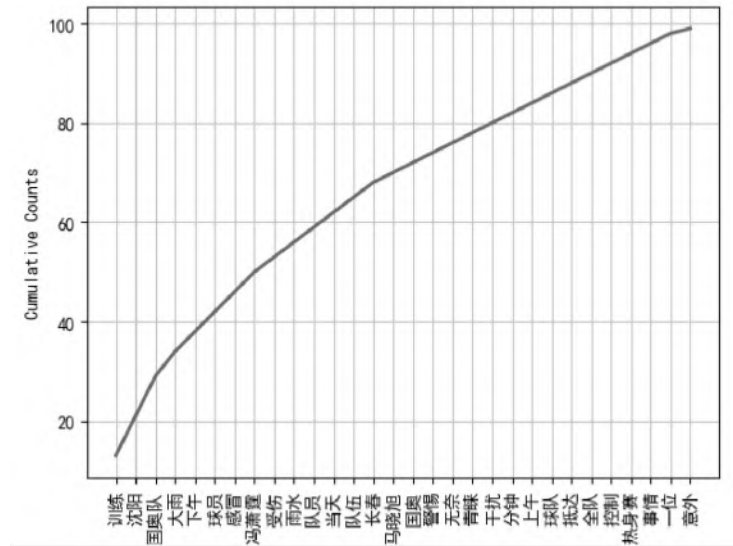


图6-19 特征词频率累计图

3. Matplotlib中文字体显示

Matplotlib模块常用于可视化处理，对于科研或者工程都是便捷的。Matplotlib对英文支持表现良好，然而对中文字体默认是识别不出来的，因此需要对字体进行配置。

(1) 查看当前使用的字体格式。

```
1 from matplotlib.font_manager import findfont, FontProperties
2 print(findfont(FontProperties(family=FontProperties().get_family(
  ))))
```

(2) 在C:\Windows\Fonts查找中文字体SimHei.ttf，并将其复制到../mpl-data/font/ttf文件夹下面。

(3) 设置使用的字体。

```
1 matplotlib.rcParams['font.sans-serif'] = 'SimHei'
```

6.5.6 基于Counter的词频统计

对于前文体育新闻中的词频统计，除了使用NLTK工具外，还可以借助Counter来实现：

```
1 # *****统计词频方法2*****
2 from collections import Counter
3 Words = Counter(word_list)
4 print(Words.keys(),Words.values())
5 wlist =[w for w in Words if len(w)>2]
6 print(wlist)
```

上述代码word_list是一个列表，存放分词的结果。通过collections.Counter方法对列表进行Key-value操作，并统计词频信息。接下来，选择词语长度大于2的词特征并打印输出。

运行结果如下：

```
dict_keys(['马晓旭', '意外', '受伤',
... '没想到', '碰到', '不解'])
dict_values([2, 1, 3, ... 1, 1, 1])
```

```
['马晓旭', '傅亚雨', '国奥队', '草草收场', '奥体中心', '阴沉沉', '气象预报',
'训练场', '停下来', '试一试', '足球队', '奥运会', '出现意外', '冯萧霆', '塞尔维亚',
'热身赛', '对雨中', '非战斗', '工作人员', '没想到']
```

6.6 自定义去高低词频

1. 词频统计模块的复用

此处，对上文介绍的模块进行复用，可以使用StopWords模块调用文件读取和分词方法，使用FreqWord模块调用词频统计等方法。首先使用下述代码导入模块：

```
1 from StopWords import readFile,seg_doc
2 from FreqWord import *
```

2. 实现高低频词的提取

在完成词频统计之后，为了更加精准地分析特征词和模型结果，需要对特征词进行进一步的处理。这里面涉及高词频和低词频的问题，对于词频的选择主要考虑以下3个方面：

- 选择高低词频。有时候高低词频更能代表文本的特征，比如一篇足球的报告，其中足球、跑位等领域特征词频明显高于普通词，此刻就对高频词赋予更大的权重。
- 选择中间词。还有些需求，特别高或者特别低的词频是一种异常值，不能突显文章的核心内容，因此需要处理，截取中位特征词。
- 至于高低词频的选择，可根据实际应用环境来设定。

编写如下代码：

```
1 def freqword(fdist):
2     wordlist = []
3     print('='*3, '打印统计的词频', '='*3)
4     for key in fdist.keys():
5         if fdist.get(key)>2 and fdist.get(key)<15:
6             wordlist.append(key+':'+str(fdist.get(key)))
7     return wordlist
```

运行结果：

```
=== 打印统计的词频 ===
['受伤:3', '大雨:5', '沈阳:8', '国奥队:8', '雨水:3', '下午:4', '训练:13',
'队员:3', '当天:3', '队伍:3', '球员:4', '感冒:4', '冯潇霆:4', '长春:3']
```

6.7 自定义规则提取特征词

特征词中有一类词特别重要，这类词指的是实体，包括人名、地名、机构名、数字、日期等，往往可以提供很大的信息价值。因此，会出现一个专门的研究方向叫作命名实体识别。可以借助上文的词性对命名实体进行简单的选择，比如需要提取人名、地名、机构团体和其他专有名词，对应的只需要选择['nr', 'ns', 'nt', 'nz']即可，编写代码如下（源代码见：Chapter6/FeatureWord.py）：

```
1 import jieba.posseg as ps
2 # 不同业务场景:评论情感判断，可以自定义特征抽取规则
3 def extract_feature_words(str_doc):
4     featWord = ""
5     stwlist = get_stop_words()
6     user_pos_list = [ 'nr', 'ns', 'nt', 'nz'] # 用户自定义特征词性列表
7     for word, pos in ps.cut(str_doc):
8         # 过滤掉停用词
9         if word not in stwlist and pos in user_pos_list:
10             if word+' '+pos+'\n' not in featWord:
11                 featWord += word+' '+pos+'\n'
12     print(featWord)
```

运行结果如下：

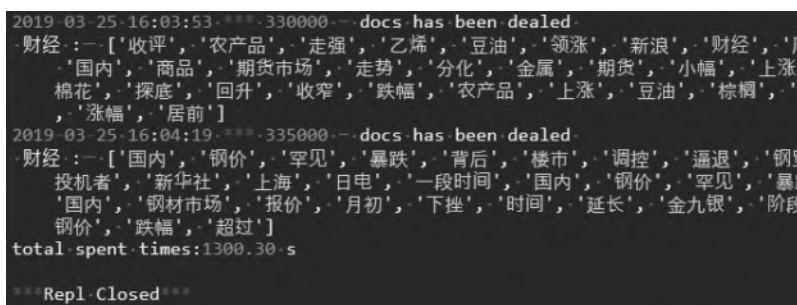
```
马晓旭 nr
国奥 nz
傅亚雨 nr
沈阳 ns
国奥队 nt
奥体中心 nt
冯萧霆 nr
长春 ns
非战斗 nz
```

6.8 实战案例：新闻文本分词处理

本案例直接调用loadFiles方法即可。本案例代码与第5章综合实战变化不大，核心变化是seg_doc(content)方法对抽取文本的分词的处理，依然遍历执行所有文件，并每隔5000个处理文件打印一条完成信息。30万条新闻信息分词处理的执行代码如下（源代码见：Chapter6/30wDealText.py）：

```
1  if __name__=='__main__':
2      start = time.time()
3      filepath = os.path.abspath(r'../Corpus/CSCMNews')
4      files = loadFiles(filepath)
5      n = 5 # n 表示抽样率
6      for i, msg in enumerate(files):
7          if i % n == 0:
8              catg = msg[0]
9              file = msg[1]
10             file = seg_doc(file)
11             if int(i/n) % 1000 == 0:
12                 print('{t} *** {i} \t docs has been dealt'
13                       .format(i=i, t=time.strftime
14                               ('%Y-%m-%d %H:%M:%S',time.localtime()))),'\n',catg,':\t',file[:30])
15             end = time.time()
16             print('total spent times:%.2f' % (end-start)+ ' s')
```

运行结果如图6-20所示。



```
2019-03-25-16:03:53-***-330000--docs has been dealt
财经：-['收评', '农产品', '走强', '乙烯', '豆油', '领涨', '新浪', '财经', '
国内', '商品', '期货市场', '走势', '分化', '金属', '期货', '小幅', '上涨
棉花', '探底', '回升', '收窄', '跌幅', '农产品', '上涨', '豆油', '棕榈', '
', '涨幅', '居前']
2019-03-25-16:04:19-***-335000--docs has been dealt
财经：-['国内', '钢价', '罕见', '暴跌', '背后', '楼市', '调控', '逼退', '钢
投机者', '新华社', '上海', '日电', '一段时间', '国内', '钢价', '罕见', '暴
国内', '钢材市场', '报价', '月初', '下挫', '时间', '延长', '金九银', '阶段
钢价', '跌幅', '超过']
total spent times:1300.30 s
*** Repl Closed***
```

图6-20 30万新闻文本分词处理的结果

6.9 本章小结

本章介绍中文分词的概况，并重点分析了结巴分词和HanLP分词，并结合NLTK技术对分词后的特征词进行处理。最后，综合以上知识点完成新闻文本数据分词的处理工作。

在分词完成后，还需要将分词结果进行向量化，这将是下一章重点讲解的内容。

第7 章

文本特征向量化

文本特征向量化属于数据预处理的重要内容，其根本目的就是 will 文本数据进行向量化，便于算法模型对数据进行处理。本章首先介绍特征集和标签集以及解析数据中缺失值的处理问题与对策，然后，着重介绍归一化处理方法，最后，对比词集模型和词袋模型完成新闻文本特征的向量化。

7.1 解析数据文件

经过数据的预处理、分词、缺失值处理等等一系列工作之后，得到了相对高效的文本数据，接下来需要将数据输入算法模型中，对具体的算法模型进行训练和改进。此时，需要进行特征集和标签集的拆分，也就是解析数据文件，其目的是将特征数据集用于训练算法模型，将预测的结果与标签集对比，然后进一步改进算法（源代码见：Chapter7/splitData.py）。

1. 数据准备

假设经过数据预处理工序得到如下的文本数据（数据文件为 dataset.txt）：

1	40920	8.326976	0.953952	3
2	14488	7.153469	1.673904	2
3	26052	1.441871	0.805124	1
4	...	...	...	...
5	16633	6.956372	1.519308	2
6	13887	0.636281	1.273984	2
7	52603	3.574737	0.075163	1

其中前3列为特征集，分别表示：特征1，年均投入时间（min）；特征2，玩游戏占时间百分比；特征3，每天看综艺的时间（h）。最后一列表示标签集，即：1表示学习专注；2表示学习正常；3表示比较贪玩。

2. 解析特征集与标签集

编写以下代码，对上述数据文件进行特征集与标签集解析。

```
1 def file_matrix(filename):
2     f = open(filename)
3     arrayLines = f.readlines()
4     returnMat = zeros((len(arrayLines),3))    # 特征数据集
5     classLabelVector = []                    # 标签集
6     index = 0
7     for line in arrayLines:
8         listFromLine = line.strip().split('\t')    # 分析数据，空格处理
9         returnMat[index,:] = listFromLine[0:3]
10        classLabelVector.append(int(listFromLine[-1]))
11        index +=1
12    return returnMat,classLabelVector
```

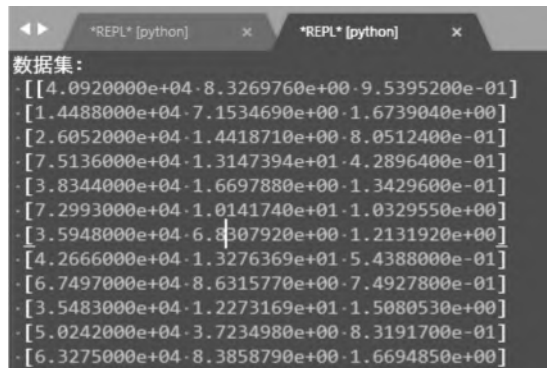
上述代码，首先调用zeros方法构建一个特征集等大的矩阵，类似的创建一个标签集的空列表；然后，对每一行数据进行拆分，并将前3列数据存放在特征数据矩阵中，最后一列的特征存储在标签集里面。

3. 执行主函数

编写以下执行主函数的代码：

```
1  if __name__ == '__main__':  
2      path = os.path.abspath(r'../Files/dataset.txt')  
3      returnMat,classLabelVector=file_matrix(path)  
4      print('数据集:\n',returnMat,'\n 标签集:\n',classLabelVector)
```

运行结果如图7-1所示。



```
数据集:  
[[4.092000e+04 8.326976e+00 9.539520e-01]  
[1.448800e+04 7.153469e+00 1.673904e+00]  
[2.605200e+04 1.441871e+00 8.051240e-01]  
[7.513600e+04 1.314739e+01 4.289640e-01]  
[3.834400e+04 1.669788e+00 1.342960e-01]  
[7.299300e+04 1.014174e+01 1.032955e+00]  
[3.594800e+04 6.830792e+00 1.213192e+00]  
[4.266600e+04 1.327636e+01 5.438800e-01]  
[6.749700e+04 8.631577e+00 7.492780e-01]  
[3.548300e+04 1.227316e+01 1.508053e+00]  
[5.024200e+04 3.723498e+00 8.319170e-01]  
[6.327500e+04 8.385879e+00 1.669485e+00]
```

图7-1 拆分特征集与标签集

7.2 处理缺失值

7.2.1 什么是数据缺失值

数据缺失值是指粗糙数据中由于缺少信息而造成的数据的聚类、分组、删失或截断。它指的是现有数据集中某个或某些属性的值是不完全的。

1. 数据缺失的原因

- 手工误操作。即网页数据由于人工输入造成的客观错误。

- 信息暂时无法获取。例如在医疗数据库中，并非所有病人的所有临床检验结果都能在给定的时间内得到，致使一部分属性值空缺出来。
- 信息被遗漏。可能是因为输入时认为不重要、忘记填写了或对数据理解错误而遗漏，也可能是由于数据采集设备的故障、存储介质的故障、传输媒体的故障、一些人为因素等原因而丢失了。
- 某个或某些属性不可用。对于一个对象来说，该属性值是不存在的，如一个未婚者的配偶姓名、一个儿童的固定收入状况等。
- 信息不重要。如一个属性的取值与给定语境是无关的，或训练数据库的设计者并不在乎某个属性的取值。
- 获取信息的代价太大。
- 实时性能要求较高，即要求得到这些信息前迅速做出判断或决策。

2. 数据缺失带来的问题

数据缺失在许多研究领域都是一个复杂的问题。对数据挖掘来说，缺失值的存在，会造成以下问题：

- 造成系统丢失大量的有用信息。
- 系统中所表现出的不确定性更加显著，系统中蕴涵的确定性成分更难把握。
- 包含空值的数据会使挖掘过程陷入混乱，导致不可靠的输出。

3. 缺失值处理方法

处理不完整数据集的方法主要有三大类：删除元组、数据补齐、不处理。其中数据补齐法包括：

- 人工填写
- 特殊值填充
- 平均值填充
- 热卡填充
- K最近距离邻法
- 组合完整化方法
- 回归
- 期望值最大化方法
- 多重填补

7.2.2 均值法处理数据缺失值

均值法处理缺少值是一种常见的处理手段，下面介绍均值法处理数据缺失值的步骤和代码实现（源代码见：Chapter7/lossval.py）。

1. 加载数据集

首先编写下述代码加载数据集：

```
1 def loadDataSet(fileName, delim='\t'):  
2     fr = open(fileName)  
3     stringArr = [line.strip().split(delim) for line in  
4                   fr.readlines()]  
5     datArr = [list(map(float, line)) for line in stringArr]  
6     return mat(datArr)
```

代码说明：

loadDataSet方法提供两个参数，其中fileName指待处理文件路径，属于必选参数；delim是每行文本信息分隔符，属于可选参数，默认是制表符。本段代码最终返回的是mat类型数据，之所以将列表数据转化为mat方法处理后的数据，主要是便于NumPy操作。

2. NaN替换成平均值

然后，编写下述代码将NaN替换成平均值函数。

```
1  '''将 NaN 替换成平均值函数'''
2  def replaceNanWithMean():
3      datMat = loadDataSet('../Files/dataset.data', ' ')
4      numFeat = shape(datMat)
5      for i in range(numFeat[1]-1):
6          # 对 value 不为 NaN 的求均值，A 返回表示矩阵的数组
7          meanVal = mean(datMat[nonzero(~isnan(datMat[:, i].A))[0], i])
8          # 将 value 为 NaN 的值赋值为均值
9          datMat[nonzero(isnan(datMat[:, i].A))[0], i] = meanVal
10     return datMat
```

代码说明：

本段代码的核心思想是找到每列数据非0、非NaN数据的均值，然后对缺少的数据进行均值填充。不容易理解的是for循环内的代码，这里先找到datMat[:, i]即特征数据的列并转化为数组，然后筛选出该列非0、非NaN数据并进行均值处理，最后将均值填充到缺失值对应的位置。

3. 执行主函数

最后，编写主函数执行代码如下：

```
1  if __name__ == '__main__':
2      # 加载数据集
3      loadDataSet('../Files/dataset.data', ' ')
4      # 均值填补缺失值
5      datMat = replaceNanWithMean()
6      print(datMat)
```

运行结果：

对dataset.txt修改为dataset.data包括数据缺失值，运行结果如图7-2所示。

```
[[ nan 8.3269760e+00 9.5395200e-01 3.0000000e+00]
 [1.4488000e+04 7.1534690e+00 1.6739040e+00 2.0000000e+00]
 [2.6052000e+04 nan 8.0512400e-01 1.0000000e+00]
 [7.5136000e+04 1.3147394e+01 4.2896400e-01 1.0000000e+00]
 [3.8344000e+04 1.6697880e+00 1.3429600e-01 1.0000000e+00]
 [7.2993000e+04 nan 1.0329550e+00 1.0000000e+00]
 [3.5948000e+04 6.8307920e+00 1.2131920e+00 3.0000000e+00]
 [4.2666000e+04 1.3276369e+01 nan 3.0000000e+00]
 [6.7497000e+04 nan nan 1.0000000e+00]
 [3.5483000e+04 1.2273169e+01 1.5080530e+00 3.0000000e+00]]
```

图7-2 带有数据缺失值的数据集

NumPy均值法处理数据缺失值的运行结果如图7-3所示。

```
[[3.33022113e+04 8.32697600e+00 9.53952000e-01 3.00000000e+00]
 [1.44880000e+04 7.15346900e+00 1.67390400e+00 2.00000000e+00]
 [2.60520000e+04 6.56491342e+00 8.05124000e-01 1.00000000e+00]
 [7.51360000e+04 1.31473940e+01 4.28964000e-01 1.00000000e+00]
 [3.83440000e+04 1.66978800e+00 1.34296000e-01 1.00000000e+00]
 [7.29930000e+04 6.56491342e+00 1.03295500e+00 1.00000000e+00]
 [3.59480000e+04 6.83079200e+00 1.21319200e+00 3.00000000e+00]
 [4.26660000e+04 1.32763690e+01 8.22827255e-01 3.00000000e+00]
 [6.74970000e+04 6.56491342e+00 8.22827255e-01 1.00000000e+00]
 [3.54830000e+04 1.22731690e+01 1.50805300e+00 3.00000000e+00]
 [5.02420000e+04 3.72349880e+00 8.31917000e-01 1.00000000e+00]]
```

图7-3 均值法处理数据缺失值

7.2.3 Pandas处理缺失值

使用Python的数据处理模块Pandas来处理数据缺失值更为方便，在实际的数据分析工作中经常使用。接下来介绍Pandas处理缺失值的步骤和代码实现。

首先，编写下述代码加载数据集：

```
1 import pandas as pd
2 import numpy as np
3
4 datMat = loadDataSet('../Files/dataset.data', ' ')
5 df = pd.DataFrame(datMat)
6 print (df) # 打印矩阵
```

代码说明：

本段代码调用数据科学计算库中的Pandas和NumPy，其中需要说明的是NumPy在数据预处理过程中，经常与其他库函数配合使用。这里需要调用DataFrame方法填充矩阵数据即二维数据（具体细节，请回顾第2章的内容），最终打印出来的结果，如图7-4所示。

```
136 -54495.0 -1.523057 -0.620528 -1.0
137 -11844.0 -2.749006 -0.169855 -2.0
138 -45465.0 -9.235393 -0.188350 -3.0
139 -31033.0 -10.555573 -0.403927 -3.0
140 -16633.0 -6.956372 -1.519308 -2.0
141 -13887.0 -0.636281 -1.273984 -2.0
142 -52603.0 -3.574737 -0.075163 -1.0

[143 rows x 4 columns]

*** Repl Closed ***
```

图7-4 Pandas加载数据集

1. 均值填充法

上文介绍了均值填充法的示例，本节介绍另一种Pandas实现的均值填充法，编写如下代码：

```
1 lossVs = [df[col].mean() for col in range(datMat.shape[1])] # 计算
    特征列均值
2 lists= [ list(df[i].fillna(lossVs[i])) for i in range(len(lossVs)) ]
3 print(mat(lists).T)
```

代码说明：

第一行代码是计算每列的均值并保存在lossVs里面，然后调用Pandas内置的fillna方法，依次将各列的均值填充到缺失值的位置。

最后经过矩阵转置处理得到如图7-5所示的结果。

```
[[3.33022113e+04 8.32697600e+00 9.53952000e-01 3.00000000e+00]
 [1.44880000e+04 7.15346900e+00 1.67390400e+00 2.00000000e+00]
 [2.60520000e+04 6.56491342e+00 8.05124000e-01 1.00000000e+00]
 [7.51360000e+04 1.31473940e+01 4.28964000e-01 1.00000000e+00]
 [3.83440000e+04 1.66978800e+00 1.34296000e-01 1.00000000e+00]
 [7.29930000e+04 6.56491342e+00 1.03295500e+00 1.00000000e+00]
 [3.59480000e+04 6.83079200e+00 1.21319200e+00 3.00000000e+00]
 [4.26660000e+04 1.32763690e+01 8.22827255e-01 3.00000000e+00]
 [6.74970000e+04 6.56491342e+00 8.22827255e-01 1.00000000e+00]
 [3.54830000e+04 1.22731690e+01 1.50805300e+00 3.00000000e+00]
 [5.02420000e+04 3.72349800e+00 8.31917000e-01 1.00000000e+00]]
```

图7-5 Pandas均值填充缺失值

2. 标量值替换法

除了均值填充法以外，还有一种常用方法是标量值替换法。也就是说采用一个对数据结果影响不大的标量值填充到缺失值里面。该方法的标量值的选择对结果影响不是很大，对数据精度要求不是很高的项目比较适用。编写如下代码：

```
1 print ("NaN replaced with '0':")
2 print (df.fillna(0))
```

本例用标量值0进行缺失值的填充，运行结果如下：

	0	1	2	3
0	0.0	8.326976	0.953952	3.0
1	14488.0	7.153469	1.673904	2.0
2	26052.0	0.000000	0.805124	1.0
3	75136.0	13.147394	0.428964	1.0

3. 向前和向后填充法

向前填充法是指在缺失值位置填充对应到列的前一个值，使用的是内置函数fillna，并将参数设为pad即可。向后填充法原理是一样的，不同的是将参数值设为backfill。实现代码如下：

```
1 print (df.fillna(method='pad'))
2 print (df.fillna(method='backfill'))
```

以下是向后填充法的示例。其中第一行第一列位置本身是缺失值，这里采用该列下的14488.0填充，部分结果如下所示：

	0	1	2	3
0	14488.0	8.326976	0.953952	3.0
1	14488.0	7.153469	1.673904	2.0
2	26052.0	13.147394	0.805124	1.0
3	75136.0	13.147394	0.428964	1.0
4	38344.0	1.669788	0.134296	1.0
5	72993.0	6.830792	1.032955	1.0

4. 忽略无效值法

该方法比较容易理解，就是将无效值即缺失值直接忽略。实现代码如下：

```
1 print("df.dropna():\n{}\n".format(df.dropna()))
```

代码中调用了内置函数dropna，运行结果如下所示：

```
138 45465.0  9.235393  0.188350  3.0
139 31033.0 10.555573  0.403927  3.0
140 16633.0  6.956372  1.519308  2.0
141 13887.0  0.636281  1.273984  2.0
142 52603.0  3.574737  0.075163  1.0
[137 rows x 4 columns]
```

7.3 数据的归一化处理

归一化是指把需要处理的数据经过处理后限制在0~1之间的统计的概率分布。归一化处理的目的是让算法收敛的更快，以提升模型拟合过程的计算效率。归一化后有两个好处：一是归一化后加快了梯度下降求最优解的速度；二是归一化有可能提高精度。

7.3.1 不均衡数据分析

选取dataset.txt的部分数据来分析不均衡数据对模型结果的影响。选择第一行数据即类别3，然后再选择第3、4行数据均属于类别1，其数据形式如表7-1所示。

表7-1 不均衡数据分布

采样编号	投入时间(m)	游戏占比(%)	综艺时间(h)	类 别
1	40920	8.33	0.95	贪玩
3	26052	1.44	0.81	专注
4	75136	13.15	0.43	专注

分析以上数据存在的几个问题：第一，数据比较的单位不同，有时间单位和百分比；第二，数值差距较大。这样，投入时间的特征权重将远远大于其他几个特征，很可能造成预测准确率低的问题。采用一个简单的欧式距离来进行一下测算，欧式距离公式为：

$$Dist = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + \dots + (n_1 - n_2)^2}, (n \in R) \quad (1)$$

得到样本3到样本4的距离：

$$D_{12} = \sqrt{(26052 - 75136)^2 + (1.44 - 13.147)^2 + (0.81 - 0.43)^2} = 49084.0013972 \quad (2)$$

样本3到样本1的距离为：

$$D_{31} = 14868.0015949 \quad (3)$$

从结果来看， $D_{31} < D_{12}$ ，即样本3与样本1距离较小，应该划归为一类数据。实际结果是样本3与样本4是同一类数据——都属于专注学习，这是由于数据不均衡造成的。由此，需要对数据进行规范化处理，常用的技术手段之一就是对数据进行归一化处理。

7.3.2 归一化的原理

数据归一化即特征值转化为0~1之间，其原理是：

$$\text{newValue} = (\text{oldValue} - \text{min}) / (\text{max} - \text{min})$$

代码实现如下（源代码见：Chapter7/normdata.py）：

```
1 def norm_dataset(dataset):
2     minVals = dataset.min(0) # 参数0列中最小值
3     maxVals = dataset.max(0)
4     ranges = maxVals - minVals
5     normdataset = zeros(shape(dataset)) # 原矩阵一样大小的0矩阵
6     m = dataset.shape[0]
7
8     # tile:复制同样大小的矩阵
9     molecular = dataset - tile(minVals, (m,1)) # 分子: (oldValue-min)
10    Denominator = tile(ranges, (m,1)) # 分母: (max-min)
11    normdataset = molecular / Denominator # 归一化结果
12
13    print('归一化的数据结果: \n'+str(normdataset))
14    return normdataset, ranges, minVals
```

执行归一化方法之后，所有的数据均落在了0~1之间，请看如下的结果：

```
[[0.42790599 0.5820766 0.56097745]
 [0.18615887 0.50004551 0.98770873]
 [0.33474674 0.45890399 0.47276384]
 [0.96543572 0.91903597 0.24980559]
 [0.49268882 0.11672239 0.07514943]
 [0.93789993 0.45890399 0.60780425]]
```

7.3.3 归一化的优点

对不同特征维度的数据进行伸缩变换的目的是使各个特征维度对目标函数的影响权重是一致的，即使得那些扁平分布的数据伸缩变换成类圆形，这样就改变了原始数据的分布。归一化是对数据的数值范

围进行特定缩放，但不改变其数据分布的一种线性特征变换。如图7-6所示是归一化前后的数据分布（此图来源于网络）。

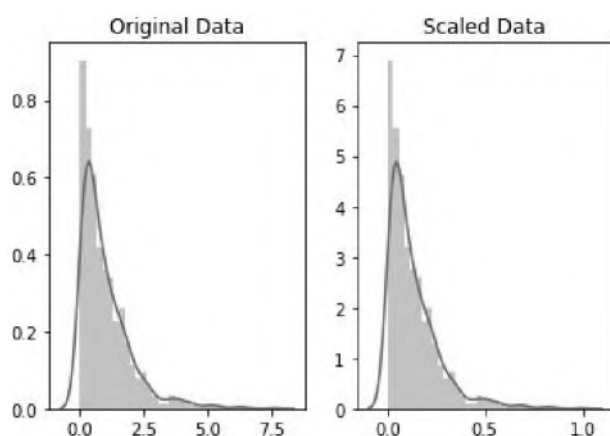


图7-6 归一化前后数据分布图

此外，归一化最显著的优点还包括：

- 提高迭代求解的收敛速度。
- 提高迭代求解的精度。

注意，并不是所有数据都需要归一化处理，也不是归一化处理都可以提高模型的性能。比如0/1取值的数据特征就不需要归一化处理，因为归一化会破坏它的稀疏性，类似的包括决策树和基于平方损失的最小二乘法也不需要归一化处理。

7.4 特征词转文本向量

由于原始数据往往并不是数值型的，这样就很难直接使用缺失值和归一化方法。本节介绍如何将文本特征的数据进行向量化处理，结合前文的分词结果进行思考，假设有两个类别的文档，其内容如下：


```
1 文档 1: 我来到成都,成都春熙路玩得很开心
2 文档 2: 成都的小酒馆很出名
3
```

首先对该文档进行数据清洗、分词及停用词处理，得到如下结果：

```
1 文档 1: ['我','来到','成都','成都','春熙路','玩','很','开心']
2 文档 2: ['成都','小','酒馆','很','出名']
3
```

然后，统计所有文档的特征词集即包含所有文档的不重复特征词，得到如下结果：

```
1 [我, 来到, 成都, 春熙路, 玩, 很, 开心, 小, 酒馆, 出名]
2
```

接下来，统计文本词频如下：

```
1 文档 1:
2 [我 1, 来到 1, 成都 2, 春熙路 1, 玩 1, 很 1, 开心 1, 小 0, 酒馆 0, 出名 0]
3 文档 2:
4 [我 0, 来到 0, 成都 1, 春熙路 0, 玩 0, 很 1, 开心 0, 小 1, 酒馆 1, 出名 1]
```

上面的列表中分别显示了各个文档中特征词对应的词频数，接下来只需要提取词频即可：

```
1 文档 1: [1,1,2,1,1,1,1,0,0,0]
2 文档 2: [0,0,1,0,0,1,0,1,1,1]
3
```

以上就是文本特征词转化为向量的整个过程，下一节进行具体的模型训练。

7.5 词频-逆词频（TF-IDF）

在模型训练的过程中，不能只是根据词频数判断词特征的权重，比如，在体育类别的文章中“进球”“比分”等特征词更具有代表性，比一般的“很棒”“漂亮”等描述性词语权重要高，这种情况下，单独以词频判断词特征权重就显得不合理。为了解决这个问题，这里引入“词频-逆词频（TF-IDF）”这个概念，关于TF-IDF的说明如下：

词频TF=某个词在文章中的出现次数 / 文章总词数

逆文档频率IDF=log（语料库的文档总数 / （包含该词的文档总数+1））

词频-逆词频应用领域比较广泛，常见的如信息检索、提取关键字、文章相似度查重、自动文摘等方面均有应用。

以下来实现文档的向量化，本节重点讲述运行原理，因此语料样例比较小，读者可以根据需求增加语料。这里主要对两个短文本语料进行处理。语料文本如下（源代码见：Chapter7/TFIDF.py）：

```
1 corpus = ["我 来到 成都 成都 春熙路 很 开心",  
2          "今天 在 宽窄巷子 耍 了 一天 ",]  
3
```

使用机器学习库sklearn来实现TF-IDF，具体代码如下：

```

1  from sklearn.feature_extraction.text import TfidfTransformer
2  from sklearn.feature_extraction.text import CountVectorizer
3  # 使用 sklearn 计算 tfidf 值特征
4  def sklearn_tfidf_feature(corpus=None):
5      # 构建词汇表
6      vectorizer = CountVectorizer()
7      # 该类会统计每个词语的 tf-idf 权重值
8      transformer = TfidfTransformer()
9      # 第一个 fit_transform 是计算 tf-idf, 第二个 fit_transform 是将文本转
      换为词频矩阵
10     tfidf = transformer.fit_transform
      (vectorizer.fit_transform(corpus))
11     # 获取词袋模型中的所有词语
12     word = vectorizer.get_feature_names()
13     # 元素 a[i][j] 表示 j 词在 i 类文本中的 tf-idf 权重
14     weight = tfidf.toarray()
15     # 第一个 for 遍历所有文本, 第二个 for 遍历某一类文本下的词语权重
16     for i in range(len(weight)):
17         print(u"-----这里输出第", i, u"类文本的词语 tf-idf 权重-----")
18         for j in range(len(word)):
19             print(word[j], weight[i][j])
20
21  if __name__ == '__main__':
22     # corpus 参数样例数据如下:
23     corpus = ["我 来到 成都 成都 春熙路 很 开心",
24              "今天 在 宽窄巷子 耍 了 一天 ",
25              "成都 整体 来说 还是 挺 安逸 的",
26              "成都 的 美食 真 巴适 惨 了"]
27     sklearn_tfidf_feature(corpus)

```

上述代码中，sklearn文本特征抽取的两个方法分别用来构建词汇表和统计TF-IDF值，代码的运行结果如图7-7所示。

```

Building prefix dict from the default dictionary...
Loading model from cache C:\Users\ADMINI~1\AppData\Local\Temp\jieba.cache
Loading model cost 0.875 seconds.
Prefix dict has been built successfully.
这里输出第 0 类文本的词语tf-idf权重
一天 0.0
今天 0.0
安逸 0.0
宽窄巷子 0.0
巴适 0.0
开心 0.4647574100194299
成都 0.5932972690785747
整体 0.0
春熙路 0.4647574100194299
来到 0.4647574100194299
来说 0.0
美食 0.0
还是 0.0
这里输出第 1 类文本的词语tf-idf权重
一天 0.5773502691896257

```

图7-7 sklearn计算各类词频-逆词频

7.6 词集模型与词袋模型

1. 打印数据集和标签

前文只是处理了两个简短的文本，本节使用真实的新闻数据，编写代码实现文本向量化和TF-IDF值的计算方法。以下代码抽样选择“体育”“娱乐”“教育”“时政”4个类别的文本信息：

```

1  from numpy import *
2  import numpy as np
3
4  '''创建数据集: 词列表 postingList, 所属类别 classVec'''
5  def loadDataSet():
6      corpus = []
7      tiyu = ['姚明', '我来', '承担', '连败', '巨人', '宣言', '酷似', '当年', '麦蒂', '新浪', '体育讯', '北京', '时间', '消息', '休斯敦', '纪事报', '专栏', '记者', '乔纳森', '费根', '报道', '姚明', '渴望', '一场', '胜利', '当年', '队友', '麦蒂', '惯用', '句式']
8      yule = ['谢婷婷', '模特', '酬劳', '仅够', '生活', '风光', '背后', '惨遭', '拖薪', '新浪', '娱乐', '金融', '海啸', 'blog', '席卷', '全球', '模特儿', '酬劳', '被迫', '打折', '全职', 'Model', '谢婷婷', '业界', '工作量', '有增无减', '收入', '仅够', '糊口', '拖薪']
9      jioayu = ['名师', '解读', '四六级', '阅读', '真题', '技巧', '考前', '复习', '重点', '历年', '真题', '阅读', '听力', '完形', '提升', '空间', '天中', '题为', '主导', '考过', '六级', '四级', '题为', '主导', '真题', '告诉', '方向', '会考', '题材', '包括']
10     shizheng = ['美国', '军舰', '抵达', '越南', '联合', '军演', '中新社', '北京', '日电', '杨刚', '美国', '海军', '第七', '舰队', '三艘', '军舰', '抵达', '越南', '岷港', '为期', '七天', '美越', '南海', '联合', '军事训练', '拉开序幕', '美国', '海军', '官方网站', '消息']
11     corpus.append(tiyu)
12     corpus.append(yule)
13     corpus.append(jioayu)
14     corpus.append(shizheng)
15     classVec = ['体育', '娱乐', '教育', '时政']
16     return corpus, classVec
17
18 if __name__ == '__main__':
19     # 1 打印数据集和标签
20     dataSet, classlab = loadDataSet()
21     print('数据集:\n', mat(dataSet), '\n 标签集:\n', mat(classlab))

```

执行主函数加载数据集和标签集，运行结果如图7-8所示。

2. 获取所有词的集合

根据上一节对词向量转化的介绍进行词汇表的抽取，代码如下：

```
数据集：
[['姚明', '我来', '承担', '连败', '巨人', '宣言', '酷似', '当年', '麦蒂', '新浪', '体育讯',
  '北京', '时间', '消息',
  '休斯敦', '纪事报', '专栏', '记者', '乔纳森', '费根', '报道', '姚明', '渴望', '一场', 'L',
  '当年', '队友',
  '麦蒂', '惯用', '句式'],
 ['谢婷婷', '模特', '酬劳', '仅够', '生活', '风光', '背后', '惨遭', '拖薪', '新浪', '娱乐',
  '金融', '海啸',
  'blog', '席卷', '全球', '模特儿', '酬劳', '被迫', '打折', '全职', 'Model', '谢婷婷', 'L',
  '工作量',
  '有增无减', '收入', '仅够', '糊口', '拖薪'],
 ['名师', '解读', '四六级', '阅读', '真题', '技巧', '考前', '复习', '重点', '历年', '真题',
  '阅读', '听力', '完形',
  '提升', '空间', '天中', '题为', '主导', '考过', '六级', '四级', '题为', '主导', '真题',
  '方向', '会考',
  '题材', '包括'],
 ['美国', '军舰', '抵达', '越南', '联合', '军演', '中新社', '北京', '日电', '杨刚', '美国',
  '海军', '第七', '舰队',
  '三艘', '军舰', '抵达', '越南', '岷港', '为期', '七天', '美越', '南海', '联合', '军事训',
  '拉开序幕', '美国',
  '海军', '官方网站', '消息']]
标签集：
[['体育', '娱乐', '教育', '时政']]
```

图7-8 加载数据集与标签集

```
1  '''获取所有词的集合:返回不含重复元素的词列表'''
2  def createVocabList(dataSet):
3      vocabSet = set([])
4      for document in dataSet:
5          vocabSet = vocabSet | set(document) # 操作符 | 用于求两个集合
        的并集
6      # print(vocabSet)
7      return list(vocabSet)
8
9  if __name__ == '__main__':
10     # 2 获取所有词的集合
11     vocabList=createVocabList(dataSet)
12     print('\n 词汇列表: \n',vocabList)
```

执行词汇列表函数，运行结果如图7-9所示。

```
词汇列表：
['模特儿', '题材', '时间', '七天', '酬劳', '休斯敦', '谢婷婷', '麦蒂', '听力',
  '天中', '第七', '官方网站', '海军', '渴望', '主导', '酷似', '姚明', '真题',
  '完形', '包括', '美国', '生活', '队友', '金融', '被迫', '报道', '胜利', '连',
  '四六级', '四级', '有增无减', '工作量', '历年', '越南', '海啸', '背后', '承',
  '阅读', '题为', '专栏', '空间', '拉开序幕', '军演', 'blog', '纪事报', '收',
  '中新社', '巨人', '费根', '名师', '会考', '句式', '考前', '舰队', '体育讯',
  '惨遭', '消息', '岷港', '宣言', '技巧', '打折', '业界', '娱乐', '全球', '考',
  '日电', '乔纳森', '抵达', '席卷', '我来', '三艘', 'Model', '提升', '模特',
  '为期', '惯用', '北京', '美越', '六级', '联合', '告诉', '记者', '全职', 'L',
  '军事训练', '拖薪', '新浪', '重点', '南海', '解读', '一场', '杨刚', '仅够',
  '复习', '糊口', '军舰', '方向', '风光', '当年']
```

图7-9 获取数据文档的词汇集

3. 词集模型：文本向量化

词集模型的原理是遍历每个文档中的所有词，如果出现了词汇表中的词，则将输出文档向量中的对应值设为1，分别对各个文档进行词频统计，代码如下（源代码见：Chapter7/wordset.py）：

```
1 def setOfWords2Vec(vocabList, DataSet):
2     # 1 所有文档的词向量
3     VecList = []
4     for inputSet in DataSet:
5         # print('-->',inputSet) # 每个文档
6         # 2 创建一个和词汇表等长的向量，并将其元素都设置为 0
7         returnVec = [0] * len(vocabList)
8         # 如果词在词汇表则修正 1
9         for word in inputSet:
10             if word in vocabList:
11                 returnVec[vocabList.index(word)] = 1
12         # 追加所有文档词向量列表
13         VecList.append(returnVec)
14     return VecList
15
16 if __name__ == '__main__':
17     #*****特征词转换为向量*****
18     # 3 词集模型:文本向量化
19     setvec = setOfWords2Vec(vocabList, dataSet)
20     print('词集模型:\n',mat(setvec))
```

执行词集模型方法，得到如图7-10所示的结果。


```

1  def TFIDF(bagvec):
2      # 词频 = 某个词在文章中出现的总次数/文章中出现次数最多的词的个数
3      tf = [ doc/sum(doc) for doc in bagvec]
4      # 逆文档频率 (IDF) = log (词料库的文档总数/包含该词的文档数+1)
5      m = len(bagvec) # 词料库的文档总数
6      ndw =sum(mat(bagvec).T!=0,axis=1).T # 包含该词的文档数
7      idf = [ log(m/(t+1)) for t in ndw]
8      tfidf = tf * np.array(idf)
9      return tfidf
10 if __name__=='__main__':
11     # 4 tf-idf 计算
12     tfidf = TFIDF(bagvec)
13     print('tf-idf:\n ',tfidf)

```

运行上述代码，调用自定义TF-IDF函数，结果如图7-11所示。

```

tf idf:
[[[0. .... 0. .... 0.02567212 0. .... 0. .... 0.02567212
...0. .... 0.02567212 0. .... 0. .... 0. .... 0.
...0. .... 0.02567212 0. .... 0.02567212 0.02567212 0.
...0. .... 0. .... 0. .... 0. .... 0.02567212 0.
...0. .... 0.02567212 0.02567212 0.02567212 0. .... 0.
...0. .... 0. .... 0. .... 0. .... 0. .... 0.
...0.02567212 0. .... 0. .... 0.02567212 0. .... 0.
...0. .... 0. .... 0. .... 0.02567212 0. .... 0.
...0.02567212 0. .... 0.01065489 0. .... 0.02567212 0.
...0. .... 0. .... 0. .... 0. .... 0. .... 0.
...0.02567212 0. .... 0. .... 0.02567212 0. .... 0.
...0. .... 0. .... 0. .... 0.02567212 0.01065489 0.
...0. .... 0. .... 0. .... 0.02567212 0. .... 0.
...0. .... 0.01065489 0. .... 0. .... 0. .... 0.02567212
...0. .... 0. .... 0. .... 0. .... 0. .... 0.
...0. .... 0.02567212]]]

```

图7-11 新闻文档词向量化

7.7 实战案例：新闻文本特征向量化

本节完成一个新闻文本特征向量化的例子。本例在上一章批量新闻文本分词处理的基础上，调用bagOfWords2VecMN方法对分词结果进行词向量转化并调用TF-IDF方法进行优化处理，最后，每隔5000行打印一条文本向量化处理的结果。案例代码实现如下（源代码见：Chapter7/30wVec.py）：

```

1  if __name__=='__main__':
2      start = time.time()
3      filepath = os.path.abspath(r'../Corpus/CSCMNews')
4      files = loadFiles(filepath)
5      n = 5 # n 表示抽样率
6      for i, msg in enumerate(files):
7          if i % n == 0:
8              catg = msg[0]
9              content = msg[1]
10             # 每个文档的 TFIDF 向量化-->所有词集 (gensim)
11             word_list = seg_doc(content)
12             vocabList=createVocabList(word_list)
13             bagvec = bagOfWords2VecMN(vocabList, word_list)
14             tfidf = TFIDF(bagvec)
15             if int(i/n) % 1000 == 0:
16                 print('{t} *** {i} \t docs has been dealed'
17                       .format(i=i, t=time.strftime('%Y-%m-%d %H:%M:%S',
18               time.localtime()))), '\n', catg, ':\t', tfidf)
19         end = time.time()
20         print('total spent times:%.2f' % (end-start)+ ' s')

```

代码说明：

首先对新闻文本进行文件遍历并返回新闻的类别和新闻内容，存放在files中。然后，遍历files文件，其中枚举方法enumerate是为了获取每行新闻信息的序号。这里，采用抽样为5的样本信息即每隔5条新闻随机抽取一个样本信息，以保证样本的均衡性。接下来对新闻内容进行seg_doc分词和词袋模型bagOfWords2VecMN的构建，将其转化为词频-逆词频TF-IDF的矩阵列表。最后，每处理完5000条信息在屏幕上打印一条处理结果。

新闻文本特征向量化的处理结果如图7-12所示。

```
2019-03-25-16:57:11-***.335000.-docs.has.been.dealed  
·财经·:-[[[0.0.0....0.0.0.]  
··[0.0.0....0.0.0.]  
··[0.0.0.....0.0.0.]  
·····  
··[0.0.0.0.....0.0.0.]  
··[0.0.0.0.....0.0.0.]  
··[0.0.0.0.....0.0.0.]]]  
total.spent.times:2901.82.s  
  
***Repl.Closed***
```

图7-12 新闻文本特征向量化

7.8 本章小结

本章介绍了数据缺失值和归一化处理的常见方法，并通过实际案例进行了剖析。在对数据处理之前，因为需要将特征词进行向量化，所以引入词集模型和词袋模型，并采用词频-逆词频进行向量优化，最终完成了新闻文本的特征向量化。

在实际项目中，读者还可以借助一款开源的第三方Python处理工具包Gensim来实现文本的向量化，这是一款多功能文档处理“神器”，可以大大提升处理文本的效率，关于该工具的使用将在下一章进行详细讨论。

第8 章

Gensim文本向量化

Gensim是一款非结构化文本处理工具包，可以便捷地构建语料库，其内置的子模块可以高效地生成TF-IDF向量，并支持大数据文本批量处理。同时，其内置的多种主题模型可以满足多任务需求，大大节约了数据预处理训练的时间。本章主要介绍使用Gensim工具实现特征向量化模型及其二次开发的相关内容。

8.1 Gensim的特性和核心概念

Gensim是一款开源的第三方Python工具包，用于从原始的非结构化文本中有效地自动抽取语义主题。它支持语料处理、LSA（Latent Semantic Analysis，潜在语义分析），LDA（Latent Dirichlet Allocation，隐含狄利克雷分布），RP（Random Projection，随机映射）、TF-IDF、word2vec和paragraph2vec等多种主题模型算法，支持流式训练，并提供了诸如相似度计算和信息检索等一些常用任务的API接口。

Gensim主要有如下特性：

- 内存独立。对于训练语料来说，没必要将整个语料都驻留在内存中。
- 有效地实现了许多流行的向量空间算法。包括tf-idf、分布式LSA、分布式LDA以及RP，并且很容易添加新算法。
- 对流行的数据格式进行了IO封装和转换。

- 在其语义表达中，可以相似查询。
- 可以实现主题建模的可扩展软件框架。

在使用Gensim工具时，会涉及以下几个核心概念：

- 语料（Corpus）。指文档的集合，用于模型训练学习，故也称作训练语料。
- 向量（Vector）。指由一组文本特征构成的列表，是一段文本在Gensim中的内部表达。
- 稀疏向量（Sparse Vector）。通常可以略去向量中多余的0元素，此时，向量中的每一个元素是一个（key，value）的元组，此向量称为稀疏向量。
- 模型（Model）。是一个抽象的术语。定义了两个向量空间的变换（即从文本的一种向量表达变换为另一种向量的表达）。

8.2 Gensim构建语料词典

训练语料的预处理指的是将文档中原始的字符文本转换成Gensim模型所能理解的稀疏向量的过程。通常要处理的原生语料是一堆文档的集合，每一篇文档又是一些原生字符的集合。在交给Gensim的模型训练之前，需要将这些原生字符解析成Gensim能处理的稀疏向量的格式。由于语言和应用的多样性，Gensim没有对预处理的接口做出任何强制性的限定，通常需要先对原始的文本进行分词、去除停用词等操作，得到每一篇文档的特征列表。例如，在词袋模型中，文档的特征就是其包含的特征词，以下是语料（Corpus）数据：

```

1 tiyu = ['姚明', '我来', '承担', '连败', '巨人', '宣言', '酷似', '当年',
  '麦蒂', '新浪', '体育讯', '北京', '时间', '消息', '休斯敦', '纪事报', '
  专栏', '记者', '乔纳森', '费根', '报道', '姚明', '渴望', '一场', '胜利',
  '当年', '队友', '麦蒂', '惯用', '句式']
2 yule = ['谢婷婷', '模特', '酬劳', '仅够', '生活', '风光', '背后', '惨遭',
  '拖薪', '新浪', '娱乐', '金融', '海啸', 'blog', '席卷', '全球', '模
  特儿', '酬劳', '被迫', '打折', '全职', 'Model', '谢婷婷', '业界', '工作
  量', '有增无减', '收入', '仅够', '糊口', '拖薪']
3 jioayu = ['名师', '解读', '四六级', '阅读', '真题', '技巧', '考前', '
  复习', '重点', '历年', '真题', '阅读', '听力', '完形', '提升', '空间', '
  天中', '题为', '主导', '考过', '六级', '四级', '题为', '主导', '真题', '
  告诉', '方向', '会考', '题材', '包括']
4 shizheng = ['美国', '军舰', '抵达', '越南', '联合', '军演', '中新社', '
  北京', '日电', '杨刚', '美国', '海军', '第七', '舰队', '三艘', '军舰', '
  抵达', '越南', '岬港', '为期', '七天', '美越', '南海', '联合', '军事训练',
  '拉开序幕', '美国', '海军', '官方网站', '消息']

```

其中，语料的每一个元素对应一篇文档。接下来依然以词袋模型为例，调用Gensim提供的API建立语料特征的索引字典，并将文本特征的原始表达转化成词袋模型对应的稀疏向量的表达。具体代码如下：

```

1 from gensim import corpora
2 def gensim_Corpus(corpus=None):
3     dictionary = corpora.Dictionary(corpus)    # 词典
4     mycorpus = [dictionary.doc2bow(text) for text in corpus]
5                                     # 词袋模型
6     print(mycorpus[0])

```

运行结果如下：

```

[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 2), (8,
1), (9, 1), (10, 2), (11, 1), (12, 1), (13, 1), (14, 1), (15, 1), (16, 1),
(17, 1), (18, 1), (19, 1), (20, 1), (21, 1), (22, 1), (23, 1), (24, 1), (25,
1), (26, 2)]

```

于是得到了语料中每一篇文档对应的稀疏向量，向量的每一个元素代表了一个词（Word）在这篇文档中出现的次数。值得注意的是，特征词大于1的仅有3个。接下来删除停用词和仅出现一次的词，编写代码如下：

```
1 # 删除停用词和仅出现一次的词
2 once_ids = [tokenid for tokenid, docfreq in dictionary.dfs.items()
               if docfreq == 1]
3 dictionary.filter_tokens(once_ids)
4 # 消除 id 序列在删除词后产生的不连续的缺口
5 dictionary.compactify()
```

对处理后的特征词进行本地化存储，代码如下：

```
1 savePath = r'../Files/mycorpus.dict'
2 dictionary.save(savePath) # 把字典保存起来，方便以后使用
```

加载存储的特征词典，代码如下：

```
1 mydict = corpora.Dictionary.load(savePath)
2 print(mydict)
```

然后，打印出处理后的词典，结果如下：

```
1 Dictionary(3 unique tokens: ['北京', '新浪', '消息'])
```

8.3 Gensim统计词频特征

关于词频统计前面章节已有介绍，本节介绍基于Gensim的实现方法，仍然采用上一节的语料。以下首先对语料进行词典构建，然后根据词袋模型打印出每个词以及对应的词频，其实现代码如下：

```

1 from gensim import corpora
2 def gensim_Corpus(corpus=None):
3     dictionary = corpora.Dictionary(corpus)
4     dfs = dictionary.dfs # 词频词典
5     print('统计词频特征:')
6     for key_id, c in dfs.items():
7         print(dictionary[key_id], c)
8     return dictionary[key_id], c

```

词频统计的结果如下所示：

统计词频特征：

姚明 1

...

麦蒂 1

新浪 2

...

军事训练 1

拉开序幕 1

官方网站 1

Repl Closed

8.4 Gensim计算TF-IDF

在8.3节已经介绍了TF-IDF的原理和实现方法，在本节使用Gensim针对多类文本实现TF-IDF计算。

首先使用以下命令自定义语料模块加载数据集与标签集：

```

1 from mydict import *
2 corpus,classVec = loadDataSet()

```

之后，进行特征词频-逆词频处理，并实现本地序列化存储，代码如下：


```

1  from gensim import corpora, models
2  def gensim_Corpus(corpus=None, classVec=''):
3      dictionary = corpora.Dictionary(corpus)
4      # 转换成 doc_bow
5      doc_bow_corpus = [dictionary.doc2bow(doc_cut) for doc_cut in
6                          corpus]
7      # 生成 tfidf 特征
8      tfidf_model = models.TfidfModel(dictionary=dictionary) # 生成
9                          tfidf 模型
10     corpus_tfidf = {} # tfidf 字典
11     i=0 # 获取类别
12     for doc_bow in doc_bow_corpus:
13         file_tfidf = tfidf_model[doc_bow] # 词袋填充
14         catg = classVec[i] # 类别
15         tmp = corpus_tfidf.get(catg, [])
16         tmp.append(file_tfidf)
17         print(tmp)
18         if tmp.__len__() == 1: # 某篇文章成功，不成功则为空
19             corpus_tfidf[catg] = tmp
20             i+=1
21     # 本地序列化存储
22     catgs = list(corpus_tfidf.keys()) # ['体育', '娱乐', '教育', '时
23     政']
24     for catg in catgs:
25         savepath = r'../Files/tfidf_corpus'
26         corpora.MmCorpus.serialize('{f}{s}{c}.mm'.format
27                                     (f=savepath, s=os.sep, c=catg), corpus_tfidf.get(catg),
28                                     id2word=dictionary)

```

代码说明：

上述代码首先对语料进行Dictionary词典化处理，然后处理成词袋模型（即特征向量化处理）。接着调用models的TfidfModel方法逐个列表进行词频-逆词频计算。最后，将各个类别的词频-逆词频计算结果进行本地序列化存储，其中调用序列化方法serialize并提供各个类别的保存路径、词频-逆词频计算结果和词典。

成功执行后，得到如图8-1所示的运行结果。

```
统计词频特征：
姚明·1
我来·1
承担·1
连败·1
巨人·1
宣言·1
酷似·1
当年·1
麦蒂·1
新浪·2
```

图8-1 TF-IDF序列化存储

8.5 Gensim实现主题模型

8.5.1 主题模型

1. 什么是主题模型

主题模型可理解为一种在大量文档中发现其主题的无监督式学习技术。这些主题本质上十分抽象，即彼此相关联的词语构成一个主题，同样，在单个文档中可以有多多个主题。可以将主题模型的示意图理解为如图8-2所示（图片来源于网络）。

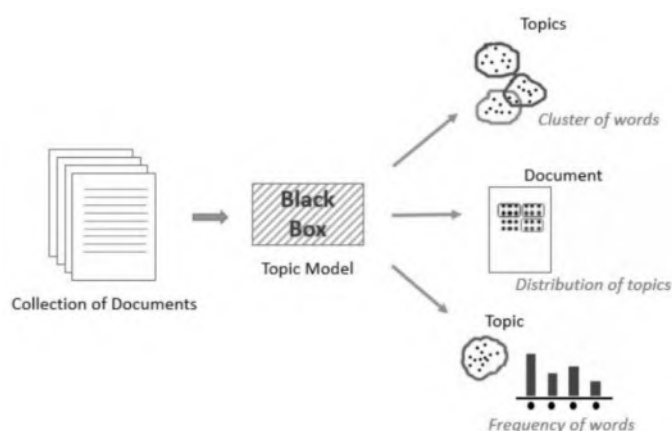


图8-2 主题模型示意图

图8-2将相似和相关的词语聚集成簇，称为主题。这些主题在文档中具有特定的分布，每个主题都根据它包含的不同词的比例来定义。

2. 主题模型的应用场景

假如要对电子文档进行归类整理，当文档的数量不太多，就可以手动完成。但是在文档数量非常多，人工远远不足以完成相关工作的情况下，主题建模就非常适用。

主题建模有助于挖掘海量文本数据、查找词簇、文本之间的相似度以及发现抽象主题。如果这些理由还不够引人注目，主题建模也可用于搜索引擎，判断搜索字段与结果的匹配程度。其中本章涉及的潜在语义分析（LDA）、隐含狄利克雷分布（LSA）和随机映射（RP）都属于主题模型的范畴。

8.5.2 潜在语义分析（LSA）

1. 什么是LSA

潜在语义分析（LSA）也叫作潜在语义索引（Latent Semantic Indexing, LSI）是一种用于知识获取和展示的计算理论和方法，出发点是文本中的词与词之间存在某种联系，即存在某种潜在的语义结构。这种潜在的语义结构隐含在文本词语的上下文使用模式中。因此采用统计计算的方法，对大量的文本中进行分析来寻找这种潜在的语义结构，它不需要确定的语义编码，仅依赖于上下文中事物的联系，并用语义结构来表示词和文本，达到消除词之间的相关性和简化文本向量的目的。

2. 理解LSA

语言学是一门复杂的学科，因此就自然语言处理而言也面临着诸多挑战，常见的难题之一就是歧义词的处理。比如一义多词和一词多义，这对机器而言很难捕捉，甚至有些场景下人类也很难区分。例如，以下两个句子：

陆庄主知道此人是湖南铁掌帮的帮主。

天晴的时候，在湖南钓鱼更容易上钩。

在第一个句子中，“湖南”指的是地名，而在第二个句子中，它的含义是湖的南边。人类能够轻松地区分这些词，是因为人类可以理解这些词背后的语境。但是机器并不能捕捉到这个概念，因为它不能理解词的上下文，这就是潜在语义分析（LSA）发挥作用的地方，它可以利用词语所在的上下文来捕捉隐藏的概念即主题。因此，简单地将词映射到文档并没有什么用，真正需要的是弄清楚词语背后的隐藏概念或主题，而LSA是一种可以发现这些隐藏主题的技术。

3. LSA的技术原理

假设有m篇文档，其中包含n个唯一词项（词），假设希望从所有文档的文本数据中提取出k个主题（k为主题数，k的取值根据实际需求由用户设置），则整个技术实现流程如下：

（1）生成 $m \times n$ 维文档-词矩阵，其中矩阵元素为TF-IDF，如图8-3所示。

		Terms					
		T1	T2	T3	...	Tn	
Documents	D1	0.2	0.1	0.5	...	0.1	
	D2	0.1	0.3	0.4	...	0.3	
	D3	0.3	0.1	0.1	...	0.5	
	...	...	...	...	...	...	
	Dm	0.2	0.1	0.2	...	0.1	

图8-3 文档-词矩阵

(2) 使用奇异值分解 (SVD) 把上述矩阵的维度降到k维。

(3) 使用降维后的矩阵构建潜在语义空间。SVD将一个矩阵分解为三个矩阵即 $A = USV^T$ ，如图8-4所示。矩阵 U_k (Document-Term Matrix) 的每个行向量代表相应的文档，这些向量的长度是k，是预期的主题数，代表数据中词的向量可以在矩阵 V_k (Term-Topic Matrix) 中找到。

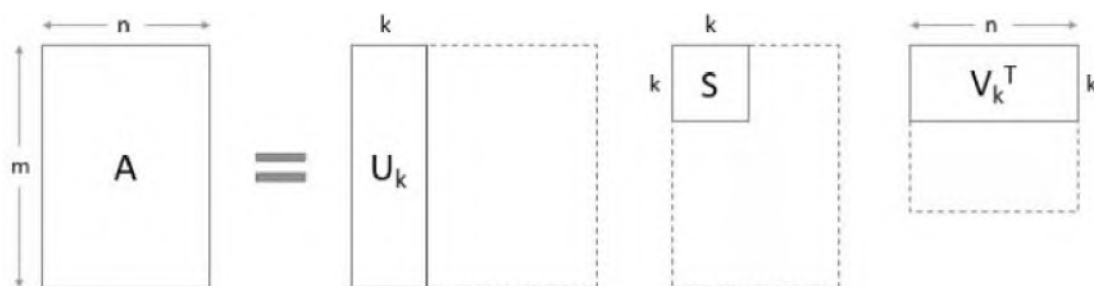


图8-4 SVD矩阵分解示意图

(4) SVD为数据中的每篇文档和每个词都提供了向量，每个向量的长度均为k。可以使用余弦相似度的方法通过这些向量找到相似的词和文档。

4. LSA模型的实现

使用8.2节构造的语料词典作为语料，采用Gensim中的LsiModel进行主题生成，代码实现如下（源代码见：Chapter8/LSA.py）：

```

1  from mydict import *
2  from gensim import corpora, models
3  import pickle as pkl
4
5  # 生成 lsi 特征 (潜在语义索引)
6  def gensim_Corpus(corpus=None):
7      dictionary = corpora.Dictionary(corpus)
8      # 转换成 doc_bow
9      doc_bow_corpus = [dictionary.doc2bow(doc_cut) for doc_cut in
10         corpus]
11      # 生成 tfidf 特征
12      tfidf_model = models.TfidfModel(dictionary=dictionary) # 生成
13         tfidf 模型
14      tfidf_corpus = [tfidf_model[doc_bow] for doc_bow in
15         doc_bow_corpus] # 将每 doc_bow 转换成对应的 tfidf_doc 向量
16      # print('TFIDF:\n',tfidf_corpus)
17
18      lsi_model = models.LsiModel(corpus=tfidf_corpus,
19         id2word=dictionary, num_topics=10) # 生成 lsi model
20
21      # 生成 corpus of lsi
22      lsi_corpus = [lsi_model[tfidf_doc] for tfidf_doc in tfidf_corpus]
23      # 转换成 lsi 向量
24      print('LSI:\n',lsi_corpus)
25      # 将 lsi 模型存储到磁盘上
26      savepath = r'../Files/lsi_model.pkl'
27      lsi_file = open(savepath, 'wb')
28      pkl.dump(lsi_model, lsi_file)
29      lsi_file.close()
30      print('--- lsi 模型已经生成 ---')

```

代码说明:

上述代码中, corpus为8.2节中构造的语料, 通过导入mydict模块, 使用corpus,classVec=loadDataSet()来加载数据。第一步是加载文档集合, 建立词汇-文本矩阵A, 主要是实现将原始语料的词袋模型转化, 最终生成TF-IDF矩阵。第二步对词汇-文本矩阵进行奇异值分解

并降维和SVD分解矩阵，幸运的是可以直接调用Gensim的models.LsiModel模块来实现。最后，需要对降维后的矩阵构建潜在语义空间并存储在本地磁盘中，以备后续使用，这样做的目的就是一次模型训练将来多次复用，从而提高了算法的性能。

5. LSA的实验结果

上述LSA主题模型构建完成以后，运行gensim_Corpus函数得到如图8-5所示结果，即输出各类文章最为重要的主题。

```
LSA生成主题:
[[ (0, -0.7122860429341998), (3, -0.7018893025549962)], [(0, -0.34158119641608475), (1, -0.8775111622043954), (3, -0.33659537498552544)], [(2, -0.9999999999999999)], [(0, -0.6250389533571518), (1, -0.4795562117277709), (3, -0.6159156976238757)]]
---lsi模型已经生成---
*** Repl Closed ***
```

图8-5 LSA生成主题

6. LSA的优缺点

LSA非常有用，但是也有其局限性。其优缺点如下：

(1) 优点

- 可以更好地刻画文本的真实含义。
- 分析词条与文本之间的相似关系时比传统的向量空间模型具有更好的灵活性。
- 用低维词条、文本向量代替原始的空间向量，可以有效地处理大规模的文本库。
- 对于原始的词条——文本矩阵，通过LSA分析提取出k维语义空间，在保留大部分信息的同时使得 $K \ll V$ 。
- 通过对大量的文本分析，LSA可以自动地模拟人类的知识获取能力，甚至分类、预测的能力。

- 低维空间表示可以刻画同义词，同义词会对应着相同或相似的主题。

(2) 缺点

- 因为它是线性模型，因此在具有非线性依赖性的数据集上可能效果不佳。
- LSA假设文本中的词服从正态分布，这可能不适用于所有问题。
- LSA涉及SVD，它是计算密集型的，当新数据出现时难以更新。
- 没有刻画term出现次数的概率模型。
- LSA具有词袋模型的缺点，即在一篇文章或者一个句子中忽略了词语的先后顺序。

8.5.3 隐含狄利克雷分布（LDA）

1. 什么是LDA

LDA是一种主题模型，它可以将文档集中每篇文档的主题按照概率分布的形式给出。同时它是一种无监督式学习算法，在训练时不需要手工标注的训练集，需要的仅仅是文档集以及指定主题的数量 k 。此外，LDA对于每一个主题均可找出一些词语来描述它。LDA是一种无监督式学习，在文本主题识别、文本分类、文本相似度计算和文章相似推荐等方面都有应用。

LDA可以被认为是一种聚类算法，主题对应聚类中心，文档对应数据集中的例子。主题和文档在特征空间中都存在，且特征向量是词频向量。LDA不是用传统的距离来衡量一个类簇，它使用的是基于文本文档生成的统计模型的函数。

2. LDA的发展历程

Papadimitriou、Raghavan、Tamaki和Vempala在1998年发表的一篇论文中提出了潜在语义索引。1999年，Thomas Hofmann又在此基础上提出了概率性潜在语义索引（Probabilistic Latent Semantic Indexing，简称PLSI）。2003年，David M. Blei、Andrew Y. Ng与Michael I. Jordan提出文档主题生成模型，LDA可能是最常见的主题模型，是一般化的PLSI。LDA允许文档拥有多种主题，其他主题模型一般是在LDA的基础上改进的，例如，HLDA（Hierarchical Latent Dirichlet Allocation，分级狄利克雷分布）模型、Labeled LDA（Labeled Latent Dirichlet Allocation，标记的狄利克雷分布）和LLDA（hierarchical Labeled Latent Dirichlet Allocation，分层标记的隐含狄利克雷分布），HLDA试图建立主题之间的关系，Labeled LDA则能够训练出带标签的主题。Labeled LDA与LDA最大的不同是，LDA是在所有主题上为某一个词进行选择某个主题，而labeled LDA则是只从文档相关的标签（Label）对应的主题中去选择，其余和LDA都是一样的。LLDA模型的好处就是在LDA模型的基础上扩展到层次结构，其思想也是很简单的，认为一个文档只是由这个文档对应的层次标签所产生。

LDA的应用十分广泛，通常会被应用在推荐系统、邮件分类和关键词提取等业务场景。

3. LDA的生成模型

LDA是一种典型的词袋模型，即它认为一篇文档是由一组词构成的一个集合，词与词之间没有顺序和先后的关系。一篇文档可以包含多个主题，文档中每一个词都由其中的一个主题生成。

看到一篇文章后，往往喜欢推测这篇文章是如何生成的，可能会认为作者先确定这篇文章的几个主题，然后围绕这几个主题遣词造句，表达成文。LDA就是根据给定的一篇文档，推测其主题分布。因

此，正如LDA贝叶斯网络结构中所描述的，在LDA模型中一篇文档生成的方式如下：

- 从LDA中取样生成文档i的主题分布。
- 从主题的多项式分布中取样生成文档i第j个词的主题。
- 从LDA中取样生成主题的词语分布。
- 从词语的多项式分布中采样最终生成词语。

8.5.4 LDA的模型实现

关于LDA本书主要给出相关概念、技术原理和实现方法，更多的理论知识会涉及大量的数学基础，读者可自行补充。本节使用8.2节构造的语料，采用Gensim中的LdaModel进行主题生成，代码实现如下（源代码见：Chapter8/LDA.py）：

```

1  from gensim import corpora, models
2  from gensim.models.doc2vec import Doc2Vec, TaggedDocument
3  from mydict import *
4
5  # 生成lda 特征(主题模型)
6  def gensim_Corpus(corpus=None):
7      dictionary = corpora.Dictionary(corpus)
8      # 1 doc_bow 转化为 tfidf 向量
9      doc_bow_corpus = [dictionary.doc2bow(doc_cut) for doc_cut in
10         corpus]
11      tfidf_model = models.TfidfModel(dictionary=dictionary) # 生成
12         tfidf 模型
13      # 将每 doc_bow 转换成对应的 tfidf_doc 向量
14      tfidf_corpus = [tfidf_model[doc_bow] for doc_bow in
15         doc_bow_corpus]
16      print('doc_bow 转换成对应的 tfidf_doc 向量:\n',tfidf_corpus)
17
18      lda_model = models.LdaModel(corpus=tfidf_corpus,
19         id2word=dictionary, num_topics=10) # 生成 lda model
20      # 2 生成 corpus of LDA
21      lda_corpus = [lda_model[tfidf_doc] for tfidf_doc in tfidf_corpus]
22      # 转换成 lda 向量
23      print('LDA 生成主题:\n',lda_corpus)
24
25      # 3 将 LDA 模型存储到磁盘上
26      savepath = r'../Files/lda_model.pkl'
27      lda_file = open(savepath, 'wb')
28      pickle.dump(lda_model, lda_file)
29      lda_file.close()
30      print('--- lda 模型已经生成 ---')

```

代码说明:

上述代码中, corpus为8.2节中构造的语料, 通过导入mydict模块, 使用corpus, classVec=loadDataSet()来加载数据。第一步就是加载文档集合, 建立词汇-文本矩阵A, 主要是实现原始语料的词袋模型转化, 最终生成TF-IDF矩阵。接着第二步直接调用Gensim的

models.LdaModel方法实现，其中参数corpus为TF-IDF向量矩阵，参数id2word为生成的词典向量，num_topics为自定义主题个数。最后将生成的结果进行本地化存储以备后续使用。

上述LDA主题模型构建完成后，运行gensim_Corpus函数得到如图8-6所示的结果，即输出各类文章前10个最为重要的主题。

```
LDA生成主题:
[[[(0, -0.016934099), (1, -0.016934095), (2, -0.016934233), (3, -0.016934099), (4, -0.8475929), (5, -0.016934179), (6, -0.016934099), (7, -0.016934099), (8, -0.016934099), (9, -0.016934099)], [(0, -0.017143842), (1, -0.017143838), (2, -0.017143838), (3, -0.017143842), (4, -0.017143944), (5, -0.84570533), (6, -0.017143842), (7, -0.017143842), (8, -0.017143842), (9, -0.017143842)], [(0, -0.017766235), (1, -0.84010386), (2, -0.017766232), (3, -0.017766235), (4, -0.017766232), (5, -0.017766232), (6, -0.017766235), (7, -0.017766235), (8, -0.017766235), (9, -0.017766235)], [(0, -0.01870277), (1, -0.018702764), (2, -0.8316748), (3, -0.01870277), (4, -0.018703056), (5, -0.018702764), (6, -0.01870277), (7, -0.01870277), (8, -0.01870277), (9, -0.01870277)]]
lda模型已经生成
*** Repl. Closed ***
```

图8-6 LDA生成主题

8.5.5 随机映射（RP）

1. 什么是RP

如果手头有一组数据 $X \in R^n$ ，它的维数太高，从而不得不进行降维至 R^k ，怎么办呢？

相信不少人会条件反射性地求它的 $x'x$ ，然后转化为沿最大方差展开的问题。这样需要求解固有值和固有向量，对于测试用的小数据尚好，但是如果面对的是几十上百GB大小的数据，对全部词降维将是一个漫长的过程，因此需要更加快速的方法。最简单的方法是：随机在高维空间里选几个单位向量 e_i ，注意，这里仅仅要求的是单位向量，并没有要求它们之间必须正交 $\langle e_i, e_j \rangle = 0$ ，因此可以随便选。最后，把高维数据投影到选择的这一组基上就完成了降维。

随机映射（RP）的目的在于减小空维度，这是一个非常高效的方法（对CPU和内存都很友好的方法），使用随机映射通过抛出一点随机

性来近似得到两个文档之间的TF-IDF距离。推荐目标维度也是成百上千，具体数值要视数据集大小而定。

2. RP的原理

- 选择映射矩阵 $R \in \mathcal{R}^{K \times N}$ 。
- 用随机数填充映射矩阵，可以选择uniform或者gaussian。
- 归一化每一个新的轴（映射矩阵中的每一行）。
- 对数据降维 $y = R\mathcal{X}$ 。

3. RP模型的实现

使用8.2节构造的语料词典作为语料，采用Gensim中的RpModel进行主题生成，代码实现如下（源代码见：Chapter8/RP.py）：

```

1  from mydict import *
2  from gensim import corpora, models
3  import pickle as pkl
4
5  # 生成随机映射 (Random Projections, RP)
6  def gensim_Corpus(corpus=None):
7      dictionary = corpora.Dictionary(corpus)
8      # 1 doc_bow 转化为 tfidf 向量
9      doc_bow_corpus = [dictionary.doc2bow(doc_cut) for doc_cut in
10         corpus]
11      tfidf_model = models.TfidfModel(dictionary=dictionary) # 生成
12         tfidf 模型
13
14      tfidf_corpus = [tfidf_model[doc_bow] for doc_bow in
15         doc_bow_corpus]
16      print('doc_bow 转换成对应的 tfidf_doc 向量:\n',tfidf_corpus)
17
18      # 2 生成 corpus of RP
19      rp_model = models.RpModel(tfidf_corpus, num_topics=10)
20      rp_corpus = [rp_model[tfidf_doc] for tfidf_doc in tfidf_corpus]
21      # 转换成随机映射 tfidf 向量
22      print('RP:\n',rp_corpus)
23
24      # 3 将随机映射 (RP) 模型存储到磁盘上
25      savepath =r'../Files/rp_model.pkl'
26      rp_file = open(savepath, 'wb')
27      pkl.dump(rp_model, rp_file)
28      rp_file.close()
29      print('--- RP 模型已经生成 ---')

```

4. RP的实验结果

上述随机映射主题模型构建完成后，运行gensim_Corpus函数得到以下结果，即输出各类文章前10个最为重要的主题，如图8-7所示。

```

RP:
[[[0, -0.24494898319244385), (1, -0.08164966106414795), (2, -0.19051587581634521)
, (3, -0.4626814126968384), (4, -0.08164966106414795), (5, -0.
24494898319244385), (6, -0.19051587581634521), (7, -0.13608276844024658), (8
, -0.5715476274490356), (9, -0.19051587581634521)], [(0, -0.
025906406342983246), (1, -0.18134468793869019), (2, -0.3885958194732666), (
3, -0.3885957598686218), (4, -0.12953190505504608), (5, -0.6994723677635193
), (6, -0.18134474754333496), (7, -0.12953191995620728), (8, -0.
4404085874557495), (9, -0.1813446581363678)], [(0, -0.2927699685096741), (1,
-0.1951800435781479), (2, -0.1951800435781479), (3, -0.7807201147079468), (
4, -0.195179988746643), (5, -0.29277002811431885), (6, -0.
19518007338047028), (7, -0.09759001433849335), (8, -0.1951800137758255), (9,
-0.09759001433849335)], [(0, -0.33183181285858154), (1, -0.33183184266090393)
, (2, -0.14221367239952087), (3, -0.42664089798927307), (4, -0.
047404564917087555), (5, -0.1422136127948761), (6, -0.04740455374121666), (8,
-0.14221367239952087), (9, -0.6636637449264526)]]
---RP模型已经生成---
*** Repl Closed ***

```

图8-7 RP生成主题

8.6 实战案例：Gensim实现新闻文本特征向量化

本节使用Gensim来实现新闻文本的特征向量化，相对于第7章自定义特征向量化的方法，本章算法实现更为简练且运行效率更高。首先对参数进行设置，其主要目的是用来本地化存储生成模型；接着生成词典模型，这个过程与前面章节原理一样，只是采用Gensim相关方法来实现；最后调用生成好的词典模型来生成TF-IDF特征向量模型，一次训练完成后，可以多次复用。本例生成的TF-IDF特征向量模型可以直接用于实际分类、聚类等算法中。

8.6.1 参数设置

本案例的实现代码如下：

```

1  import os,time,sys
2
3  if __name__=='__main__':
4      start = time.time()
5      n = 5 # n 表示抽样率
6      path_doc_root = '../Corpus/CSCMNews' # 根目录，即存放按类分类好的
      文本数据集
7      path_tmp = '../Corpus/CSCMNews_model' # 存放中间结果的位置
8      path_dictionary = os.path.join(path_tmp, 'CSCMNews.dict')
9      path_tmp_tfidf = os.path.join(path_tmp, 'tfidf_corpus')

```

代码说明：

其中，n表示抽样率，依旧设置为5，即每隔5条抽取一个新闻文本，当然n的值可以根据需求而设置。pathdocroot为30万新闻文本的根目录，也就是待处理文本数据存放的目录。pathtmp是用来存放中间结果的目录路径，其中包括pathdictionary和pathtmptfidf两个子文件夹。pathdictionary是对30万新闻文本处理后选择处理的词典文件，pathtmp_tfidf用来存放TF-IDF模型生成的各类新闻文本的向量模型。

8.6.2 生成词典模型

本节实现30万新闻文本词典的生成。需要导入StopWords模块的seg_doc方法用来分词并去除停用词，还需要导入Gensim的corpora和models模块，其中corpora模块用来构造词典及其词袋模型，models为其内容的主题模块。具体实现代码如下（源代码见：Chapter8/30wVec.py）：


```

1  from gensim import corpora, models
2  from StopWords import *
3
4  if __name__=='__main__':
5      #====第一阶段，遍历文档，生成词典，并去掉频率较少的词====
6      if not os.path.exists(path_tmp):
7          os.makedirs(path_tmp)
8      # 如果指定的位置没有词典，则重新生成一个。如果有，则跳过该阶段
9      if not os.path.exists(path_dictionary):
10         print('=== 未检测到有词典存在，开始遍历生成词典 ===')
11         dictionary = corpora.Dictionary()
12         files = loadFiles(path_doc_root)
13         for i, msg in enumerate(files):
14             if i % n == 0:
15                 catg = msg[0]
16                 content = msg[1]
17                 content = seg_doc(content)
18                 dictionary.add_documents([file])
19                 if int(i/n) % 1000 == 0:
20                     print('{t} *** {i} \t docs has been dealed'
21                           .format(i=i, t=time.strftime
22                                   ('%Y-%m-%d %H:%M:%S',time.localtime()))))
23                 # 去掉词典中出现次数过少的词
24                 small_freq_ids = [tokenid for tokenid, docfreq in
25                                   dictionary.dfs.items() if docfreq < 5 ]
26                 dictionary.filter_tokens(small_freq_ids)
27                 dictionary.compactify() # 重新产生连续的编号
28                 dictionary.save(path_dictionary)
29                 print('=== 词典已经生成 ===')
30             else:
31                 print('=== 检测到词典已经存在，跳过该阶段 ===')

```

代码说明：

本段代码用来生成词典模型，是生成IF-IDF模型的前置工作。首先判断是否存在生成结果的文件夹pathtmp，如果不存在则创建该目录。反之，判断是否已经生成词典pathdictionary，未检测到有词典

存在，则开始遍历生成词典。如果检测到词典已经存在，跳过该阶段即可。

在生成词典的过程中，通过高效遍历文件的loadFiles方法来装载所有文本信息并返回该文本类别和内容构建语料词典。然后对文本信息通过seg_doc方法分词并剔除停用词，最后加载到语料向量dictionary中。接着对装载好的语料根据词语出现频率进行选择，这一步的目的是初步降维，去除词语权重较小的词汇。最后调用dictionary.save方法将处理好的词典保存即可。

代码的执行结果如图8-8所示。

```
2019-04-01-17:10:54.***.294000.--docs has been dealed
2019-04-01-17:11:01.***.297000.--docs has been dealed
2019-04-01-17:11:08.***.300000.--docs has been dealed
2019-04-01-17:11:17.***.303000.--docs has been dealed
2019-04-01-17:11:24.***.306000.--docs has been dealed
2019-04-01-17:11:31.***.309000.--docs has been dealed
2019-04-01-17:11:38.***.312000.--docs has been dealed
2019-04-01-17:11:46.***.315000.--docs has been dealed
2019-04-01-17:11:54.***.318000.--docs has been dealed
2019-04-01-17:12:02.***.321000.--docs has been dealed
2019-04-01-17:12:09.***.324000.--docs has been dealed
2019-04-01-17:12:16.***.327000.--docs has been dealed
2019-04-01-17:12:24.***.330000.--docs has been dealed
2019-04-01-17:12:33.***.333000.--docs has been dealed
2019-04-01-17:12:42.***.336000.--docs has been dealed
===词典已经生成===
```

图8-8 生成新闻词典

8.6.3 生成TF-IDF模型

本节生成TF-IDF模型，需要使用Gensim中的models模块，其中models模块包含tfidfModel方法，可以直接构造TF-IDF模型，具体实现代码如下：

```

1  from gensim import corpora, models
2  from StopWords import *
3
4  if __name__=='__main__':
5      start = time.time()
6      dictionary = None
7      if not os.path.exists(path_tmp_tfidf):
8          print('=== 未检测到有 tfidf 文件夹存在, 开始生成 tfidf 向量 ===')
9          if not dictionary: # 如果有跳过了第一阶段, 则从指定位置读取词典
10             dictionary = corpora.Dictionary.load(path_dictionary)
11             os.makedirs(path_tmp_tfidf)
12             files = loadFiles(path_doc_root)
13             tfidf_model = models.TfidfModel(dictionary=dictionary)
14             corpus_tfidf = {}
15             for i, msg in enumerate(files):
16                 if i % n == 0:
17                     catg = msg[0]
18                     content = msg[1]
19                     word_list = seg_doc(content)
20                     file_bow = dictionary.doc2bow(word_list)
21                     file_tfidf = tfidf_model[file_bow]
22                     tmp = corpus_tfidf.get(catg, [])
23                     tmp.append(file_tfidf)
24                     if tmp.__len__() == 1:
25                         corpus_tfidf[catg] = tmp
26                 if i % 10000 == 0:
27                     print('{i} files is dealed'.format(i=i))
28             # 将 tfidf 中间结果保存起来
29             catgs = list(corpus_tfidf.keys())
30             for catg in catgs:
31                 corpora.MmCorpus.serialize('{f}{s}{c}.mm'.format
32                 (f=path_tmp_tfidf, s=os.sep, c=catg), corpus_tfidf.get(catg),
33                 id2word=dictionary)
34             print('catg {c} has been transformed into tfidf
35             vector'.format(c=catg))
36             print('=== tfidf 向量已经生成 ===')
37         else:
38             print('=== 检测到 tfidf 向量已经生成, 跳过该阶段 ===')
39     end = time.time()
40     print('total spent times: %.2f' % (end-start)+ ' s')

```

代码说明：

本段代码是TF-IDF模型的生成过程，首先判断TF-IDF模型保存路径pathtmpfidf是否存在，如果不存在就执行训练过程。接着寻找dictionary，如上文已经生成则可以直接跳过词典训练过程，用corpora.Dictionary.load方法加载词典即可。然后通过loadFiles方法返回文本的类别和内容，调用segdoc方法进行分词并去停用词。之后通过dictionary.doc2bow方法转化为词袋模型，并有tfidfmodel方法生成TF-IDF向量。这个过程前面章节已经做了详细的介绍。完成之后根据类别-TF-IDF向量保存到corpus_tfidf字典中。最后根据各个新闻类别，调用序列化corpora.MmCorpus.serialize方法进行本地存储。整个TF-IDF训练结果可以用于具体的算法应用中，即完成了整个数据预处理的过程，其执行结果如图8-9所示。

```
320000 files is dealt
330000 files is dealt
catg 体育 has been transformed into tfidf vector
catg 娱乐 has been transformed into tfidf vector
catg 教育 has been transformed into tfidf vector
catg 时政 has been transformed into tfidf vector
catg 科技 has been transformed into tfidf vector
catg 财经 has been transformed into tfidf vector
=== tfidf向量已经生成 ===
total spent times:1697.02 s
***Repl Closed***
```

图8-9 生成TF-IDF特征向量模型

8.7 本章小结

本章介绍了基于Gensim工具包实现新闻文本的特征向量化，并详细介绍了Gensim构建语料词典和统计词频特征。重点介绍了Gensim中的models模块中的TF-IDF向量生成以及LSA、LDA、RP三种主题模型的实现过程。最后通过综合实战完成词典模型和TF-IDF模型的生成。

下一章将开始介绍主成分分析PCA模型及其PCA降维的原理及实现方法。

第9 章

PCA降维技术

PCA (Principal Components Analysis, 主成分分析) 是一种分析、简化数据集的技术。PCA经常用于减少数据集的维数, 同时保持数据集中对方差贡献最大的特征。PCA常常应用在文本处理、人脸识别、图片识别、自然语言处理等领域, 可以说在数据预处理阶段非常重要的一环。本章首先对PCA的基本概念进行介绍, 然后给出PCA的算法思想、流程、优缺点等, 最后通过一个综合案例来实现其应用。

9.1 什么是降维

降维是对数据高维度特征的一种预处理方法。降维是将高维度的数据保留下最重要的一些特征, 去除噪声和不重要的特征, 从而实现提升数据处理速度的目的。在实际的生产和应用中, 降维在一定的信息损失范围内, 可以节省大量的时间和成本。降维也成为应用非常广泛的数据预处理方法。

比如, 通过电视观看体育比赛, 在电视的显示器上有一个足球, 显示器大概包含了100万像素点, 而球则可能是由较少的像素点组成, 例如一千个像素点。人们实时地将显示器上的百万像素转换成为一个三维图像, 该图像就给出运动场上球的位置。在这个过程中, 人们已经将百万像素点的数据降至三维。这个过程就称为降维 (Dimensionality Reduction)。

数据降维的目的是:

- 使数据集更容易使用。
- 确保变量是相互独立的。
- 降低很多算法的计算开销。
- 去除噪音。
- 使得结果易懂。

数据降维可用于已标注或未标注的数据上，本文主要关注未标注数据上的降维技术，该技术同样也可以应用于已标注的数据。

常见的降维技术主要有：

- PCA（主成分分析）。就是找出一个最主要的特征，然后进行分析。例如：考察一个人的智力情况，从数学、语文、英语成绩中，直接看数学成绩就行。
- FA（Factor Analysis，因子分析）。是指将多个实测变量转换为少数几个综合指标。它反映一种降维的思想，通过降维将相关性高的变量聚在一起，从而减少需要分析的变量的数量，以减少问题分析的复杂性。例如：考察一个人的整体情况，可组合3科成绩（数学、语文、英语成绩，隐变量），看3科成绩的平均成绩即可。FA的应用的领域包括社会科学、金融等。在FA中，假设观察数据的成分中有一些观察不到的隐变量（Latent Variable）或假设观察到的数据是这些隐变量和某些噪音的线性组合，那么隐变量的数据可能比观察数据的数目少，也就是说通过找到隐变量就可以实现数据的降维。
- ICA（Independ Component Analysis，独立成分分析）。ICA认为，如果观测到的信号是若干个独立信号的线性组合，ICA可对其进行解混。例如：去KTV唱歌，想辨别是原唱还是主唱，使用ICA可以实现分辨原唱或主唱的目的（从2个独立的声音中分辨出原唱或主唱）。

ICA假设数据是由N个数据源混合组成，这一点和FA有些类似，这些数据源之间在统计上是相互独立的，而在PCA中只假设数据是不相关（线性关系）的。同FA一样，如果数据源的数目少于观察数据的数目，则可以实现降维过程。

9.2 PCA概述

PCA由卡尔·皮尔逊于1901年发明，用于分析数据及建立数理模型。其方法主要是通过对协方差矩阵进行特征分解，以得出数据的主成分（即特征向量）与它们的权重值（即特征值）。PCA是最简单的以特征量来分析多元统计分布的方法，它的结果可以理解为对原数据中的方差做出解释：哪一个方向上的数据值对方差的影响最大？换言之，PCA提供了一种降低数据维度的有效办法；如果分析者在原数据中除掉最小的特征值所对应的成分，那么所得的低维度数据必定是最优化的（即这样降低维度必定是失去信息最少的方法）。PCA在分析复杂数据时尤为有用，比如人脸识别。

PCA是最简单的以特征量来分析多元统计分布的方法。通常情况下，这种运算可以被看作是揭露数据的内部结构，从而更好地解释数据的变量。如果一个多元数据集能够在一个高维数据空间坐标系中被显现出来，那么PCA就能够提供一幅比较低维度的图像，在这幅图像上信息最多的点就是原对象的一个“投影”。这样就可以利用少量的主成分使得数据的维度降低。PCA跟因子分析密切相关，并且已经有很多混合这两种分析的统计包。

1. PCA的思想

PCA解决问题的主要过程是：

- （1）去除平均值。

- (2) 计算协方差矩阵。
- (3) 计算协方差矩阵的特征值和特征向量。
- (4) 将特征值排序。
- (5) 保留前N个最大的特征值对应的特征向量。
- (6) 将数据转换到上一步得到的N个特征向量构建的新空间中（实现了特征压缩）。

2. PCA的原理

- (1) 找出第一个主成分的方向，也就是数据方差最大的方向。
- (2) 找出第二个主成分的方向，也就是数据方差次大的方向，并且该方向与第一个主成分方向正交（Orthogonal，如果是二维空间就叫垂直）。
- (3) 通过这种方式计算出所有的主成分方向。
- (4) 通过数据集的协方差矩阵及其特征值分析，可以得到这些主成分的值。
- (5) 一旦得到了协方差矩阵的特征值和特征向量，就可以保留最大的N个特征。这些特征向量也给出了N个最重要特征的真实结构，就可以通过将数据乘上这N个特征向量，从而将它转换到新的空间上。

3. PCA的算法流程

下面来看一下PCA具体的算法流程。

输入： m维样本集 $D = (x^{(1)}, x^{(2)}, \dots, x^{(m)})$ ，以及要降维到的维数n。

输出： 降维后的样本集 D' 。

完成上述任务的PCA算法流程是：

- (1) 对所有的样本进行中心化： $\bar{x}^{(i)} = x^{(i)} - \frac{1}{m} \sum_{j=1}^m x^{(j)}$ 。
- (2) 计算样本的协方差矩阵 XX^T 。
- (3) 对矩阵 XX^T 进行特征值分解。
- (4) 取出最大的 n 个特征值对应的特征向量 $(w_1, w_2, \dots, w_n)$ ，将所有特征向量标准化后，组成特征向量矩阵 W 。
- (5) 对样本集中的每一个样本 $x^{(i)}$ ，转化为新的样本 $z^{(i)} = W^T x^{(i)}$ 。
- (6) 得到输出样本集 $D' = (z^{(1)}, z^{(2)}, \dots, z^{(n)})$ 。

4. PCA的优缺点

优点： 降低了数据的复杂性，可识别出最重要的多个特征。

缺点： 不一定需要，且可能损失有用信息。

适用的数据类型： 数值数据类型。

9.3 PCA应用场景

真实的训练数据总是存在各种各样的问题，这时就可以考虑通过PCA技术来解决，比如以下情况：

- (1) 一辆汽车的数据样本，里面既有以“千米 / 每小时”度量的最大速度特征，也有“公里 / 小时”的最大速度特征，显然这两个特征有一个是多余的。

(2) 一个数学系的本科生期末考试成绩单，里面有三列，一列是对数学的兴趣程度，一列是复习时间，还有一列是考试成绩。知道要学好数学，需要有浓厚的兴趣，所以第二项与第一项强相关，第三项和第二项也是强相关。那么是不是可以合并第一项和第二项呢？

(3) 一个样本特征非常多，而样例特别少，这样用回归去直接拟合非常困难，容易过度拟合。比如北京的房价：假设房子的特征是（大小、位置、朝向、是否学区房、建造年代、是否二手、层数、所在层数），这么多特征只有不到十套房子的样例。要拟合房子特征？房价的这么多特征，就会造成过度拟合。

(4) 与第二点类似，假设在信息检索（IR）中建立的文档-词矩阵中，有两个词为learn和study，在传统的向量空间模型中，认为两者是独立的。然而，从语义的角度来讲，两者是相似的，而且这两者出现的频率也类似，是不是可以合成为一个特征呢？

(5) 在信号传输过程中，由于信道不是理想的，信道另一端收到的信号会有噪音扰动，那么怎样滤去这些噪音呢？这时可以采用PCA的方法来解决。PCA的思想是将 n 维特征映射到 k 维上（ $k < n$ ）， k 维是全新的正交特征。 k 维特征称为主元，是重新构造出来的 k 维特征，而不是简单地从 n 维特征中去除其余的 $n-k$ 维的特征。

9.4 PCA的算法实现

本节将通过一个小案例学习PCA降维技术。

9.4.1 准备数据

这里选择二维坐标系数，数据在源代码Files文件夹中的文件中（源代码见：Files/testSet.txt）。部分数据的格式如下：

```
1 10.235186 11.321997
2 10.122339 11.810993
3 9.190236 8.904943
4 9.306371 9.847394
5 8.330131 8.340352
6 10.152785 10.123532
7 10.408540 10.821986
8 9.003615 10.039206
9 9.534872 10.096991
10 9.498181 10.825446
11 9.875271 9.233426
12 10.362276 9.376892
13 10.191204 11.250851
```

这些文本数据还不能直接使用，需要将数据进行格式化处理以满足PCA数据准备。首先加载文本数据，然后将其进行矩阵化处理，代码实现如下（源代码见：Chapter9/loadData.py）：

```
1 from numpy import *
2
3 '''加载数据集'''
4 def loadDataSet(fileName, delim='\t'):
5     fr = open(fileName)
6     stringArr = [line.strip().split(delim) for line in
7 fr.readlines()]
8     datArr = [list(map(float, line)) for line in stringArr]
9     # 注意这里和 Python2 的区别，需要在 map 函数外加一个 list()，否则显示结果
10 为 map at 0x3fed1d0
11     return mat(datArr)
12
13 if __name__ == "__main__":
14     # 1 加载数据，并把数据类型转换为 float
15     dataMat = loadDataSet('../Files/testSet.txt')
16     print('坐标系原始数据:\n', dataMat)
```

执行以上代码，得到数据的加载结果，如图9-1所示。

```
坐标系原始数据:
·[[10.235186, 11.321997]
·[10.122339, 11.810993]
·[-9.190236, 8.904943]
·...
·[-9.854922, 9.201393]
·[-9.11458, 9.134215]
·[10.334899, 8.543604]]
*** Repl Closed ***
```

图9-1 加载坐标系数据

9.4.2 PCA数据降维

在等式 $Av = \lambda v$ 中， v 是特征向量， λ 是特征值，该等式表示如果特征向量 v 被某个矩阵 A 左乘，那么它就等于某个标量 λ 乘以 v 。幸运的是，NumPy 中有寻找特征向量和特征值的模块 `linalg`，它有 `eig()` 方法，该方法用于求解特征向量 `eigVects` 和特征值 `eigVals`。具体实现代码如下（源代码见：Chapter9/pca.py）：

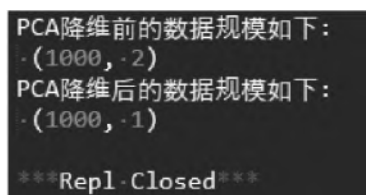
```

1  from numpy import *
2  from loadData import *
3  def pca(dataMat, topNfeat=99999999):
4      # 1 计算每一列的均值
5      meanVals = mean(dataMat, axis=0) # axis=0 表示列, axis=1 表示行
6      print('各列的均值: \n', meanVals)
7
8      # 2 去平均值, 每个向量同时都减去均值
9      meanRemoved = dataMat - meanVals
10     print('每个向量同时都减去均值:\n', meanRemoved)
11
12     # 3 计算协方差矩阵的特征值与特征向量, eigVals 为特征值, eigVects 为特征
    向量
13     # rowvar=0, 传入的数据一行代表一个样本, 若为非 0, 传入的数据一列代表一个
    样本
14     covMat = cov(meanRemoved, rowvar=0)
15     eigVals, eigVects = linalg.eig(mat(covMat))
16     print('特征值:\n', eigVals, '\n 特征向量:\n', eigVects)
17
18     # 4 将特征值排序, 特征值的逆序就可以得到 topNfeat 个最大的特征向量
19     eigValInd = argsort(eigVals) # 特征值从小到大排序, 返回从小到大的
    index 序号
20     # print('eigValInd1=', eigValInd)
21
22     # 5 保留前 N 个特征。-1 表示倒序, 返回 topN 的特征值[-1 到 -(topNfeat+1)
    不包括-(topNfeat+1)]
23     eigValInd = eigValInd[:-(topNfeat+1):-1]
24     # 重组 eigVects 最大到最小
25     redEigVects = eigVects[:, eigValInd]
26     print('重组 n 特征向量最大到最小:\n', redEigVects.T)
27
28     # 6 将数据转换到新空间
29     lowDDataMat = meanRemoved * redEigVects
30     reconMat = (lowDDataMat * redEigVects.T) + meanVals
31     return lowDDataMat, reconMat #降维后的数据集, 新的数据集空间
32
33 if __name__ == "__main__":
34     # 1 加载数据, 并把数据类型转换为 float
35     dataMat = loadDataSet('../Files/testSet.txt')
36     # 2 主成分分析降维特征向量的设置
37     lowDmat, reconMat = pca(dataMat, 1)
38     print('PCA 降维前的数据规模如下:\n', shape(dataMat))
39     print('PCA 降维后的数据规模如下:\n', shape(lowDmat))

```

代码说明：其中的参数dataMat是原数据集矩阵，参数topNfeat是应用的N个特征。

加载二维坐标系中的数据并处理成数组形式，通过前文介绍的PCA构造步骤，完成降维函数，用户输入原始数据并指定维度（这里将2维数据降低为1维数据），结果如图9-2所示。



```
PCA降维前的数据规模如下:
(1000, 2)
PCA降维后的数据规模如下:
(1000, 1)
***Repl - Closed***
```

图9-2 PCA降维前后特征规模对比

通过上述结果不难发现，降维前后的数据维度发生了变化，但整个数据条数保持不变，这是空间压缩的问题。为了更加形象化，下一小节采用可视化技术来对比PCA降维前后数据的变化。

9.4.3 高维向低维数据映射

本小节来查看降维后的数据与原始数据的可视化效果，将原始数据采用绿色△表示，降维后的数据采用红色○表示，具体的可视化代码如下：

```

1  from numpy import *
2  from pca import *
3  import matplotlib
4  import matplotlib.pyplot as plt
5
6  '''降维后的数据和原始数据可视化'''
7  def show_picture(dataMat, reconMat):
8      fig = plt.figure()
9      ax = fig.add_subplot(111)
10     # flatten()方法能将matrix的元素变成一维的，A能使matrix变成
    array,A[0]或者数组数据
11     ax.scatter(dataMat[:, 0].flatten().A[0], dataMat[:,
12     1].flatten().A[0], marker='^', s=90,c='green')
13     ax.scatter(reconMat[:, 0].flatten().A[0], reconMat[:,
14     1].flatten().A[0], marker='o', s=50, c='red')
15     plt.show()
16
17 if __name__ == "__main__":
18     # 1 加载数据，并把数据类型转化为float
19     dataMat = loadDataSet('../Files/testSet.txt')
20     # 2 主成分分析降维特征向量的设置
21     lowDmat, reconMat = pca(dataMat,1)
22     # 3 将降维后的数据和原始数据一起可视化
23     show_picture(dataMat, reconMat)

```

PCA对二维坐标系降维后，将原始的二维数据（绿色图标<三角形形状>）降维到一维数据（红色图标<圆球形状>），其可视化对比结果如图9-3所示。注：因为书籍黑白印刷的问题，所以无法看到读者在计算机运行中看到的绿色和红色。在图9-3中三角形图标就是对应计算机显示的绿色，圆球形状就是对应计算机显示的红色。

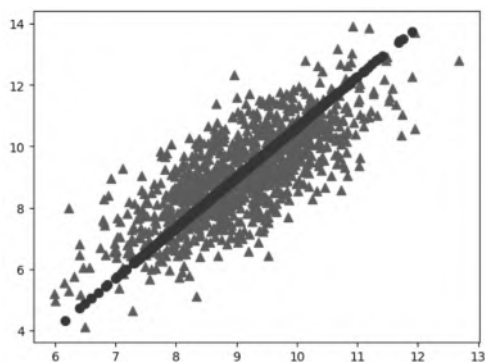


图9-3 PCA降维后的可视化结果

9.5 实战案例：PCA技术实现新闻文本特征降维

9.5.1 加载新闻数据

每条新闻文本信息由不同的词语组成，比如“进球”一般会出现在体育新闻之中，“学校”一般会出现在教育新闻中，此类词语在文章中占据着比较重要的位置，利用PCA降维技术处理此类新闻文本就是将一些代表性不强的词语剔除，这样处理之后就形成了低维度的特征词组成的新闻文本。本例中我们采集了1567篇不同类型的新闻文章，每篇文章由分词并去除停用词后的590个特征词组成。由于词袋模型构造的原因，必然有些词在一些文章中不存在，即存在所谓的缺失值。对于数据缺失值的问题，将缺失值NaN（Not a Number缩写）全部用平均值来替代（用0来处理的策略不可取）。整个新闻文本数据存放在Files文件夹中的news.data文件中，该文本文件的部分数据格式如下：

```
83.3971 9.5126 50.617 64.2588 49.383 66.3141 86.9555 117.5132 61.29
4.515 70 352.7173 10.1841 130.3691 723.3092 1.3072 141.2282 1 624.3145
218.3174 0 4.592 4.841 2834 0.9317 0.9484 4.7057 -1.7264 350.9264 10.6231
108.6427 16.1445 21.7264 29.5367 693.7724 0.9226 148.6009 1 608.17 84.0793
NaN NaN 0 0.0126 -0.0206 0.0141 -0.0307 -0.0083 -0.0026 -0.0567 -0.0044
7.2163 0.132 NaN 2.3895 0.969 1747.6049 0.1841 8671.9301 -0.3274 -0.0055
-0.0001 0.0001 0.0003 -0.2786
```


下面使用均值法来处理缺失值问题，代码实现如下（源代码见：Chapter9/loadnews.py）：

```
1  from numpy import *
2  from loadData import *
3
4  '''将NaN替换成平均值函数，secom.data 为1567篇文章，每篇文章有590个词'''
5  def replaceNanWithMean():
6      datMat = loadDataSet('../Files/news.data', ' ')
7      numFeat = shape(datMat)[1]
8      for i in range(numFeat):
9          # 对 value 不为 NaN 的求均值
10         meanVal = mean(datMat[nonzero(~isnan(datMat[:, i]).A))[0], i])
11         datMat[nonzero(isnan(datMat[:, i]).A)[0], i] = meanVal
12     return datMat
13
14  if __name__ == "__main__":
15     dataMat = replaceNanWithMean()
16     print('处理缺少值的新闻数据:\n', dataMat)
17     print('新闻矩阵数据规模:\n', shape(dataMat))
```

最后对新闻数据进行格式化处理，结果如图9-4所示。

```
处理缺少值的新闻数据:
[[3.03093000e+03 2.56400000e+03 2.18773330e+03 ... 1.64749042e-02
 5.28333333e-03 9.96700663e+01]
 [3.09578000e+03 2.46514000e+03 2.23042220e+03 ... 2.01000000e-02
 6.00000000e-03 2.08204500e+02]
 [2.93261000e+03 2.55994000e+03 2.18641110e+03 ... 4.84000000e-02
 1.48000000e-02 8.28602000e+01]
 ...
 [2.97881000e+03 2.37978000e+03 2.20630000e+03 ... 8.60000000e-03
 2.50000000e-03 4.35231000e+01]
 [2.89492000e+03 2.53201000e+03 2.17703330e+03 ... 2.45000000e-02
 7.50000000e-03 9.34941000e+01]
 [2.94492000e+03 2.45076000e+03 2.19544440e+03 ... 1.62000000e-02
 4.50000000e-03 1.37784400e+02]]
新闻矩阵数据规模:
(1567, 590)
*** Repl. Closed ***
```

图9-4 处理带有缺失值的新闻数据

9.5.2 前N个主成分特征

得到新闻数据之后，使用PCA方法对其进行处理。这时会遇到一个棘手的问题，即保留的维度为多少合适？9.4节介绍了将2维数据降为1维数据，目标比较清晰。本例特征值较多，很容易确定合适的维度，可以使用方差的概念来解决，根据累积方差可以判断特征保留的维度（关于方差与累积方差，请读者自行学习）。分析数据特征以查看保留不同特征对结果的影响，具体代码实现如下（源代码见：Chapter9/analyse.py）：

```

1  from numpy import *
2  from loadnews import *
3
4  '''分析数据'''
5  def analyse_data(dataMat,topNfeat = 20):
6      # 去除平均值
7      meanVals = mean(dataMat, axis=0)
8      meanRemoved = dataMat-meanVals
9      # 计算协方差矩阵
10     covMat = cov(meanRemoved, rowvar=0)
11     # 特征值和特征向量
12     eigvals, eigVects = linalg.eig(mat(covMat))
13     eigValInd = argsort(eigvals)
14     # 保留前 N 个特征
15     eigValInd = eigValInd[:-(topNfeat+1):-1]
16     # 对特征的主成分分析
17     cov_all_score = float(sum(eigvals))
18     sum_cov_score = 0
19     for i in range(0, len(eigValInd)):
20         line_cov_score = float(eigvals[eigValInd[i]])
21         sum_cov_score += line_cov_score
22         print('主成分: %s, 方差占比: %s%%, 累积方差占比: %s%%' % (format
23             (i+1, '2.0f'), format(line_cov_score/cov_all_score*100, '4.2f'),
24             format(sum_cov_score/cov_all_score*100, '4.1f')))
25
26
27 if __name__ == "__main__":
28     # 加载新闻数据
29     dataMat = replaceNaNWithMean()
30     # 分析数据: 要求满足 99%即可
31     line_cov_score=analyse_data(dataMat,20)

```

获取到新闻数据后，按照PCA的算法原理及步骤保留前N个有序（根据特征的重要性由大到小排序）特征词。再采用累积方差的方法对不同的特征数量进行测试，直到满足99%（假设99%符合需求）的准确率即可。运行结果如图9-5所示。

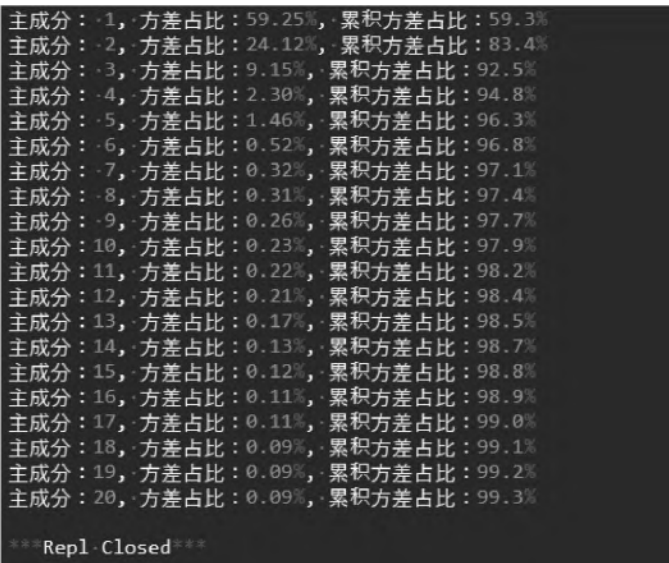


图9-5 前20个主成分的累积方差

从图9-5所示的结果中可以发现，选取前17个特征值的时候就达到了99.0%，即满足预设的要求。可见，针对590个特征值，只需要处理17个（即前2.9%）的特征值即可，这样大大简化了工作量。注意，这里的99%是预设值的一个满足条件值，具体条件应根据实际情况来设置。根据实验结果，我们绘制了新闻文本数据中前17个主要成分所占的方差百分比的表格，如表9-1所示。

表9-1 新闻文本中前17个主成分特征

主 成 分	方差百分比（%）	累积方差百分比（%）
1	59.25	59.3
2	24.12	83.4
3	9.15	92.5
4	2.30	94.8
5	1.46	96.3
6	0.52	96.8
7	...	...
17	0.09	99.0

9.5.3 PCA新闻特征降维可视化

本节通过分析数据特征，选取前17个特征值，然后将降维后的结果进行可视化显示。这里使用的是二维图形，所以在选择特征值的时候，分别选择原始数据和降维后的数据第一列和第二列的值，代码实现如下（源代码见：Chapter9/pcanews.py）：

```
1  from numpy import *
2  import matplotlib
3  import matplotlib.pyplot as plt
4
5  '''降维后的数据和原始数据可视化'''
6  def show_picture(dataMat, reconMat):
7      fig = plt.figure()
8      ax = fig.add_subplot(111)
9      ax.scatter(dataMat[:, 0].flatten().A[0], dataMat[:,
10     1].flatten().A[0], marker='^', s=5, c='green')
11     ax.scatter(reconMat[:, 0].flatten().A[0], reconMat[:,
12     1].flatten().A[0], marker='o', s=5, c='red')
13     plt.show()
14
15  if __name__ == "__main__":
16     dataMat = replaceNaNWithMean()
17     # 分析数据
18     lowDmat, reconMat = pca(dataMat, 17)
19     print(shape(lowDmat)) # 1567 篇文章，提取前 20 个词
20     show_picture(dataMat, reconMat)
```

PCA技术实现了新闻文本数据的特征降维，原始的新闻文本特征比较多，而且存在异常值等问题。经过降维以后，可以获取到结果最大的前n个特征，特征维度降低，权重更加紧凑，大大优化了运算的性能。选取不同特征值的可视化结果如图9-6所示。

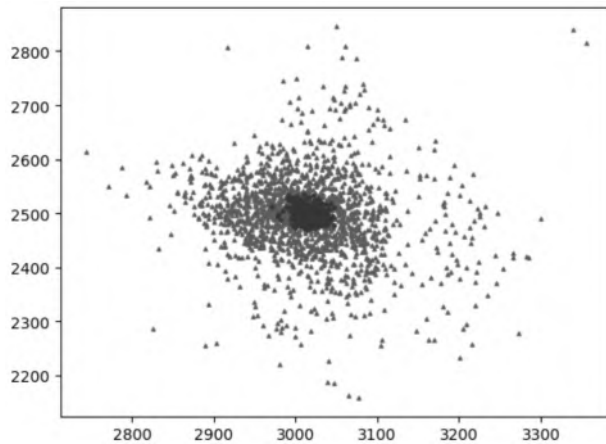


图9-6 PCA对新闻特征降维的可视化

9.6 本章小结

本章主要介绍了降维技术和方法，针对常见的PCA降维技术进行了深入的解析。首先介绍了PCA降维技术理论（定义、算法原理、实现流程），接着通过实战案例进一步深化降维技术的理解。最后应用到新闻文本特征降维的实际案例中。下一章，我们将对可视化技术进行全面的讲解。

第10 章

数据可视化

在自然语言处理相关的工作过程中，数据可视化方法能够使数据分析的结果更加一目了然，所谓“一图胜千言”就是这个意思。本章主要介绍Matplotlib数据可视化方法，通过本章的学习使读者对Matplotlib操作更加轻车熟路。

10.1 Matplotlib概述

10.1.1 认识Matplotlib

Matplotlib是一个基于Python的绘图库，完全支持二维图形，有限支持三维图形，Matplotlib是Python编程语言及其数据科学扩展包NumPy的可视化操作界面库。它利用通用的图形用户界面工具包（如Tkinter、wxPython、Qt、FLTK、Cocoa toolkits或GTK+）向应用程序嵌入式绘图提供了应用程序接口（API）。此外，Matplotlib还有一个基于图像处理库（如图形库OpenGL）的pylab接口，其设计与MATLAB非常类似。SciPy就是用Matplotlib进行图形绘制。

Matplotlib最初由John D. Hunter开发的，它拥有一个活跃的开发社区，并且根据BSD样式许可证分发。截至到2019年1月1日，Matplotlib 2.2.x支持Python 2和Python 3版本，Matplotlib 3.0只支持Python 3相关版本。Matplotlib 1.2是Matplotlib的第一个版本，支持Python 3.x，Matplotlib 1.4是Matplotlib支持Python 2.6

的最后一个版本。Matplotlib的官网网址为：
<https://matplotlib.org/index.html>，如图10-1所示。

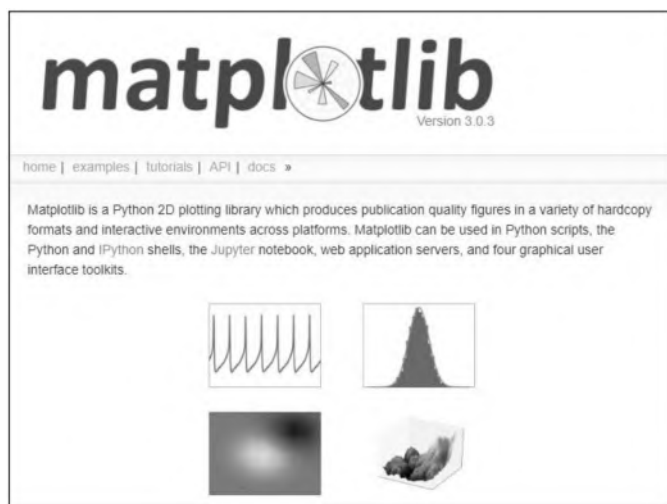


图10-1 matplotlib官网

读者可以通过以下3种方法安装Matplotlib。

方法一 在 Anaconda Prompt 下执行 conda 命令安装 Matplotlib，可以通过执行以下命令自动安装：

```
conda install matplotlib
```

方法二 可以通过pip自动安装，执行如下命令：

```
pip install matplotlib
```

方法三 在 GitHub 上下载 Matplotlib 源代码 (<https://github.com/matplotlib/matplotlib>)，然后在源代码根目录下执行如下命令：

```
python setup.py install
```

安装完成后，检测是否成功。首先打开Sublime，然后按F6键进入Python环境，最后输入以下命令：


```
import matplotlib
```

得到如图10-2所示状态即表示安装成功。

```
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64-bit (AMD64)] ::  
Anaconda, Inc. on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import matplotlib  
>>> |
```

图10-2 导入matplotlib

10.1.2 Matplotlib的架构

Matplotlib按照逻辑性划分为三层，即Backend Layer、Artist Layer、Scripting Layer。Backend Layer是Matplotlib的最底层，负责软件与绘图硬件的交互，负责绘图的主要框架搭建。该层主要实现了FigureCanvas、Renderer和Event的抽象接口类，其中，FigureCanvas实现绘图表面概念的封装，Renderer封装了执行绘图操作。Event封装了处理键盘与鼠标事件的用户输入。例如，下面的代码用于实现当用户键入“t”时，切换显示Axes窗口中的线段。

```
1  import numpy as np  
2  import matplotlib.pyplot as plt  
3  
4  def on_press(event):  
5      if event.inaxes is None: return  
6      for line in event.inaxes.lines:  
7          if event.key=='t':  
8              visible = line.get_visible()  
9              line.set_visible(not visible)  
10         event.inaxes.figure.canvas.draw()  
11  
12  fig, ax = plt.subplots(1)  
13  fig.canvas.mpl_connect('key_press_event', on_press)  
14  ax.plot(np.random.rand(2, 20))  
15  plt.show()
```

运行结果如图10-3所示。

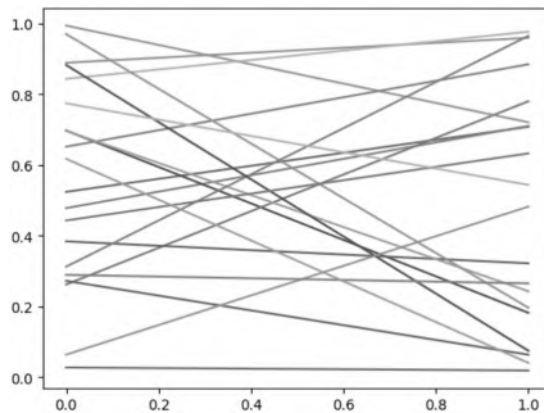


图10-3 Axes窗口线段

Artist Layer又被称作美工层，该层结构处于Matplotlib的中间层，是对Backend Layer的进一步封装，该层级结构中有两种类型的Artist：基础Artist和复合Artist。基础Artist是我们在图形中能看到的一类对象，如Line2D、Rectangle、Circle与Text。复合Artist是Artist的集合，如Axis、Tick、Axes与Figure。每个复合Artist可能包含基础Artist或者其他复合Artist。例如，Figure包含一个或多个Axes，而且Figure的背景是基础Artist的Rectangle。如图10-4和图10-5所示，标题、坐标轴、刻度以及图像等都对应着某个Artist实例。

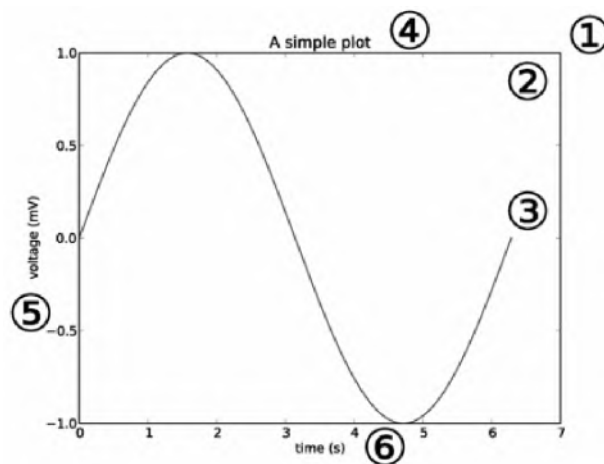


图10-4 正弦曲线图

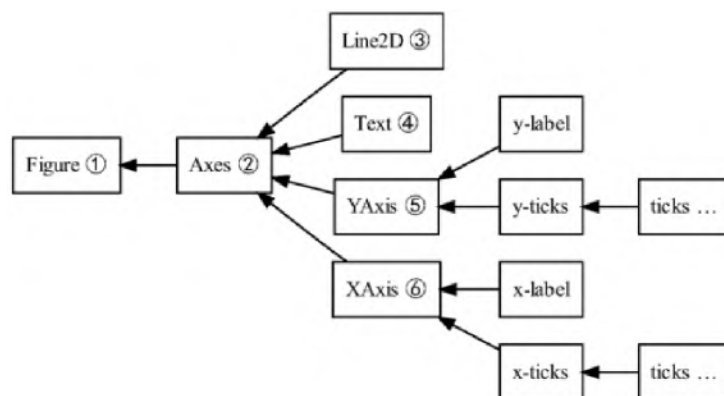


图10-5 图层解析

另外，Axes不仅包含构成绘图背景的图形元素，还包括了创建基础Artist并将其添加到Axes实例中的辅助函数。例如，表10-1列出了部分Axes函数，这些函数主要实现对象的绘制并将它们存储在Axes实例中。

表10-1 Axes方法的样例

方 法	创 建	存 储 在
Axes.imshow	一个或多个 matplotlib.image.AxesImages	Axes.images
Axes.hist	多个 matplotlib.patch.Rectangles	Axes.patches
Axes.plot	多个 matplotlib.patch.Rectangles	Axes.lines

下面运用上述的层级结构和函数来实现一个小练习，首先定义Backend并链接Figure，然后使用NumPy创建正态分布的随机数，最后绘制出随机数的直方图。

```

1  # 选择 Backend 导入 FigureCanvas, 并将其链接到 Figure
2  from matplotlib.backends.backend_agg import FigureCanvasAgg as
    FigureCanvas
3  from matplotlib.figure import Figure
4  import numpy as np
5
6  fig = Figure()
7  canvas = FigureCanvas(fig)
8  x = np.random.randn(10000) # 生成随机数
9  # 使用 figure 创建一个 Axes artist, Axes artist 自动添加到 figure 容器中,
    参数 111 的意思是: 创建一个 1 行 1 列的网格且坐标轴位于网格的第一个单元
10 ax = fig.add_subplot(111)
11 # 调用 hist 生成直方图; hist 为每个直方图 Rectangle artist, 并将它们添加到
    Axes 容器中。这里“50”表示创建 50 个条形图
12 ax.hist(x, 50)
13 # 为直方图添加标题并存储
14 ax.set_title('Normal distribution with  $\mu=0, \sigma=1$ ')
15 fig.savefig('matplotlib_histogram.png')

```

运行结果如图10-6所示。

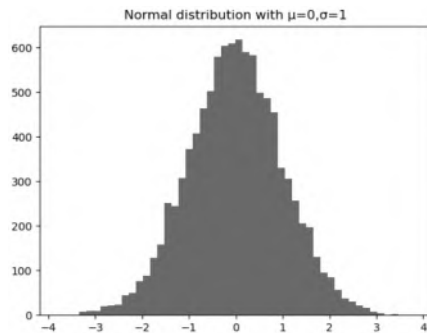


图10-6 正态分布图

Scripting Layer是Matplotlib的更高层封装，也就是我们平时所用的高级封装pyplot模块，使用Scripting不用设计如何构建图片，只需要根据需求设计图片显示效果即可。Backend、Artist需要有很多代码才能完成的事，而使用Scripting一段代码就可以实现。例如，使用下面代码即可快速设计并实现上面的小练习。

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 x = np.random.randn(10000)
4 plt.hist(x, 50)
5 plt.title(r'Normal distribution with  $\mu=0$ ,  $\sigma=1$ ')
6 plt.savefig('matplotlib_histogram.png')
7 plt.show()
```

10.2 Matplotlib绘制折线图

折线图可用于显示数据在一个连续的时间间隔或者时间跨度上的变化，它的特点是反映事物随时间或有序类别而变化的趋势。在折线图中，数据是递增还是递减、增减的速率、增减的规律（周期性、螺旋性等）、峰值等特征都可以清晰地反映出来。因此，折线图常用来分析数据随时间变化的趋势，也可用来分析多组数据随时间变化的相互作用和相互影响。

10.2.1 折线图的应用场景

折线图常用于有序因变量的场景，例如，某监控系统的折线图表，显示了请求次数和响应时间随时间的变化趋势。但是，当水平轴的数据类型为无序的分类或者垂直轴的数据类型为连续时间时，不适合使用折线图；另外，当折线的条数过多时，也不建议使用折线图。

10.2.2 折线图的绘制示例

本例使用Matplotlib绘制一个简单的折线图，再对其进行定制，以实现信息更加丰富的数据可视化。下面是绘制横坐标为（1，2，3，4，5，6）的折线图的相关代码：

```
1 import matplotlib
2 import matplotlib.pyplot as plt
3 # 加入中文显示
4 import matplotlib.font_manager as fm
5
6 # 解决中文乱码, 本案例使用的字体为宋体
7 myfont=fm.FontProperties(fname=r"C:\\Windows\\Fonts\\simSun.ttc")
8
9 def line_chart(xvalues,yvalues):
10     # 绘制折线图, c 为颜色设置, alpha 表示透明度
11     plt.plot(xvalues,yvalues,linewidth=5,alpha=0.5,c='red')
12     # num_squares 为数据值, linewidth 用于设置线条的粗细
13     # 设置折线图标题和横纵坐标标题
14     plt.title("Python 绘制折线图",fontsize=20,fontname='宋体',
15             fontproperties=myfont)
16     plt.xlabel('横坐标',fontsize=15,fontname='宋体',
17             fontproperties=myfont)
18     plt.ylabel('纵坐标',fontsize=15,fontname='宋体',
19             fontproperties=myfont)
20     # 设置刻度标记大小, axis='both' 参数影响横纵坐标, labelsizes 为刻度大小
21     plt.tick_params(axis='both',labelsize=10)
22     # 显示图形
23     plt.show()
```

运行结果如图10-7所示。

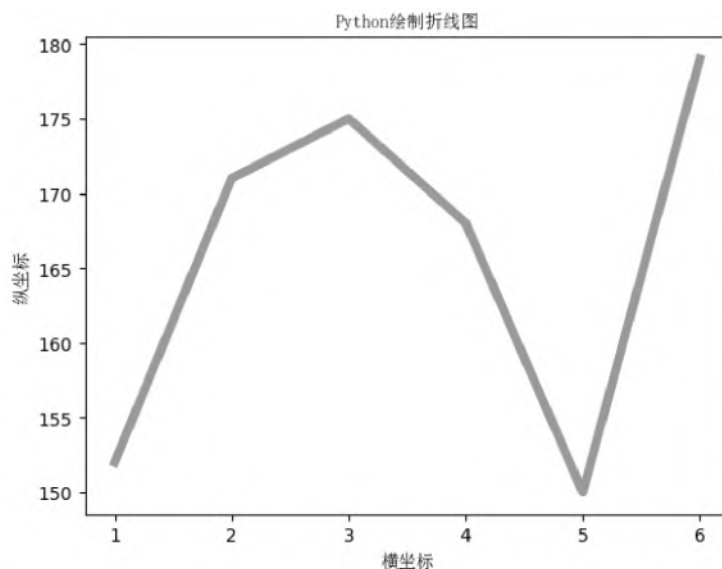


图10-7 折线图

10.3 Matplotlib绘制散点图

散点图也叫X-Y图，它是将所有的数据以点的形式展现在直角坐标系上，以显示变量之间的相互影响程度，点的位置由变量的数值决定。

通过观察散点图上数据点的分布情况，我们可以推断出变量间的相关性。如果变量之间不存在相互关系，那么在散点图上就会表现为随机分布的离散的点，如果存在某种相关性，那么大部分的数据点就会相对密集并以某种趋势呈现。数据的相关关系主要分为：正相关（两个变量值同时增长）、负相关（一个变量值增加另一个变量值下降）、不相关、线性相关、指数相关等。那些离点集群较远的点称为离群点或者异常点。

散点图经常与回归线（最准确地贯穿所有点的线）结合使用，用来归纳分析现有数据以进行预测分析。

10.3.1 散点图的应用场景

散点图通常用于显示和比较数值。不仅可以显示趋势，还能显示数据集群的形状，以及在数据云团中各数据点的关系。常见的散点图主要用于男女身高和体重等离散型数值的展示。

10.3.2 散点图的绘制示例

本例使用Matplotlib绘制一个简单的散列点图，再对其进行定制，以实现信息更加丰富的数据可视化，以下为绘制散点图的相关代码：


```

1  import matplotlib
2  import matplotlib.pyplot as plt
3
4  # 加入中文显示
5  import matplotlib.font_manager as fm
6  # 解决中文乱码，本案例使用的字体为宋体
7  myfont=fm.FontProperties(fname=r"C:\\Windows\\Fonts\\simsum.ttc")
8
9  def scatter_chart(xvalues,yvalues):
10     # 绘制散点图，s 设置点的大小，c 为数据点的颜色，edgecolors 为数据点的轮廓
11     plt.scatter(xvalues,yvalues,c='green',edgecolors='none',s=20)
12     # 设置散点图标题和横纵坐标标题
13     plt.title("Python 绘制折线图",fontsize=30,fontname='宋体',
14               fontproperties=myfont)
15     plt.xlabel('横坐标',fontsize=20,fontname='宋体',
16               fontproperties=myfont)
17     plt.ylabel('纵坐标',fontsize=20,fontname='宋体',
18               fontproperties=myfont)
19     # 设置刻度标记大小，axis='both' 参数影响横纵坐标，labelsize 为刻度大小
20     plt.tick_params(axis='both',which='major',labelsize=10)
21     # 设置每个坐标轴的取值范围
22     plt.axis([80,100,6400,10000])
23     # 显示图形
24     plt.show()
25     # 自动保存图表，bbox_inches 剪除图片空白区
26     # plt.savefig('squares_plot.png',bbox_inches='tight')
27
28 if __name__ == '__main__':
29     xvalues = list(range(1,100)) # 校正坐标点，即横坐标值列表
30     yvalues = [x**2 for x in xvalues] # 纵坐标值列表
31     scatter_chart(xvalues,yvalues)

```

运行结果如图10-8所示。

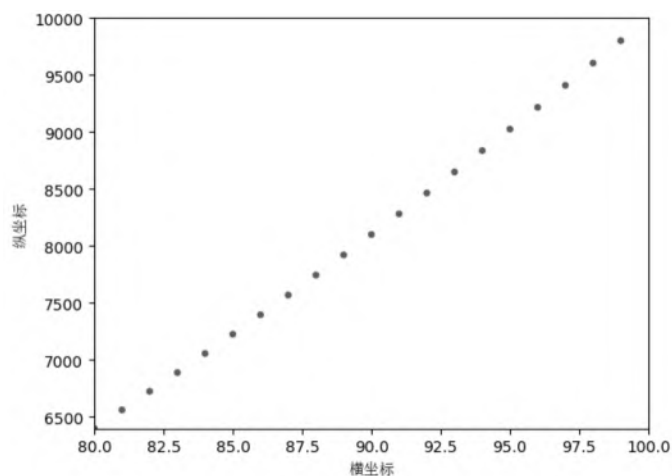


图10-8 散点图

10.4 Matplotlib绘制直方图

直方图是一种对数据分布情况的图形表示，是一种二维统计图表，它的两个坐标分别是统计样本和该样本对应的某个属性的度量。利用直方图可以很清晰地看出每个类的总和及各个属性的比例，但是不容易看出各个属性的频数。

10.4.1 直方图的应用场景

直方图适合应用于分类数据对比，通常使用矩形的长度（宽度）来对比分类数据的大小，非常方便临近的数据进行大小的对比，比如一个游戏销量的图表，展示不同游戏类型的销量对比。但是当分类太多时，则不适合使用直方图，如对比不同省份的人口数量。另外在展示连续数据趋势等问题时，也不适合使用直方图。

10.4.2 直方图的绘制示例

本例使用Matplotlib绘制一个简单的直方图，再对其进行定制，以实现信息更加丰富的数据可视化。下面为绘制直方图的相关代码：

```
1  def histogram(xvalues,yvalues):
2      # 绘制直方图
3      hist = pygal.Bar()
4      # 设置散点图标题和横纵坐标标题
5      hist.title = '事件频率的直方图'
6      hist.x_title = '事件的结果'
7      hist.y_title = '事件的频率'
8      # 绘制气温图，设置图形大小
9      fig = plt.figure(dpi=100,figsize=(10,6))
10     # 事件的结果
11     hist.x_labels = xvalues
12     # 事件的统计频率
13     hist.add('事件',yvalues)
14     # 保存文件路径
15     hist.render_to_file('die_visual.svg')
16
17  if __name__ == '__main__':
18     x_result = [1,2,3,4,5,6]
19     y_frequencies = [152,171,175,168,150,179]
20     histogram(x_result,y_frequencies)
```

运行结果如图10-9所示。

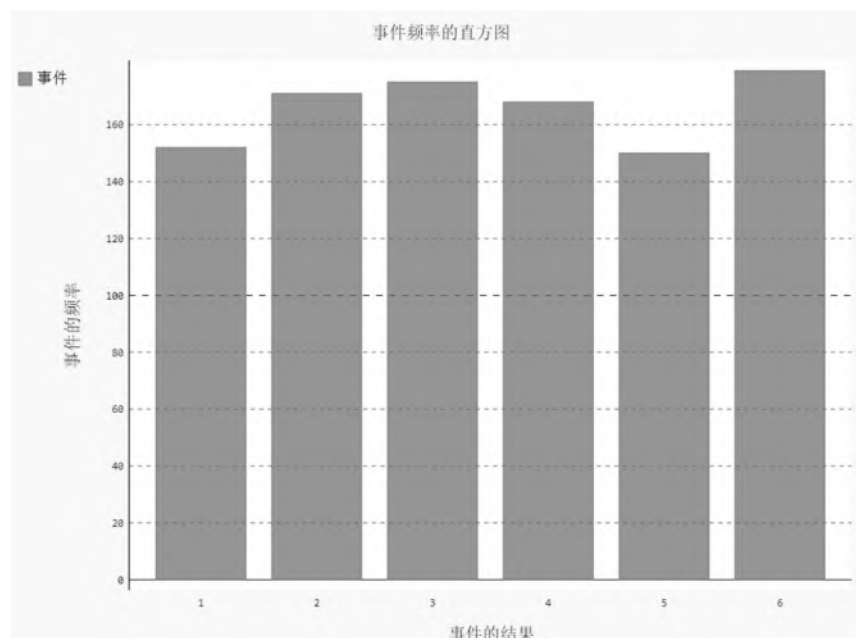


图10-9 直方图

10.5 练习：Matplotlib绘制气温图

这一节我们主要实现一个数据可视化的小练习，对某城市气温的CSV文件进行处理，提取天气数据绘制气温图。在绘制气温图中添加图例，绘制最高气温和最低气温的折线图，对气温区域进行着色。下面为某城市2018年7月气温图绘制的代码（源代码见：Chapter10/temper.py）：

```

1  def temper_char():
2      fig = plt.figure() # 将画布划分为1行1列1块
3      dates, highs, lows = [], [], []
4      with open(r'./weather07.csv') as f:
5          reader = csv.reader(f)
6          header_row = next(reader) # 返回文件第一行
7          for row in reader:
8              current_date = datetime.strptime(row[0], "%Y-%m-%d")
9              dates.append(current_date)
10             highs.append(int(row[1]))
11             lows.append((int(row[3])))
12
13     # 接收数据并绘制图形, facecolor 为填充区域颜色
14     plt.plot(dates, highs, c='red', linewidth=2, alpha=0.5)
15     plt.plot(dates, lows, c='green', linewidth=2, alpha=0.5)
16     plt.fill_between(dates, highs, lows, facecolor='blue', alpha=0.2)
17     # 设置散点图标题和横纵坐标标题
18     plt.title("日常最高气温, 2018 年 7 月", fontsize=10, fontname='宋体',
fontproperties=myfont)
19     plt.xlabel('横坐标', fontsize=10, fontname='宋体',
fontproperties=myfont)
20     plt.ylabel('温度', fontsize=10, fontname='宋体',
fontproperties=myfont)
21     # 绘制的日期
22     fig.autofmt_xdate()
23     # 设置刻度标记大小, axis='both' 参数影响横纵坐标, labelszize 为刻度大小
24     plt.tick_params(axis='both', which='major', labelszize=8)
25     # 显示图形
26     plt.show()
27
28  if __name__ == '__main__':
29     xvalues = list(range(1,100)) #校正坐标点, 即横坐标值列表
30     yvalues = [x**2 for x in xvalues] # 纵坐标值列表
31
32     x_result = [1,2,3,4,5,6]
33     y_frequencies = [152,171,175,168,150,179]
34
35     temper_char()

```

运行结果如图10-10所示。

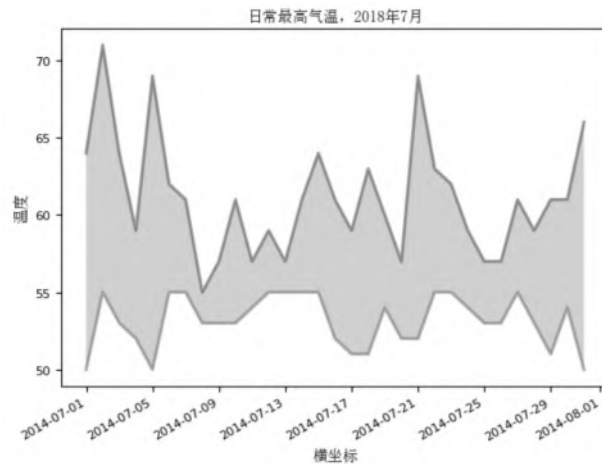


图10-10 某市最低、最高气温图

10.6 练习：Matplotlib绘制三维图

本节主要实现两个数据可视化的小练习：绘制三维梯度下降图和绘制三维散点图。

10.6.1 练习1：绘制三维梯度下降图

在机器学习领域，梯度下降法作为基础优化算法具有较多的应用。使用Matplotlib绘制三维梯度下降，能够详细展示算法梯度下降的过程，从而优化改进算法的不足之处。下面为绘制三维梯度下降图的代码：

```

1  # 绘制三维梯度下降图
2  def d3_hookface():
3      fig = plt.figure() # 得到画面
4      ax = fig.gca(projection='3d') # 得到三维坐标的图
5      X = np.arange(-5, 5, 0.1)
6      Y = np.arange(-5, 5, 0.1)
7      X,Y = np.meshgrid(X, Y) # 将坐标向量变为坐标矩阵，列为 x 的长度，行为
    y 的长度
8      R = np.sqrt(X**2 + Y**2)
9      Z = np.sin(R)
10     # 曲面, x,y,z 坐标, 横向步长, 纵向步长, 颜色, 线宽, 是否渐变
11     surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
    cmap=cm.coolwarm, linewidth=0, antialiased=False)
12     ax.set_zlim(-1.01, 1.01)
13
14     ax.set_xlabel("x-label", color='r')
15     ax.set_ylabel("y-label", color='g')
16     ax.set_zlabel("z-label", color='b')
17
18     ax.zaxis.set_major_locator(LinearLocator(10)) # 设置 z 轴标度
19     ax.zaxis.set_major_formatter(FormatStrFormatter('%0.02f')) #
    设置 z 轴精度
20     # shrink 颜色条伸缩比例 0-1, aspect 颜色条宽度 (反比例, 数值越大宽度越窄)
21     fig.colorbar(surf, shrink=0.5, aspect=5)
22
23     plt.savefig("d3_hookface.png")
24     plt.show()
25
26 if __name__ == '__main__':
27     # 绘制三维梯度下降图
28     d3_hookface()

```

运行结果如图10-11所示。

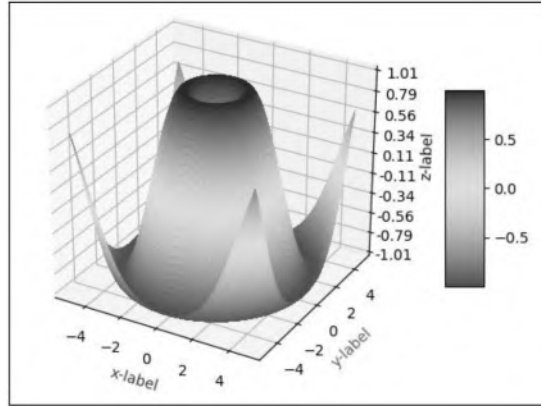


图10-11 三维梯度图

10.6.2 练习2：绘制三维散点图

三维散点图适用于若干数据系列中各数值之间关系的展示，例如构建XYZ轴展示数据在3个变量之间是否存在某种关联。常用的平面散点图对于处理值的分布和数据点的分簇，散点图都很理想，但如果需要展示数据三维空间中的关联，那么三维散点图便是最佳图表类型。下面为随机生成数据的三维散点图代码：


```

1  def d3_scatter():
2      fig = plt.figure()
3      ax = fig.add_subplot(111, projection='3d')
4      n = 50
5      for c, m, zlow, zhigh in [('r', 'o', -50, -25), ('g', '+', -20,
15), ('b', '^', -30, -5)]:
6          xs = randrange(n, 23, 32)
7          ys = randrange(n, 0, 100)
8          zs = randrange(n, zlow, zhigh)
9          ax.scatter(xs, ys, zs, c=c, marker=m)
10     ax.set_xlabel('X Label')
11     ax.set_ylabel('Y Label')
12     ax.set_zlabel('Z Label')
13     plt.show()
14
15 def randrange(n, vmin, vmax):
16     return (vmax - vmin) * np.random.rand(n) + vmin
17
18 if __name__ == '__main__':
19     # 绘制三维散点图
20     d3_scatter()

```

运行结果如图10-12所示。

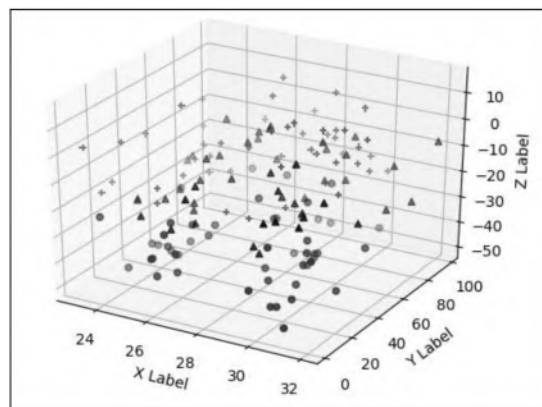


图10-12 三维数据散点图

10.7 本章小结

本章介绍了如何利用相关数据可视化工具让我们更好地理解数据分析的结果，还介绍了Python数据预处理的数据可视化，解释了Matplotlib架构各层所负责的功能，着重介绍了数据可视化常用的图形及其应用场景，最后通过案例练习介绍了绘制了气温图和三维图的代码实现。下一章我们将综合以上数据处理过程，将其应用在XGBoost文本分类算法上。

第11 章

竞赛神器XGBoost

本章介绍一款号称竞赛神器的XGBoost，很多Kaggle（一个数据科学竞赛的平台）比赛的冠军采用的都是XGBoost算法。XGBoost近些年在学术界取得的成果捷报连连，在诸多算法模型和深度学习框架中，XGBoost倍受关注的直接原因是其拥有运行速度快（以特征为单位并行处理）、内置交叉验证、正则化提升，适合不均衡数据，支持自定义优化目标和评价指标、自动处理缺失值和字符型特征、自动处理相关特征等优势，总之省心又省事。本章我们将从认知XGBoost到模型参数调优等方面深入学习该工具的使用。

11.1 XGBoost概述

11.1.1 认识XGBoost

XGBoost是一个开源软件库，它提供了一个梯度提高框架，可采用C++、Java、Python、R和Julia编程，它适用于Linux、Windows和Mac OS根据项目的描述，它的目的在于提供一个可扩展的、便携式和可分布的梯度提高（GBM、GBRT、GBDT）库。除了在单一的机器上运行，XGBoost还支持分布式框架Apache Hadoop、Apache Spark、Apache Flink。近几年，由于受到许多机器学习竞赛获奖团队的青睐，XGBoost逐渐成为机器学习领域具有较强实用性的算法工具之一。XGBoost 也是一款开源项目，其GitHub源码网址为：<https://github.com/dmlc/xgboost>，如图11-1所示。

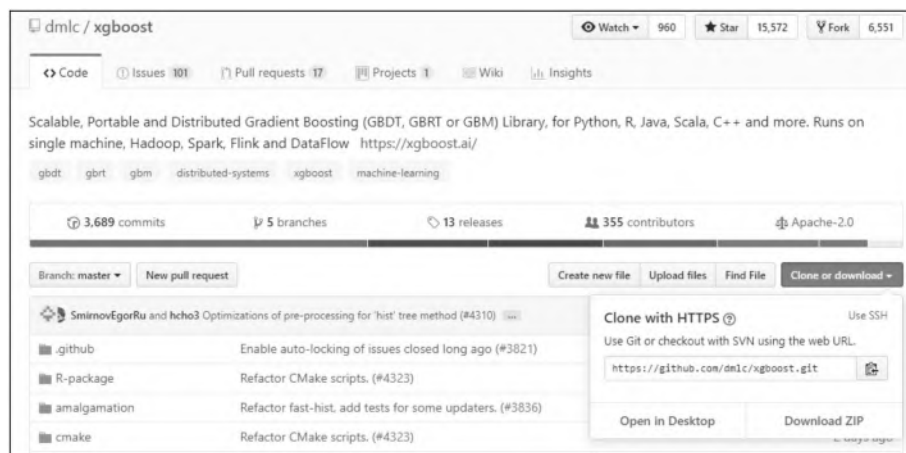


图11-1 XGBoost的GitHub源码

XGBoost最初是一个研究项目，由当时在Distributed (Deep) Machine Learning Community (DMLC) 组里的陈天奇负责。它开始是作为一个可由libsvm配置文件进行配置的终端应用程序，当其在Higgs机器学习挑战中获胜之后，开始在机器学习竞赛圈子中被大家熟知，不久之后，相应的Python和R的包被开发出来。现在，许多其他语言如Julia、Scala、Java等也有对应的包，这使得更多的开发者认识了XGBoost，也让其在Kaggle社区大受欢迎，并用于大量的比赛。

XGBoost与其他多个软件包一起使用，使其更易于在各自的社区中使用，还可以与Python用户的scikit-learn以及R的Caret集成。另外，也可以使用抽象的Rabit和XGBoost4J集成到Apache Hadoop、Apache Spark、Apache Flink等数据流框架中。

11.1.2 XGBoost的应用场景

XGboost能够在一系列的问题上取得良好的效果，这些问题包括存销预测、物理事件分类、网页文本分类、顾客行为预测、点击率预测、动机探测、产品分类等。XGboost在所有场景中提供可扩展的功能，该特性使其可扩展性保证了相比其他系统更快的速度。XGBoost算法的优势具体体现在：处理稀疏数据的新颖的树的学习算法和近似学

习的分布式加权直方图。XGBoost能够基于外存的计算，保障了大数据的计算，使用少量的节点资源可处理大量的数据。XGBoost的主要贡献是：

- 构建了高可扩展的端到端的boosting系统。
- 提出了具有合理理论支撑的分布分位调整框架。
- 介绍了一个新颖的并行适应稀疏处理树学习算法。
- 提出了基于缓存块的结构，便于外存树的学习。

11.2 XGBoost的优点

XGBoost可以防止过度拟合，支持各种分布式 / 并行处理，样本量和特征数据类型要求没那么苛刻，适用范围更广。陈天奇认为：基于树模型的XGBoost能很好地处理表格数据，同时还拥有一些深度神经网络所没有的特性（如模型的可解释性、输入数据的不变性、更易于调参等）。XGBoost的优点有：

- 正则化：XGBoost在代价函数里加入了正则项，用于控制模型的复杂度。使学习出来的模型更加简单，防止过度拟合，这也是XGBoost优于传统GBDT的一个特性。
- 并行处理：XGBoost工具支持并行。决策树的学习最耗时的一个步骤就是对特征值进行排序，XGBoost在训练之前，预先对数据进行了排序，然后保存为block结构，后面的迭代中重复地使用这个结构，大大减小了计算量。这个block结构也使得并行计算成为可能，因为在进行节点的分裂时，可以计算每个特征的增益，最终选增益最大的那个特征去做分裂，那么各个特征的增益计算就可以多线程进行。
- 灵活性高：XGBoost支持用户自定义目标函数和评估函数，只要目标函数二阶可导就行。

- 缺失值自动处理：对于特征值有缺失的样本，XGBoost可以自动学习出它的分裂方向。
- 内置交叉验证：XGBoost允许在每一轮boosting迭代中使用交叉验证。因此，可以方便地获得最优boosting迭代次数，而GBM（Gradient Boosting Machine）使用网格搜索，只能检测有限个值。

11.3 使用XGBoost预测毒蘑菇

本节首先介绍XGBoost的开发环境，然后通过毒蘑菇预测案例进一步了解XGBoost的运行过程。

11.3.1 XGBoost的开发环境及安装

本例使用的开发环境如下：

- Python版本——Python 3.7.1
- 操作系统——Windows 10 64bit
- 集成开发环境——Sublime Text 3.1.1

可以使用以下命令安装XGBoost： `pip install xgboost`。然后检查XGBoost模块是否安装成功，成功安装的结果如图11-2所示。

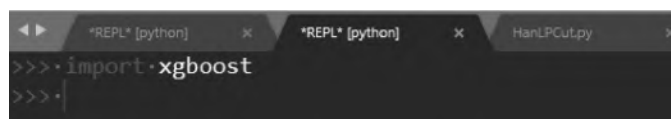


图11-2 成功导入XGBoost

11.3.2 数据准备

首先从官方网站（<https://github.com/rosefun/xgboost-1/tree/master/demo>）下载蘑菇数据集的训练数据agaricus.txt.train和测试数据agaricus.txt.test。摘取前3条test数据，其形式如下：

```
1  0 1:1 9:1 19:1 21:1 24:1 34:1 36:1 39:1 42:1 53:1 56:1 65:1 69:1 77:1
   86:1 88:1 92:1 95:1 102:1 106:1 117:1 122:1
2  1 3:1 9:1 19:1 21:1 30:1 34:1 36:1 40:1 41:1 53:1 58:1 65:1 69:1 77:1
   86:1 88:1 92:1 95:1 102:1 106:1 118:1 124:1
3  0 1:1 9:1 20:1 21:1 24:1 34:1 36:1 39:1 41:1 53:1 56:1 65:1 69:1 77:1
   86:1 88:1 92:1 95:1 102:1 106:1 117:1 122:1
```

其中每一行表示一个样本，第一列数字是类别标签，表示样本所属的类别，1代表有毒的蘑菇，0代表没有毒的蘑菇。第二列的1:1、3:1、1:1表示“特征索引：特征值”。后面各列分别表示特征索引和值。同时类别标签支持概率标签，取值范围为[0, 1]，表示样本属于某个类别的可能性。读取蘑菇数据集的训练集和测试集，代码如下（数据见：Chapter11/Data）：

```
1  dtrain = xgb.DMatrix(r'data/agaricus.txt.train')
2  dtest = xgb.DMatrix(r'data/agaricus.txt.test')
```

11.3.3 参数设置

接下来进行参数设置（具体参数的意义参见11.4节）。由于本例是一个二分类的逻辑回归问题，因此代码中使用了“objective’:’binary:logistic”，具体参数设置的代码如下：

```
1 param = {
2     'max_depth':2,
3     'eta':1,
4     'silent':1,
5     'objective':'binary:logistic'
6 }
```

11.3.4 模型训练

了解了训练数据和测试数据以及参数设置的情况，接下来实现毒蘑菇预测的算法过程，代码实现如下（源代码见：Chapter11/agaricus.py）：

```
1 import xgboost as xgb
2 # 读取数据
3 dtrain = xgb.DMatrix(r'data/agaricus.txt.train')
4 dtest = xgb.DMatrix(r'data/agaricus.txt.test')
5 # 通过 map 指定参数
6 param = {
7     'max_depth':2,
8     'eta':1,
9     'silent':1,
10    'objective':'binary:logistic'
11 }
12 num_round = 10
13 bst = xgb.train(param, dtrain, num_round)
14 # 预测
15 preds = bst.predict(dtest)
16 print('预测结果:\n',preds)
```

在上述代码中，第一行是导入XGBoost模块，第3、4行用于加载训练集和数据集。第13行是本算法的核心工作，其中param是参数设置，也是影响算法性能好坏的关键所在。dtrain是加载训练集模型进行学习，num_round用于控制迭代的次数。第15行就是对结果的预测。

运行结果如下：

```
[16:11:29] 6513x127 matrix with 143286 entries loaded from data/
agaricus.txt.train
[16:11:29] 1611x127 matrix with 35442 entries loaded from data/
agaricus.txt.test
```

以上结果显示：训练数据6513x127表示6513个蘑菇数据样本，每个样本包含127个数据特征。测试数据1611x127表示1611个蘑菇数据样本，每个样本包含127个数据特征。

预测结果如下：

```
[0.00501517 0.9884467 0.00501517 ... 0.9981102 0.00285519
0.9981102 ]
```

预测结果显示判断类别的概率，以前3个样本为例，预测有毒的蘑菇概率分别是：0.5%、98.8%和0.5%，即分类为（0，1，0）。那么，实际值是否与预测结果一致呢？下面对预测结果与实际结果进行对比验证，代码如下：

```

1 # 验证数据集
2 watch_list = [(dtest, 'eval'), (dtrain, 'train')]
3 # 模型训练
4 bst = xgb.train(param, dtrain, num_round, watch_list)
5 运行结果如下:
6 [16:21:36] 6513x127 matrix with 143286 entries loaded from
   data/agaricus.txt.train
7 [16:21:36] 1611x127 matrix with 35442 entries loaded from
   data/agaricus.txt.test
8 [0]    eval-error:0.042831 train-error:0.046522
9 [1]    eval-error:0.021726 train-error:0.022263
10 [2]    eval-error:0.006207 train-error:0.007063
11 [3]    eval-error:0.018001 train-error:0.0152
12 [4]    eval-error:0.006207 train-error:0.007063
13 [5]    eval-error:0      train-error:0.001228
14 [6]    eval-error:0      train-error:0.001228
15 [7]    eval-error:0      train-error:0.001228
16 [8]    eval-error:0      train-error:0.001228
17 [9]    eval-error:0      train-error:0

```

得到如下预测结果:

```

[0.00501517 0.9884467 0.00501517 ... 0.9981102 0.00285519
0.9981102 ]

```

从上述结果可以看到，每次迭代的验证错误和训练错误。随着数据不断地训练，正常情况下，这两个指标都会呈现减小的情况，理想状态下两个误差值为0，说明验证和训练的正确率都为100%。如果参数验证误差比较大，这个时候就需要进行调参，比如调节学习率、最大深度、随机种子等参数，直到满足我们的需求。

11.3.5 可视化特征排名

接下来查看一下毒蘑菇数据样本特征的重要程度，对代码进行以下修改：

```
1 from matplotlib import pyplot
2 xgb.plot_importance(bst)
3 pyplot.show()
```

特征重要程度可视化结果如图11-3所示。

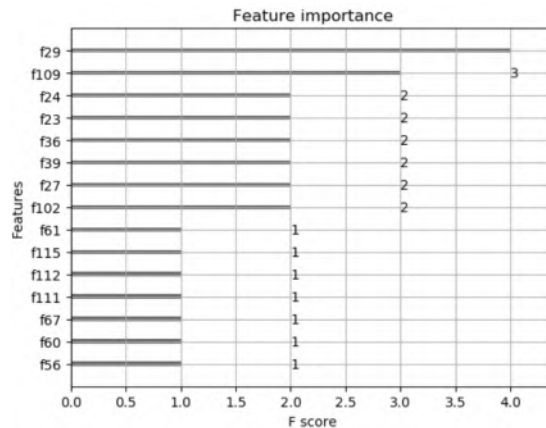


图11-3 毒蘑菇特征重要程度排名

从图11-3所示的可视化结果不难发现，特征f29最重要，F得分达到4，依次是f109和f24等，这些数据怎么理解呢？我们知道，毒蘑菇可能色彩比较鲜亮，形状比较奇特，假如这是判断毒蘑菇的主要标志，那么，f29就可以代表色彩，f109可以代表形状，而重量、高度等特征对是否为毒蘑菇的判断影响权重不大，可以是特征f60和f56。通过这个可视化结果，可以获取毒蘑菇的主要特征，为判断毒蘑菇提供了依据。

11.4 XGBoost优化调参

上节介绍了使用XGBoost训练分类器模型预测毒蘑菇的例子，在实际使用中，通常还要对分类器模型效果进行测评，如果其获得的结果不能满足需要，就要对其进行改进，本节来看看如何进行模型性能评估和参数调优。

11.4.1 参数解读

在运行XGBoost之前，必须设置三种类型的参数：常规参数（General Parameter），模型参数（Booster Parameter）和任务参数（Task Parameter）。

- 常规参数与我们用于提升的提升器有关，通常是树模型或线性模型。
- 模型参数取决于所选择的提升器。
- 任务参数决定了学习场景。

1. 常规参数

```
1 booster
2     gbtrees 树模型作为基分类器（默认）
3     gblinear 线性模型作为基分类器
4 silent
5     silent=0 时，不输出中间过程（默认）
6     silent=1 时，输出中间过程
7 nthread
8     nthread=-1 时，使用全部 CPU 进行并行运算（默认）
9     nthread=1 时，使用 1 个 CPU 进行运算。
```

- `booster [default=gbtree]`：可以选择`gbtree`和`gblinear`两种模型。`gbtree`使用基于树的模型进行提升计算，`gblinear`使用线性模型进行提升计算。默认值为`gbtree`。
- `silent [default=0]`：取0时表示打印出运行时的信息，取1时表示以缄默方式运行，不打印运行时的信息。默认值为0。
- `nthread`：XGBoost运行时的线程数。默认值是当前系统可以获得的最大线程数。

2. 模型参数

```
1  n_estimators
2      含义：总共迭代的次数，即决策树的个数。
3      调参：
4  early_stopping_rounds
5      含义：在验证集上，当连续 n 次迭代，分数没有提高后，提前终止训练。
6      调参：防止过度拟合。
7  max_depth
8      含义：树的深度，默认值为 6，典型值 3~10。
9      调参：值越大，越容易过度拟合；值越小，越容易欠拟合。
10 min_child_weight
11     含义：默认值为 1。
12     调参：值越大，越容易欠拟合；值越小，越容易过度拟合（值较大时，避免模型学习到局部的特殊样本）。
13 subsample
14     含义：训练每棵树时，使用的数据占全部训练集的比例。默认值为 1，典型值为 0.5-1。
15     调参：防止过度拟合。
16 colsample_bytree
17     含义：训练每棵树时，使用的特征占全部特征的比例。默认值为 1，典型值为 0.5-1。
18     调参：防止过度拟合。
19 scale_pos_weight
20     正样本的权重，在二分类任务中，当正负样本比例失衡时，设置正样本的权重，模型效果更好。例如，当正负样本比例为 1:10 时，scale_pos_weight=10。
```

参数的进一步说明：

- `Min_child_weight[default=1]`：每个叶子里面的h的和至少是多少，这个参数非常影响结果，控制叶子节点中二阶导的和的最小值，该参数越小，越容易过度拟合。
- `subsample[default=1]`：随机样本，该参数越大，越容易过度拟合，但设置过大也会造成过度拟合。
- `colsample_bytree[default=1]`：列采样，对每棵树生成时用的特征进行列采样，一般设置为0.5~1。

- `scaleposweight[default=1]`: 如果取值大于0, 在类别样本偏斜时, 有助于快速收敛。

3. 学习任务参数

```

1  learning_rate
2      含义：学习率，控制每次迭代更新权重时的步长，默认 0.3。
3      调参：值越小，训练越慢。
4      典型值为 0.01-0.2。
5  objective 目标函数
6      回归任务
7          reg:linear (默认)
8          reg:logistic
9      二分类
10         binary:logistic    概率
11         binary: logitraw    类别
12     多分类
13         multi: softmax    num_class=n    返回类别
14         multi: softprob    num_class=n    返回概率
15     rank:pairwise
16 eval_metric
17     回归任务 (默认 rmse)
18         rmse--均方根误差
19         mae--平均绝对误差
20     分类任务 (默认 error)
21         auc--roc 曲线下面积
22         error--错误率 (二分类)
23         merror--错误率 (多分类)
24         logloss--负对数似然函数 (二分类)
25         mlogloss--负对数似然函数 (多分类)
26
27 gamma
28     惩罚项系数，指定节点分裂所需的最小损失函数下降值。
29     调参：
30 alpha
31     L1 正则化系数，默认为 1。
32 lambda
33     L2 正则化系数，默认为 1。

```

参数的进一步说明：

- `objective [default=reg:linear]`: 定义学习任务及相应的学习目标，可选的目标函数如下：`reg:linear`表示线性回归，`reg:logistic`表示逻辑回归，`binary:logistic`表示二分类的逻辑回归，`multi:softmax`表示多分类问题，同时需设置参数`num_class`（类别个数），`multi:softprob`和`softmax`一样，但是输出的是`ndata×nclass`的向量，可以将该向量重新转换（`reshape`）成`ndata`行`nclass`列的矩阵。`num_class`表类别个数，与`multissoftmax`并用。
- `eval_metric[default according to objective]`: 用于衡量验证数据的参数，即评价标准，常用参数如图11-4所示。
- `gamma [default=0]`: 用于控制是否后剪枝的参数，越大越保守，一般为0.1、0.2。取值范围为：`[0, ∞]`。
- `alpha[default=1]`: 模型的L1正则化参数，参数越大，越不容易过度拟合。
- `lambda [default=1]`: 模型的L2正则化参数，参数越大，越不容易过度拟合。

Scoring	Function	Comment
Classification		
'accuracy'	metrics.accuracy_score	
'average_precision'	metrics.average_precision_score	
'f1'	metrics.f1_score	for binary targets
'f1_micro'	metrics.f1_score	micro-averaged
'f1_macro'	metrics.f1_score	macro-averaged
'f1_weighted'	metrics.f1_score	weighted average
'f1_samples'	metrics.f1_score	by multilabel sample
'neg_log_loss'	metrics.log_loss	requires predict_proba support
'precision' etc.	metrics.precision_score	suffixes apply as with 'f1'
'recall' etc.	metrics.recall_score	suffixes apply as with 'f1'
'roc_auc'	metrics.roc_auc_score	
Clustering		
'adjusted_mutual_info_score'	metrics.adjusted_mutual_info_score	
'adjusted_rand_score'	metrics.adjusted_rand_score	
'completeness_score'	metrics.completeness_score	
'fowlkes_mallows_score'	metrics.fowlkes_mallows_score	
'homogeneity_score'	metrics.homogeneity_score	
'mutual_info_score'	metrics.mutual_info_score	
'normalized_mutual_info_score'	metrics.normalized_mutual_info_score	
'v_measure_score'	metrics.v_measure_score	
Regression		
'explained_variance'	metrics.explained_variance_score	
'neg_mean_absolute_error'	metrics.mean_absolute_error	
'neg_mean_squared_error'	metrics.mean_squared_error	
'neg_mean_squared_log_error'	metrics.mean_squared_log_error	
'neg_median_absolute_error'	metrics.median_absolute_error	
'r2'	metrics.r2_score	

图11-4 eval_metric评价指标规范

11.4.2 调参原则

先固定一个参数，最优化后继续调整，一般调参步骤如下：

(1) 确定学习速率和`tree_based`，给一个常见的初始值，根据是否类别不平衡来进行调整。

- `max_depth=3`，起始值在4~6之间都是不错的选择。
- `min_child_weight`，设置比较小的值解决极不平衡的分类问题，如设置为1。
- `subsample, colsample_bytree=0.8`：最常见的初始值。
- `scale_pos_weight=1`：设置为1表示类别十分不平衡。

(2) `max_depth`和`min_weight`的值对最终结果会产生很大的影响，如以下示例：

- `max_depth:range(3, 10, 2)`
- `min_childweight:range(1, 6, 2)`

可以先大范围地粗调参数，然后再小范围微调。

(3) `gamma`参数调优。

- `gamma:[i/10.0 for i in range(0, 5)]`

(4) 调整`subsample`和`colsample_bytree`参数。

- `subsample:[i/100.0 for i in range(75, 90, 5)]`,
- `colsample_bytree:[i/100.0 for i in range(75, 90, 5)]`

(5) 正则化参数调优。

- `reg_alpha:[1e-5, 1e-2, 0.1, 1, 100]`

(6) 降低学习速率。

- `learning_rate=0.01,`

11.4.3 调参技巧

1. 控制过度拟合

当观察到训练精度高，但测试精度低时，可能是遇到了过度拟合的问题。通常有两种方法可以用于控制XGBoost中的过度拟合。

- 第一种方法是直接控制模型的复杂度，即调整`max_depth`、`min_child_weight`和`gamma`参数的值。
- 第二种方法是增加随机性，使训练对噪声的鲁棒性，包括：
 - ◆ 调整`subsample`和`colsample_bytree`的值。
 - ◆ 也可以减小步长`eta`，但是当这么做时，需要记得增加`num_round`的值。

2. 处理不平衡的数据集

对于广告点击日志等常见情况，数据集是极不平衡的。这可能会影响XGBoost模型的训练，通过以下两种方法可以改善这种情况。

- 如果只关心预测的排名顺序（AUC），那么，
 - ◆ 可以通过调整`scale_pos_weight`的值来平衡`positive`（正面）和`negative`（负面）的权重。
 - ◆ 使用AUC进行评估。

- 如果关心预测正确的概率，则无法重新平衡数据集，此时，将参数max_delta_step设置为有限数字（比如说1）将有助于收敛。

11.5 预测糖尿病患者

11.5.1 数据准备

本节以数据科学竞赛平台Kaggle数据分析比赛涉及的预测糖尿病发病的项目为例进行实战学习，其中数据diabetes.csv可以在本书源代码中下载，数据的表示形式如图11-5所示。

	A	B	C	D	E	F	G	H	I
1	Pregnanci	Glucose	BloodPres	SkinThick	Insulin	EMI	DiabetesF	Age	Outcome
2	6	148	72	35	0	33.6	0.627	50	1
3	1	85	66	29	0	26.6	0.351	31	0
4	8	183	64	0	0	23.3	0.672	32	1
5	1	89	66	23	94	28.1	0.167	21	0
6	0	137	40	35	168	43.1	2.288	33	1
7	5	116	74	0	0	25.6	0.201	30	0
8	3	78	50	32	88	31	0.248	26	1
9	10	115	0	0	0	35.3	0.134	29	0
10	2	197	70	45	543	30.5	0.158	53	1
11	8	125	96	0	0	0	0.232	54	1
12	4	110	92	0	0	37.6	0.191	30	0
13	10	168	74	0	0	38	0.537	34	1
14	10	139	80	0	0	27.1	1.441	57	0
15	1	189	60	23	846	30.1	0.398	59	1
16	5	166	72	19	175	25.8	0.587	51	1

图11-5 糖尿病发病预测病历数据

数据中的每一行表示一个病历样本，每一列表示病症特征，最后一列表示样本所属的类别，1代表患有糖尿病，0代表没有患糖尿病。其中各个特征的中文含义如下：

- Pregnancies: 怀孕次数
- Glucose: 葡萄糖
- BloodPressure: 血压（mm Hg）
- SkinThickness: 皮层厚度（mm）
- nsulin: 胰岛素2小时血清胰岛素（mu U/ml）

- BMI：体重指数（体重 / 身高）²
- DiabetesPedigreeFunction：糖尿病谱系功能
- Age：年龄（岁）
- Outcome：类标变量（0或1）

11.5.2 预测器模型构建

1. 导入模块

本项目需要导入的模块包括Numpy、Pandas、XGBoost、scikit-learn、Matplotlib，其中Numpy和Pandas用于数据处理，XGBoost用于模型训练，scikit-learn用于调用机器学习库评估模型性能，Matplotlib用于可视化显示，完整的模块导入代码如下（源代码见：Chapter11/dome.py）：

```
1 import pandas as pd # 数据科学计算工具
2 import xgboost as xgb
3 from xgboost.sklearn import XGBClassifier,XGBRegressor
4 from sklearn import metrics
5 from sklearn.model_selection import train_test_split,GridSearchCV
6 from sklearn.metrics import accuracy_score
7 from xgboost import plot_importance
8 from matplotlib import pyplot
```

2. 加载数据集

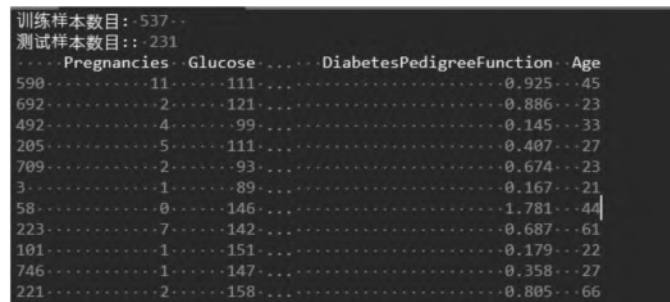
数据集为糖尿病病历信息，以CSV格式存储（见图11-5）。需要加载数据的特征值和特征标签，其中前8列为特征数据，最后一列为分类标签数据。提取的特征数据用于训练模型，而标签数据则用于验证模型的性能。这里调用sklearn内置的`traintestsplit`方法划分数据集，其中`seed`为随机种子，`test_size`为测试数据集占总数据集的比例，具体实现代码如下：

```

1  # 1 加载数据集
2  def loadData(seed = 7, test_size = 0.3):
3      pima = pd.read_csv("../data/diabetes.csv")
4      trains = pima.iloc[:,0:8] # 特征值
5      labels = pima.iloc[:,8]   # 标签值
6      train_data, test_data, train_label, test_label =
        train_test_split(trains, labels, test_size=test_size,
        random_state=seed)
7      print("训练样本数目:", len(train_label), " \n 测试样本数目::",
        len(test_label))
8      return train_data, test_data, train_label, test_label

```

运行loadData方法对数据进行特征集和标签集划分，并将数据转化为特定格式便于数据处理。其运行结果如图11-6所示。



```

训练样本数目: 537
测试样本数目: 231
Pregnancies Glucose DiabetesPedigreeFunction Age
590 11 111 0.925 45
692 2 121 0.886 23
492 4 99 0.145 33
205 5 111 0.407 27
709 2 93 0.674 23
3 1 89 0.167 21
58 0 146 1.781 44
223 7 142 0.687 61
101 1 151 0.179 22
746 1 147 0.358 27
221 2 158 0.805 66

```

图11-6 加载糖尿病病历数据

3. 模型训练

因为是二分类问题，所以我们选择XGBClassifier方法。首先对其进行参数设置，各个参数初步设置为默认值或常用值，其参数含义请参照上文的参数说明。然后进行模型填充model.fit，其中调整参数train_data、train_label、evalmetric='auc'、verbose=True、evalset=[(testdata, testlabel)]、early_stopping_rounds=100。这里的traindata和trainlabel分别代表训练特征数据和训练标签数据。Eval_metric为对模型性能评价的方法，可参考eval_metric评价指标规范。eval_set用来监控每一步的运行结果。

`early_stopping_rounds`为设置多少次迭代没有找到模型最佳得分而停止。

通过预测结果和真实的结果对比，对预测器性能进行综合评估，这里直接调用sklearn的metrics方法。主要评估参数是AUC和ACC，其中AUC是ROC曲线下的面积，介于0.1~1之间，AUC作为数值可以直观地评价分类器的好坏，值越大越好。ACC则用来反映模型预测的准确率。完整的代码如下：

```

1  def xgboost_train(train_data, test_data, train_label, test_label):
2      model = xgb.XGBClassifier(
3          max_depth=3,          # 构建树的深度，默认为 6
4          min_child_weight=1,   # 值过高欠拟合，调整参数 cv
5          learning_rate=0.1,    # 学习率
6          n_estimators=500,     # 迭代次数，值太小易欠拟合
7          silent=1,            # 取 0 时表示打印出运行时的信息
8          objective='binary:logistic', # 二分类的逻辑回归
9          gamma=0,              # 控制是否后剪枝的参数
10         max_delta_step=0,      # 限制每棵树权重改变的最大步长。
11         subsample=0.8,         # 随机采样的比例，值过小会欠拟合
12         colsample_bytree=0.8,  # 控制每棵树随机采样的列数的占比
13         reg_alpha=0,           # L1 正则化参数，越大越不容易过度拟合
14         reg_lambda=0,          # L2 正则化的惩罚系数，默认为 1
15         scale_pos_weight=1,    # 类别高度不平衡参数，设置为 0 加快收敛
16         seed=1,                # 随机数的种子，缺省值为 0
17     )
18     model.fit(train_data, train_label, eval_metric='auc',
19             verbose=True,
20             eval_set=[(test_data, test_label)],
21             early_stopping_rounds=100)
22     y_pre = model.predict(test_data)
23     y_pro = model.predict_proba(test_data)[:, 1]
24     # AUC: Roc 曲线下的面积，介于 0.1 和 1 之间。Auc 作为数值可以直观地评价分类
25     # 器的好坏，值越大越好
26     print("AUC Score : %f" % metrics.roc_auc_score(test_label,
27     y_pro))
28     print("Accuracy : %.4g" % metrics.accuracy_score(test_label,
29     y_pre))
30
31 if __name__ == '__main__':
32     train_data, test_data, train_label, test_label = loadData()
33     xgboost_train(train_data, test_data, train_label, test_label)

```

4. 结果分析

执行XGBoost模型训练方法得到的结果如图11-7所示，其中在迭代到第45次时模型最佳，AUC得分为0.8384，其准确率为77.49%。注意，如果迭代了100次没有出现最佳AUC得分时，这时可以考虑适当提高early_stopping_rounds的值，直到最佳得分。到目前为止，预测器效果并不是特别理想，接下来通过模型调参来提高性能。

```
[140]—validation_0 auc:0.817622
[141]—validation_0 auc:0.818027
[142]—validation_0 auc:0.81827
[143]—validation_0 auc:0.818027
[144]—validation_0 auc:0.818027
[145]—validation_0 auc:0.817217
Stopping. Best iteration:
[45]—validation_0 auc:0.838354

AUC Score : 0.838354
Accuracy : 0.7749

***Repl Closed***
```

图11-7 预测器模型的性能

5. 可视化特征排名

查看一下对于糖尿病指标特征的重要程度，可以将代码修改如下：

```
1 from matplotlib import pyplot
2 # 重要特征
3 plot_importance(model)
4 pyplot.show()
```

特征重要程度可视化结果如图11-8所示。

从图11-8的结果可以发现，横坐标为各个指标特征的F得分，F得分越高，影响因子越大；横坐标为各个指标特征的名称。其中对糖尿病影响最大的三个指标是DiabetesPedigreeFunction（糖尿病谱系功能）、BMI（体重指数）、Glucose（葡萄糖），这也符合常识。

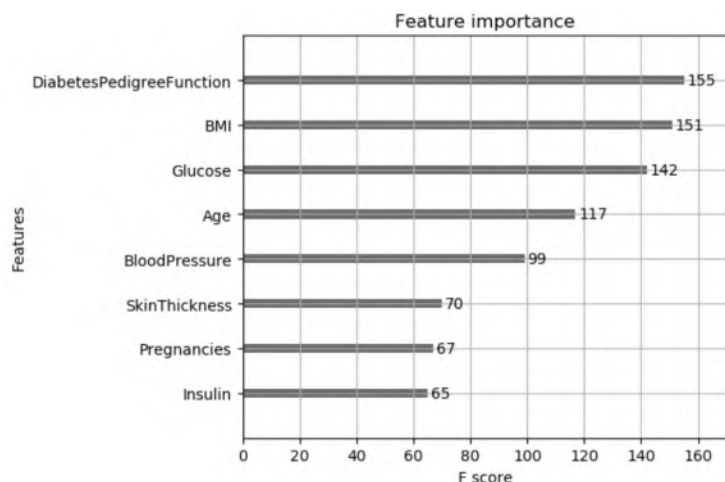


图11-8 糖尿病指标特征排名

11.5.3 调参提高预测器的性能

按照调参原则，我们选择对 `n_estimators`、`max_depth`、`min_weight`、`gamma`、`subsample` 及 `colsample_bytree` 这几个参数调出最优值。这里需要调用 `sklearn` 中的 `GridSearchCV` 方法进行调参，该方法的详细说明请参考官方文档，网址为：https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html。

1. 最佳迭代次数

本代码中的 `GridSearchCV` 方法中包含4个参数，可以根据实际情况来设置，其中的 `estimator` 为分类器模型参数、`paramgrid` 为需要验证调试的参数且为字典形式、`scoring` 为评价指标（可以根据需求设置不同的评价方法）、`cv` 是多折交叉验证（经过测试默认选择5是一个不错的选择）。方法 `GridSearchCV` 返回的三个值，分别是 `cvresults`（每一步执行结果）、`best_params`（最佳参数）和 `best_score_`（最佳模型得分）。

在保持其他参数不变的前提下，对n_estimators进行调参，由于n_estimators默认等于500，这里调参范围可以选择[300, 400, 500, 600, 700]，将默认值设置为一个中位数。根据每轮最优值的偏大或偏小，随时改变参数，直到参数为最优值。具体实现的代码如下：

```

1  # 1 最佳迭代次数
2  def n_estimators():
3      param_test = {
4          'n_estimators':[5,10,20,30,400,500,700]
5      }
6      model = xgb.XGBClassifier(
7          max_depth=3,          # 构建树的深度，默认为 6
8          min_child_weight=1,   # 值过高欠拟合，调整参数 CV
9          learning_rate=0.1,    # 学习率
10         # n_estimators=500,    # 迭代次数，值太小易欠拟合
11         silent=1,              # 取 0 时表示打印出运行时的信息
12         objective='binary:logistic', # 二分类的逻辑回归
13         gamma=0,                # 控制是否后剪枝的参数
14         max_delta_step=0,        # 限制每棵树权重改变的最大步长
15         subsample=0.8,          # 随机采样的比例，值过小会欠拟合
16         colsample_bytree=0.8,   # 控制每棵树随机采样的列数的占比
17         reg_alpha=0,            # L1 正则化参数，越大越不容易过度拟合
18         reg_lambda=0,          # L2 正则化的惩罚系数，默认为 1
19         scale_pos_weight=1,    # 类别高度不平衡参数，设置为 0 加快收敛
20         seed=1,                # 随机数的种子，缺省值为 0
21     )
22     gsearch = GridSearchCV(estimator = model,param_grid = param_test,
23                             scoring='f1', cv=5)
24     gsearch.fit(train_data,train_label)
25     print('每轮迭代运行结果:\n',gsearch.cv_results_, '\n 参数的最佳取值:\n',gsearch.best_params_, '\n 最佳模型得分:\n',
26           gsearch.best_score_)
27
28 if __name__ == '__main__':
29     n_estimators() # 最佳迭代次数

```

运行代码，得到最佳决策树的数目，如图11-9所示。

```

71434552,-0.74567509,-0.77867963,-1. ....,
.....1. ....,1. ....]),-'std_train_score':array([0.01840223,-0.
0164958,-0.01908946,-0.01344233,-0. ....,
.....0. ....,0. ....])}
参数的最佳取值:
{'n_estimators':30}
最佳模型得分:
0.6423724257213981
*** Repl Closed ***

```

图11-9 最佳决策树数目

结果分析： 从输出结果可以看出，在学习速率为0.1时，理想的决策树数目是30，最佳模型得分为0.6424。那么，在下一次调参时，将最优n_estimators代入即可。

2. 最佳min_child_weight和max_depth

本代码中GridSearchCV方法的使用类似于上面“最佳迭代次数”部分，不再赘述。

在保持其他参数不变的前提下，对max_depth和min_child_weight进行调参，将决策树数目n_estimators的最佳值设为30，这里调参范围可以选择'max_depth':[1, 2, 3, 4, 5]和'min_child_weight':[0, 1, 2]，将默认值设置为一个中位数。根据每轮最优值的偏大或偏小，随时改变参数，直到参数为最优值。具体实现的代码如下：

```

1  # 2 调试的参数是min_child_weight 和max_depth
2  def max_depthANDmin_child_weight():
3      param_test2 = {
4          'max_depth':[1, 2, 3, 4,5],
5          'min_child_weight':[0,1, 2]
6      }
7      model = xgb.XGBClassifier(
8          # max_depth=3,          # 构建树的深度，默认为 6
9          # min_child_weight=1,    # 值过高欠拟合，调整参数 cv
10         learning_rate=0.1,      # 学习率
11         n_estimators=30,        # 迭代次数，值太小易欠拟合
12         silent=1,              # 取 0 时表示打印出运行时的信息
13         objective='binary:logistic', # 二分类的逻辑回归
14         gamma=0,               # 控制是否后剪枝的参数
15         max_delta_step=0,       # 限制每棵树权重改变的最大步长
16         subsample=0.8,          # 随机采样的比例，值过小会欠拟合
17         colsample_bytree=0.8,    # 控制每棵树随机采样的列数的占比
18         reg_alpha=0,            # L1 正则化参数，越大越不容易过度拟合
19         reg_lambda=0,           # L2 正则化的惩罚系数，默认为 1
20         scale_pos_weight=1,     # 类别高度不平衡参数，设置为 0 加快收敛
21         seed=1,                 # 随机数的种子，缺省值为 0
22     )
23     gsearch2 = GridSearchCV(estimator = model,param_grid =
24     param_test2, scoring='f1', cv=5)
25     gsearch2.fit(train_data,train_label)
26     print('每轮迭代运行结果:\n',gsearch2.cv_results_, '\n 参数的最佳取
27     值:\n',gsearch2.best_params_,'\n 最佳模型得分:\n',
28     gsearch2.best_score_)

```

运行代码，得到最佳树的深度和叶子节点，如图11-10所示。

```

.....0.01509153, -0.01708128, -0.01686646, -0.0174764, -0.01344233,
.....-0.00580852, -0.00719666, -0.01706701, -0.0134679, -0.01338011,
.....-0.01536377, -0.00863192, -0.00981681, -0.01498867, -0.01069785]])
参数的最佳取值:
{'max_depth': 4, 'min_child_weight': 2}
最佳模型得分:
0.6517955539184589

Repl Closed

```

图11-10 最佳树的深度和叶子节点

结果分析： 从输出结果可以看出，其他参数不变，最佳决策树数目为30时，理想的决策树深度是4，叶子节点为2。最佳模型得分为0.6518，略微提升了一点。在下一次调整参数时将最优值代入即可。

3. 最佳gamma

本代码中GridSearchCV方法的使用类似于前面“最佳迭代次数”部分，不再赘述。

在保持其他参数不变的前提下，对gamma进行调参，将上述最佳值代入。这里调参范围可以选择'gamma':[0, 0.1, 0.2]，将默认值设置为一个中位数。根据每轮最优值的偏大或偏小，随时改变参数，直到参数为最优值。具体实现的代码如下：

```

1  # 3 调试参数: gamma
2  def gamma():
3      # 在树的叶子节点上进行进一步划分所需的最小损失。越大, 算法就越保守。
4      param_test3 = {
5          gamma':[0.2,0.3,0.4]
6      }
7      model = xgb.XGBClassifier(
8          max_depth=4,          # 构建树的深度, 默认为 6
9          min_child_weight=2,   # 值过高欠拟合, 调整参数 cv
10         learning_rate=0.1,    # 学习率
11         n_estimators=30,      # 迭代次数, 值太小易欠拟合
12         silent=1,            # 取 0 时表示打印出运行时的信息
13         objective='binary:logistic', # 二分类的逻辑回归
14         # gamma=0,            # 控制是否后剪枝的参数
15         max_delta_step=0,     # 限制每棵树权重改变的最大步长
16         subsample=0.8,        # 随机采样的比例, 值过小会欠拟合
17         colsample_bytree=0.8,  # 控制每棵树随机采样的列数的占比
18         reg_alpha=0,          # L1 正则化参数, 越大越不容易过度拟合
19         reg_lambda=0,         # L2 正则化的惩罚系数, 默认为 1
20         scale_pos_weight=1,   # 类别高度不平衡参数, 设置为 0 加快收敛
21         seed=1,               # 随机数的种子, 缺省值为 0
22     )
23     gsearch3 = GridSearchCV(estimator = model,param_grid =
24     param_test3, scoring='f1', cv=5)
25     gsearch3.fit(train_data,train_label)
26     print('每轮迭代运行结果:\n',gsearch3.cv_results_, '\n 参数的最佳取值:
27     \n',gsearch3.best_params_, '\n 最佳模型得分:\n',
28     gsearch3.best_score_)

```

运行代码, 得到最佳惩罚系数, 如图11-11所示。

```

array([[0.81428571, 0.8113879, 0.8113879, 0.8113879, 0.82517483]], '.'
split4_train_score': array([[0.84892086, 0.84892086, 0.84892086, 0.84892086,
0.85614035]], 'mean_train_score': array([[0.82097897, 0.82039941, 0.
82039941, 0.81921864, 0.82432506]], 'std_train_score': array([[0.01443744, 0.
01474929, 0.01474929, 0.01582577, 0.01773923]]).
参数的最佳取值:
{'gamma': 0.3}
最佳模型得分:
0.6499788965616453
*** Repl. Closed ***

```

图11-11 最佳惩罚系数

结果分析： 从输出结果可以看出，其他参数不变，最佳决策树数目为30，决策树深度是4，叶子节点为2时。最理想的惩罚项系数为0.3，最佳模型得分为0.65。下一次调整参数时将最优值代入即可。

4. 最佳subsample和colsample_bytree

本代码中GridSearchCV方法的使用类似于前面“最佳迭代次数”部分，不再赘述。

在保持其他参数不变的前提下，对随机采样的比例subsample和每棵树随机采样的列数的占比colsample_bytree进行调参，将上述最佳值代入。这里调参范围可以选择'subsample':[0.6, 0.7, 0.8, 0.9]和'colsample_bytree':[0.6, 0.7, 0.8, 0.9]，将默认值设置为一个中位数。根据每轮最优值的偏大或偏小，随时改变参数，直到参数为最优值。具体实现的代码如下：


```

1  # 4 调整的数据是: subsample 和 colsample_bytree
2  def subsample_colsample_bytree():
3      param_test4 = {
4          'subsample':[0.6, 0.7, 0.8, 0.9],
5          'colsample_bytree': [0.6, 0.7, 0.8, 0.9]
6      }
7      model = xgb.XGBClassifier(
8          max_depth=4,          # 构建树的深度, 默认为 6
9          min_child_weight=2,   # 值过高欠拟合, 调整参数 cv
10         learning_rate=0.1,    # 学习率
11         n_estimators=30,      # 迭代次数, 值太小易欠拟合
12         silent=1,            # 取 0 时表示打印出运行时的信息
13         objective='binary:logistic', # 二分类的逻辑回归
14         gamma=0.3,           # 控制是否后剪枝的参数
15         max_delta_step=0,     # 限制每棵树权重改变的最大步长
16         # subsample=0.8,      # 随机采样的比例, 值过小会欠拟合
17         # colsample_bytree=0.8, # 控制每棵树随机采样的列数的占比
18         reg_alpha=0,          # L1 正则化参数, 越大越不容易过度拟合
19         reg_lambda=0,         # L2 正则化的惩罚系数, 默认为 1
20         scale_pos_weight=1,   # 类别高度不平衡参数, 设置为 0 加快收敛
21         seed=1,               # 随机数的种子, 默认值为 0
22     )
23     gsearch4 = GridSearchCV(estimator = model, param_grid =
24     param_test4, scoring='f1', cv=5)
25     gsearch4.fit(train_data, train_label)
26     print('每轮迭代运行结果:\n', gsearch4.cv_results_, '\n 参数的最佳取值:\n',
27     gsearch4.best_params_, '\n 最佳模型得分:\n',
28     gsearch4.best_score_)

```

最佳采样比和采样列数比, 运行结果如图11-12所示。

```

array([0.81428571, 0.8113879, 0.8113879, 0.8113879, 0.82517483]),
split4_train_score': array([0.84892086, 0.84892086, 0.84892086, 0.84892086,
0.85614035]), 'mean_train_score': array([0.82097897, 0.82039941, 0.
82039941, 0.81921864, 0.82432506]), 'std_train_score': array([0.01443744, 0
.01474929, 0.01474929, 0.01582577, 0.01773923])}.
参数的最佳取值:
{'gamma': 0.3}
最佳模型得分:
0.6499788965616453
***Repl. Closed***

```

图11-12 最佳采样比和采样列数比

结果分析： 从输出结果可以看出，其他参数不变，最佳决策树数目为30，决策树深度是4，叶子节点为2，gamma为0.3时，最理想的采样比和采样列数比均为0.8保持不变，最佳模型得分为0.65。

5. 调参后的预测器模型

选择几个常见参数调试最优值得到：最佳决策树数目是30，最佳决策树深度是4，最佳叶子节点是2，最佳gamma是0.3，最佳采样比是0.8，最佳采样列数比是0.8。将以上各最优值代入参数中，新的预测器模型和性能实现代码如下：

```

1  # 调参后的模型
2  def new_xgboost_train(x_train, x_test, y_train, y_test):
3      model = xgb.XGBClassifier(
4          max_depth=4,          # 构建树的深度，默认为 6
5          min_child_weight=2,   # 值过高欠拟合，调整参数 cv
6          learning_rate=0.1,    # 学习率
7          n_estimators=30,      # 迭代次数，值太小易欠拟合
8          silent=1,            # 取 0 时表示打印出运行时的信息
9          objective='binary:logistic', # 二分类的逻辑回归
10         gamma=0.3,            # 控制是否后剪枝的参数
11         max_delta_step=0,      # 限制每棵树权重改变的最大步长
12         subsample=0.8,         # 随机采样的比例，值过小会欠拟合
13         colsample_bytree=0.8,  # 控制每棵树随机采样的列数的占比
14         reg_alpha=0,           # L1 正则化参数，越大越不容易过度拟合
15         reg_lambda=0,          # L2 正则化的惩罚系数，默认为 1
16         scale_pos_weight=1,    # 类别高度不平衡参数，设置为 0 加快收敛
17         seed=1,                # 随机数的种子，默认值为 0
18     )
19     model.fit(train_data, train_label, eval_metric='auc',
20             verbose=True,
21             eval_set=[(test_data, test_label)],
22             early_stopping_rounds=100)
23     y_pre = model.predict(test_data)
24     y_pro = model.predict_proba(test_data)[: , 1]
25     # AUC: Roc 曲线下的面积，介于 0.1 和 1 之间。Auc 作为数值可以直观地评价分类
26     # 器的好坏，值越大越好。
27     print( "AUC Score : %f" % metrics.roc_auc_score(test_label,
28     y_pro))
29     print("Accuracy : %.4g" % metrics.accuracy_score(test_label,
30     y_pre))
31
32 if __name__ == '__main__':
33     train_data, test_data, train_label, test_label =loadData()
34     new_xgboost_train(train_data,test_data,train_label,test_label)

```

新的预测器模型运行结果如图11-13所示。

```
[24]—validation_0 auc:0.836978
[25]—validation_0 auc:0.837626
[26]—validation_0 auc:0.835115
[27]—validation_0 auc:0.834629
[28]—validation_0 auc:0.834143
[29]—validation_0 auc:0.833414
AUC Score : -0.841270
Accuracy : -0.7879
*** Repl Closed ***
```

图11-13 调整最佳参数后的结果

结果分析： 调参前预测器模型AUC得分为0.838354，准确率得分为0.7749；调参后预测器模型AUC得分为0.841270，准确率得分为0.7879。整体AUC提升了0.002916，准确率提升了1.3%。很显然效果并不明显。为什么会出现这种现象呢？主要有以下几个原因：

- 仅仅靠参数的调整和模型的小幅优化，想要让模型的表现有大幅度的提升是不可能的。
- 要想模型的表现有一个质的飞跃，需要依靠如特征工程，模型组合，以及堆叠等手段。
- 模型性能提升与数据集划分和数据预处理的质量也有很大的关系。

6. 改变数据集划分提升模型性能

为了验证上述原因三，我们改变数据集的划分，将随机种子设为1，测试集设为0.2，代码修改如下：

```
1 train_data, test_data, train_label, test_label =loadData(1,0.2)
2 new_xgboost_train(train_data,test_data,train_label,test_label)
```

代码修改后的运行结果如图11-14所示。

```
[24]—validation_0 auc:0.884665
[25]—validation_0 auc:0.883563
[26]—validation_0 auc:0.884298
[27]—validation_0 auc:0.884481
[28]—validation_0 auc:0.880808
[29]—validation_0 auc:0.881543
AUC Score : 0.884665
Accuracy : 0.8117
***Repl. Closed***
```

图11-14 修改随机种子和数据集划分后的运行结果

从图11-14中可以很明显地发现，AUC直接提升了0.04，较调参前的0.002916提升非常明显，而准确率也提升了近3个百分点。那么，是不是测试集越大或越小效果越好呢？我们重新设置测试集 `testsize=0.1`，结果显示 `AUC Score : 0.898707` 和 `Accuracy : 0.7922`；而设置 `testsize=0.3` 时，结果显示为 `AUC Score : 0.879371` 和 `Accuracy : 0.8095`，可见，一个合适的划分比例也很重要。

总之，XGBoost模型调参时，参数的选择和调整方法对结果有影响。此外，不同的数据划分比例和数据规模对结果也有影响。数据本身的质量和数据处理的结果对模型也有影响，其中影响最大的还是数据本身。所以，在保证数据质量和预处理比较好的情况下，调参会起到锦上添花的作用。

11.6 本章小结

本章主要介绍了XGBoost的概念和应用场景，通过预测毒蘑菇案例介绍了XGBoost的使用。最后介绍了一些优化调参知识，并通过预测糖尿病患者案例详细介绍了调参步骤和技巧。下一章将综合前面数据预处理知识来实现新闻文本分类，目的在于既让读者学会数据预处理，又让读者懂得如何使用处理后的数据。

第12 章

XGBoost实现新闻文本分类

随着互联网技术的快速发展，对浩如烟海的文献资料和数据进行自动分类、组织和管理，是一个具有重要用途的研究课题。由于采用人工的方法处理这么庞大的数据耗时耗力，于是出现了文本分类技术。本章首先概述文本分类及其原理，然后对数据预处理全过程进行封装，最后结合XGBoost实现文本分类并应用到实际场景之中（源代码见：Chapter12/classifier.py）。

12.1 文本分类概述

互联网已经成为人们日常信息获取的重要途径之一，成为日常生活的一部分。伴随互联网发展而来的是数据量成指数增长，据IDC《数字宇宙》（Digital Universe）的研究报告表明，中国的数据量会在2020年超过8ZB，比2012年增长22倍。

面对如此浩瀚的数据，进行更加深层次的挖掘开发和信息的归类就变得十分重要。通过对数据的归类，可以让互联网用户在信息查询和检索变得更加快速而准确；可以使得个性化服务更加准确，针对不同的用户制定不同的信息服务；可以使得搜索引擎查询出的信息更加符合用户的实际需要，虽然有些搜索引擎通过关键词等方法查询排序，但是采用文本分类技术可以更加快速和准确地进行查询，提高了用户的体验。

文本分类是数据挖掘领域的一个热门研究方向，在日常生活中的应用也很广泛，例如，垃圾邮件的过滤、个性化服务定制、网页分类

等。据不完全统计，目前存储的数据80%是以文本形式存储的，所以对文本的分类归纳同样存在很高的商业价值。

目前对文本分类尚未有一个统一的定义，其中宗成庆关于文本分类的定义较为业界认可，其对文本分类描述如下：文本分类是预定义的分类体系下，根据文本特征（内容或属性），将给定文本与一个或多个类别相关联的过程。

文本分类常用的方法包括：

- 支持向量机分类方法（Support Vector Machine, SVM）。
- 朴素贝叶斯分类方法（Naïve Bayesian Classifier）。
- 神经网络分类方法（Neural Network）。
- Rocchion分类方法。
- 决策树分类方法（Decision Tree）。
- 线性最小平方分类方法（Linear Least-Squares Fit）。
- KNN分类方法（K Nearer Neighbors）。
- XGBoost分类方法等等。

目前，文本分类主要在以下领域得到应用：

- 银行业务：客户贷款风险（信用）分类。
- 网安领域：日志数据分类检测非法入侵。
- 图像处理：检测图像是否有人脸（人脸识别）。
- 垃圾邮件：分类识别垃圾邮件。
- 学术论文：计算机论文的领域。
- 其他应用领域。

12.2 文本分类的原理

12.2.1 文本分类的数学描述

文本分类以文本内容特征为基础，通过计算机的帮助自动将文本划分为不同类别，以新闻文本分类为例，可以划分为娱乐、体育、教育、军事、财经等类别。

文本分类的一般过程其数学描述如下：首先需要预定义文档的类别集合 $C = \{c_1, c_2, \dots, c_k\}$ ，用集合 $W = \{w_1, w_2, \dots, w_s\}$ 表示需要分类的文本，通过函数 $\phi: W \times C \rightarrow \{T, F\}$ 来描述文本分类的过程；若用T来表示正确的分类，F来表示错误的分类，即如果 $w_j \in W, c_j \in C, \langle w_j, c_j \rangle \rightarrow T$ ，那么 w_j 属于类 c_j ；如果 $w_j \in W, c_j \in C, \langle w_j, c_j \rangle \rightarrow F$ ，那么， w_j 不属于类 c_j 。这个映射函数就是文本分类器。

12.2.2 文本分类的形式化描述

文本分类从数学角度看，其实就是一个映射的过程，利用模型对未标注的样本进行类别映射。文本分类总体上可分为三个部分，即文本表示部分、模型训练部分和文本分类部分。

文本表示部分采用一种能够完好的反应样本内容并且具有一定区分能力的表示方法，目前用得较多的方法是向量空间模型（Vector Space Model, VSM），用于把样本处理成适合机器学习的特征向量。模型训练部分利用文本表示部分产生的特征向量，通过机器学习方法训练分类模型。文本分类部分则对未分类的样本先进行特征化处理，然后通过分类模型进行分类。文本的分类过程模型如图12-1所示。

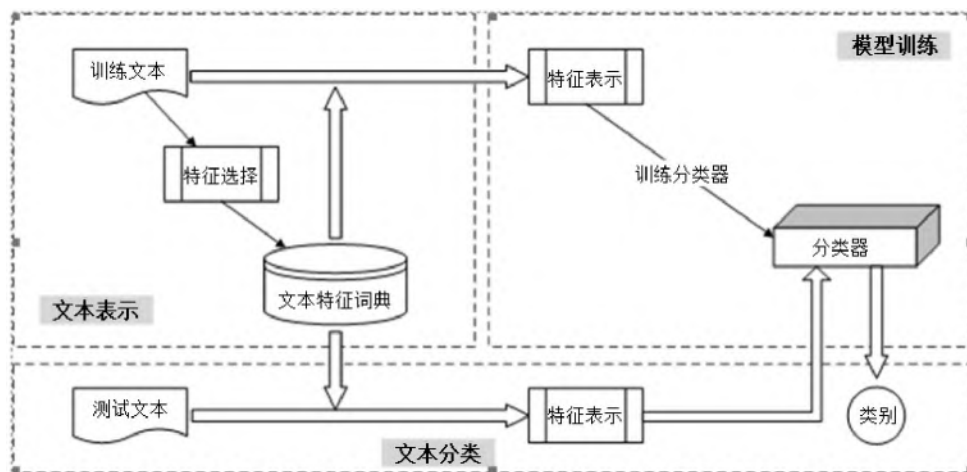


图12-1 文本分类的过程模型

12.3 分类模型评估

1. 模型评估指标

对分类模型的性能评估，找到分类器存在的不足，针对不足进行算法改进，这也是文本分类工作的一部分。目前文本分类常见的评估方法有准确率（Precision，简称为P）、召回率（Recall，简称为R）和 F 测度值。

准确率P又称为正确率或者查准率，主要是分类器正确预测某类别的文档数和被预测到某类别所有文档的比例。召回率R又称为查全率是分类器正确预测到某类别的文档数除以某类别原有文档数。测度值F是综合了准确率和召回率，通过调整它们在衡量评估中的比例大小从而提供更加综合的度量。

这里根据文本分类器输出的几种结果计算准确率和召回率，其中：

- A 表示分类器正确的预测某类别的文档个数。
- B 表示分类器错误的预测到某类别外的文档个数。

- C 表示分类器错误预测到某类别的文档个数。
- D 表示分类器正确的预测到某类别以外的文档个数。

文本分类器评估指标如表12-1所示。

表12-1 文本分类器的评估指标

	分类器分为 C 类的文档数	分类器没有分为 C 类的文档数
属于 C 类文档数	A	B
不属于 C 类文档数	C	D

准确率（P），召回率（R）计算方法如下：

$$P = \frac{A}{A + C} \quad (1)$$

$$R = \frac{A}{A + B} \quad (2)$$

F 测度值是准确率和召回率的调和均值，赋予准确率和召回率相等的权重。设 λ 为调节因子，当取0时， F 测度值即为准确率，当 λ 取无穷大时， F 测度值近似为召回率。通常认为准确率P和召回率R同样重要，一般将 λ 的值置为1，即 F_1 测度值。 F 测度值的表示如下：

$$F_\lambda = \frac{(\lambda^2 + 1)PR}{(\lambda^2 P) + R} \quad (3)$$

2. 算法模型的选择

到目前为止，我们知道了文本分类的常见算法和算法模型评估方法，那么具体到项目应用场景中该如何去选择呢？到底哪一种方法更加有利呢？带着这样的疑问，我们随机采用部分新闻数据集，并选择支持向量机SVM、决策树DT、逻辑回归LR、随机森林RF、基因表达式编程GEP、梯度迭代决策树GBDT、XGBoost等几种分类算法测试分类结果，如图12-2所示。

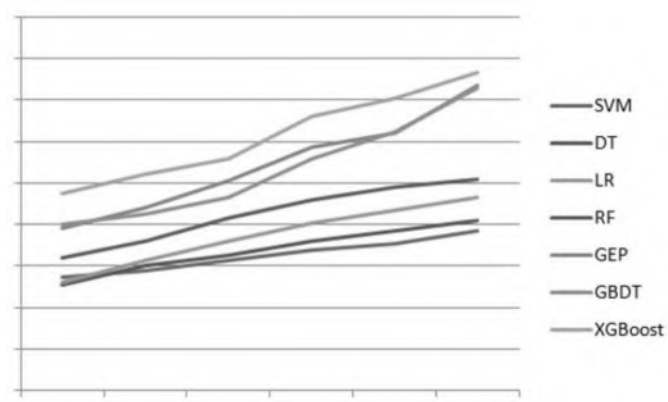


图12-2 分类模型的性能对比

在图12-2中，横坐标为数据语料的数量，纵坐标为算法性能的 F 度测度值，通过实验结果可以发现，随着数据量的增加，XGBoost算法表现得更好，所以我们选择XGBoost分类进行文本分类。实际上，有时候算法性能对实验结果的提升，不如数据特征处理更加明显，典型的的就是词典的引入。所以数据预处理在整个工程流程中所占比重很大。

12.4 数据预处理

本节主要介绍数据预处理的全过程，12.4.1节主要介绍如何高效批量读取文本，并对文本信息进行分词等处理；12.4.2节主要是词典生成阶段，对分词数据进行处理成词典模型并进行本地化存储；12.4.3节主要是对词典向量化，加载词典模型和进行向量化处理，最终将词典向量化模型进行本地化存储；12.4.4节主要是生成主题模

型，也可以理解成为特征降维阶段，加载向量化主题模型并进行特征降维，将得到的高质量数据集进行本地化存储。12.5节对这些数据加载完成分类。在进入数据预处理核心阶段之前，首先了解一下这个项目所用的数据语料。

原始新闻语料达到千万条级别，为了适合讲解，笔者选择了6类新闻的平衡语料30余万条存放在Corpus文件夹下的CSCMNews中，具体包括财经新闻37098篇，教育新闻41936篇，科技新闻65534篇，时政新闻63086篇，体育新闻65534篇和娱乐新闻65534篇。将这些语料生成的中间模型结果保存在源代码中的Corpus文件夹下的CSCMNews_model中，这两个目录如下（本节数据与生成模型均保存在源代码下的Corpus文件夹中）：

```
1  n = 5 # n 表示抽样率
2  if __name__ == '__main__':
3      path_doc_root = '../Corpus/CSCMNews' # 根目录，存放分类好的数据集
4      path_tmp = '../Corpus/CSCMNews_model' # 存放中间结果的位置
```

其中，n是一个全局变量表示抽样率，如果n设为1，代表每间隔1条信息进行一次抽样，默认设置为5。

12.4.1 通用的类库

在数据预处理的各个阶段都需要对文件夹遍历和文本信息的分词处理，将这两份方法提炼出来，封装在通用的类库中，这就是本节的主要内容。

1. 高效遍历文件

前面已经阐述了文件高效遍历的方法，并将其封装在通用的类库之中，返回新闻的类别和内容。具体的实现代码如下：

```

1 class loadFolders(object): # 迭代器
2     def __init__(self, par_path):
3         self.par_path = par_path
4     def __iter__(self):
5         for file in os.listdir(self.par_path):
6             file_abspath = os.path.join(self.par_path, file)
7             if os.path.isdir(file_abspath): # if file is a folder
8                 yield file_abspath
9
10 class loadFiles(object):
11     def __init__(self, par_path):
12         self.par_path = par_path
13     def __iter__(self):
14         folders = loadFolders(self.par_path)
15         for folder in folders: # level directory
16             catg = folder.split(os.sep)[-1]
17             for file in os.listdir(folder): # secondary directory
18                 file_path = os.path.join(folder, file)
19                 if os.path.isfile(file_path):
20                     this_file = open(file_path, 'rb') # rb 读取方式更快
21                     content = this_file.read().decode('utf8')
22                     yield catg, content
23                     this_file.close()

```

2. 分词并去除停用词

高效遍历文件之后，对每个文本需要进行分词处理。在分词处理前需要用正则表达式的方法对原始文本数据中不符合要求的字符进行清洗，这样可以加快后续分词的效率。

```

1 # 以正则表达式来处理字符
2 def textParse(str_doc):
3     str_doc = re.sub('\u3000', '', str_doc)
4     return str_doc

```

然后利用分词工具进行分词处理，这个过程中对分词结果还需要进一步的处理，比如去掉停用词、单字、数字、空字符等，以列表的形式返回新闻文本数据的特征。

```
1  # 分词并去除停用词
2  def seg_doc(str_doc):
3      sent_list = str_doc.split('\n')
4      sent_list = map(textParse, sent_list) # 去掉一些字符，例如\u3000
5      stwlist = get_stop_words()
6      word_2dlist = [rm_tokens(jieba.cut(part), stwlist) for part in
7                      sent_list] # 分词
8      word_list = sum(word_2dlist, [])
9      return word_list
10
11 # 获取停用词列表
12 def get_stop_words(path=r'../Files/NLPIR_stopwords.txt'):
13     file = open(path, 'r', encoding='utf-8').read().split('\n')
14     return set(file)
15
16 # 去掉一些停用词和数字
17 def rm_tokens(words, stwlist):
18     words_list = list(words)
19     stop_words = stwlist
20     for i in range(words_list.__len__())[:-1]:
21         if words_list[i] in stop_words: # 去除停用词
22             words_list.pop(i)
23         elif words_list[i].isdigit(): # 去除数字
24             words_list.pop(i)
25         elif len(words_list[i]) == 1: # 去除单个字符
26             words_list.pop(i)
27         elif words_list[i] == " ": # 去除空字符
28             words_list.pop(i)
29     return words_list
```

以上两个loadFiles通用方法直接遍历文件，seg_doc对单文本数据进行清洗，两者配合可以得到相对干净的特征词。但是，这个词汇

特征一方面比较庞大且特征维度较高，还存在特征重复率高的情况；另一个方面，有效的可用特征占比不高，后面还是需要进行特征降维处理及词典操作。

12.4.2 阶段1：生成词典

数据预处理的第一个阶段就是加载原始数据，并对其进行分词、去停用词、词频选择等系列操作，最终生成词典模型本地化存储。这么做的目的就是在阶段2中直接加载词典模型，节约训练时间，提高处理效率。本节采用Python模块中的Gensim对语料进行处理。将生成的词典以CSCMNews.dict命名，作为中间结果存放在Corpus/CSCMNews_model文件夹下。实现代码如下（源代码见：Chapter12/classifier.py中的GeneModel方法）：

```
1 path_doc_root = '../Corpus/CSCMNews' # 根目录即分类好的文本数据集
2 path_tmp = '../Corpus/CSCMNews_model' # 存放中间结果的位置
3 #阶段1：生成词典并去掉低率词，如果词典不存在则重新生成，反之跳过该阶段
4 path_dictionary = os.path.join(path_tmp, 'CSCMNews.dict') # 词典路径
5 if os.path.exists(path_dictionary):
6     print('=== 检测到词典已生成，跳过数据预处理阶段1 ===')
7 else:
8     os.makedirs(path_tmp) # 创建中间结果保存路径
9     GeneDict(path_doc_root, path_dictionary)
```

生成的模型封装在GeneModel方法中，首先判断词典模型是否已经存在，如果词典模型已经存在，则跳入生成向量化模型阶段；如果不存在生成词典模型，词典模型生成的代码如下：

```

1  # *****预处理阶段 1: 生成词典*****
2  '''
3  path_doc_root: 存放训练数据的根路径
4  path_dictionary: 存放中间输出结果的根路径（为第二阶段使用）
5  '''
6  def GeneDict(path_doc_root,path_dictionary):
7      # 阶段 1: 生成词典并去掉低率词，如果词典不存在则重新生成，反之跳过该阶段
8      print('=== 未检测到有词典存在，开始遍历生成词典 ===')
9      dictionary = corpora.Dictionary()
10     files = loadFiles(path_doc_root)
11     for i, msg in enumerate(files):
12         if i % n == 0:
13             catg = msg[0]
14             content = seg_doc(msg[1]) # 对文本内容进行分词处理
15             dictionary.add_documents([content])
16             if int(i/n) % 1000 == 0:
17                 print('{t} *** {i} \t docs has been dealed'
18                       .format(i=i,
19                               t=time.strftime('%Y-%m-%d %H:%M:%S',time.localtime()))))
19         # 去掉词典中出现次数过少的词
20         small_freq_ids = [tokenid for tokenid, docfreq in
21                            dictionary.dfs.items() if docfreq < 5 ]
22         dictionary.filter_tokens(small_freq_ids)
23         dictionary.compactify() # 重新产生连续的编号
24         dictionary.save(path_dictionary)
25         print('=== 词典已经生成 ===')

```

上述第6行代码中调用gensim.corpora.Dictionary()方法生成一个词典对象，用于存放生成的词典。第10行代码返回文本类别catg和文本的内容content。这里将两个列表保存在files中。第11行enumerate方法的作用是打印文本的编号。第11~18行的作用是遍历文本信息，将分词清洗的结果存放在dictionary词典中，每处理5000条新闻文本就在屏幕上打印一次完成的信息。第20行是对词典库中低词频信息的处理。第23行的作用是将处理好的词典模型保存到指定的位置，下次使用的时候直接加载指定路径的词典模型即可，而不用重新

对模型进行训练，因而可以大大节约时间成本并提高可用性。词典生成过程如图 12-3 所示（生成的词典模型见：Corpus/CSCMNews_model/CSCMNews.dict）。

```
Start...2019-04-18-15:22:45

*****.分类器训练过程.*****
---未检测到有词典存在，开始遍历生成词典---
Building prefix dict from the default dictionary...
Loading model from cache C:\Users\ADMINI~1\AppData\Local\Temp\jieba.cache
Loading model cost 0.914 seconds.
Prefix dict has been built successfully.
2019-04-18-15:22:46-***0---docs has been dealed
2019-04-18-15:22:55-***5000---docs has been dealed
2019-04-18-15:23:04-***10000---docs has been dealed
2019-04-18-15:23:12-***15000---docs has been dealed
```

图12-3 词典生成的过程

12.4.3 阶段2：词典向量化TF-IDF

对原始数据经过词典生成之后，形成有序的词特征。之后进入数据预处理的第二个阶段即词典向量化处理，也就是说将中文特征词转变为概率矩阵，便于计算机识别计算。将生成的TF-IDF模型保存在源代码的CSCMNews_model/tfidf_corpus文件夹下。代码的实现如下：

```
1 # 数据预处理阶段 2：开始将文档转化成 tfidf
2 path_tmp_tfidf = os.path.join(path_tmp, 'tfidf_corpus') # tfidf 存
  储路径
3 if os.path.exists(path_tmp_tfidf):
4     print('=== 检测到 tfidf 向量已经生成，跳过数据预处理阶段 2 ===')
5 else:
6     GeneTFIDF(path_doc_root, path_dictionary, path_tmp_tfidf)
```

词典向量化封装在GeneModel方法中，参数path_doc_root为待处理文本的根目录，参数path_dictionary为词典模型保存的路径，参数path_tmp_tfidf为生成各类TF-IDF模型的保存路径。在主函数执行到生成TF-IDF阶段，首先判断词典向量化模型是否存在，若存在直接跳过，反之则进入词典向量化训练模型。实现代码如下：

```

1  # *****预处理阶段 2：词典向量化*****
2  '''
3  path_doc_root: 存放训练数据的根路径
4  path_dictionary: 存放中间输出结果的根路径
5  path_tmp_tfidf: 存档 TF-IDF 特征向量的目录（为第三阶段使用）
6  '''
7  def GeneTFIDF(path_doc_root,path_dictionary,path_tmp_tfidf):
8      print('=== 未检测到有 tfidf 文件夹存在，开始生成 tfidf 向量 ===')
9      dictionary = corpora.Dictionary.load(path_dictionary)
10     os.makedirs(path_tmp_tfidf)
11     files = loadFiles(path_doc_root)
12     tfidf_model = models.TfidfModel(dictionary=dictionary)
13     corpus_tfidf = {}
14     for i, msg in enumerate(files):
15         if i % n == 0:
16             catg = msg[0]
17             word_list = seg_doc(msg[1])
18             file_bow = dictionary.doc2bow(word_list)
19             file_tfidf = tfidf_model[file_bow]
20             tmp = corpus_tfidf.get(catg, [])
21             tmp.append(file_tfidf)
22             if tmp.__len__() == 1:
23                 corpus_tfidf[catg] = tmp
24         if i % 5000 == 0:
25             print('{t} *** {i} \t docs has been dealed'.format(i=i,
26 t=time.strftime('%Y-%m-%d %H:%M:%S',time.localtime()))))
27             # 将 tfidf 中间结果保存起来
28             catgs = list(corpus_tfidf.keys())
29             for catg in catgs:
30                 corpora.MmCorpus.serialize('{f}{s}{c}.mm'.format
31 (f=path_tmp_tfidf, s=os.sep, c=catg),corpus_tfidf.get(catg),
32 id2word=dictionary )
33             print('catg {c} has been transformed into tfidf
34 vector'.format(c=catg))
35         print('=== tfidf 向量已经生成 ===')

```

上述代码第9行加载阶段1中的词典模型的目的是将其进行向量化处理，结合第12行gensim.models.TfidfModel方法构建向量化模型。第14~25行是对文本进行向量化处理，采用词袋模型和TF-IDF模型相结合的策略，每处理5000条就在屏幕上打印一次完成信息。第27行实现文本类别列表。第28~29行是将词典向量按照类别有序地进行本地化存储。目的在于为下一个阶段主题模型（数据降维）提供中间结果，节约训练时间，提高模型效率。生成TF-IDF词典向量模型过程如图12-4所示（词典向量模型见：Corpus/CSCMNews_model/tfidf_corpus）。

```
Start....2019-04-18-15:42:51

*****-分类器训练过程-*****

===-检测到词典已生成，跳过数据预处理阶段1-===
===-未检测到有tfidf文件夹存在，开始生成tfidf向量-===
Building prefix dict from the default dictionary...
Loading model from cache C:\Users\ADMINI~1\AppData\Local\Temp\jieba.cache
Loading model cost 0.911 seconds.
Prefix dict has been built successfully.
0 files is dealt
5000 files is dealt
10000 files is dealt
15000 files is dealt
20000 files is dealt
```

图12-4 各类生成TFIDF的过程

12.4.4 阶段3：生成主题模型

数据预处理最后一个阶段是对词特征进行选择，也就是通常说的特征选择与降维。首先判断是否有主题模型，实现代码如下：

```

1  # 数据预处理阶段 3: TFIDF 向量模型生成 LSI 主题模型
2  num_topics = 300 # 特征维度
3  test_size = 0.33 # 训练集和测试集比率
4  param = str(n)+'_'+str(num_topics)+'_'+str(test_size)
5  print("\n 特征维度:", num_topics, "\n 测试集划分:", test_size)
6  # 存放中间结果的位置
7  path_tmp = os.path.join(path_tmp, param)
8  path_tmp_lsi = os.path.join(path_tmp, 'lsi_corpus_'+param)
9  # lsi 模型的保存路径
10 path_tmp_lsamodel = os.path.join(path_tmp,
    lsi_model_'+param+'.pkl')
11 if os.path.exists(path_tmp_lsi):
12     print('=== 检测到 LSI 主题模型已经生成, 跳过数据预处理阶段 3 ===')
13 else:
14     os.makedirs(path_tmp_lsi)
15     GeneLSI(path_dictionary, path_tmp_tfidf, path_tmp_lsi,
        path_tmp_lsamodel, num_topics)

```

生成主题方法封装在GeneModel方法中，上述代码第2行是特征选择的维度，这里选择300个特征维度（维度的选择，请参考第9章内容）。第3行中test_size表示1/3的数据量作为测试集，2/3的数据量作为训练集。第4~7行设置主题模型的中间结果的保存路径。第10行是生成主题模型的保存路径。第15行是主题模型具体的实现方法，其中参数path_dictionary是词典模型的保存路径，参数path_tmp_tfidf是生成各类TFIDF模型的保存路径，参数path_tmp_lsi为各类主题模型的保存路径，参数path_tmp_lsamodel为主题模型的保存路径，参数num_topics为自定义的特征维度。生成主题模型的代码实现如下：

```

1  # *****预处理阶段 3: TFIDF 向量模型生成 LSI 主题模型
   *****
2  '''
3  num_topics: 设置保存权重最大的前 N 个数据特征, 默认为 300
4  path_tmp_tfidf: 保存 TF-IDF 特征向量的目录
5  path_tmp_lsamodel: lsi 模型的保存路径 (为使用)
6  '''
7  def GeneLSI(path_dictionary, path_tmp_tfidf,
   path_tmp_lsamodel, path_tmp_lsi, num_topics=300):
8      print('=== 未检测到有 lsi 文件夹存在, 开始生成 lsi 向量 ===')
9      dictionary = corpora.Dictionary.load(path_dictionary)
10     # 从对应文件夹中读取所有类别
11     catg_list = []
12     for file in os.listdir(path_tmp_tfidf):
13         t = file.split('.')[0]
14         if t not in catg_list:
15             catg_list.append(t)
16
17     # 从磁盘中读取语料 (corpus)
18     corpus_tfidf = {}
19     for catg in catg_list:
20         path = '{f}{s}{c}.mm'.format(f=path_tmp_tfidf, s=os.sep,
   c=catg)
21         corpus = corpora.MmCorpus(path)
22         corpus_tfidf[catg] = corpus
23     print('tfidf 文档读取完毕, 开始转化成 lsi 向量 ...')
24
25     # 生成 lsi 模型
26     corpus_tfidf_total = []
27     for catg in list(corpus_tfidf.keys()):
28         tmp = corpus_tfidf.get(catg)
29         corpus_tfidf_total += tmp
30     lsi_model = models.LsiModel(corpus=corpus_tfidf_total,
   id2word=dictionary, num_topics=num_topics)
31
32     # 将 lsi 模型保存到磁盘上
33     lsi_file = open(path_tmp_lsamodel, 'wb')
34     pickle.dump(lsi_model, lsi_file)
35     lsi_file.close()
36     del corpus_tfidf_total # lsi model 已经生成, 释放变量空间
37     print('--- lsi 模型已经生成 ---')
38
39     # 生成 corpus of lsi, 并逐步去掉 corpus of tfidf
40     corpus_lsi = {}
41     for catg in list(corpus_tfidf.keys()):
42         corpu = [lsi_model[doc] for doc in corpus_tfidf.get(catg)]
43         corpus_lsi[catg] = corpu
44         corpus_tfidf.pop(catg)
45         corpora.MmCorpus.serialize('{f}{s}{c}.mm'.format
   (f=path_tmp_lsi, s=os.sep, c=catg), corpu, id2word=dictionary)
46     print('=== lsi 向量已经生成 ===')

```

上述代码第11~15行用来获取所有类别名称。第18~23行加载词典向量模型。第26~30行是生成主题模型，这里按照指定的维度处理。第33~36行保存数据降维后的主题模型。第45行将主题模型本地化存储。为分类器提供语料。其过程如图12-5所示。（词典向量模型见：Corpus/CSCMNews_model/5_300_0.7/lsi_model_5_300_0.7.pkl）。

```
*****分类器训练过程*****
===检测到词典已生成，跳过数据预处理阶段1===
===检测到tfidf向量已经生成，跳过数据预处理阶段2===

特征维度：300
测试集划分：0.33
===未检测到有lsi文件夹存在，开始生成lsi向量===
tfidf文档读取完毕，开始转化成lsi向量...
---lsi模型已经生成---
===lsi向量已经生成===
```

图12-5 主题特征降维的过程

注 意

- ① 扩大语料时，只需把数据预处理各个阶段的训练过程执行一遍，替换之前的生成模型即可，无须修改代码。
- ② 文件夹5_300_0.7命名：5代表取样数即每隔5条，300代表保留特征数，0.7代表70%的训练集。

12.5 XGBoost分类器

在12.4节中我们完成了整个数据预处理工作，并将处理好的数据存储在../lsi_model_5_300_0.7.pkl中。接着进行分类器构建工作，同样的原理，我们对将生成好的分类器进行本地存储，如果已经生成好分类器则跳过该阶段，反之生成分类器。实现代码如下：

```

1  # 生成分类器阶段: xgboost 训练分类模型
2  path_tmp_predictor = os.path.join(path_tmp,
    predictor_+'param+'.pkl')
3  if os.path.exists(path_tmp_predictor):
4      print('=== 检测到分类器已经生成, 跳过生成分类器阶段 ===')
5  else:
6      GeneClassifier(path_tmp_lsi,path_tmp_predictor,test_size)

```

分类器方法封装在GeneModel方法中, 参数path_tmp_lsi是生成的主题模型路径, path_tmp_predictor是训练好的分类器保存路径, test_size是测试集与训练集划分比例, 默认为70%训练集和30%测试集。分类器训练过程如下:

```

1  #####生成分类器阶段#####
2  '''
3  path_tmp_lsi 各类主题 mm 保存的父目录
4  path_tmp_predictor: 分类器模型的保存路径
5  train_test_ratio: 训练集与测试集划分比例, 默认为 70%训练集和 30%测试集
6  '''
7  def GeneClassifier(path_tmp_lsi,path_tmp_predictor,
    train_test_ratio=0.7):
8      print('=== 未检测到分类器存在, 开始进行分类过程 ===')
9      print('--- 未检测到 lsi 文档, 开始从磁盘中读取 ---')
10     catg_list = []
11     for file in os.listdir(path_tmp_lsi):
12         t = file.split('.')[0]
13         if t not in catg_list:
14             catg_list.append(t)
15
16     # 从磁盘中读取语料 (corpus)
17     corpus_lsi = {}

```

```

18     for catg in catg_list:
19         path = '{f}{s}{c}.mm'.format(f=path_tmp_lsi, s=os.sep,
    c=catg)
20         corpus = corpora.MmCorpus(path)
21         corpus_lsi[catg] = corpus
22     print('--- lsi 文档读取完毕, 开始进行分类 ---')
23     # 类别标签、文档数、语料主题
24     tag_list, doc_num_list, corpus_lsi_total = [], [], []
25     for count, catg in enumerate(catg_list):
26         tmp = corpus_lsi[catg]
27         tag_list += [count]*tmp.__len__()
28         doc_num_list.append(tmp.__len__())
29         corpus_lsi_total += tmp
30         corpus_lsi.pop(catg)
31     print("文档类别数目:", len(doc_num_list))
32     # 将 gensim 中的 mm 表示转化成 numpy 矩阵表示
33     print("LSI 语料总大小:", len(corpus_lsi_total))
34
35     data, rows, cols = [], [], []
36     line_count = 0
37     for line in corpus_lsi_total:
38         for elem in line:
39             rows.append(line_count)
40             cols.append(elem[0])
41             data.append(elem[1])
42         line_count += 1
43     lsi_matrix = csr_matrix((data, (rows, cols))).toarray()
44     print("LSI 矩阵规模:", lsi_matrix.shape)
45     print("数据样本数目:", line_count)
46     # 生成训练集和测试集
47     rarray = np.random.random(size=line_count)
48     train_set, train_tag, test_set, test_tag = [], [], [], []
49     for i in range(line_count):
50         if rarray[i] < train_test_ratio:
51             train_set.append(lsi_matrix[i, :])
52             train_tag.append(tag_list[i])
53         else:
54             test_set.append(lsi_matrix[i, :])
55             test_tag.append(tag_list[i])

```



```
56     # 生成分类器
57     predictor = xgboost_multi_classify(train_set,
    test_set, train_tag, test_tag)
58     x = open(path_tmp_predictor, 'wb')
59     pkl.dump(predictor, x)
60     x.close()
```

在上述代码中，第10~14行获取所有新闻类别名称。第17~21行读取特征选择后的数据集。第24~30行按照类别有序读取数据语料并以“键-值对”（Key-Value Pair）的形式进行缓存处理。第35~42行是将数据语料进行矩阵化处理。第47~55行是对数据集的拆分，按照比例划为训练集和测试集。第57行是调用XGBoost算法进行模型构建（下文详细介绍该方法）。第58~60行是对训练好的分类器进行本地化存储。

XGBoost 分类器的实现代码如下（分类器模型见：Corpus/CSCMNews_model/5_300_0.7/predictor_5_300_0.7.pkl）：

```

1  #*****xgboost 训练分类模型*****
2  def xgboost_multi_classify(train_set, test_set, train_tag,
    test_tag):
3      # 统计信息
4      print("训练集大小:", len(train_tag), " 测试集大小:", len(test_tag))
5      train_info = {k: train_tag.count(k) for k in train_tag}
6      print("训练集类别对应的样本数:", train_info)
7      test_info = {k: test_tag.count(k) for k in test_tag}
8      print("测试集类别对应的样本数", test_info)
9
10     # XGBoost
11     data_train = xgb.DMatrix(train_set, label=train_tag)
12     data_test = xgb.DMatrix(test_set, label=test_tag)
13     watch_list = [(data_test, 'eval'), (data_train, 'train')]
14     param = {
15         'objective': 'multi:softmax', # 多分类的问题
16         'num_class': 6, # 类别数, 与 multisoftmax 并用
17         'max_depth': 8, # 构建树的深度, 越大越容易过度拟合
18         'eta': 0.3, # 如同学习率
19         'eval_metric': 'merror',
20         'silent': 1, # 设置为 1 则没有运行信息的输出, 最好是设置为 0
21         'subsample': 0.9, # 随机采样训练样本
22     } # 参数
23     xgb_model = xgb.train(param, data_train, num_boost_round=250,
        evals=watch_list) # num_boost_round 用于控制迭代的次数
24     y_hat = xgb_model.predict(data_test)
25     validateModel(test_tag, y_hat)
26     return xgb_model

```

在上述代码中，第2行中参数train_set是训练集样本，参数test_set是测试集样本，train_tag是训练集标签，test_tag是测试集标签。第4~8行分析打印数据语料的统计信息。第11~12行分别加载训练数据与测试数据。第13行作为验证参数，目的是每次遍历查看训练误差。第14~22行是分类器参数的设置。第23行是分类器训练过程，其中param是模型参数，data_train是训练数据集，num_boost_round是

迭代的次数，evals是模型验证。第24行对测试数据进行分类结果预测。第25行用于查询分类器的性能。第26行返回分类器模型。

代码的运行结果如图12-6所示。

```
===== 分类器训练过程 =====
=== 检测到词典已生成, 跳过数据预处理阶段1 ===
=== 检测到tfidf向量已经生成, 跳过数据预处理阶段2 ===
=== 检测到LSI主题模型已经生成, 跳过数据预处理阶段3 ===

特征维度: 300
训练集和测试集划分比率: 0.7
=== 未检测到分类器存在, 开始进行分类过程 ===
=== 未检测到lsi文档, 开始从磁盘中读取 ===
=== lsi文档读取完毕, 开始进行分类 ===
文档类别数目: 6
LSI语料总大小: 67745
LSI矩阵规模: (67745, 300)
数据样本数目: 67745
训练集大小: 47169 · 测试集大小: 20576
训练集类别对应的样本数: {0: 9066, 1: 9147, 2: 5881, 3: 8730, 4: 9173, 5: 5172}
测试集类别对应的样本数: {0: 4041, 1: 3960, 2: 2506, 3: 3887, 4: 3934, 5: 2248}
[0]-eval merror:0.061285——train merror:0.045729
[1]-eval merror:0.055161——train merror:0.038479
[2]-eval merror:0.052537——train merror:0.035108
```

图12-6 分类器模型的训练过程

至此，完成了分类器模型的训练过程，分类器到底是好是坏，准确率如何？需要对模型进行评估，我们来看下分类器训练过程中第25行代码中分类器性能的评估方法，代码如下：

```

1  #*****xgboost 分类结果验证*****
2  '''
3  y_true 文本对应的正确类别
4  y_pred 分类器预测的类别
5  '''
6  def validateModel(y_true, y_pred):
7      classify_report = metrics.classification_report(y_true, y_pred)
8      confusion_matrix = metrics.confusion_matrix(y_true, y_pred)
9      overall_accuracy = metrics.accuracy_score(y_true, y_pred)
10     acc_for_each_class = metrics.precision_score(y_true, y_pred,
11         average=None)
12     average_accuracy = np.mean(acc_for_each_class)
13     score = metrics.accuracy_score(y_true, y_pred)
14     print('classify_report : \n', classify_report)
15     print('confusion_matrix : \n', confusion_matrix)
16     print('acc_for_each_class : \n', acc_for_each_class)
17     print('average_accuracy: {0:f}'.format(average_accuracy))
18     print('overall_accuracy: {0:f}'.format(overall_accuracy))
19     print('score: {0:f}'.format(score))

```

在上述代码中，第7行调用sklearn.metrics.classification_report模块生成模型分类6行5列的报告，每行代表一类新闻，每列分别代表新闻类别、准确率、召回率、F得分和支持度，以及每类的平均得分。第8行调用sklearn.metrics.confusion_matrix模块打印出模型的混淆矩阵。第9行打印出整体得分。第10行打印出每一类的准确率，第11行打印出平均分类的准确率，第12行打印出总体分类的准确率。

代码的运行结果如图12-7所示。

结果分析： 上述XGBoost训练出来的分类器性能究竟如何？我们进行了模型性能评估。结果显示在分类报告中，体育新闻准确率为99%，召回率为98%， F_1 得分98%，训练样本数4041；娱乐新闻准确率为96%，召回率为98%， F_1 得分97%，训练样本数3960；教育新闻准确

率为96%，召回率为96%， F_1 得分96%，训练样本数2506；时政新闻准确率为95%，召回率为95%， F_1 得分95%，训练样本数3887；科技新闻准确率为97%，召回率为98%， F_1 得分97%，训练样本数3934；财经新闻准确率为97%，召回率为97%， F_1 得分97%，训练样本数2248；以上类别平均准确率为97%，召回率为97%， F_1 得分97%，效果非常不错。

混淆矩阵结果显示，体育新闻训练样本4041个，准确预测3956个。娱乐新闻训练样本3960个，准确预测3865个，教育新闻训练样本2506个，准确预测2409个，时政新闻训练样本3887个，准确预测3693个；科技新闻训练样本3934个，准确预测3795个；财经新闻训练样本2248个，准确预测2174个。其中对角线表示准确预测的个数，非对角线是错误归类的数目。

```
[249]—eval merror:0.033243—train merror:0
classify_report :-
.....precision.....recall.....f1 score.....support
.....0.....0.99.....0.98.....0.98.....4041
.....1.....0.96.....0.98.....0.97.....3960
.....2.....0.96.....0.96.....0.96.....2506
.....3.....0.95.....0.95.....0.95.....3887
.....4.....0.97.....0.96.....0.97.....3934
.....5.....0.97.....0.97.....0.97.....2248

.. micro avg .....0.97.....0.97.....0.97.....20576
.. macro avg .....0.97.....0.97.....0.97.....20576
weighted avg .....0.97.....0.97.....0.97.....20576

confusion_matrix :-
[[3956...57...6...14...6...2]
[...23...3865...20...32...14...6]
[...2...19...2409...51...13...12]
[...12...37...46...3693...71...28]
[...9...21...17...66...3795...26]
[...2...17...3...36...16...2174]]
acc_for_each_class :-
[0.98801199 0.9624004 0.96321471 0.94886948 0.96934866 0.96708185]
average_accuracy: 0.966488
overall_accuracy: 0.966757
score: 0.966757
*****.分类器训练结束.*****
```

图12-7 分类器的性能评估

体育、娱乐、教育、时政、科技、财经准确率分别是98.8%、96.2%、96.3%、94.9%、96.9%和96.7%。其中体育类准确率最高达到98.8%，而时政类准确率最低达到94.9%。总体平均分类准确率达到96.6%，总体分类准确率达到96.7%，总体得分达到96.7%。所以分类器

的整体性能结果很好。当然，如果要求更高的准确率，可以通过提升数据样本量、提高特征维度、提高数据预处理质量和参数调优等方法来改进。

12.6 新闻文本分类应用

我们将 12.5 节构建好的分类器模型本地化存储为 `predictor_5_300_0.7.pkl`，在项目应用的时候直接加载 `pkl` 文件即可。那么，该分类器在实际应用环境中效果如何呢？以下选用训练数据范围以外的数据来进行一下测试，我们从新闻网站中选取一篇最新的新闻文本作为分类测试语料，代码实现如下：

```

1 #####主函数#####
2 if __name__ == '__main__':
3     print(""*15,'分类开始',"*"*15)
4
5     demo_doc = ""今天在上午 FIFA 国际足联公布了一条信息：我们在此宣布：2023
        年#FIFA 女足世界杯# 将从 24 队扩军到 32 队。
6     官宣！2023 年女足世界杯正式扩军，从 24 队扩充到 32 队
7     在今年夏天结束的法国女足世界杯之后，国际足联主席因凡蒂诺表示，今年在法国举行的
        女足世界杯取得了令人惊讶的成功，证明需要保持这种良好势头，切实采取措施促进女足
        运动发展。同时因凡蒂诺还表示：女足世界杯扩军的意义远不仅仅是增加了 8 支参赛球
        队，还意味着从现在起，有更多成员协会将启动女足运动发展项目，因为他们有机会获得
        世界杯名额。
8     官宣！2023 年女足世界杯正式扩军，从 24 队扩充到 32 队
9     需要说的是国际足联并未公布各个大洲的名额，扩军后的女足世界杯共分为 8 个小组，每
        组 4 支球队。国际足联相关机构将与各大洲足球联合会商讨名额分配方案，该方案需通过
        国际足联理事会通过。
10    官宣！2023 年女足世界杯正式扩军，从 24 队扩充到 32 队
11    中国女足在法国世界杯中取得了 16 强的成绩，这样扩军的消息对于球迷们来说并非全都
        是利好，这意味着会有更多实力不是很强的队伍进入到世界杯赛场，如果遇到德国和美国
        等强队的时候，或许会经常上演美国队 13 球血洗泰国女足的情况。    ""
12    print("未知文本类别内容为：\n\n",demo_doc,'\n')
13    # 模型参数
14    path_dictionary = r'templates/files/CSCModel/CSCMNews.dict'
15    path_tmp_lsimodel =
        r'templates/files/CSCModel/530070/lsimodel.pkl' # 主题模型路径
16    path_tmp_predictor =
        r'templates/files/CSCModel/530070/predictormodel.pkl'
17    path_tmp_lsi = r'templates/files/CSCModel/530070/tfidf'
18
    TestClassifier(demo_doc,path_dictionary,path_tmp_lsimodel,path_tmp_
        predictor,path_tmp_lsi)

```

上述代码第5~11行是随机在“凤凰新闻”网站摘取的一篇新闻语料作为测试数据。第14~17行是训练好的模型参数，第18行是分类器的分类过程。将以上参数传入分类方法TestClassifier，编写以下代码：

```

1 #####新闻文本进行分类#####
2 '''
3 demo_doc 待分类的新闻文本
4 path_dictionary 训练好的词典模型的保存路径
5 path_tmp_lsimodel 训练好的主题模型的保存路径
6 path_tmp_predictor 训练的的分类器模型存储路径
7 path_tmp_lsi 各类主题 mm 存放的路径
8 '''
9 def TestClassifier(demo_doc, path_dictionary , path_tmp_lsimodel,
10 path_tmp_predictor, path_tmp_lsi):
11     # 1 文本数据预处理, 分词并转化为 TF-IDF 向量
12     dictionary = corpora.Dictionary.load(path_dictionary) # 加载词典
13     demo_bow = dictionary.doc2bow(list(jieba.cut(demo_doc,
14 cut_all=False)))
15     tfidf_model = models.TfidfModel(dictionary=dictionary)
16     demo_tfidf = tfidf_model[demo_bow]
17
18     # 2 文本转化主题模型
19     lsi_file = open(path_tmp_lsimodel, 'rb')
20     lsi_model = pickle.load(lsi_file)
21     lsi_file.close()
22     demo_lsi = lsi_model[demo_tfidf]
23
24     # 3 主题 mm 类型数据表示成 numpy 矩阵
25     data,cols,rows = [],[],[]
26     for item in demo_lsi:
27         data.append(item[1])
28         cols.append(item[0])
29         rows.append(0)
30     demo_matrix = csr_matrix((data, (rows, cols))).toarray()
31
32     # 4 加载分类器
33     x = open(path_tmp_predictor, 'rb')
34     predictor = pickle.load(x)
35     x.close()
36     DMatrix = xgb.DMatrix(demo_matrix)
37     y_hat = predictor.predict(DMatrix)
38
39     # 5 获取所有类别
40     catg_list = []
41     for file in os.listdir(path_tmp_lsi):
42         t = file.split('.')[0]
43         if t not in catg_list:
44             catg_list.append(t)
45
46     print('\n 分类结果为: ',catg_list[int(y_hat[0])])

```


在上述代码中，第11~14行是对测试新闻语料的数据预处理过程，转化为向量化特征。第17~20行是对测试数据特征选择处理。第23~28行是将测试数据进行矩阵转化。第31~35行读取本地的预测器并预测新闻类别。第38~44行是返回新闻预测的结果，如图12-8所示。

结果分析： 随机采用一篇新闻，其分类器预测结果是体育新闻，人工判断也是体育新闻，预测准确。整个预测花费0.9s（即0.9秒），时间主要花费在加载模型，实际预测可以达到毫秒级。

```
*****.分类开始*****
未知文本类别内容为：

在今天上午FIFA国际足联公布了一条信息：我们在此宣布：2023年#FIFA女足世界杯#将从24队扩军到32队。官宣！2023年女足世界杯正式扩军，从24队扩充到32队。在今年夏天结束的法国女足世界杯之后，国际足联主席因凡蒂诺表示，今年在法国举行的女足世界杯取得了令人惊讶的成功，证明需要保持这种良好势头，切实采取措施促进女足运动发展。同时因凡蒂诺还表示：女足世界杯扩军的意义远不仅仅是增加了8支参赛球队，还意味着从现在起，有更多成员协会将启动女足运动发展项目，因为他们有机会获得世界杯名额。官宣！2023年女足世界杯正式扩军，从24队扩充到32队。需要说的是国际足联并未公布各个大洲的名额，扩军后的女足世界杯共分为8个小组，每组4支球队。国际足联相关机构将与各大洲足球联合会商讨名额分配方案，该方案需通过国际足联理事会通过。中国女足在法国世界杯中取得了16强的成绩，这样扩军的消息对于球迷们来说并非全都是利好，这意味着会有更多实力不是很强的队伍进入到世界杯赛场，如果遇到德国和美国等强队的时候，或许会经常上演美国队13球血洗泰国女足的情况。
.....

Building prefix dict from the default dictionary...
Loading model from cache C:\Users\ADMINI~1\AppData\Local\Temp\jieba.cache
Loading model cost 0.908 seconds.
Prefix dict has been built successfully.
['体育', '娱乐', '教育', '时政', '科技', '财经']

分类结果为：体育
*****.分类结束*****
```

图12-8 分类器对应用于实际的网络文本

至此，完成了数据预处理的整个过程及其分类器的构建与应用。该模型一经构建完成就可以重复利用。在非算法层面改进的情况下，只需扩充语料重新生成模型即可。如果项目需要在Web端部署，也很方便，并不需要编写大量的代码，只需拷贝项目Corpus文件夹的代码，然后按照本节方法加载参数即可。

12.7 本章小结

本章主要介绍了文本分类的定义和原理，其中包括形式化和数学化描述及其文本分类模型的评估问题。通过几种常见分类模型的对比，得出了XGBoost算法更优的结论。最后结合新闻文本数据预处理，打造了一款通用的新闻文本分类器。

参考文献

- [1] Kotsiantis S B, Kanellopoulos D, Pintelas P E. Data preprocessing for supervised learning[J]. International Journal of Computer Science, 2006, 1(2): 111-117.
- [2] Famili A, Shen W M, Weber R, et al. Data preprocessing and intelligent data analysis[J]. Intelligent data analysis, 1997, 1(1): 3-23.
- [3] García S, Luengo J, Herrera F. Data preprocessing in data mining[M]. New York:Springer, 2015.
- [4] Lommen A. MetAlign: interface-driven, versatile metabolomics tool for hyphenated full-scan mass spectrometry data preprocessing[J]. Analytical chemistry, 2009, 81(8): 3079-3086.
- [5] Hand D J. Data Mining[J]. Encyclopedia of Environmetrics, 2006, 2.
- [6] Tanasa D, Trousse B. Advanced data preprocessing for intersites web usage mining[J]. IEEE Intelligent Systems, 2004, 19(2): 59-65.
- [7] 菅志刚, 金旭. 数据挖掘中数据预处理的研究与实现 [J] . 计算机应用研究, 2004, 7: 117-118.
- [8] 彭高辉, 王志良. 数据挖掘中的数据预处理方法 [J] . 华北水利水电学院学报, 2008, 29(6): 61-63.
- [9] Davis J J, Clark A J. Data preprocessing for anomaly based network intrusion detection: A review[J]. computers & security, 2011, 30(6-7): 353-375.
- [10] Kamiran F, Calders T. Data preprocessing techniques for classification without discrimination[J]. Knowledge and Information Systems, 2012, 33(1): 1-33.

- [11] Munk M, Kapusta J, Švec P. Data preprocessing evaluation for web log mining:reconstruction of activities of a web visitor[J]. Procedia Computer Science, 2010, 1(1):2273-2280.
- [12] Van Der Walt S, Colbert S C, Varoquaux G. The NumPy array: a structure for efficient numerical computation[J]. Computing in Science & Engineering, 2011, 13(2): 22.
- [13] NumPy官网: <http://www.numpy.org/>
- [14] NumPy 官方教程 : <https://docs.scipy.org/doc/numpy/user/quickstart.html>
- [15] scipy官网: <https://www.scipy.org/>
- [16] scipy 官方教程 : <https://docs.scipy.org/doc/scipy/reference/tutorial/index.html>
- [17] pandas官网: <https://pandas.pydata.org/>
- [18] pandas 官方教程 : http://pandas.pydata.org/pandas-docs/stable/getting_started/tutorials.html
- [19] 百易教程: <https://www.yiibai.com/>
- [20] Moore D, Shannon C. Code-Red: a case study on the spread and victims of an Internet worm[C]//Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment. ACM, 2002: 273-284.
- [21] Spafford E H. The Internet worm program: An analysis[J]. ACM SIGCOMM Computer Communication Review, 1989, 19(1): 17-57.
- [22] 刘金红, 陆余良. 主题网络爬虫研究综述 [J] . 计算机应用研究, 2007, 24(10): 26-29.
- [23] 孙立伟, 何国辉, 吴礼发. 网络爬虫技术的研究 [J] . 电脑知识与技术, 2010, 6(15): 4112-4115.
- [24] 徐远超, 刘江华, 刘丽珍, 等. 基于Web的网络爬虫的设计与实现 [J] . 微计算机信息, 2007, 23(7): 119-121.
- [25] Li P, Salour M, Su X. A survey of internet worm detection and containment[J]. IEEE Communications Surveys & Tutorials, 2008, 10(1): 20-35.

- [26] Spafford E H. The internet worm incident[C]//European Software Engineering Conference. Springer, Berlin, Heidelberg, 1989: 446-468.
- [27] Denning P J. The science of computing: The Internet worm[J]. American Scientist, 1989, 77(2): 126-128.
- [28] 孙承杰, 关毅. 基于统计的网页正文信息抽取方法的研究 [J] . 中文信息学报, 2004, 18(5): 18-23.
- [29] Finkel J R, Grenager T, Manning C. Incorporating non-local information into information extraction systems by gibbs sampling[C]//Proceedings of the 43rd annual meeting on association for computational linguistics. Association for Computational Linguistics, 2005: 363-370.
- [30] Banko M, Cafarella M J, Soderland S, et al. Open information extraction from the web[C]//IJCAI. 2007, 7: 2670-2676.
- [31] Cowie J, Wilks Y. Information extraction[J]. Handbook of Natural Language Processing, 2000, 56: 57.
- [32] Grishman R. Information extraction: Techniques and challenges[C]//International summer school on information extraction. Springer, Berlin, Heidelberg, 1997: 10-27.
- [33] 刘迁, 焦慧, 贾惠波. 信息抽取技术的发展现状及构建方法的研究 [J] . 计算机应用研究, 2007, 24(7): 6-9.
- [34] 胡军伟, 秦奕青, 张伟. 正则表达式在Web信息抽取中的应用 [J] . 北京信息科技大学学报 (自然科学版), 2011, 6 (86) : 89.
- [35] 梁晗, 陈群秀, 吴平博. 基于事件框架的信息抽取系统 [J] . 中文信息学报, 2006, 20(2): 42-48.
- [36] Chang C H, Kayed M, Girgis M R, et al. A survey of web information extraction systems[J]. IEEE transactions on knowledge and data engineering, 2006, 18(10):1411-1428.
- [37] Rahm E, Do H H. Data cleaning: Problems and current approaches[J]. IEEE Data Eng. Bull., 2000, 23(4): 3-13.

- [38] 王曰芬, 章成志, 张蓓蓓, 等. 数据清洗研究综述 [J]. 数据分析与知识发现, 2007, 2(12): 50-56.
- [39] Dasu T, Johnson T. Exploratory data mining and data cleaning[M]. John Wiley & Sons, 2003.
- [40] Guyon I, Matic N, Vapnik V. Discovering Informative Patterns and Data Cleaning[J]. 1996.
- [41] Bohannon P, Fan W, Geerts F, et al. Conditional functional dependencies for data cleaning[C]//2007 IEEE 23rd international conference on data engineering. IEEE, 2007:746-755.
- [42] 孙铁利, 刘延吉. 中文分词技术的研究现状与困难 [J]. 信息技术, 2009, 7: 187-189.
- [43] 赵海, 揭春雨. 基于有效子串标注的中文分词 [J]. 中文信息学报, 2007, 21(5): 8-13.
- [44] Chao S, Lihui C. Feature dimension reduction for microarray data analysis using locally linear embedding[C]//Proceedings Of The 3rd Asia-Pacific Bioinformatics Conference. 2005: 211-217.
- [45] 胡洁. 高维数据特征降维研究综述 [J]. 计算机应用研究, 2008, 25(9): 2601-2606.
- [46] 张玉芳, 万斌候, 熊忠阳. 文本分类中的特征降维方法研究 [J]. 计算机应用研究, 2012, 29(7): 2541-2543.
- [47] Li H J, Yang S H. Using range profiles as feature vectors to identify aerospace objects[J]. IEEE Transactions on Antennas and Propagation, 1993, 41(3): 261-268.
- [48] Gamon M. Sentiment classification on customer feedback data: noisy data, large feature vectors, and the role of linguistic analysis[C]//Proceedings of the 20th international conference on Computational Linguistics. Association for Computational Linguistics, 2004:841.
- [49] Bellet A, Habrard A, Sebban M. A survey on metric learning for feature vectors and structured data[J]. arXiv preprint

arXiv:1306.6709, 2013.

- [50] Shapiro L S, Brady J M. Feature-based correspondence: an eigenvector approach[J]. Image and vision computing, 1992, 10(5): 283-288.
- [51] Bohm C, Pryakhin A, Schubert M. The gauss-tree: Efficient object identification in databases of probabilistic feature vectors[C]//22nd International Conference on Data Engineering (ICDE'06). IEEE, 2006: 9-9.
- [52] Kotropoulos C L, Tefas A, Pitas I. Frontal face authentication using discriminating grids with morphological feature vectors[J]. IEEE Transactions on Multimedia, 2000, 2(1):14-26.
- [53] Chen T, Guestrin C. Xgboost: A scalable tree boosting system[C]//Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. ACM, 2016: 785-794.
- [54] Chen T, He T, Benesty M, et al. Xgboost: extreme gradient boosting[J]. R package version 0.4-2, 2015: 1-4.
- [55] Torlay L, Perrone-Bertolotti M, Thomas E, et al. Machine learning - XGBoost analysis of language networks to classify patients with epilepsy[J]. Brain informatics, 2017, 4(3): 159.
- [56] Torlay L, Perrone-Bertolotti M, Thomas E, et al. Machine learning - XGBoost analysis of language networks to classify patients with epilepsy[J]. Brain informatics, 2017, 4(3): 159.
- [57] Zhang L, Zhan C. Machine learning in rock facies classification: an application of XGBoost[C]//International Geophysical Conference, Qingdao, China, 17-20 April 2017. Society of Exploration Geophysicists and Chinese Petroleum Society, 2017: 1371-1374.
- [58] Chen Z, Jiang F, Cheng Y, et al. Xgboost classifier for ddos attack detection and analysis in SDN-based cloud[C]//2018

IEEE International Conference on Big Data and Smart Computing (BigComp). IEEE, 2018: 251-256.

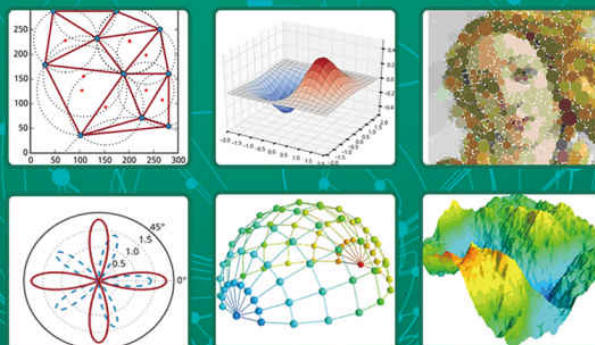
- [59] 代六玲, 黄河燕, 陈肇雄. 中文文本分类中特征抽取方法的比较研究[J]. 中文信息学报, 2004, 18(1): 27-33.
- [60] McCallum A, Nigam K. A comparison of event models for naive bayes text classification[C]//AAAI-98 workshop on learning for text categorization. 1998, 752(1):41-48.
- [61] Lodhi H, Saunders C, Shawe-Taylor J, et al. Text classification using string kernels[J]. Journal of Machine Learning Research, 2002, 2(Feb): 419-444.
- [62] Forman G. An extensive empirical study of feature selection metrics for text classification[J]. Journal of machine learning research, 2003, 3(Mar): 1289-1305.
- [63] Zhang X, Zhao J, LeCun Y. Character-level convolutional networks for text classification[C]//Advances in neural information processing systems. 2015: 649-657.
- [64] McCallum A. Multi-label text classification with a mixture model trained by EM[C]//AAAI workshop on Text Learning. 1999: 1-7.

从 $e^{i\pi} + 1$ 开始



Python

科学计算 (第2版)



张若愚 著

- ★ NumPy—快速处理数据
- ★ SciPy—数值计算库
- ★ matplotlib—绘制精美的图表
- ★ Pandas—方便的数据分析库
- ★ SymPy—符号运算好帮手
- ★ Traits & TraitsUI—轻松制作图形界面
- ★ TVTK与Mayavi—数据的三维可视化
- ★ OpenCV—图像处理和计算机视觉
- ★ Cython—编译Python程序
- ★ 使用泊松混合合成图像
- ★ 经典力学模拟
- ★ 频域信号处理
- ★ 布尔可满足性问题求解器
- ★ 分形

清华大学出版社

Python科学计算 (第2版)

张若愚 著

清华大学出版社
北京

内容简介

本书详细介绍Python科学计算中最常用的扩展库NumPy、SciPy、matplotlib、Pandas、SymPy、TTK、Mayavi、OpenCV、Cython，涉及数值计算、界面制作、三维可视化、图像处理、提高运算效率等多方面的内容。所附光盘中包含所有章节的Notebook以及便携式运行环境WinPython，以方便读者运行书中所有实例。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

图书在版编目（CIP）数据

Python科学计算/张若愚 著. —2版. —北京：清华大学出版社，
2016

ISBN 978-7-302-42658-5

I . ①P… II . ①张… III . ①软件工具—程序设计 IV .
①TP311.56

中国版本图书馆CIP数据核字（2016）第014000号

责任编辑：李维杰

装帧设计：牛静敏

责任校对：成凤进

责任印制：

出版发行：清华大学出版社

网 址： <http://www.tup.com.cn>, <http://www.wqbook.com>

地 址：北京清华大学学研大厦A座

邮 编： 100084

社 总 机： 010-62770175

邮 购： 010-62786544

投稿与读者服务： 010-62776969, c-service@tup.tsinghua.edu.cn

质量反馈： 010-62772015, zhiliang@tup.tsinghua.edu.cn

印 装 者：

经 销： 全国新华书店

开 本： 185mm×260mm

彩 插： 6

印 张： 45.75

字 数： 1083 千字

版 次： 2012年1月第1版 2016年4月第2版

印 次： 2016年4月第1次印刷

印 数： 1~3500

产品编号：

Preface

Python is rightfully viewed as a general purpose language, well suited for web development, system administration, and general purpose business applications. It's has earned this reputation well by powering web sites such as YouTube, installation tools integral to Red Hat's operating system, and large corporate IT systems from cloud cluster management to investment banking. Python has also established itself firmly in the world of scientific computing covering a wide range of applications from seismic processing for oil exploration to quantum physics. This breadth of applicability is significant because these seemingly disparate uses often overlap in important ways. Applications that can easily connect to databases publish information to the web, and efficiently carry out complex calculations are now critical in many industries. Python's primary strength is that it allows developers to build such tools quickly.

Python's scientific computing roots actually go quite deep. Guido van Rossum created the language while at CWI, the Center for Mathematics and Computer Science, in the Netherlands. As interest developed outside the center, others began to contribute. The first several Python workshops, starting in 1994, were held an ocean away at scientific institutions such as NIST (National Institute of Instruments

and Technology), the US Geological Society, and LLNL (Lawrence Livermore National Laboratories), all science centric institutions. At the time, Python 1.0 had recently been released and the attendees were just beginning to hammer out the design of its mathematical tools. A decade and a half later, it is gratifying to see how far we have come both in the amazing capabilities of the tool set and the diversity of the community. It is somehow fitting that the first comprehensive book (that I know of) covering the primary scientific computing tools for Python is composed and published, another ocean away, in Chinese. Looking forward a decade and a half, I can hardly wait to see what we will all build together.

Guido, himself, was not a scientist or engineer. He sat squarely in the computer science branch of CWI and created Python to ease the pain of building system administration tools for the Amoeba operating system. At the time, the tools were being written in C. Python was to be the tool that “bridged the gap between shell scripting and C.” Operating system tools are not even in the same neighborhood as matrix inversions or fast Fourier transforms, but, as the language emerged, scientists around the world were some of its earliest adopters. Guido had succeeded in creating an elegantly expressive language that coupled nicely with their existing C and Fortran code. And, in Guido, they had a language designer willing to listen and add critical

features, such as complex numbers, specifically for the scientific community. With the creation of Numeric, the precursor to NumPy, Python gained a fast and powerful number crunching tool that solidified Python's role as a leading computational language in the coming decades.

For some, the term “scientific programming” conjures up visions of intricate algorithms described from “Numerical Recipes in C” or forged in late night programming sessions by graduate students. But the reality is the domain encompasses a much wider range of programming tasks from low level algorithms to GUI development with advanced graphics. This latter topic is too often underestimated in terms of importance and effort. Fortunately, Ruoyu Zhang has done us the service of covering all facets of the scientific programming in this book. Beginning with the foundational Numpy library the algorithmic toolboxes in SciPy he provides the fundamental tools for any scientific application. He then aptly covers the 2D plotting and 3D visualization libraries provided by matplotlib, chaco, and mayavi. Application and GUI development with Traits and Traits UI, and coupling to legacy C libraries through Cython, Weave, ctypes, and SWIG are well covered as well. These core tools are rounded out by coverage of symbolic mathematics with SymPy and various other useful topics.

It's truly gratifying to see all of these topics aggregated into a single volume. It provides a one-stop shop that can lead you from the beginning steps to a polished and full featured application for analysis and simulation.

Eric Jones

2011/12/8

第1版序

Python理所当然地被视为一门通用的程序设计语言，非常适合于网站开发、系统管理以及通用的业务应用程序。它为诸如YouTube这样的网站系统、Red Hat操作系统中不可或缺的安装工具以及从云管理到投资银行等大型企业的IT系统提供技术支持，从而赢得了如此高的声誉。Python还在科学计算领域建立了牢固的基础，覆盖了从石油勘探的地震数据处理到量子物理等范围广泛的应用场景。Python这种广泛的适用性在于，这些看似不同的应用领域通常在某些重要的方面是重叠的。易于与数据库连接、在网络上发布信息并高效地进行复杂计算的应用程序对于许多行业是至关重要的，而Python最主要的长处就在于它能让开发者迅速地创建这样的工具。

实际上，Python与科学计算的关系源远流长。吉多·范罗苏姆创建这门语言，还是在他在荷兰阿姆斯特丹的国家数学和计算机科学研究学会（CWI）的时候。当时只是作为“课余”的开发，但是很快其他人也开始为之做出贡献。从1994年开始的头几次Python研讨会，都是在大洋彼岸的科研机构举行的。例如国家标准技术研究所（NIST）、美国地质学会以及劳伦斯利福摩尔国家实验室（LLNL），所有这些都是以科研为中心的机构。当时Python 1.0刚刚发布，与会者们就已经开始打造Python的数学计算工具。10多年过去了，我们欣喜地看到，我们在开发具有惊人能力的工具集以及建设多彩的社区方面做出了如此多的成绩。很合时宜的是，就我所知的第一本涵盖了Python的主要科学计算工具的综合性著作，在另一个海洋之遥的中国编著并出版了。展望今后的十几年，我迫不及待地想看到我们能共同创建出怎样的未来。

吉多他本人并不是科学家或工程师。他在CWI的计算机科学部门时，为了缓解为阿米巴（Amoeba）操作系统创建系统管理工具的痛苦，他创建了Python。当时那些系统管理工具都是用C语言编写的。于是Python就成了填补shell脚本和C语言之间空白的工具。操作系统工具与计算逆矩阵或快速傅立叶变换是完全不同的领域，但是从Python诞生开始，世界各地的许多科学家就成了它最早期的采用者。吉多成功地创建了一门能与他们的C和Fortran代码完美结合的、具有优雅表现力的程序语言。并且，吉多是一位愿意听取建议并添加关键功能的语言设计师，例如支持复数就是专门针对科学领域的。随着NumPy的前身——Numeric的诞生，Python获得了一个高效且强大的数值运算工具，它巩固了在未来几十年中，Python作为领先的科学计算语言的地位。

对于一些人来说，“科学计算编程”会让人联想起*Numerical Recipes in C*中描述的那些复杂算法，或是研究生们在深夜中努力打造程序的场景。但是真实情况所涵盖的范围更广泛——从底层的算法设计到具有高级绘图功能的用户界面开发。而后者的重要性却常常被忽视了。幸运的是在本书中，作者为我们介绍了科学计算编程所需的各个方面。从NumPy库和SciPy算法工具库的基础开始，介绍了任何科学计算应用程序所需的基本工具。然后，本书很恰当地介绍了二维绘图以及三维可视化库——matplotlib、Chaco、Mayavi。用Traits和TraitsUI进行应用程序和界面开发，以及用Cython、Weave、ctypes和SWIG等与传统的C语言库相互结合等内容在书中也有很好的介绍。除了这些核心的工具之外，本书还介绍了使用SymPy进行数学符号运算以及其他各种有用的主题。

所有这些主题都被汇编到一本书中真是一件令人欣喜的事情。本书所提供的一站式服务，能够指导读者从最初的入门直到创建一个漂亮的、全功能的分析与模拟应用程序。

Eric Jones

2011年12月8日

关于序言作者

Eric Jones是Enthought公司的CEO，他在工程和软件开发领域拥有广泛的背景，指导Enthought公司的产品工程和软件设计。在共同创建Enthought公司之前，他在杜克大学电机工程学系从事数值电磁学以及遗传优化算法方面的研究，并获得了该系的硕士和博士学位。他教授过许多用Python做科学计算的课程，并且是Python软件基金会的成员。

关于Enthought公司

Enthought是一家位于美国得克萨斯州首府奥斯汀的软件公司，主要使用Python从事科学计算工具的开发。本书中介绍的NumPy、SciPy、Traits、TraitsUI、Chaco、TVTK以及Mayavi均为该公司开发或维护的开源程序库。

前言

Python世界的发展日新月异，在本书第1版出版之后，Python在数据分析、科学计算领域又出现了许多令人兴奋的进展：

- IPython从增强的交互式解释器发展到Jupyter Notebook项目，它已经成为Python科学计算界的标准配置。
- Pandas经过几个版本的更新，目前已经成为数据清洗、处理和分析的不二选择。
- OpenCV官方的扩展库cv2已经正式发布，它的众多图像处理函数能直接对NumPy数组进行处理，编写图像处理、计算机视觉程序变得更方便、简洁。
- matplotlib 2.0即将发布，它将使用更美观的默认样式。
- Cython内置支持NumPy数组，它已经逐渐成为编写高效运算扩展库的首选工具。
- NumPy、SciPy等也经历了几个版本的更新，许多计算变得更快捷，功能也更加丰富。
- WinPython、Anaconda等新兴的Python集成环境无须安装，使得开发与共享Python程序更方便快捷。

本书第2版紧随各个扩展库的发展，将最新、最实用的内容呈现给读者。除了数值计算之外，本书还包含了界面制作、三维可视化、图像处理、提高运算效率等方面的内容。最后一章综合使用本书介绍的各个扩展库，完成几个有趣的实例项目。

本书完全采用IPython Notebook编写，保证了书中所有代码及输出的正确性。附盘中附带所有章节的Notebook以及便携式运行环境

WinPython，以方便读者运行书中所有实例。

本书适合于工科高年级本科生、研究生、工程技术人员以及计算机开发人员阅读，也适合阅读过第1版的读者了解各个扩展库的最新进展，进一步深入学习。

阅读本书的读者需要掌握Python语言的一些基础知识，Cython章节需要读者能够阅读C语言代码。

除封面署名的作者外，参加本书编写的人员还有张佑林、张东等人，在此一并表示感谢。

目 录

[Preface](#)

[第1版序](#)

[前言](#)

[第1章 Python科学计算环境的安装与简介](#)

[1.1 Python简介](#)

[1.1.1 Python 2还是Python 3](#)

[1.1.2 开发环境](#)

[1.1.3 集成开发环境\(IDE\)](#)

[1.2 IPython Notebook入门](#)

[1.2.1 基本操作](#)

[1.2.2 魔法\(Magic\)命令](#)

[1.2.3 Notebook的显示系统](#)

[1.2.4 定制IPython Notebook](#)

[1.3 扩展库介绍](#)

[1.3.1 数值计算库](#)

[1.3.2 符号计算库](#)

[1.3.3 绘图与可视化](#)

[1.3.4 数据处理和分析](#)

[1.3.5 界面设计](#)

[1.3.6 图像处理和计算机视觉](#)

[1.3.7 提高运算速度](#)

[第2章 NumPy-快速处理数据](#)

[2.1 ndarray对象](#)

[2.1.1 创建](#)

[2.1.2 元素类型](#)

[2.1.3 自动生成数组](#)

[2.1.4 存取元素](#)

[2.1.5 多维数组](#)

[2.1.6 结构数组](#)

[2.1.7 内存结构](#)

[2.2 ufunc函数](#)

[2.2.1 四则运算](#)

[2.2.2 比较运算和布尔运算](#)

[2.2.3 自定义ufunc函数](#)

[2.2.4 广播](#)

[2.2.5 ufunc的方法](#)

[2.3 多维数组的下标存取](#)

[2.3.1 下标对象](#)

[2.3.2 整数数组作为下标](#)

[2.3.3 一个复杂的例子](#)

[2.3.4 布尔数组作为下标](#)

[2.4 庞大的函数库](#)

[2.4.1 随机数](#)

[2.4.2 求和、平均值、方差](#)

[2.4.3 大小与排序](#)

[2.4.4 统计函数](#)

[2.4.5 分段函数](#)

[2.4.6 操作多维数组](#)

[2.4.7 多项式函数](#)

[2.4.8 多项式函数类](#)

[2.4.9 各种乘积运算](#)

[2.4.10 广义ufunc函数](#)

[2.5 实用技巧](#)

[2.5.1 动态数组](#)

[2.5.2 和其他对象共享内存](#)

[2.5.3 与结构数组共享内存](#)

[第3章 SciPy-数值计算库](#)

[3.1 常数和特殊函数](#)

[3.2 拟合与优化-optimize](#)

[3.2.1 非线性方程组求解](#)

[3.2.2 最小二乘拟合](#)

[3.2.3 计算函数局域最小值](#)

[3.2.4 计算全域最小值](#)

[3.3 线性代数-linalg](#)

[3.3.1 解线性方程组](#)

[3.3.2 最小二乘解](#)

[3.3.3 特征值和特征向量](#)

[3.3.4 奇异值分解-SVD](#)

[3.4 统计-stats](#)

[3.4.1 连续概率分布](#)

[3.4.2 离散概率分布](#)

[3.4.3 核密度估计](#)

[3.4.4 二项分布、泊松分布、伽玛分布](#)

[3.4.5 学生t-分布与t检验](#)

[3.4.6 卡方分布和卡方检验](#)

[3.5 数值积分-integrate](#)

[3.5.1 球的体积](#)

[3.5.2 解常微分方程组](#)

[3.5.3 ode类](#)

[3.6 信号处理-signal](#)

[3.6.1 中值滤波](#)

[3.6.2 滤波器设计](#)

[3.6.3 连续时间线性系统](#)

[3.7 插值-interpolate](#)

[3.7.1 一维插值](#)

[3.7.2 多维插值](#)

[3.8 稀疏矩阵-sparse](#)

[3.8.1 稀疏矩阵的存储形式](#)

[3.8.2 最短路径](#)

[3.9 图像处理-ndimage](#)

[3.9.1 形态学图像处理](#)

[3.9.2 图像分割](#)

[3.10 空间算法库-spatial](#)

[3.10.1 计算最近旁点](#)

[3.10.2 凸包](#)

[3.10.3 沃罗诺伊图](#)

[3.10.4 德劳内三角化](#)

[第4章 matplotlib-绘制精美的图表](#)

[4.1 快速绘图](#)

[4.1.1 使用pyplot模块绘图](#)

[4.1.2 面向对象方式绘图](#)

[4.1.3 配置属性](#)

[4.1.4 绘制多子图](#)

[4.1.5 配置文件](#)

[4.1.6 在图表中显示中文](#)

[4.2 Artist对象](#)

[4.2.1 Artist的属性](#)

[4.2.2 Figure容器](#)

[4.2.3 Axes容器](#)

[4.2.4 Axis容器](#)

[4.2.5 Artist对象的关系](#)

[4.3 坐标变换和注释](#)

[4.3.1 4种坐标系](#)

[4.3.2 坐标变换的流水线](#)

[4.3.3 制作阴影效果](#)

[4.3.4 添加注释](#)

[4.4 块、路径和集合](#)

[4.4.1 Path与Patch](#)

[4.4.2 集合](#)

[4.5 绘图函数简介](#)

[4.5.1 对数坐标图](#)

[4.5.2 极坐标图](#)

[4.5.3 柱状图](#)

[4.5.4 散列图](#)

[4.5.5 图像](#)

[4.5.6 等值线图](#)

[4.5.7 四边形网格](#)

[4.5.8 三角网格](#)

[4.5.9 箭头图](#)

[4.5.10 三维绘图](#)

[4.6 matplotlib技巧集](#)

[4.6.1 使用agg后台在图像上绘图](#)

[4.6.2 响应鼠标与键盘事件](#)

[4.6.3 动画](#)

[4.6.4 添加GUI面板](#)

[第5章 Pandas-方便的数据分析库](#)

[5.1 Pandas中的数据对象](#)

[5.1.1 Series对象](#)

[5.1.2 DataFrame对象](#)

[5.1.3 Index对象](#)

[5.1.4 MultiIndex对象](#)

[5.1.5 常用的函数参数](#)

[5.1.6 DataFrame的内部结构](#)

[5.2 下标存取](#)

[5.2.1 \[\]操作符](#)

[5.2.2 .loc\[\]和.iloc\[\]存取器](#)

[5.2.3 获取单个值](#)

[5.2.4 多级标签的存取](#)

[5.2.5 query\(\)方法](#)

[5.3 文件的输入输出](#)

[5.3.1 CSV文件](#)

[5.3.2 HDF5文件](#)

[5.3.3 读写数据库](#)

[5.3.4 使用Pickle序列化](#)

[5.4 数值运算函数](#)

[5.5 时间序列](#)

[5.5.1 时间点、时间段、时间间隔](#)

[5.5.2 时间序列](#)

[5.5.3 与NaN相关的函数](#)

[5.5.4 改变DataFrame的形状](#)

[5.6 分组运算](#)

[5.6.1 groupby\(\)方法](#)

[5.6.2 GroupBy对象](#)

[5.6.3 分组—运算—合并](#)

[5.7 数据处理和可视化实例](#)

[5.7.1 分析Pandas项目的提交历史](#)

[5.7.2 分析空气质量数据](#)

[第6章 SymPy-符号运算好帮手](#)

[6.1 从例子开始](#)

[6.1.1 封面上的经典公式](#)

[6.1.2 球体体积](#)

[6.1.3 数值微分](#)

[6.2 数学表达式](#)

[6.2.1 符号](#)

[6.2.2 数值](#)

[6.2.3 运算符和函数](#)

[6.2.4 通配符](#)

[6.3 符号运算](#)

[6.3.1 表达式变换和化简](#)

[6.3.2 方程](#)

[6.3.3 微分](#)

[6.3.4 微分方程](#)

[6.3.5 积分](#)

[6.4 输出符号表达式](#)

[6.4.1 lambdify](#)

[6.4.2 用autowrap\(\)编译表达式](#)

[6.4.3 使用cse\(\)分步输出表达式](#)

[6.5 机械运动模拟](#)

[6.5.1 推导系统的微分方程](#)

[6.5.2 将符号表达式转换为程序](#)

[6.5.3 动画演示](#)

[第7章 Traits&TraitsUI-轻松制作图形界面](#)

[7.1 Traits类型入门](#)

[7.1.1 什么是Traits属性](#)

[7.1.2 Trait属性的功能](#)

[7.1.3 Trait类型对象](#)

[7.1.4 Trait的元数据](#)

[7.2 Trait类型](#)

[7.2.1 预定义的Trait类型](#)

[7.2.2 Property属性](#)

[7.2.3 Trait属性监听](#)

[7.2.4 Event和Button属性](#)

[7.2.5 动态添加Trait属性](#)

[7.3 TraitsUI入门](#)

[7.3.1 默认界面](#)

[7.3.2 用View定义界面](#)

[7.4 用Handler控制界面和模型](#)

[7.4.1 用Handler处理事件](#)

[7.4.2 Controller和UIInfo对象](#)

[7.4.3 响应Trait属性的事件](#)

[7.5 属性编辑器](#)

[7.5.1 编辑器演示程序](#)

[7.5.2 对象编辑器](#)

[7.5.3 自定义编辑器](#)

[7.6 函数曲线绘制工具](#)

[第8章 TVTK与Mayavi-数据的三维可视化](#)

[8.1 VTK的流水线\(Pipeline\)](#)

[8.1.1 显示圆锥](#)

[8.1.2 用ivtk观察流水线](#)

[8.2 数据集](#)

[8.2.1 ImageData](#)

[8.2.2 RectilinearGrid](#)

[8.2.3 StructuredGrid](#)

[8.2.4 PolyData](#)

[8.3 TVTK的改进](#)

[8.3.1 TVTK的基本用法](#)

[8.3.2 Trait属性](#)

[8.3.3 序列化](#)

[8.3.4 集合迭代](#)

[8.3.5 数组操作](#)

[8.4 TVTK可视化实例](#)

[8.4.1 切面](#)

[8.4.2 等值面](#)

[8.4.3 流线](#)

[8.4.4 计算圆柱的相贯线](#)

[8.5 用mlab快速绘图](#)

[8.5.1 点和线](#)

[8.5.2 Mayavi的流水线](#)

[8.5.3 二维图像的可视化](#)

[8.5.4 网格面mesh](#)

[8.5.5 修改和创建流水线](#)

[8.5.6 标量场](#)

[8.5.7 矢量场](#)

[8.6 将TVTK和Mayavi嵌入界面](#)

[8.6.1 TVTK场景的嵌入](#)

[8.6.2 Mayavi场景的嵌入](#)

[第9章 OpenCV-图像处理和计算机视觉](#)

[9.1 图像的输入输出](#)

[9.1.1 读入并显示图像](#)

[9.1.2 图像类型](#)

[9.1.3 图像输出](#)

[9.1.4 字节序列与图像的相互转换](#)

[9.1.5 视频输出](#)

[9.1.6 视频输入](#)

[9.2 图像处理](#)

[9.2.1 二维卷积](#)

[9.2.2 形态学运算](#)

[9.2.3 填充-floodFill](#)

[9.2.4 去瑕疵-inpaint](#)

[9.3 图像变换](#)

[9.3.1 几何变换](#)

[9.3.2 重映射-remap](#)

[9.3.3 直方图](#)

[9.3.4 二维离散傅立叶变换](#)

[9.3.5 用双目视觉图像计算深度信息](#)

[9.4 图像识别](#)

[9.4.1 用霍夫变换检测直线和圆](#)

[9.4.2 图像分割](#)

[9.4.3 SURF特征匹配](#)

[9.5 形状与结构分析](#)

[9.5.1 轮廓检测](#)

[9.5.2 轮廓匹配](#)

[9.6 类型转换](#)

[9.6.1 分析cv2的源程序](#)

[9.6.2 Mat对象](#)

[9.3.3 在cv和cv2之间转换图像对象](#)

[第10章 Cython-编译Python程序](#)

[10.1 配置编译器](#)

[10.2 Cython入门](#)

[10.2.1 计算矢量集的距离矩阵](#)

[10.2.2 将Cython程序编译成扩展模块](#)

[10.2.3 C语言中的Python对象类型](#)

[10.2.4 使用cdef关键字声明变量类型](#)

[10.2.5 使用def定义函数](#)

[10.2.6 使用cdef定义C语言函数](#)

[10.3 高效处理数组](#)

[10.3.1 Cython的内存视图](#)

[10.3.2 用降采样提高绘图速度](#)

[10.4 使用Python标准对象和API](#)

[10.4.1 操作list对象](#)

[10.4.2 创建tuple对象](#)

[10.4.3 用array.array作为动态数组](#)

[10.5 扩展类型](#)

[10.5.1 扩展类型的基本结构](#)

[10.5.2 一维浮点数向量类型](#)

[10.5.3 包装ahocorasick库](#)

[10.6 Cython技巧集](#)

[10.6.1 创建ufunc函数](#)

[10.6.2 快速调用DLL中的函数](#)

[10.6.3 调用BLAS函数](#)

[第11章 实例](#)

[11.1 使用泊松混合合成图像](#)

[11.1.1 泊松混合算法](#)

[11.1.2 编写代码](#)

[11.1.3 演示程序](#)

[11.2 经典力学模拟](#)

[11.2.1 悬链线](#)

[11.2.2 最速降线](#)

[11.2.3 单摆模拟](#)

[11.3 推荐算法](#)

[11.3.1 读入数据](#)

[11.3.2 推荐性能评价标准](#)

[11.3.3 矩阵分解](#)

[11.3.4 使用最小二乘法实现矩阵分解](#)

[11.3.5 使用Cython迭代实现矩阵分解](#)

[11.4 频域信号处理](#)

[11.4.1 FFT知识复习](#)

[11.4.2 合成时域信号](#)

[11.4.3 观察信号的频谱](#)

[11.4.4 卷积运算](#)

[11.5 布尔可满足性问题求解器](#)

[11.5.1 用Cython包装PicoSAT](#)

[11.5.2 数独游戏](#)

[11.5.3 扫雷游戏](#)

[11.6 分形](#)

[11.6.1 Mandelbrot集合](#)

[11.6.2 迭代函数系统](#)

[11.6.3 L-System分形](#)

[11.6.4 分形山脉](#)

[附录：参看CD链接](#)

[返回总目录](#)

第1章 Python科学计算环境的安装与简介

1.1 Python简介

Python是一种解释型、面向对象、动态的高级程序设计语言。自从20世纪90年代初Python语言诞生至今，它逐渐被广泛应用于处理系统管理任务和开发Web系统。目前Python已经成为最受欢迎的程序设计语言之一。

由于Python语言的简洁、易读以及可扩展性，在国外用Python做科学计算的研究机构日益增多，一些知名大学已经采用Python教授程序设计课程。众多开源的科学计算软件包都提供了Python的调用接口，例如计算机视觉库OpenCV、三维可视化库VTK、复杂网络分析库igraph等。而Python专用的科学计算扩展库就更多了，例如三个十分经典的科学计算扩展库：NumPy、SciPy和matplotlib，它们分别为Python提供了快速数组处理、数值运算以及绘图功能。因此Python语言及其众多的扩展库所构成的开发环境十分适合工程技术、科研人员处理实验数据、制作图表，甚至开发科学计算应用程序。近年随着数据分析扩展库Pandas、机器学习扩展库scikit-learn以及IPython Notebook交互环境的日益成熟，Python也逐渐成为数据分析领域的首选工具。

说起科学计算，首先会被提到的可能是MATLAB。然而除了MATLAB的一些专业性很强的工具箱目前还无法替代之外，MATLAB的大部分常

用功能都可以在Python世界中找到相应的扩展库。和MATLAB相比，用Python做科学计算有如下优点：

- 首先，MATLAB是一款商用软件，并且价格不菲。而Python完全免费，众多开源的科学计算库都提供了Python的调用接口。用户可以在任何计算机上免费安装Python及其绝大多数扩展库。
- 其次，与MATLAB相比，Python是一门更易学、更严谨的程序设计语言。它能让用户编写出更易读、更易维护的代码。
- 最后，MATLAB主要专注于工程和科学计算。然而即使在计算领域，也经常会遇到文件管理、界面设计、网络通信等各种需求。而Python有着丰富的扩展库，可以轻易完成各种高级任务，开发者可以用Python实现完整应用程序所需的各种功能。

1.1.1 Python 2还是Python 3

自从2008年发布以来，Python3经历了5个小版本的更迭，无论是语法还是标准库都发展得十分成熟。许多重要的扩展库也已经逐渐同时支持Python2和Python3。但是由于Python3不向下兼容，目前大多数开发者仍然在生产环境中使用Python 2.7。在PyCon2014大会上，Python之父宣布Python 2.7的官方支持延长至2020年。因此本书仍然使用Python 2.7作为开发环境。

在本书涉及的扩展库中，IPython、NumPy、SciPy、matplotlib、Pandas、SymPy、Cython、Spyder和OpenCV等都已经支持Python 3，而

Traits、TraitsUI、TVTK、Mayavi等扩展库则尚未着手Python 3的移植。虽然一些新兴的三维可视化扩展库正朝着替代Mayavi的方向努力，但目前Python环境中尚未有能替代VTK和Mayavi的专业级别的三维可视化扩展库，因此本书仍保留第1版中相关的章节。

1.1.2 开发环境

和MATLAB等商用软件不同，Python的众多扩展库由许多社区分别维护和发布，因此要一一将其收集齐全并安装到计算机中是一件十分耗费时间和精力事情。本节介绍两个科学计算用的Python集成软件包。读者只需要下载并执行一个安装程序，就能安装好本书涉及的所有扩展库。

1. WinPython



<https://winpython.github.io/>

WinPython的下载地址。

WinPython只能在Windows系统中运行，其安装包不会修改系统的任何配置，各种扩展库的用户配置文件也保存在WinPython的文件夹之下。因此可将整个运行环境复制到U盘中，在任何安装了Windows操作系统的计算机上运行。WinPython提供了一个安装扩展库的WinPython

Control Panel界面程序，通过它可以安装Python的各种扩展库。可以通过下面的链接下载已经编译好的二进制扩展库安装包，然后通过WinPython Control Panel来安装。



<http://www.lfd.uci.edu/~gohlke/pythonlibs/>

从该网址可以下载各种Python扩展库的Windows安装文件。

图1-1显示了通过WinPython Control Panel安装本书介绍的几个扩展库。通过“Add packages”按钮添加扩展库的安装程序之后，单击“Install packages”按钮一次性安装勾选的所有扩展库。

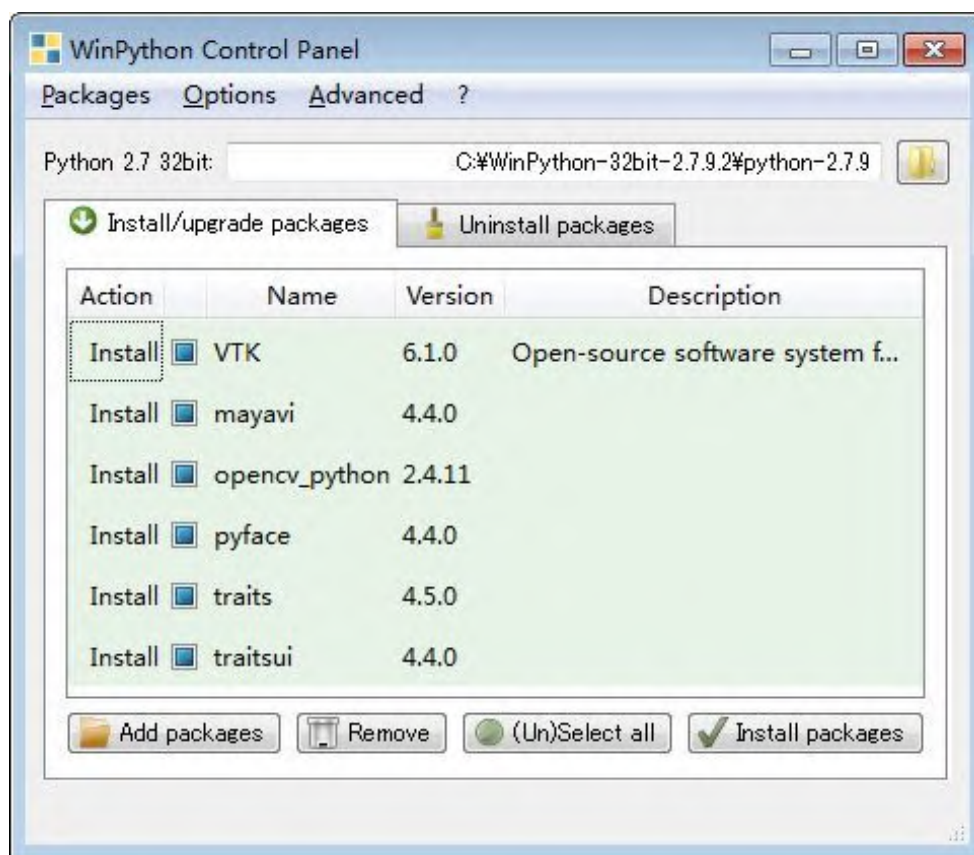


图1-1 通过WinPython Control Panel安装扩展库

虽然手动安装扩展库有些麻烦，不过这种方式适合没有网络连接或者网速较慢的计算机。例如在笔者的工作环境中，有大量的实验用计算机不允许连接互联网。

如果读者从WinPython的官方网站下载WinPython开发环境，为了运行本书的所有实例程序，还需要安装如下扩展库：

- VTK、Mayavi、pyface、Traits和TraitsUI：在图形界面以及三维可视化章节需要用到这些扩展库。
- OpenCV：在图像处理章节需要用到该扩展库。

2. Anaconda

 <https://store.continuum.io/cshop/anaconda/>

Anaconda的下载地址。

由CONTINUUM开发的Anaconda开发环境支持Windows、Linux和Mac OSX。安装时会提示是否修改PATH环境变量和注册表，如果希望手工激活Anaconda环境，请取消选择这两个选项。

在命令行中运行安装路径之下的批处理文件Scripts\anaconda.bat以启动Anaconda环境，然后就可以输入表1-1中的conda命令来管理扩展库了。

表1-1 conda命令及说明

命令	说明
conda list	列出所有的扩展库
conda update 扩展库名	升级扩展库
conda install 扩展库名	安装扩展库
conda search 模板	搜索符合模板的扩展库

conda命令本身也是一个扩展库，因此通常在执行上述命令之前，可以先运行conda update conda来尝试升级到最新版本。conda默认从官方频道下载扩展库，如果未找到指定的扩展库，还可以使用anaconda命令从Anaconda网站的其他频道搜索指定的扩展库。例如下面的命令用于搜索可使用conda安装的OpenCV扩展库：

```
binstar search -t conda opencv
```

找到包含目标扩展库的频道之后，输入下面的命令来从指定的频道rsignell安装：

```
conda install opencv-python -c rsignell
```

还可以使用pip命令安装下载的扩展库文件，例如从前面介绍的网址下载文件opencv_python-2.4.11-cp27-none-win32.whl之后，切换到该文件所在的路径并输入pip install opencv_python-2.4.11-cp27-none-win32.whl即可安装该扩展库。

3. 使用附赠光盘中的开发环境

本书的附赠光盘中包含了能运行本书所有实例程序的WinPython压缩包：winpython.zip。请读者将之解压到C盘根目录之下，该压缩包会创建C:\WinPython-32bit-2.7.9.2目录。

然后将本书附赠光盘中提供的代码目录scipybook2复制到计算机的硬盘中，为了保证代码正常运行，请确保该代码目录的完整路径中不包含空格和中文字符。在scipybook2中包含三个子目录：

- codes：其中的scpy2子目录下包含本书提供的示例程序，该示例程序库采用包的形式管理，因此需要将它添加进Python的包搜索路径环境变量PYTHONPATH中才能正确运行scpy2中的示例程序。在scipybook2目录下的批处理文件run_console.bat和run_notebook.bat中会自动设置该环境变量。
- notebooks：本书完全使用IPython Notebook编写，该目录下的Notebook文件中保存了本书所有章节的标题以及示例代码。读者可以通过run_notebook.bat批处理文件启动本书的编写环境。为了保护本书版权，除本章之外的其他所有章节的文字解说内容都已被删除。
- settings：其中保存了各种扩展库的配置文件。这些文件会保存在HOME环境变量所设置的目录之下，默认值为C:\Users\用户名。为了避免与读者系统中的配置文件发生冲突，在批处理文件中将HOME环境变量修改为该settings目录。

为了确认开发环境正确安装，请读者运行run_console.bat批处理文件，然后在命令行中执行python -m scpy2，并检查是否打印出开发环境中各个扩展库的版本信息。



如果读者将winpython.zip文件解压到别的路径之下，可以修改env.bat文件中第二行中的路径。

```
!python -m scpy2
Welcome to Scpy2
Python: 2.7.9
          executable:      C:\WinPython-32bit-2.7.9.2\python-
2.7.9\python.exe
Cython          : 0.22
matplotlib      : 1.4.3
numpy-MKL       : 1.9.1
opencv_python   : 2.4.11
pandas          : 0.16.0
scipy           : 0.15.0
sympy           : 0.7.6
```

1.1.3 集成开发环境(IDE)

本节介绍两个常用的Python集成开发环境，它们能实现自动完成、定义跳转、自动重构、调试等常用的IDE功能，并集成了IPython的交互环境以及查看数组、绘制图表等科学计算开发中常用的功能。熟练使用这些工具能极大地提高编程效率。

1. Spyder

Spyder是由WinPython的作者开发的一个简单的集成开发环境，可通过WinPython的安装目录下的Spyder.exe来运行。如果读者希望在本书的开发环境中运行Spyder，可以在run_console.bat开启的命令行中输入spyder命令。

和其他的Python开发环境相比，它最大的优点就是模仿MATLAB的“工作空间”的功能，可以很方便地观察和修改数组的值。图1-2是Spyder的界面截图。

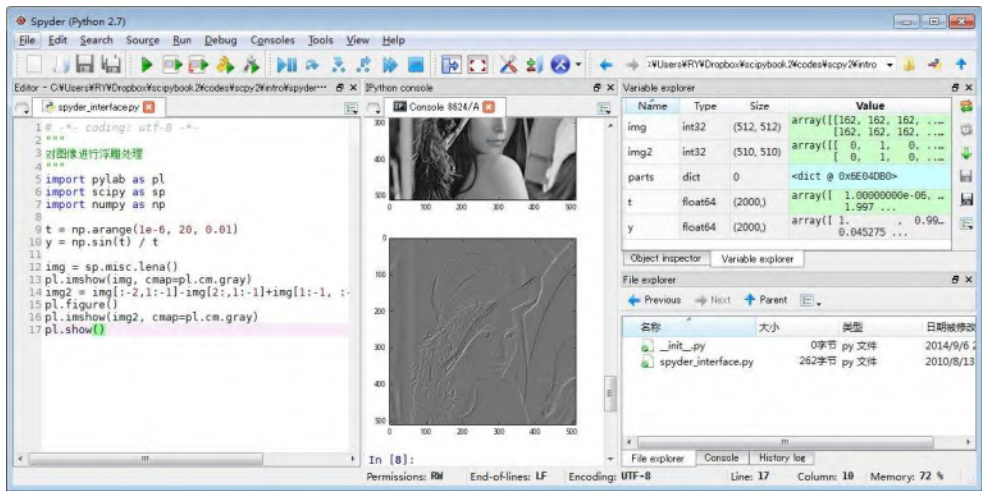


图1-2 在Spyder中执行图像处理的程序

Spyder的界面由许多泊坞窗口构成，用户可以根据自己的喜好调整它们的位置和大小。当多个窗口在一个区域中时，将使用标签页的形式显示。例如在图1-2中，可以看到“Editor”、“Variable explorer”、“File explorer”、“IPython console”等窗口。在View菜单中可以设置是否显示这些窗口。表1-2中列出了Spyder的主要窗口及其作用：

表1-2 Spyder的主要窗口及其作用

窗口名	功能
Editor	编辑程序，以标签页的形式编辑多个程序文件
Console	在别的进程中运行的Python控制台
Variable explorer	显示Python控制台中的变量列表
Object inspector	查看对象的说明文档和源程序
File explorer	文件浏览器，用来打开程序文件或者切换当前路径

按F5键将在另外的控制台进程中运行当前编辑器中的程序。第一次运行程序时，将弹出一个如图1-3所示的运行配置对话框。在此对话框中可以对程序的运行进行如下配置：

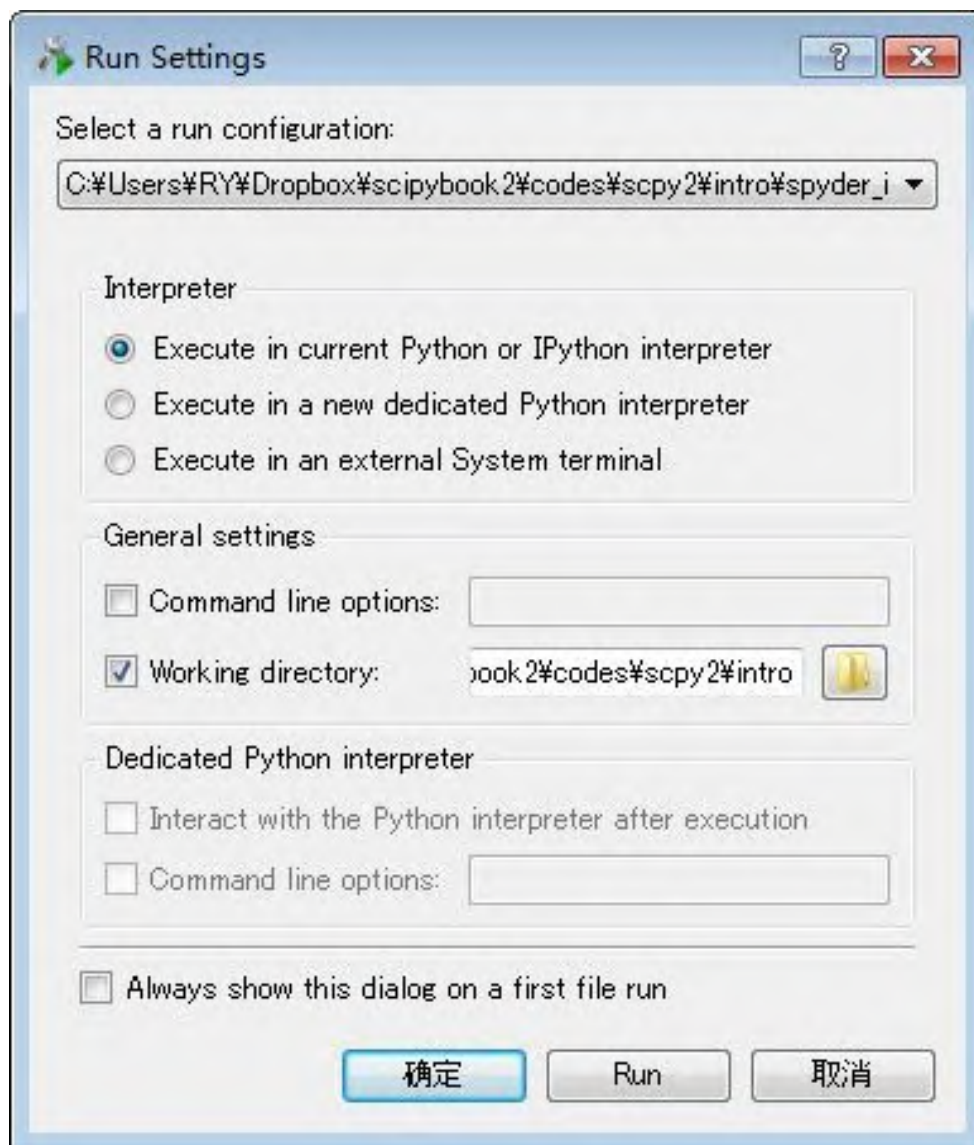


图1-3 运行配置对话框

- Command line options: 输入程序的运行参数。
- Working directory: 输入程序的运行路径。
- Execute in current Python or IPython interpreter: 在当前的Python控制台中运行程序。程序可以访问此控制台中的所有全局对象，控制台中已经载入的模块不需要重新载入，因此程序的启动速度较快。

- **Execute in a new dedicated Python interpreter:** 新开一个Python控制台并在其中运行程序，程序的启动速度较慢，但是由于新控制台中没有多余的全局对象，因此更接近真实运行的情况。当选择此项时，还可以勾选“Interact with the Python interpreter after execution”，这样当程序结束运行之后，控制台进程继续运行，可以通过它查看程序运行之后的所有全局对象。此外，还可以在“Command line options”中输入新控制台的启动参数。
- **Execute in an external System terminal:** 选择该选项则完全脱离Spyder运行程序。

运行配置对话框只会在第一次运行程序时出现，如果想修改程序的运行配置，可以按F6键来打开运行配置对话框。

控制台中的全局对象可以在“Variable explorer”窗口中找到。此窗口支持数值、字符串、元组、列表、字典以及NumPy的数组等对象的显示和编辑。图1-4（左）是“Variable explorer”窗口的截图，列出了当前运行环境中的变量名、类型、大小及其内容。右键单击变量名，会弹出对此变量进行操作的菜单。在菜单中选择Edit选项，弹出图1-4（右）所示的数组编辑窗口。此编辑窗口中的单元格的背景颜色直观地显示了数值的大小。当有多个控制台运行时，“Variable explorer”窗口显示当前控制台中的全局对象。

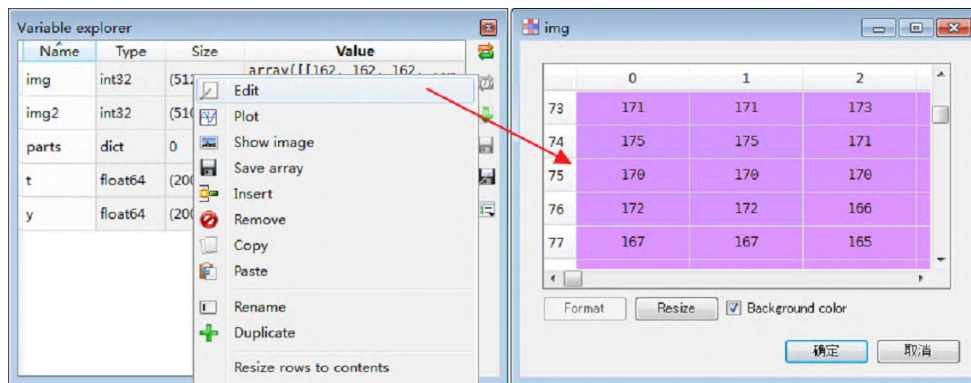


图1-4 使用“Variable explorer”查看和编辑变量内容

选择菜单中的Plot选项，将弹出如图1-5所示的绘图窗口。在绘图窗口的工具栏中单击最右边的按钮，将弹出一个编辑绘图对象的对话框。图中使用此对话框修改了曲线的颜色和线宽。

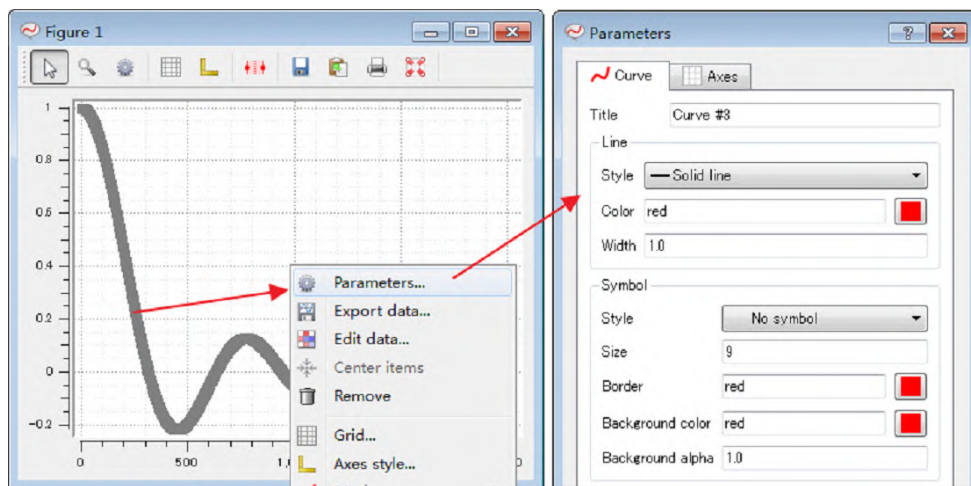


图1-5 在“Variable explorer”中将数组绘制成曲线

Spyder的功能比较多，这里仅介绍一些常用的功能和技巧：

- 默认配置下，“Variable explorer”中不显示大写字母开头的变量，可以单击工具栏中的配置按钮（最后一个按钮），在菜单中取消“Exclude capitalized references”的勾选状态。

- 在控制台中，可以按Tab键自动补全。在变量名之后输入“?”，可以在“Object inspector”窗口中查看对象的说明文档。此窗口的Options菜单中的“Show source”选项可以开启显示函数的源程序。
- 可以通过“Working directory”工具栏修改工作路径，用户程序运行时，将以此工作路径作为当前路径。只需要修改工作路径，就可以用同一个程序处理不同文件夹下的数据文件。
- 在程序编辑窗口中按住Ctrl按键，并单击变量名、函数名、类名或模块名，可以快速跳转到其定义位置。如果是在别的程序文件中定义的，将打开此文件。在学习一个新的模块库的用法时，经常需要查看模块中的某个函数或某个类是如何定义的，使用此功能可以帮助我们快速查看和分析各个库的源程序。

2. PyCharm

PyCharm是由JetBrains开发的集成开发环境，具有项目管理、代码跳转、代码格式化、自动完成、重构、自动导入、调试等功能。虽然专业版价格比较高，但是提供的免费社区版具有开发Python程序所需的所有功能。如果读者需要开发较大的应用程序，使用它可以提高开发效率，保证代码的质量。



<http://www.jetbrains.com/pycharm>

PyCharm的官方网站。

如果读者使用本书提供的便携WinPython版本，那么需要在PyCharm中设置Python解释器。通过菜单“File”→“Settings”打开配置对话框，在左栏中找到“Project Interpreter”，然后通过右侧的齿轮按钮，并选择弹出菜单中的“Add Local”选项，即可打开如图1-6所示的对话框。

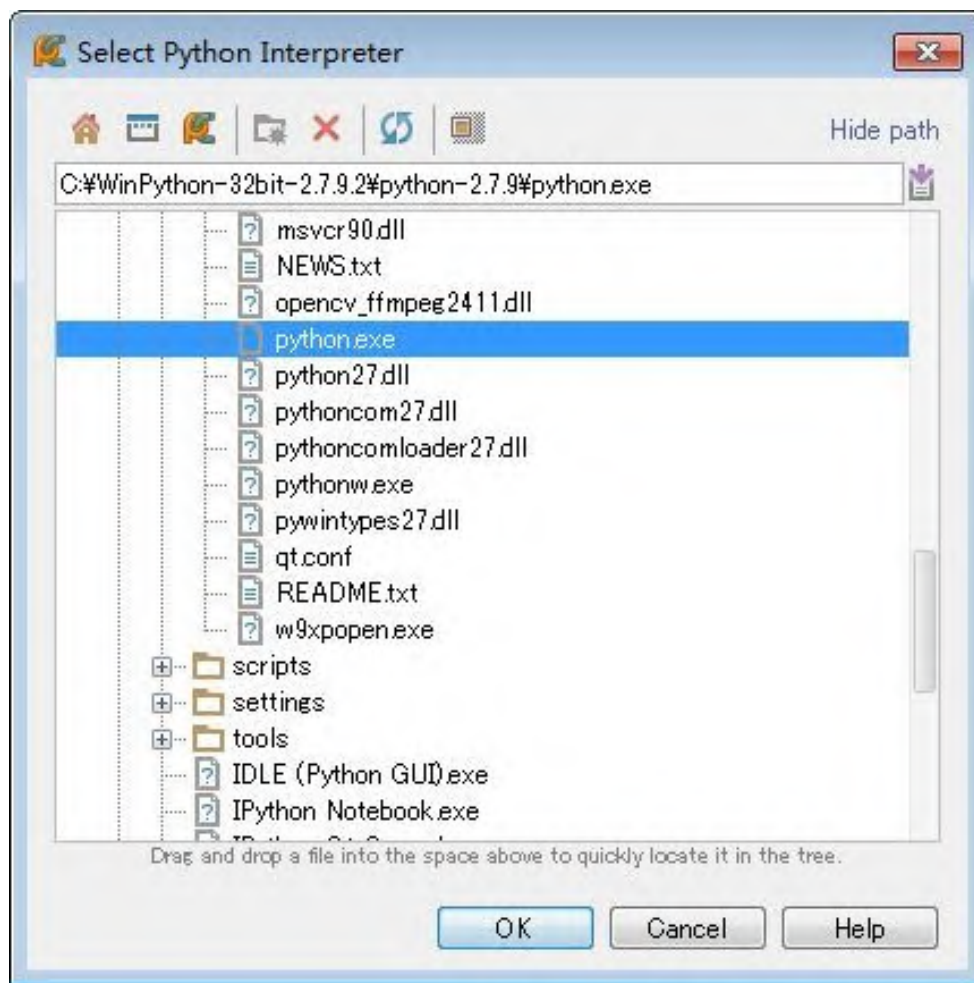


图1-6 配置Python解释器的路径

由于本书提供的代码没有复制到Python的库搜索路径中，可以将scipy2的路径添加进PYTHONPATH环境变量，或者在PyCharm中将scipy2所在的路径添加进Python的库搜索路径。单击上面提到的齿轮按钮，并选择“More...”，将打开图1-7中左侧的对话框，选择解释器之后，单击右侧工具栏中最下方的按钮，打开路径配置对话框，通过此对话框添加本书提供的scipy2库所在的路径。

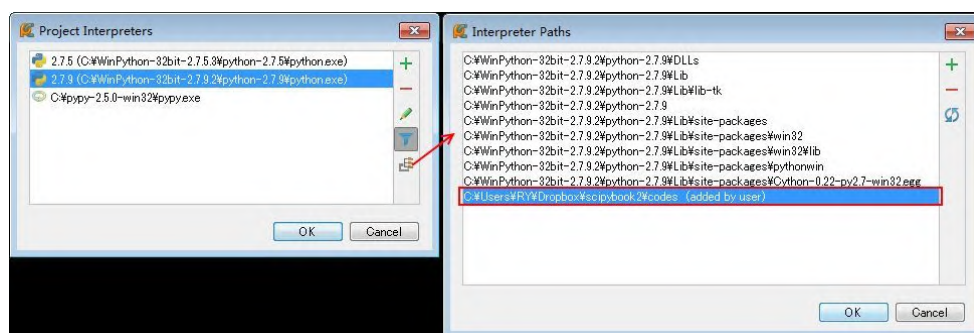


图1-7 添加库搜索路径

1.2 IPython Notebook入门



与本节内容对应的Notebook为：01-intro/intro-200-ipython.ipynb。

自从IPython 1.0发布以来，越来越多的科学家、研究者、教师使用IPython Notebook处理数据、写研究报告，甚至编写书籍。可以使用下面的nbviewer网站查看在网络上公开的Notebook：



<http://nbviewer.ipython.org/>

通过这个网站可以快速查看网络上任何Notebook的内容。

在IPython的官方网站上收集了许多开发者发布的Notebook:



<https://github.com/ipython/ipython/wiki/A-gallery-of-interesting-IPython-Notebooks>

上面有许多有趣的Notebook。

本节简要介绍IPython Notebook的基本使用方法、魔法命令以及显示系统等方面的内容。

1.2.1 基本操作

1. 运行IPython Notebook

使用系统的命令行工具切换到保存Notebook文档的目录，输入
ipython notebook命令即可启动Notebook服务器，并通过系统的默认

浏览器打开地址<http://127.0.0.1:8888>。建议读者最好使用Firefox或Chrome浏览Notebook。

本书提供的代码目录scipybook2中包含了一个启动Notebook的批处理文件run_notebook.bat。运行该批处理文件之后，在浏览器的Notebook列表中依次单击01-intro→intro-100-ipython.ipynb，就能打开与本节对应的Notebook文档。

如图1-8所示，Notebook采用浏览器作为界面，首页显示当前路径下的所有Notebook文档和文件夹。单击“New Notebook”按钮或文档名将打开一个新的页面，同时启动一个运算核进程与其交互。每个打开的Notebook页面都有单独的Python进程与之对应，在Notebook中输入的所有命令都将由浏览器传递到服务器程序，再转发到该进程运行。文档的读取和保存工作由服务器进程完成，而运算核进程则负责运行用户的程序。因此即使用户程序造成运算核进程异常退出，也不会丢失任何用户输入的数据。在关闭服务器进程之前，确保所有的Notebook都已保存。

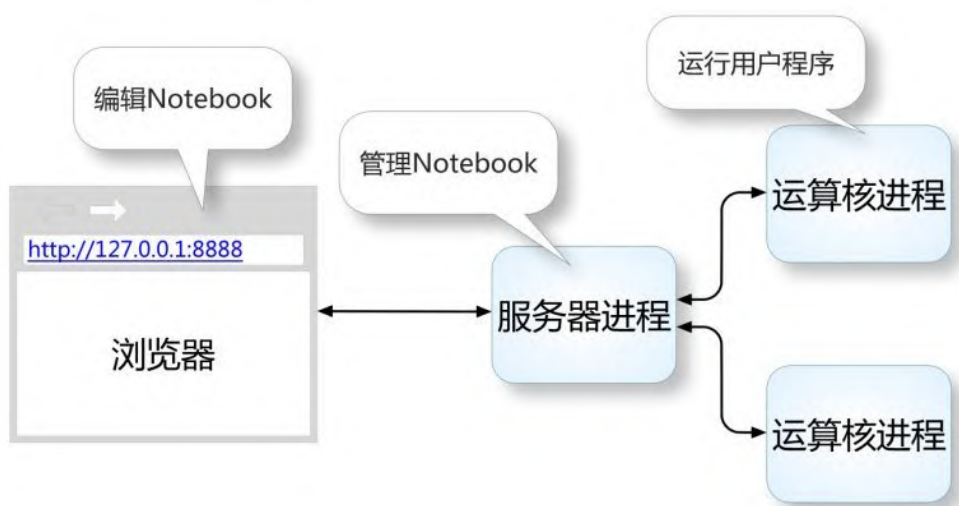


图1-8 IPython Notebook架构示意图

Notebook有自动存档和恢复功能，可通过File → Revert to Checkpoint菜单恢复到以前的版本。此外为了确保安全，打开他人创建的Notebook时，不会运行其中的Javascript程序和显示SVG图像。如果确信来源可靠，可以通过File → Trusted Notebook信任该Notebook。

2. 操作单元



notebooks\01-intro\notebook-train.ipynb: Notebook的操作教程，读者可以使用它练习Notebook的基本操作。

Notebook由多个竖向排列的单元构成，每个单元可以有以下两种样式：

- Code: Code单元中的文本将被作为代码执行，执行代码时按Shift+Enter快捷键，即同时按下Shift和Enter键。
- Markdown: 使用Markdown的格式化文本，可以通过简单的标记表示各种显示格式。

单元的样式可以通过工具栏中的下拉框或快捷键来选择。为了快速操作这些单元格，需要掌握一些快捷键，完整的快捷键列表可以通过菜单Help → Keyboard Shortcuts查看。

Notebook有两种编辑模式：命令模式和单元编辑模式。在命令模式中，被选中的单元格的边框为灰色。该模式用来对整个单元格进行

操作，例如删除、添加、修改格式等。按Enter键进入单元编辑模式，边框的颜色变为绿色，并且上方菜单条的右侧会出现铅笔图标，表示目前处于编辑状态。按Esc键可返回命令模式。

3. 安装MathJax

编写技术资料少不了输入数学公式，Notebook使用MathJax将输入的LaTeX文本转换成数学公式。由于MathJax库较大，没有集成到IPython中，而是直接从MathJax官网载入，因此如果计算机没有联网，就无法正确显示数学公式。为了解决这个问题，可以在单元中输入如下程序，它将会下载MathJax到本地硬盘：

```
from IPython.external.mathjax import install_mathjax,
default_dest
install_mathjax()
```

MathJax完整解压之后，约需100MB空间，其中大都是为旧版浏览器准备的PNG字体图像文件。执行下面的语句可以快速删除存放PNG字体图片的文件夹：

```
from os import path
import shutil

png_path = path.join(default_dest, "fonts/HTML-
CSS/TeX/png")
shutil.rmtree(png_path)
```

运行完上面的命令之后，在命令模式下按m键将单元样式切换到Markdown。然后输入如下LaTeX文本：

```

$$e^{i\pi} + 1 = 0$$

```

按Shift+Enter快捷键之后，其内容将被转换成数学公式显示：
$$e^{i\pi} + 1 = 0$$
。

在本书提供的scipybook2下的settings目录下已经安装了MathJax，因此不必联网也可以看到数学公式。

4. 操作运算进程

在代码单元中输入的代码都将在运算核进程的运行环境中执行。当执行某些代码出现问题时，可以通过Kernel菜单中的选项操作该进程：

- Interrupt：中断运行当前的程序，当程序进入死循环时可以通过它中断程序运行。
- Restart：当运算核进程在扩展模块的程序中进入死循环，无法通过Interrupt菜单中断时，可以通过此选项重新启动运算核进程。

一旦运算核进程被关闭，运行环境中的对象将不复存在，此时可以通过Cell → Run All菜单再次执行所有单元中的代码。代码将按照从上到下的顺序执行。由于用户在编写Notebook时，可以按照任意顺

序执行单元，因此为了保证能再现运行环境中的所有对象，请记住调整单元的先后顺序。

1.2.2 魔法(Magic)命令

IPython提供了许多魔法命令，使得在IPython环境中的操作更加得心应手。魔法命令都以%或%%开头，以%开头的为行命令，以%%开头的为单元命令。行命令只对命令所在的行有效，而单元命令则必须出现在单元的第一行，对整个单元的代码进行处理。

执行%magic可以查看关于各个命令的说明，而在命令之后添加?可以查看命令的详细说明。此外扩展库可以提供自己的魔法命令，这些命令可以通过%load_ext载入。例如%load_ext cython载入%%cython命令，以该命令开头的单元将调用Cython编译其中的代码。

1. 显示matplotlib图表

matplotlib是Python世界中最著名的绘图扩展库，支持输出多种格式的图形图像，并且可以使用多种GUI界面库交互式地显示图表。使用%matplotlib命令可以将matplotlib的图表直接嵌入到Notebook中，或者使用指定的界面库显示图表，它有一个参数指定matplotlib图表的显示方式。

在下面的例子中，inline表示将图表嵌入到Notebook中。因此由最后一行pl.plot()创建的图表将直接显示在该单元之下：

```
%matplotlib inline  
  
import pylab as pl  
  
pl.seed(1)  
  
data = pl.randn(100)  
  
pl.plot(data)
```

内嵌图表的输出格式默认为PNG，可以通过%config命令修改这个配置。%config命令可以配置IPython中的各可配置对象，其中InlineBackend对象为matplotlib输出内嵌图表时所使用的配置，我们配置它的figure_format="svg"，这样可将内嵌图表的输出格式修改为SVG。

```
%config InlineBackend.figure_format="svg"  
  
pl.plot(data)
```

内嵌图表很适合制作图文并茂的Notebook，然而它们是静态的，无法进行交互。可以将图表输出模式修改为使用GUI界面库，下面的qt4表示使用QT4界面库显示图表。请读者根据自己系统的配置，选择合适的界面库：gtk、osx、qt、qt4、tk、wx。

执行下面的语句将弹出一个窗口显示图表，可以通过鼠标和键盘与此图表交互。请注意该功能只能在运行IPython Kernel的机器上显示图表。

```
%matplotlib qt4  
  
pl.plot(data)
```

2. 性能分析

性能分析对编写处理大量数据的程序非常重要，特别是Python这样的动态语言，一条语句可能会执行很多内容，有的是动态的，有的调用扩展库。不做性能分析，就无法对程序进行优化。IPython提供了性能分析的许多魔法命令。

`%timeit`调用`timeit`模块对单行语句重复执行多次，计算出执行时间。下面的代码测试修改列表的单个元素所需的时间：

```
a = [1,2,3]
%timeit a[1] = 10
10000000 loops, best of 3: 69.3 ns per loop
```

`%%timeit`则用于测试整个单元中代码的执行时间。下面的代码测试空列表中循环添加10个元素所需的时间：

```
%%timeit
a = []
for i in xrange(10):
    a.append(i)
1000000 loops, best of 3: 1.82 µs per loop
```

`timeit`命令会重复执行代码多次，而`time`则只执行一次代码，输出代码的执行情况。和`timeit`命令一样，`time`可以作为行命令和单元命令。下面的代码统计往空列表中添加10万个元素所需的时间：

```
%%time
a = []
for i in xrange(100000):
```

```
a.append(i)
Wall time: 18 ms
```

`time`和`timeit`命令都使用`print`输出信息，如果希望用程序分析这些信息，可以使用`%%capture`命令，将单元格的输出保存为一个对象。下面的程序对不同长度的列表调用`random.shuffle()`以打乱顺序，用`%time`记录下`shuffle()`的运行时间：

```
%%capture time_results
import random
for n in [1000, 5000, 10000, 50000, 100000, 500000]:
    print "n={0}".format(n)
    alist = range(n)
    %time random.shuffle(alist)
```

`time_results.stdout`属性保存标准输出管道中的输出信息：

```
print time_results.stdout
n=1000
Wall time: 1 ms
n=5000
Wall time: 5 ms
n=10000
Wall time: 10 ms
n=50000
Wall time: 40 ms
n=100000
Wall time: 62 ms
```



```
n=500000  
Wall time: 400 ms
```

如果在调用`%timeit`命令时添加`-o`参数，则返回一个表示运行时间信息的对象。下面的程序对不同长度的列表调用`sorted()`排序，并使用`%timeit`命令统计排序所需的时间：

```
timeit_results = []  
for n in [5000, 10000, 20000, 40000, 80000, 160000,  
320000]:  
    alist = [random.random() for i in xrange(n)]  
    res = %timeit -o sorted(alist)  
    timeit_results.append((n, res))  
1000 loops, best of 3: 1.56 ms per loop  
100 loops, best of 3: 3.32 ms per loop  
100 loops, best of 3: 7.57 ms per loop  
100 loops, best of 3: 16.4 ms per loop  
10 loops, best of 3: 35.8 ms per loop  
10 loops, best of 3: 81 ms per loop  
10 loops, best of 3: 185 ms per loop
```

图1-9显示了排序的耗时结果。横坐标为对数坐标轴，表示数组的长度；纵坐标为平均每个元素所需的排序时间。可以看出每个元素所需的平均排序时间与数组长度的对数成正比，因此可以计算出排序函数`sorted()`的时间复杂度为： $O(n\log n)$ 。

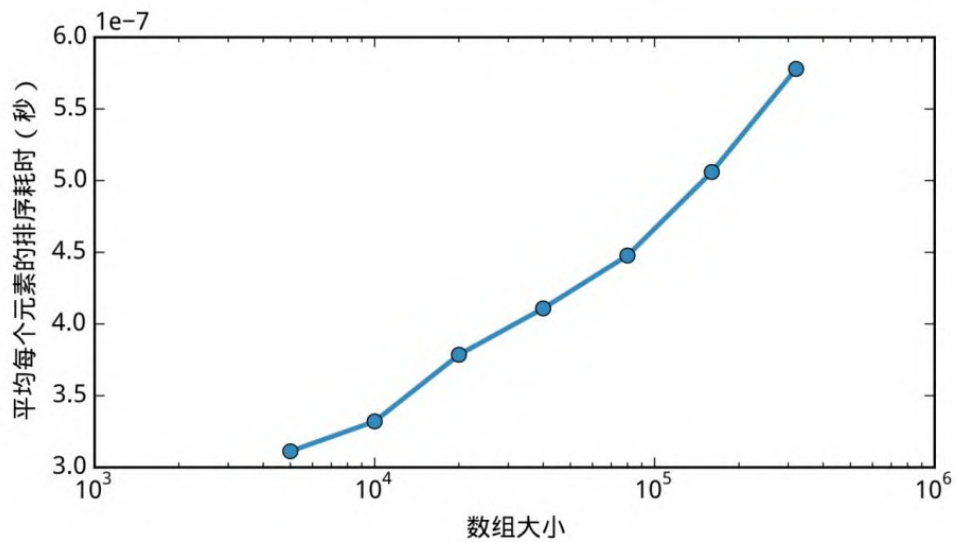


图1-9 sorted() 函数的时间复杂度

%%prun命令调用profile模块，对单元中的代码进行性能剖析。下面的性能剖析显示fib()运行了21891次，而fib_fast()则只运行了20次：

```

%%prun
def fib(n):
    if n < 2:
        return 1
    else:
        return fib(n-1) + fib(n-2)

def fib_fast(n, a=1, b=1):
    if n == 1:
        return b
    else:
        return fib_fast(n-1, b, a+b)

fib(20)
fib_fast(20)

```

21913 function calls (4 primitive calls) in 0.007 seconds

Ordered by: internal time

ncalls	totttime	percall	cumtime	percall	filename:lineno(function)
21891/1	0.007	0.000	0.007	0.007	<string>:2(fib)
20/1	0.000	0.000	0.000	0.000	<string>:8(fib_fast)
1	0.000	0.000	0.007	0.007	<string>:2(<module>)
1	0.000	0.000	0.000	0.000	{method 'disable' of '_lsprof.Profiler' objects}

3. 代码调试

%debug命令用于调试代码，它有两种用法：一种是在执行代码之前设置断点进行调试；另一种则是在代码抛出异常之后，执行%debug命令查看调用堆栈。下面先演示第二种用法：

```

import math

def sinc(x):
    return math.sin(x) / x

[sinc(x) for x in range(5)]

```

```

-----
ZeroDivisionError                                Traceback (most
recent call last)
<ipython-input-28-9b69eaad97fe> in <module>()
      4     return math.sin(x) / x
      5
----> 6 [sinc(x) for x in range(5)]

<ipython-input-28-9b69eaad97fe> in sinc(x)
      2
      3 def sinc(x):
----> 4     return math.sin(x) / x
      5
      6 [sinc(x) for x in range(5)]

ZeroDivisionError: float division by zero

```

上面的程序抛出了ZeroDivisionError异常，下面用%debug查看调用堆栈。在调试模式下可以使用pdb模块提供的调试命令，例如用命令p x显示变量x的值：

```

%debug
><ipython-input-28-9b69eaad97fe>(4) sinc()
      3 def sinc(x):
----> 4     return math.sin(x) / x
      5

```

```
ipdb> p x
0
ipdb> q
```

还可以先设置断点，然后运行程序。但是%debug的断点需要指定文件名和行号，使用起来并不是太方便。本书提供了%%func_debug单元命令，可以通过它指定中断运行的函数。在下面的例子中，程序将在numpy.unique()的第一行中断运行，然后通过输入命令n单步运行程序，最后输入命令c继续运行：

```
%%func_debug np.unique
np.unique([1, 2, 5, 4, 2])
      Breakpoint 1 at c:\winpython-32bit-2.7.9.2\python-2.7.9\lib\site-packages\numpy\lib\arraysetops.py:96
      NOTE: Enter 'c' at the ipdb> prompt to continue execution.
      > c:\winpython-32bit-2.7.9.2\python-2.7.9\lib\site-packages\numpy\lib\arraysetops.py(173)
unique()
      172      """
--> 173      ar = np.asanyarray(ar).flatten()
      174

ipdb> n
      >c:\winpython-32bit-2.7.9.2\python-2.7.9\lib\site-packages\numpy\lib\arraysetops.py(175)
unique()
```

```

174
--> 175         optional_indices = return_index or
return_inverse
176         optional_returns = optional_indices or
return_counts

ipdb> c

```

4. 自定义的魔法命令



`scipy2.utils.nbmagics`: 该模块中定义了本书提供的魔法命令，如果读者使用本书提供的批处理运行Notebook，则该模块已经载入。`notebooks\01-intro\scipy2-magics.ipynb`是这些魔法命令的使用说明。

IPython提供了很方便的自定义魔法命令的方法。最简单的方法就是使用`register_line_magic`和`register_cell_magic`装饰器将函数转换为魔法命令。下面的例子使用`register_line_magic`定义了一个行魔法命令`%find`，它在指定的对象中搜索与目标匹配的属性名：

```

from IPython.core.magic import register_line_magic

@register_line_magic
def find(line):

```

```

from IPython.core.getipython import get_ipython
from fnmatch import fnmatch

items = line.split() ❶
patterns, target = items[:-1], items[-1]
ipython = get_ipython() ❷
names = dir(ipython.ev(target)) ❸

results = []
for pattern in patterns:
    for name in names:
        if fnmatch(name, pattern):
            results.append(name)
return results

```

当调用`%find`行魔法命令时，魔法命令后面的所有内容都传递给`line`参数。❶按照空格对`line`进行分隔，除最后一个元素之外，其余的元素都作为搜索模板，而最后一个参数则为搜索的目标。❷通过`get_ipython()`函数获得表示IPython运算核的对象，通过该对象可以操作运算核。❸调用运算核的`ev()`方法对表达式`target`求值以得到实际的对象，并用`dir()`获取该对象的所有属性名。

最后使用`fnmatch`模块对搜索模板和属性名进行匹配，将匹配结果保存到`results`并返回。下面使用`%find`命令在`numpy`模块中搜索所有以`array`开头或包含`mul`的属性名：

```
import numpy as np
names = %find array* *mul* np
names
```

['array',	'array2string',	'array_equal',	'array_equiv',
'array_repr',	'array_split',	'array_str',	'multiply',
'polymul',	'ravel_multi_index']		

下面的例子使用`register_cell_magic`注册`%%cut`单元命令。在调试代码时，我们经常会添加`print`语句以输出中间结果。但如果输出的字符串太多，会导致浏览器的速度变慢甚至失去响应。此时可以使用`%%cut`限制程序输出的行数和字符数。

`cut()`函数有两个参数：`line`和`cell`，其中`line`为单元第一行中除了魔法命令之外的字符串，而`cell`为除了单元中第一行之外的所有字符串。`line`通常为魔法命令的参数，而`cell`则为需要执行的代码。IPython提供了基于装饰器的参数分析函数。下面的例子使用`argument()`声明了两个参数`-l`和`-c`，它们分别指定最大行数和最大字符数，它们的默认值分别为100和10000：

```
from IPython.core.magic import register_cell_magic
from IPython.core.magic_arguments import argument,
magic_arguments, parse_argstring

@magic_arguments()
    @argument('-l', '--lines', help='max lines', type=int,
default=100)
    @argument('-c', '--chars', help='max chars', type=int,
default=10000)

@register_cell_magic
```



```
def cut(line, cell):  
    from IPython.core.getipython import get_ipython  
    from sys import stdout  
    args = parse_argstring(cut, line) ❶  
    max_lines = args.lines  
    max_chars = args.chars  
  
    counters = dict(chars=0, lines=0)  
  
    def write(string):  
        counters["lines"] += string.count("\n")  
        counters["chars"] += len(string)  
  
        if counters["lines"] >= max_lines:  
            raise IOError("Too many lines")  
        elif counters["chars"] >= max_chars:  
            raise IOError("Too many characters")  
        else:  
            old_write(string)  
  
    try:  
        old_write, stdout.write = stdout.write, write❷  
        ipython = get_ipython()  
        ipython.run_cell(cell) ❸  
    finally:  
        del stdout.write❹
```

❶调用`parse_argstring()`分析行参数，它的第一个参数是使用`argument`装饰器修饰过的魔法命令函数，第二个参数为行命令字符串。❷在调用单元代码之前，将`stdout.write()`替换为限制输出行数和字符数的`write()`函数。❸调用运算核对象的`run_cell()`来运行单元代码。❹运行完毕之后将`stdout.write()`删除，恢复到原始状态。

下面是使用`%%cut`限制输出行数的例子：

```
%%cut -l 5

for i in range(10000):
    print "I am line", i

I am line 0
I am line 1
I am line 2
I am line 3
I am line 4
-----

-----

IOError                                Traceback (most
recent call last)

<ipython-input-9-5d2e5180be18> in <module>()
      1 for i in range(10000):
----> 2     print "I am line", i

<ipython-input-8-e0ddfb5e18b6> in write(string)
     20
     21     if counters["lines"] >= max_lines:
```

```
---> 22             raise IOError("Too many lines")
      23             elif counters["chars"] >= max_chars:
      24             raise IOError("Too many characters")
```

```
IOError: Too many lines
```

1.2.3 Notebook的显示系统

若单元中代码的最后一行没有缩进，并且不以分号结尾，则在单元的输出栏中显示运行该代码后得到的对象。此外运算核的标准输出被重定向到单元的输出框中，因此可以使用print语句输出任何信息。例如在下面的程序中，使用循环进行累加计算，在循环体中使用print输出中间结果，而最后一行的运算结果就是变量s的值：

```
s = 0
for i in range(4):
    s += i
    print "i={}, s={}".format(i, s)

s
i=0, s=0
i=1, s=1
i=2, s=3
i=3, s=6
6
```

1. display模块

由于Notebook采用浏览器作为界面，因此除了可以显示文本之外，还可以显示图像、动画、HTML等多种形式的数​​据。有关显示方面的功能均在IPython.display模块中定义。其中提供了表1-3所示的对象，用于显示各种格式的数据。

表1-3 IPython.display模块提供的用于显示各种格式的数据的类

类名	说明
Audio	将二进制数据、文件或网址显示为播放声音的控件
FileLink	将文件夹路径显示为一个超链接
FileLinks	将文件夹路径显示为一组超链接
HTML	将字符串、文件或网址显示为HTML
Image	将表示图像的​​二进制字符串、文件或网址显示为图像
Javascript	将字符串作为Javascript代码在浏览器中运行
Latex	将字符串作为LaTeX代码显示，主要用于显示数学公式
SVG	将字符串、文件或网址显示为SVG图形

当对单元中程序的最后一行求值并得到上述类型的对象时，将在单元的输出栏中显示对应的格式。也可以使用display模块中的display()函数在程序中输出这些对象。下面的程序使用Latex对象输出了3个数学公式，其中前两个使用display()输出，而由于最后一行的求值结果为Latex()对象，因此它也会被显示为数学公式。

```
from IPython import display

for i in range(2, 4):
    display.display(display.Latex("$x^{i} + y^{i}$".format(i=i)))
display.Latex("$x^4 + y^4$")
```

$$x^2 + y^2$$

$$x^3 + y^3$$

$$x^4 + y^4$$

Image对象可以用于显示图像，当用url参数时，它会从指定的网址获取图像，并显示在Notebook中。如果embed参数为True，图像的数据将直接嵌入到Notebook之中，这样此后打开此Notebook时，即使没有联网也可以显示该图像。

```
logourl = "https://www.python.org/static/community_logos/python-logo-master-v3-TM.png"
display.Image(url=logourl, embed=True)
```

在后续的章节中经常会将NumPy数组显示为图像，这时可以使用matplotlib中提供的函数将数组转换成PNG图像的字符串，然后通过Image将图像数据嵌入到Notebook中。下面的as_png()使用matplotlib中的imsave()将数组转换成PNG图像数据：

```
def as_png(img, **kw):
    "将数组转换成PNG格式的字符串数据"
    import io
    from matplotlib import image
    from IPython import display
    buf = io.BytesIO()
    image.imsave(buf, img, **kw)
    return buf.getvalue()
```

下面的程序通过公式 $\sin(x^2 + 2 \cdot y^2 + x \cdot y)$ 生成二维数组z，并调用as_png()将其转换为字符串png，查看该字符串的头10个字节，可以看出该字符串就是PNG图像文件中的数据。最后使用Image()将该字符串使用PNG图像显示出来，结果如图1-10（左）所示。

```
import numpy as np
y, x = np.mgrid[-3:3:300j, -6:6:600j]
z = np.sin(x**2 + 2*y**2 + x*y)
png = as_png(z, cmap="Blues", vmin=-2, vmax=2)
print repr(png[:10])
display.Image(png)
'\x89PNG\r\n\x1a\n\x00\x00'
```

2. 自定义对象的显示格式

有两种方式可以自定义对象在Notebook中的显示格式：

- 给类添加相应的显示方法。
- 为类注册相应的显示函数。

当我们自己编写类的代码时，使用第一种方法最为便捷。和Python的`__str__()`方法类似，只需要定义`_repr_*_()`等方法即可，这里的*可以是html、svg、javascript、latex、png等格式的名称。

在下面的例子中，Color类中定义了IPython用的两个显示方法：`_repr_html_()`和`_repr_png_()`，它们分别使用HTML和PNG图像显示颜色信息。

```
class Color(object):

    def __init__(self, r, g, b):
        self.rgb = r, g, b

    def html_color(self):
        return '#{0:02x}{0:02x}{0:02x}'.format(*self.rgb)

    def invert(self):
        r, g, b = self.rgb
        return Color(255-r, 255-g, 255-b)
```

```

def _repr_html_(self):
    color = self.html_color()
    inv_color = self.invert().html_color()
    template = '<span
                                style="background-color:{c};color:
{ic};padding:5px;">{c}</span>'

    return template.format(c=color, ic=inv_color)

def _repr_png_(self):
    img = np.empty((50, 50, 3), dtype=np.uint8)
    img[:, :, :] = self.rgb
    return as_png(img)

```

下面创建Color对象，并直接查看它，IPython会自动选择最合适的显示格式。由于Notebook是基于HTML的，HTML格式的优先级别最高，因此查看Color对象时，_repr_html_()方法将被调用：

```

c = Color(255, 10, 10)
c

```

为了使用其他格式显示对象，可以调用display.display_*()函数，这里调用display_png()将Color对象转换成PNG图像显示：

```

display.display_png(c)

```

每种输出格式都对应一个Formatter对象，它们被保存在DisplayFormatter对象的formatters字典中，下面获取该字典中与PNG格式对应的Formatter对象：


```

shell = get_ipython()

                                png_formatter
shell.display_formatter.formatters[u'image/png']

```

调用`Formatter.for_type_by_name()`可以为该输出格式添加指定的格式显示函数，其前两个参数分别为模块名和类名。由于使用字符串指定类，因此添加格式显示函数时不需要载入目标类。下面的代码为NumPy的数组添加显示函数`as_png()`：

```

png_formatter.for_type_by_name("numpy", "ndarray",
as_png)

```

下面查看前面创建的数组`z`，它将以图像的形式呈现，结果如图1-10（右）所示。

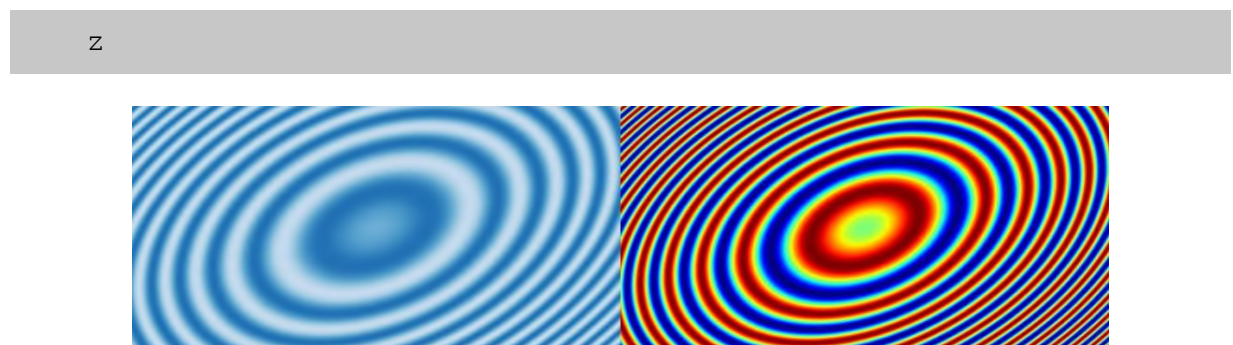


图1-10 使用`as_png()`将数组显示为PNG图像(左)，将`as_png()`注册为数组的显示格式(右)

如果目标类已被载入，可以使用`for_type()`方法为其添加格式显示函数。下面的代码将表示分数的`Fraction`类使用LaTeX的数学公式进行显示：

```

from fractions import Fraction

latex_formatter =
shell.display_formatter.formatters[u"text/latex"]

def fraction_formatter(obj):
    return '$$\frac{%d}{%d}$$' % (obj.numerator,
obj.denominator)

latex_formatter.for_type(Fraction, fraction_formatter)

Fraction(3, 4) ** 4 / 3

```

$$\frac{27}{256}$$

1.2.4 定制IPython Notebook

虽然IPython只提供了最基本的编辑、运行Notebook的功能，但是它具有丰富的可定制性，用户可以根据自己的需要打造出独特的Notebook开发环境。如图1-8所示，IPython Notebook系统由浏览器、服务器和运算核三部分组成。IPython分别提供了这三部分的定制方法。

1. 用户配置(profile)

每次启动IPython时都会从指定的用户配置(profile)文件夹下读取配置信息。下面的代码输出当前的用户配置文件夹的路径：该路径由HOME环境变量、.ipython和profile_配置名构成。



在本书提供的运行IPython Notebook的批处理文件中配置了HOME环境变量，因此能将配置文件夹和Notebook文件一起打包。

```
import os
ipython = get_ipython()
print "HOME环境变量:", os.environ["HOME"]
print "IPython配置文件夹:", ipython.ipython_dir
print "  当前的用户配置文件夹:",
ipython.config.ProfileDir.location
HOME环境变量: C:\Users\RY\Dropbox\scipybook2\settings
IPython 配置文件夹:
C:\Users\RY\Dropbox\scipybook2\settings\.ipython
当前的用户配置文件夹:
C:\Users\RY\Dropbox\scipybook2\settings\.ipython\profile_scipyb
ook2
```

当前的用户配置文件夹:

```
ipython profile create test
```

修改用户配置文件夹之下的配置文件之后，在启动Notebook时通过--profile参数指定所采用的用户配置:

```
ipython notebook --profile test
```

2. 服务器扩展插件和Notebook扩展插件

在`.ipython`文件夹之下还有两个子文件夹——`extensions`和`nbextensions`，它们分别用于保存服务器和浏览器的扩展程序。

- `extensions`：存放用Python编写的服务器扩展程序。
- `nbextensions`：存放Notebook客户端的扩展程序，通常为JavaScript和CSS样式表文件。

Notebook的服务器基于tornado服务器框架开发，因此编写服务器的扩展程序需要了解tornado框架，而开发Notebook客户端（浏览器的界面部分）的扩展程序则需要了解HTML、JavaScript和CSS样式表等方面的内容。这些内容与本书的主题无关，就不再详细叙述了。下面看看如何安装他人开发的扩展程序。



<https://github.com/ipython-contrib/IPython-notebook-extensions/wiki/config-extension>

安装IPython扩展程序的说明。

首先执行下面的语句来安装Notebook客户端的扩展程序，`user`参数为`True`表示将扩展安装在HOME环境变量路径之下的`.ipython`文件夹中：

```
import IPython.html.nbextensions as nb

ext= 'https://github.com/ipython-contrib/IPython-notebook-extensions/archive/3.x.zip'

nb.install_nbextension(ext, user=True)
```

上面的程序将在nbextensions文件夹下创建IPython-notebook-extensions-3.x文件夹，其中包含了许多客户端扩展程序。接下来按照如下步骤完成安装：

(1) 将 nbextensions\IPython-notebook-extensions-3.x\config移到nbextensions文件夹之下。

(2) 将nbextensions\config\nbextensions.py移到extensions文件夹之下。

(3) 在.ipython之下创建templates文件夹。

(4) 将 nbextensions\config\nbextensions.html 移到 templates文件夹之下。

(5) 将nbextensions\config\ipython_notebook_config.py中的代码添加到profile_default\ipython_notebook_config.py中。

(6) 访问<http://localhost:8888/nbextensions/>，在该页面上可以管理nbextensions文件夹下安装的客户端扩展程序。

当Notebook服务器启动时，会运行用户配置(profile)文件夹之下的ipython_notebook_config.py文件，并使用其中的配置。

下面是ipython_notebook_config.py中的配置代码。❶首先将extensions文件夹添加到Python的模块搜索路径之下，因此该路径之下的nbextensions.py文件可以通过import nbextensions载入。❷指定服务器扩展程序的模块名，由于之前添加了搜索路径，因此Python可以直接通过模块名'nbextensions'找到对应的文件nbextensions.py。❸将templates文件夹添加到服务器扩展程序的网页模板的搜索路径，让服务器可以找到nbextensions.html文件。

```
from IPython.utils.path import get_ipython_dir
import os
import sys

ipythondir = get_ipython_dir()
extensions = os.path.join(ipythondir, 'extensions')
sys.path.append( extensions ) ❶

c = get_config()
c.NotebookApp.server_extensions = [ 'nbextensions' ] ❷
c.NotebookApp.extra_template_paths =
[os.path.join(ipythondir, 'templates')] ❸
```

nbextensions 扩展程序为服务器添加了一个新的URL——<http://localhost:8888/nbextensions/>，通过该路径可以开启或禁止指定的客户端扩展程序。nbextensions 扩展程序通过递归搜索nbextensions文件夹下的YAML文件识别客户端扩展程序，IPython-notebook-extensions-3.x目录下只有部分扩展程序附带了YAML文件，

读者可以仿照这些文件为其他的扩展程序添加相应的YAML文件，这样就可以通过nbextensions页面管理扩展程序了。

3. 添加新的运算核

由于执行用户代码的运算核与Notebook服务器是独立的进程，因此不同的Notebook可以使用不同版本的Python，甚至是其他语言的运算核。IPython的下一个版本将改名为Jupyter，其目标是创建通用的科学计算的开发环境，支持Julia、Python和R等在数据处理领域流行的语言。下面以Python3-64bit为例介绍如何添加新的运算核。

首先从WinPython的网址下载WinPython-64bit-3.4.3.3.exe，并安装在C盘根目录之下。然后运行下面的代码来创建运算核配置文件：

```
import os

from os import path

import json


ipython = get_ipython()

kernels_folder = path.join(ipython.ipython_dir, "kernels")

if not path.exists(kernels_folder):

    os.mkdir(kernels_folder)


python3_path = "C:\\WinPython-64bit-3.4.3.3\\scripts\\python.bat"
```

```

kernel_settings = {
    "argv": [python3_path,
              "-m", "IPython.kernel", "-f", "{connection_file}"],
    "display_name": "Python3-64bit",
    "language": "python"
}

kernel_folder = path.join(kernels_folder,
kernel_settings["display_name"])

if not path.exists(kernel_folder):
    os.mkdir(kernel_folder)

kernel_fn = path.join(kernel_folder, "kernel.json")

with open(kernel_fn, "w") as f:
    json.dump(kernel_settings, f, indent=4)

```

上面的代码创建.ipython\kernels\python3-64bit\kernel.json文件，它是一个JSON格式的字典，其中“argv”键为运算核的启动命令，“display_name”为运算核的显示名称，“language”为运算核的语言。

刷新Notebook的索引页面之后，可以在“New”下拉菜单中找到“Python3-64bit”选项，单击它将打开一个以Python3 64bit解释器为运算核的Notebook页面。在Notebook页面中也可以使用“Kernel”

菜单更改当前的运算核。运算核的配置保存在Notebook文件中，因此下一次开启Notebook时，将自动使用最后一次选择的运算核。

感兴趣的读者可以试试添加更多的运算核，笔者在Windows系统下成功地安装了PyPy、Julia、R、NodeJS等运算核。

1.3 扩展库介绍



与本节内容对应的Notebook为：`01-intro/intro-300-library.ipynb`。

Python科学计算方面的内容由许多扩展库构成。本书将对编写科学计算软件时常用的一些扩展库进行详细介绍，这里先简要介绍本书涉及的扩展库。

1.3.1 数值计算库

NumPy为Python带来了真正的多维数组功能，并且提供了丰富的函数库来处理这些数组。在下面的例子中，使用如下公式计算 π ，可以看到在NumPy中使用数组运算替代通常需要借助循环的运算：

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \frac{4}{13} - \dots$$

```
import numpy as np

n = 100000

np.sum(4.0 / np.r_[1:n:4, -3:-n:-4])

3.141572653589833
```

SciPy则在NumPy基础上添加了众多的科学计算所需的各种工具，它的核心计算部分都是一些久经考验的Fortran数值计算库，例如：

- 线性代数使用LAPACK库
- 快速傅立叶变换使用FFTPACK库
- 常微分方程求解使用ODEPACK库
- 非线性方程组求解以及最小值求解等使用MINPACK库

在下面的例子中，使用SciPy中提供的数值积分函数quad() 计算 π ：

$$\pi = 2 \int_{-1}^1 \sqrt{1-x^2} \, dx$$

```
from scipy.integrate import quad

quad(lambda x: (1-x**2)**0.5, -1, 1)[0] * 2

3.141592653589797
```

1.3.2 符号计算库

SymPy是一套数学符号运算的扩展库，虽然与一些专门的符号运算软件相比，SymPy的功能以及运算速度都还是较弱的，但是由于它完全采用Python编写，因此能够很好地与其他的科学计算库相结合。

下面用SymPy提供的符号积分函数`integrate()`对上面的公式进行积分运算，可以看到运算的结果为符号表示的 π ：

```
from sympy import symbols, integrate, sqrt
x = symbols("x")
integrate(sqrt(1-x**2), (x, -1, 1)) * 2
pi
```

1.3.3 绘图与可视化

`matplotlib`是Python最著名的绘图库，它提供了一整套和MATLAB类似的绘图函数集，十分适合编写短小的脚本程序进行快速绘图。此外，`matplotlib`采用面向对象的技术来实现，因此组成图表的各个元素都是对象，在编写较大的应用程序时通过面向对象的方式使用`matplotlib`将更加有效。

下面的程序绘制隐函数 $(x^2 + y^2 - 1)^3 - x^2 y^3 = 0$ 的曲线，结果如图1-11所示。

```
x, y = np.mgrid[-2:2:500j, -2:2:500j]
z = (x**2 + y**2 - 1)**3 - x**2 * y**3
pl.contourf(x, y, z, levels=[-1, 0], colors=["red"])
pl.gca().set_aspect("equal")
```

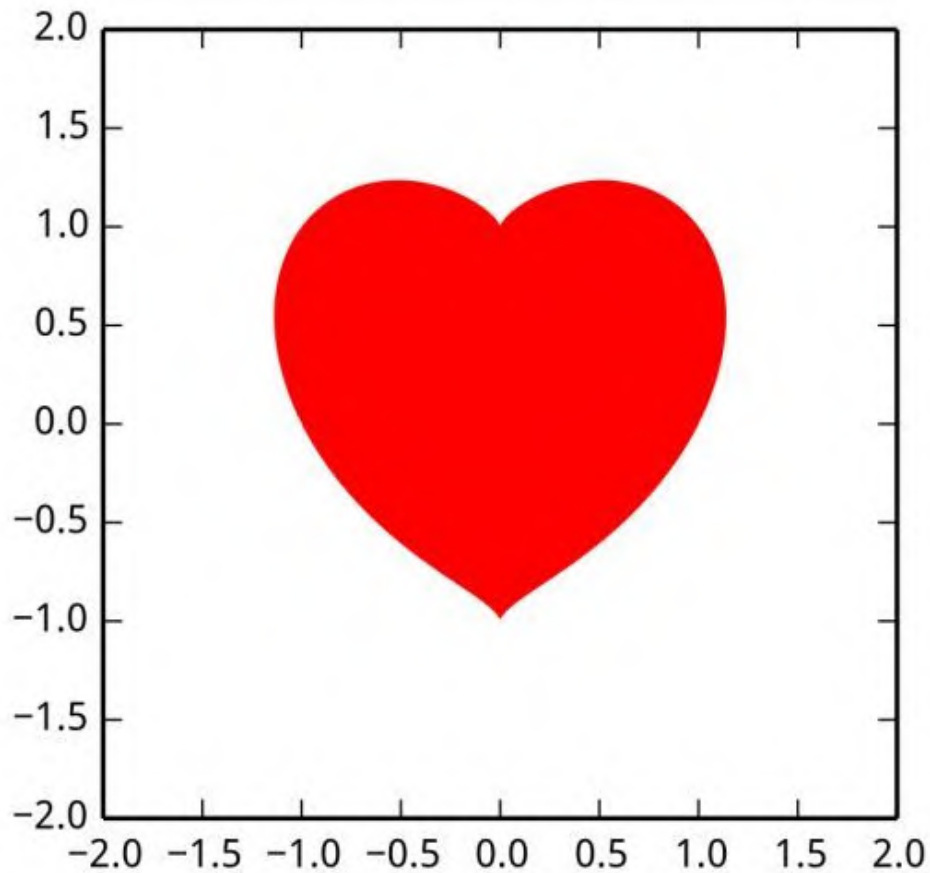


图1-11 matplotlib绘制心形隐函数曲线

VTK是一套功能十分强大的三维数据可视化库，TVTK库在标准的VTK库之上用Traits库进行封装。而Mayavi2则在TVTK的基础上添加了一套面向应用的方便工具，它既可以单独作为3D可视化程序使用，也可以很方便地嵌入到TraitsUI编写的界面程序中。在下面的例子中，使用Mayavi绘制如下隐函数的曲面，结果如图1-12所示。

$$(x^2 + \frac{9}{4}y^2 + z^2 - 1)^3 - x^2z^3 - \frac{9}{80}y^2z^3 = 0$$

```
%%mlab_plot  
  
from mayavi import mlab  
  
x, y, z = np.mgrid[-3:3:100j, -1:1:100j, -3:3:100j]
```

```
f = (x**2 + 9.0/4*y**2 + z**2 - 1)**3 - x**2 * z**3 -  
9.0/80 * y**2 * z**3  
  
contour = mlab.contour3d(x, y, z, f, contours=[0], color=  
(1, 0, 0))
```



图1-12 使用Mayavi绘制心形隐函数曲面

1.3.4 数据处理和分析

Pandas 在 NumPy 的基础之上提供类似电子表格的数据结构 DataFrame，并以此为核心提供大量的数据的输入输出、清洗、处理和分析函数。其核心运算函数使用 Cython 编写，在不失灵活性的前提下保证了函数库的运算速度。

在下面的例子中，从电影打分数据 MovieLens 中读入用户数据文件 u.user，并显示其中的头 5 条数据：

```
import pandas as pd
columns = 'user_id', 'age', 'sex', 'occupation', 'zip_code'
df = pd.read_csv("../data/ml-100k/u.user",
                  delimiter="|", header=None, names=columns)
print df.head()
```

	user_id	age	sex	occupation	zip_code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213

下面使用职业栏对用户数据进行分组，计算每组的平均年龄，按年龄排序之后将结果显示为柱状图，如图1-13所示。可以看到如此复杂的运算在Pandas中可以使用一行代码完成：

```
df.groupby("occupation").age.mean().order().plot(kind="bar",
figsize=(12, 4))
```

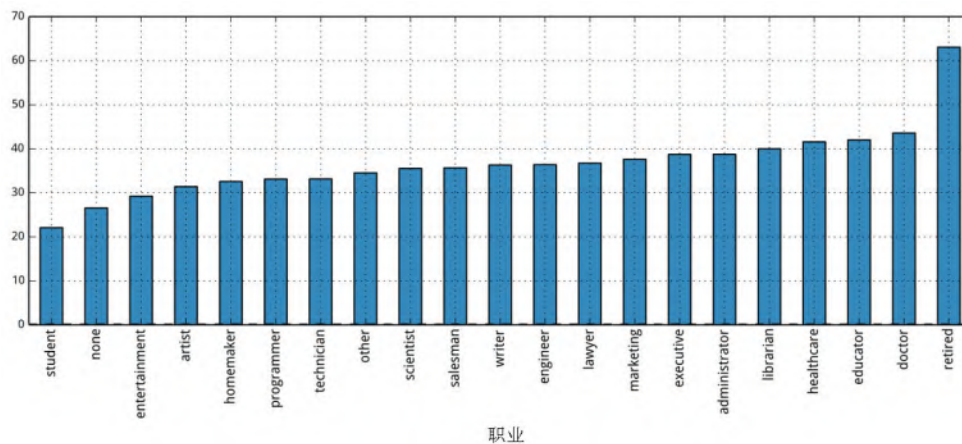


图1-13 使用Pandas统计电影打分用户的职业

1.3.5 界面设计

Python可以使用多种界面库编写GUI程序，例如标准库中自带的以TK为基础的Tkinter、以wxWidgets为基础的wxPython和以QT为基础的pyQt4等界面库。但是使用这些界面库编写GUI程序仍然是一件十分繁杂的工作。为了让读者不在界面设计上耗费大量精力，从而能把注意力集中到如何处理数据上去，本书详细介绍了使用Traits库设计图形界面程序的方法。

Traits库分为Traits和TraitsUI两大部分，Traits为Python添加了类型定义的功能，使用它定义的Trait属性具有初始化、校验、代理、事件等诸多功能。

TraitsUI库基于Traits库，使用MVC（模型—视图—控制器）模式快速定义用户界面，在最简单的情况下，甚至不需要写一句界面相关的代码，就可以通过Traits的属性定义获得一个可以使用的图形界面。使用TraitsUI库编写的程序自动支持wxPython和pyQt两个经典的界面库。

1.3.6 图像处理和计算机视觉

OpenCV是一套开源的跨平台计算机视觉库，可用于开发实时的图像处理、计算机视觉以及模式识别程序。它提供的Python包装模块可调用OpenCV提供的大部分功能。由于它采用NumPy数组表示图像，因此能很方便地与其他扩展库共享图像数据。

在下面的例子中，读入图像moon.jpg，并转换为二值图像。找到二值图像中黑白区域相交的边线，并计算周长和面积。然后通过这两个参数计算 π 。

```

import cv2

img = cv2.imread("moon.jpg", cv2.IMREAD_GRAYSCALE)

_, bimg = cv2.threshold(img, 50, 255, cv2.THRESH_BINARY)

contour, _ = cv2.findContours(bimg, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_TC89_L1)

contour = cv2.approxPolyDP(contour[0], epsilon=2,
closed=False)

area = cv2.contourArea(contour)

perimeter = cv2.arcLength(contour, True)

perimeter**2 / (4 * area)

3.176088313869952

```

1.3.7 提高运算速度

Python的动态特性虽然方便了程序的开发，但也会极大地降低程序的运行速度。使用Cython可以将添加了类型声明的Python程序编译成C语言源代码，再编译成扩展模块，从而提高程序的运算速度。使用Cython既能实现C语言的运算速度，也能使用Python的所有动态特性，极大地方便了扩展库的编写。

下面是按照前面介绍的公式使用循环计算 π 的源程序，使用cdef关键字定义变量的类型，从而提高程序的运行效率：

```

%%cython

import cython

```



```
@cython.cdivision(True)
def calc_pi(int n):
    cdef double pi = 0
    cdef int i
    for i in range(1, n, 4):
        pi += 4.0 / i
    for i in range(3, n, 4):
        pi -= 4.0 / i
    return pi
```

调用`calc_pi()`来计算近似值:

```
calc_pi(1000000)
3.141590653589821
```

下面使用`%timeit`比较`calc_pi()`和NumPy库来计算的运算时间:

```
n = 1000000
%timeit calc_pi(n)
%timeit np.sum(4.0 / np.r_[1:n:4, -3:-n:-4])
100 loops, best of 3: 3.9 ms per loop
100 loops, best of 3: 5.94 ms per loop
```


第2章 NumPy-快速处理数据

标准的Python中用列表（list）保存一组值，可以用来当作数组使用。但由于列表的元素可以是任何对象，因此列表中保存的是对象的指针。这样的话，为了保存一个简单的列表，比如[1, 2, 3]，需要三个指针和三个整数对象。对于数值运算来说，这种结构显然比较浪费内存和CPU计算时间。

此外，Python还提供了array模块，它所提供的array对象和列表不同，能直接保存数值，和C语言的一维数组类似。但是由于它不支持多维数组，也没有各种运算函数，因此也不适合做数值运算。

NumPy的诞生弥补了这些不足，NumPy提供了两种基本的对象：

- ndarray：英文全称为n-dimensional array object，它是存储单一数据类型的数据的多维数组，后统一称之为数组。
- ufunc：英文全称为universal function object，它是一种能够对数组进行处理的特殊函数。

本书采用NumPy 1.9版本，请读者运行下面的程序以查看NumPy的版本号：

```
import numpy
numpy.__version__
'1.9.0'
```

2.1 ndarray对象



与本节内容对应的Notebook为：[02-numpy/numpy-100-ndarray.ipynb](#)。

本书的示例程序假设用以下推荐的方式导入NumPy函数库：

```
import numpy as np
```

NumPy中使用ndarray对象表示数组，它是整个库的核心对象，NumPy中所有的函数都是围绕ndarray对象进行处理的。ndarray的结构并不复杂，但是功能却十分强大。不但可以用它高效地存储大量的数值元素，从而提高数组计算的运算速度，还能用它与各种扩展库进行数据交换。本节的内容可能会有些枯燥，但是为了打下一个良好的基础，让我们从深入理解ndarray对象开始学习Python科学计算之旅。

2.1.1 创建

首先需要创建数组才能对其进行运算和操作。可以通过给array()函数传递Python的序列对象来创建数组，如果传递的是多层嵌套的序列，将创建多维数组(下例中的变量c)：

```
a = np.array([1, 2, 3, 4])
b = np.array((5, 6, 7, 8))
c = np.array([[1, 2, 3, 4], [4, 5, 6, 7], [7, 8, 9, 10]])
```

b	c
[5, 6, 7, 8]	[[1, 2, 3, 4], [4, 5, 6, 7], [7, 8, 9, 10]]

数组的形状可以通过其shape属性获得，它是一个描述数组各个轴的长度的元组(tuple)：

a.shape	b.shape	c.shape
(4,)	(4,)	(3, 4)

数组a的shape属性只有一个元素，因此它是一维数组。而数组c的shape属性有两个元素，因此它是二维数组，其中第0轴的长度为3，第1轴的长度为4。还可以通过修改数组的shape属性，在保持数组元素个数不变的情况下，改变数组每个轴的长

度。下面的例子将数组c的shape属性改为(4, 3)，注意从(3, 4)改为(4, 3)并不是对数组进行转置，而只是改变每个轴的大小，数组元素在内存中的位置并没有改变：

```
c.shape = 4, 3
c
array([[ 1,  2,  3],
       [ 4,  4,  5],
       [ 6,  7,  7],
       [ 8,  9, 10]])
```

当设置某个轴的元素个数为-1时，将自动计算此轴的长度。由于数组c有12个元素，因此下面的程序将数组c的shape属性改成了(2, 6)：

```
c.shape = 2, -1
c
array([[ 1,  2,  3,  4,  4,  5],
       [ 6,  7,  7,  8,  9, 10]])
```

使用数组的reshape()方法，可以创建指定形状的新数组，而原数组的形状保持不变：

```
d = a.reshape((2,2)) # 也可以用a.reshape(2,2)

      d          a
-----
[[1, 2], [1, 2, 3, 4]
 [3, 4]]
```

数组a和d其实共享数据存储空间，因此修改其中任意一个数组的元素都会同时修改另一个数组的内容。注意在下面的例子中，数组d中的2也被改成了100：

```
a[1] = 100      # 将数组a的第1个元素改为100

      a          d
-----
```

```
[ 1, 100, 3, 4] [[ 1, 100],  
[ 3, 4]]
```

2.1.2 元素类型

数组的元素类型可以通过dtype属性获得。在前面的例子中，创建数组所用的序列的元素都是整数，因此所创建的数组的元素类型是整型，并且是32位的长整型。这是因为笔者所使用的Python是32位的，如果使用64位的操作系统和Python，那么默认整数类型的长度为64位。

```
c.dtype  
dtype('int32')
```

可以通过dtype参数在创建数组时指定元素类型，注意float类型是64位的双精度浮点类型，而complex是128位的双精度复数类型：

```
ai32 = np.array([1, 2, 3, 4], dtype=np.int32)  
af = np.array([1, 2, 3, 4], dtype=float)  
ac = np.array([1, 2, 3, 4], dtype=complex)  
  
ai32.dtype      af.dtype      ac.dtype  
-----  
dtype('int32') dtype('float64') dtype('complex128')
```

在上面的例子中，传递给dtype参数的都是类型（type）对象，其中float和complex为Python内置的浮点数类型和复数类型，而np.int32是NumPy定义的新的数据类型——32位符号整数类型。

NumPy也有自己的浮点数类型：float16、float32、float64和float128。当使用float64作为dtype参数时，其效果和内置的float类型相同。

在需要指定dtype参数时，也可以传递一个字符串来表示元素的数值类型。NumPy中的每个数值类型都有几种字符串表示方式，字符串和类型之间的对应关系都存储在typeDict字典中。下面的程序获得与float64类型对应的所有键值：

```
[key for key, value in np.typeDict.items() if value is np.float64]
[12, 'd', 'float64', 'float_', 'float', 'f8', 'double', 'Float64']
```

完整的类型列表可以通过下面的语句得到，它将typeDict字典中所有的值转换为一个集合，从而去除其中的重复项：

set(np.typeDict.values())			
{numpy.bool_,	numpy.object_,	numpy.string_,	numpy.unicode_,
numpy.void,	numpy.int8,	numpy.int16,	numpy.int32,
numpy.int32,	numpy.int64,	numpy.uint8,	numpy.uint16,
numpy.uint32,	numpy.uint32,	numpy.uint64,	numpy.float16,
numpy.float32,	numpy.float64,	numpy.float64,	numpy.datetime64,
numpy.timedelta64,	numpy.complex64,	numpy.complex128,	numpy.complex128}

上面显示的数值类型与数组的dtype属性是不同的对象。通过dtype对象的type属性可以获得与其对应的数值类型：

```
c.dtype.type
numpy.int32
```

通过NumPy的数值类型也可以创建数值对象，下面创建一个16位的符号整数对象，它与Python的整数对象不同的是，它的取值范围有限，因此计算200*200会溢出，得到一个负数，这一点与C语言的16位整数的结果相同：

```
a = np.int16(200)
a*a
-25536
```

另外值得注意的是，NumPy的数值对象的运算速度比Python的内置类型的运算速度慢很多，如果程序中需要大量地对单个数值运算，应当尽量避免使用NumPy的数值对象。下面比较了Python内置的float类型与NumPy的双精度浮点数值float64的乘法运算的速度：

```
v1 = 3.14
v2 = np.float64(v1)
%timeit v1*v1
%timeit v2*v2
```

```
10000000 loops, best of 3: 70.1 ns per loop
10000000 loops, best of 3: 178 ns per loop
```

使用`astype()`方法可以对数组的元素类型进行转换，下面将浮点数数组`t1`转换为32位整数数组，将双精度的复数数组`t2`转换成单精度的复数数组：

```
t1 = np.array([1, 2, 3, 4], dtype=np.float)
t2 = np.array([1, 2, 3, 4], dtype=np.complex)
t3 = t1.astype(np.int32)
t4 = t2.astype(np.complex64)
```

2.1.3 自动生成数组

前面的例子都是先创建一个Python的序列对象，然后通过`array()`将其转换为数组，这样做显然效率不高。因此NumPy提供了很多专门用于创建数组的函数。下面的每个函数都有一些关键字参数，具体用法请查看函数说明。

`arange()`类似于内置函数`range()`，通过指定开始值、终值和步长来创建表示等差数列的一维数组，注意所得到的结果中不包含终值。例如下面的程序创建开始值为0、终值为1、步长为0.1的等差数组，注意终值1不在数组中：

```
np.arange(0, 1, 0.1)
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9])
```

`linspace()`通过指定开始值、终值和元素个数来创建表示等差数列的一维数组，可以通过`endpoint`参数指定是否包含终值，默认值为`True`，即包含终值。下面两个例子分别演示了`endpoint`为`True`和`False`时的结果，注意`endpoint`的值会改变数组的等差步长：

```
np.linspace(0, 1, 10) # 步长为 1/9
array([ 0.          ,  0.11111111,  0.22222222,  0.33333333,  0.44444444,
        0.55555556,  0.66666667,  0.77777778,  0.88888889,  1.          ])
```



```
np.linspace(0, 1, 10, endpoint=False) # 步长为1/10
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9])
```

`logspace()` 和 `linspace()` 类似，不过它所创建的数组是等比数列。下面的例子产生从 10^0 到 10^2 、有5个元素的等比数列，注意起始值0表示 10^0 ，而终值2表示 10^2 ：

```
np.logspace(0, 2, 5)
array([ 1.          ,  3.16227766, 10.          , 31.6227766 , 100.          ])
```

基数可以通过 `base` 参数指定，其默认值为10。下面通过将 `base` 参数设置为2，并设置 `endpoint` 参数为 `False`，创建一个比例为 $2^{1/12}$ 的等比数组，此等比数组的比值是音乐中相差半音的两个音阶之间的频率比值，因此可以用它计算一个八度中所有半音的频率：

```
np.logspace(0, 1, 12, base=2, endpoint=False)
array([ 1.          ,  1.05946309,  1.12246205,  1.18920712,  1.25992105,
        1.33483985,  1.41421356,  1.49830708,  1.58740105,  1.68179283,
        1.78179744,  1.88774863])
```

`zeros()`、`ones()`、`empty()` 可以创建指定形状和类型的数组。其中 `empty()` 只分配数组所使用的内存，不对数组元素进行初始化操作，因此它的运行速度是最快的。下面的程序创建一个形状为 (2,3)、元素类型为整数的数组，注意其中的元素值没有被初始化：

```
np.empty((2,3), np.int)
array([[1078523331, 1065353216, 1073741824],
       [1077936128, 1082130432, 1084227584]])
```

而 `zeros()` 将数组元素初始化为0，`ones()` 将数组元素初始化为1。下面创建一个长度为4、元素类型为整数的一维数组，并且元素全部被初始化为0：

```
np.zeros(4, np.int)
array([0, 0, 0, 0])
```

`full()` 将数组元素初始化为指定的值：

```
np.full(4, np.pi)
array([ 3.14159265,  3.14159265,  3.14159265,  3.14159265])
```

此外，`zeros_like()`、`ones_like()`、`empty_like()`、`full_like()`等函数创建与参数数组的形状和类型相同的数组，因此`zeros_like(a)`和`zeros(a.shape, a.dtype)`的效果相同。

`frombuffer()`、`fromstring()`、`fromfile()`等函数可以从字节序列或文件创建数组。下面以`fromstring()`为例介绍它们的用法，先创建含8个字符的字符串s：

```
s = "abcdefgh"
```

Python的字符串实际上是一个字节序列，每个字符占一个字节。因此如果从字符串s创建一个8位的整数数组，所得到的数组正好就是字符串中每个字符的ASCII编码：

```
np.fromstring(s, dtype=np.int8)
array([ 97,  98,  99, 100, 101, 102, 103, 104], dtype=int8)
```

如果从字符串s创建16位的整数数组，那么两个相邻的字节就表示一个整数，把字节98和字节97当作一个16位的整数，它的值就是 $98*256+97 = 25185$ 。可以看出，16位的整数是以低位字节在前(little-endian)的方式保存在内存中的。

```
print 98*256+97
np.fromstring(s, dtype=np.int16)
25185
array([25185, 25699, 26213, 26727], dtype=int16)
```

如果把整个字符串转换为一个64位的双精度浮点数数组，那么它的值是：

```
np.fromstring(s, dtype=np.float)
array([ 8.54088322e+194])
```

显然这个结果没有什么意义，但是如果我们用C语言的二进制方式写了一组double类型的数值到某个文件中，那就可以从此文件读取相应的数据，并通过

`fromstring()` 将其转换为 `float64` 类型的数组，或者直接使用 `fromfile()` 从二进制文件读取数据。

`fromstring()` 会对字符串的字节序列进行复制，而使用 `frombuffer()` 创建的数组与原始字符串共享内存。由于字符串是只读的，因此无法修改所创建的数组的内容：

```
buf = np.frombuffer(s, dtype=np.int16)
buf[1] = 10
-----
---
ValueError                                Traceback (most recent call
last)
  <ipython-input-52-f523db231ae5> in <module>()
      1 buf = np.frombuffer(s, dtype=np.int16)
----> 2 buf[1] = 10

ValueError: assignment destination is read-only
```

Python中还有一些类型也支持buffer接口，例如 `bytearray`、`array.array` 等。在后面的章节中，我们会介绍如何使用这些对象实现动态数组的功能。

还可以先定义一个从下标计算数值的函数，然后用 `fromfunction()` 通过此函数创建数组：

```
def func(i):
    return i % 4 + 1

np.fromfunction(func, (10,))
array([ 1.,  2.,  3.,  4.,  1.,  2.,  3.,  4.,  1.,  2.]
```

`fromfunction()` 的第一个参数是计算每个数组元素的函数，第二个参数指定数组的形状。因为它支持多维数组，所以第二个参数必须是一个序列。上例中第二个参数是长度为1的元组 `(10,)`，因此创建了一个有10个元素的一维数组。

下面的例子创建一个表示九九乘法表的二维数组，输出的数组a中的每个元素a[i, j]都等于func2(i, j)：

```
def func2(i, j):  
    return (i + 1) * (j + 1)  
np.fromfunction(func2, (9,9))  
array([[ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.],  
       [ 2.,  4.,  6.,  8., 10., 12., 14., 16., 18.],  
       [ 3.,  6.,  9., 12., 15., 18., 21., 24., 27.],  
       [ 4.,  8., 12., 16., 20., 24., 28., 32., 36.],  
       [ 5., 10., 15., 20., 25., 30., 35., 40., 45.],  
       [ 6., 12., 18., 24., 30., 36., 42., 48., 54.],  
       [ 7., 14., 21., 28., 35., 42., 49., 56., 63.],  
       [ 8., 16., 24., 32., 40., 48., 56., 64., 72.],  
       [ 9., 18., 27., 36., 45., 54., 63., 72., 81.]])
```

2.1.4 存取元素

可以使用和列表相同的方式对数组的元素进行存取：

```
a = np.arange(10)  
a  
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

- a[5]：用整数作为下标可以获取数组中的某个元素。
- a[3:5]：用切片作为下标获取数组的一部分，包括a[3]但不包括a[5]。
- a[:5]：切片中省略开始下标，表示从a[0]开始。
- a[:-1]：下标可以使用负数，表示从数组最后往前数。

a[5]	a[3:5]	a[:5]	a[:-1]
5	[3, 4]	[0, 1, 2, 3, 4]	[0, 1, 2, 3, 4, 5, 6, 7, 8]

- `a[1:-1:2]`: 切片中的第三个参数表示步长，2表示隔一个元素取一个元素。
- `a[::-1]`: 省略切片的开始下标和结束下标，步长为-1，整个数组头尾颠倒。
- `a[5:1:-2]`: 步长为负数时，开始下标必须大于结束下标。

<code>a[1:-1:2]</code>	<code>a[::-1]</code>	<code>a[5:1:-2]</code>
-----	-----	-----
[1, 3, 5, 7]	[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]	[5, 3]

下标还可以用来修改元素的值：

```
a[2:4] = 100, 101
a
array([ 0,  1, 100, 101,  4,  5,  6,  7,  8,  9])
```

和列表不同的是，通过切片获取的新的数组是原始数组的一个视图。它与原始数组共享同一块数据存储空间。下面的程序将b的第2个元素修改为-10，a的第5个元素也同时被修改为-10，因为它们在内存在中的地址相同。

```
b = a[3:7] # 通过切片产生一个新的数组b，b和a共享同一块数据存储空间
b[2] = -10 # 将b的第2个元素修改为-10

      b                                     a
-----
[101,  4, -10,  6] [ 0,  1, 100, 101,  4, -10,  6,  7,  8,  9]
```

除了使用切片下标存取元素之外，NumPy还提供了整数列表、整数数组和布尔数组等几种高级下标存取方法。

当使用整数列表对数组元素进行存取时，将使用列表中的每个元素作为下标。使用列表作为下标得到的数组不和原始数组共享数据：

```
x = np.arange(10, 1, -1)
x
array([10,  9,  8,  7,  6,  5,  4,  3,  2])
```

- `x[[3, 3, 1, 8]]`: 获取x中的下标为3、3、1、8的4个元素，组成一个新的数组。
- `x[[3, 3, -3, 8]]`: 下标可以是负数，-3表示取倒数第3个元素（从1开始计数）。

```
a = x[[3, 3, 1, 8]]
b = x[[3, 3, -3, 8]]

      a              b
-----
[7, 7, 9, 2]  [7, 7, 4, 2]
```

下面修改`b[2]`的值，但是由于它和`x`不共享内存，因此`x`的值不变：

```
b[2] = 100

      b                                x
-----
[ 7,  7, 100,  2]  [10,  9,  8,  7,  6,  5,  4,  3,  2]
```

整数序列下标也可以用来修改元素的值：

```
x[[3,5,1]] = -1, -2, -3

x
array([10, -3,  8, -1,  6, -2,  4,  3,  2])
```

当使用整数数组作为数组下标时，将得到一个形状和下标数组相同的新数组，新数组的每个元素都是用下标数组中对应位置的值作为下标从原数组获得的值。当下标数组是一维数组时，结果和用列表作为下标的结果相同：

```
x = np.arange(10,1,-1)
x[np.array([3,3,1,8])]
array([7, 7, 9, 2])
```

而当下标是多维数组时，得到的也是多维数组：

```
x[np.array([[3,3,1,8],[3,3,-3,8]])]  
array([[7, 7, 9, 2],  
       [7, 7, 4, 2]])
```

可以将上述操作理解为：先将下标数组展平为一维数组，并作为下标获得一个新的二维数组，然后将其形状修改为下标数组的形状：

```
x[[3,3,1,8,3,3,-3,8]].reshape(2,4) # 改变数组形状  
array([[7, 7, 9, 2],  
       [7, 7, 4, 2]])
```

当使用布尔数组b作为下标存取数组x中的元素时，将获得数组x中与数组b中True对应的元素。使用布尔数组作为下标获得的数组不和原始数组共享数据内存，注意这种方式只对应于布尔数组，不能使用布尔列表。

```
x = np.arange(5,0,-1)  
x  
array([5, 4, 3, 2, 1])
```

布尔数组中下标为0,2的元素为True，因此获取x中下标为0, 2的元素：

```
x[np.array([True, False, True, False, False])]  
array([5, 3])
```

如果是布尔列表，就把True当作1，把False当作0，按照整数序列方式获取x中的元素：

```
x[[True, False, True, False, False]]  
array([4, 5, 4, 5, 5])
```



在NumPy 1.10之后的版本中，布尔列表会被当作布尔数组，因此上面的运行结果会变成array([5, 3])。

布尔数组的长度不够时，不够的部分都当作False：

```
x[np.array([True, False, True, True])]
array([5, 3, 2])
```

布尔数组的下标也可以用来修改元素：

```
x[np.array([True, False, True, True])] = -1, -2, -3
x
array([-1,  4, -2, -3,  1])
```

布尔数组一般不是手工产生，而是使用布尔运算的ufunc函数产生，关于ufunc函数请参照下一节的介绍。下面我们举一个简单的例子说明布尔数组下标的用法：

```
x = np.random.randint(0, 10, 6) # 产生一个长度为6，元素值为0到9的随机整数数组
      x                                x > 5
-----
[8, 1, 5, 6, 2, 7] [ True, False, False,  True, False,  True]
```

表达式`x > 5`将数组`x`中的每个元素和5进行大小比较，得到一个布尔数组，`True`表示`x`中对应的值大于5。我们可以使用`x > 5`所得到的布尔数组收集`x`中所有大于5的数值：

```
x[x > 5]
array([8, 6, 7])
```

2.1.5 多维数组

多维数组的存取和一维数组类似，因为多维数组有多个轴，所以它的下标需要用多个值来表示。NumPy采用元组作为数组的下标，元组中的每个元素和数组的每个轴对应。图2-1显示了一个shape为(6, 6)的数组`a`，图中用不同颜色和线型标出各个下标所对应的选择区域。

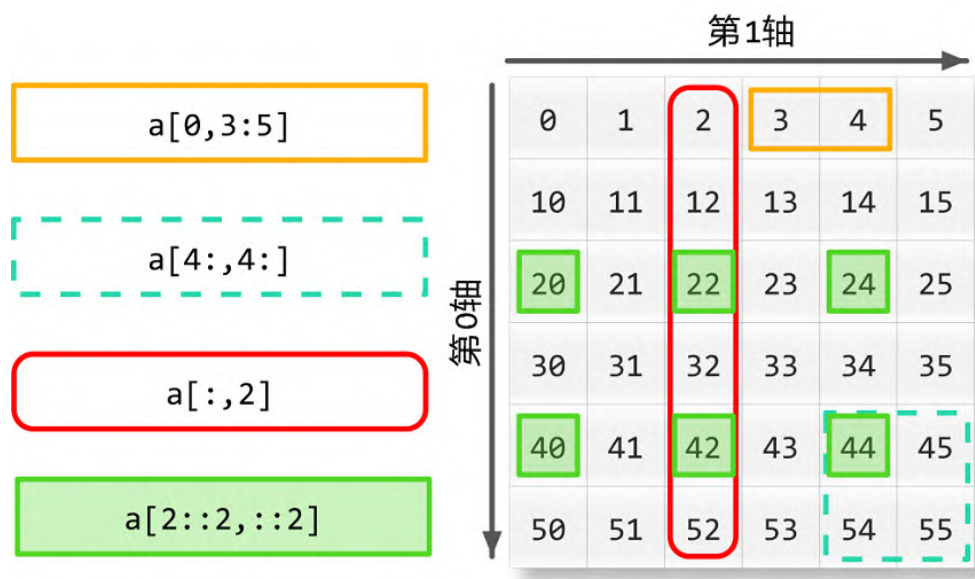


图2-1 使用数组切片语法访问多维数组中的元素

为什么使用元组作为下标

Python的下标语法（用[]存取序列中的元素）本身并不支持多维，但是可以使用任何对象作为下标，因此NumPy使用元组作为下标存取数组中的元素，使用元组可以很方便地表示多个轴的下标。虽然在Python程序中经常用圆括号将元组的元素括起来，但其实元组的语法只需要用逗号隔开元素即可，例如x, y = y, x就是用元组交换变量值的一个例子。因此a[1, 2]和a[(1, 2)]完全相同，都是使用元组(1, 2)作为数组a的下标。

读者也许会对如何创建图中的二维数组感到好奇。它实际上是一个加法表，由纵向量(0, 10, 20, 30, 40, 50)和横向量(0, 1, 2, 3, 4, 5)的元素相加而得。可以用下面的语句创建它，至于其原理，将在后面的章节进行讨论。

```
a = np.arange(0, 60, 10).reshape(-1, 1) + np.arange(0, 6)
a
array([[ 0,  1,  2,  3,  4,  5],
       [10, 11, 12, 13, 14, 15],
       [20, 21, 22, 23, 24, 25],
       [30, 31, 32, 33, 34, 35],
       [40, 41, 42, 43, 44, 45],
       [50, 51, 52, 53, 54, 55]])
```

```
[30, 31, 32, 33, 34, 35],
[40, 41, 42, 43, 44, 45],
[50, 51, 52, 53, 54, 55]])
```

图2-1中的下标都是有二个元素的元组，其中的第0个元素与数组的第0轴(纵轴)对应，而第1个元素与数组的第1轴(横轴)对应。下面是图中各种多维数组切片的运算结果：

a[0, 3:5]	a[4:, 4:]	a[:, 2]	a[2::2, ::2]
-----	-----	-----	-----
[3, 4]	[[44, 45], [54, 55]]	[2, 12, 22, 32, 42, 52]	[[20, 22, 24], [40, 42, 44]]

如果下标元组中只包含整数和切片，那么得到的数组和原始数组共享数据，它是原数组的视图。下面的例子中，数组b是a的视图，它们共享数据，因此修改b[0]时，数组a中对应的元素也被修改：

```
b = a[0, 3:5]
b[0] = -b[0]
a[0, 3:5]
array([-3,  4])
```

因为数组的下标是一个元组，所以我们可以将下标元组保存起来，用同一个元组存取多个数组。在下面的例子中，a[idx]和a[:,2,:]相同，a[idx][idx]和a[:,2,:][:,2,:]相同。

```
idx = slice(None, None, 2), slice(2, None)
a[idx]      a[idx][idx]
-----
[[ 2, -3,  4,  5],  [[ 4,  5],
 [22, 23, 24, 25],   [44, 45]]
 [42, 43, 44, 45]]
```

切片(slice)对象

根据Python的语法，在[]中可以使用以冒号隔开的两个或三个整数表示切片，但是单独生成切片对象时需要使用slice()来创建。它有三个参数，分别为开始值、结束值和间隔步长，当这些值需要省略时可以使用None。例如，a[slice(None, None, None), 2]和a[:, 2]相同。

用Python的内置函数slice()创建下标比较麻烦，因此NumPy提供了一个s_对象来帮助我们创建数组下标，请注意s_实际上是IndexExpression类的一个对象：

```
np.s_[:, 2:]  
(slice(None, None, 2), slice(2, None, None))
```

s_为什么不是函数

根据Python的语法，只有在中括号[]中才能使用以冒号隔开的切片语法，如果s_是函数，那么这些切片必须使用slice()创建。类似的对象还有mgrid和ogrid等，后面我们会学习它们的用法。Python的下标语法实际上会调用__getitem__()方法，因此我们可以很容易自己实现s_对象的功能：

```
class S(object):  
    def __getitem__(self, index):  
        return index
```

在多维数组的下标元组中，也可以使用整数元组或列表、整数数组和布尔数组，如图2-2所示。当下标中使用这些对象时，所获得的数据是原始数据的副本，因此修改结果数组不会改变原始数组。

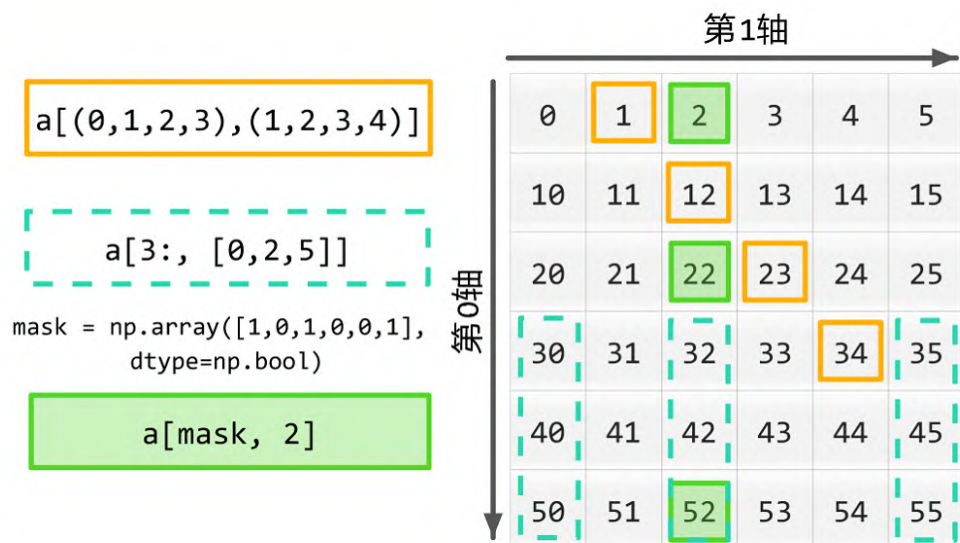


图2-2 使用整数序列和布尔数组访问多维数组中的元素

在`a[(0, 1, 2, 3), (1, 2, 3, 4)]`中，下标仍然是一个有两个元素的元组，元组中的每个元素都是一个整数元组，分别对应数组的第0轴和第1轴。从两个序列的对应位置取出两个整数组成下标，于是得到的结果是：`a[0, 1]`、`a[1, 2]`、`a[2, 3]`、`a[3, 4]`。

```
a[(0, 1, 2, 3), (1, 2, 3, 4)]
array([ 1, 12, 23, 34])
```

在`a[3:, [0, 2, 5]]`中，第0轴的下标是一个切片对象，它选取第3行之后的所有行；第1轴的下标是整数列表，它选取第0、第2和第5列。

```
a[3:, [0, 2, 5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])
```

在`a[mask, 2]`中，第0轴的下标是一个布尔数组，它选取第0、第2和第5行；第1轴的下标是一个整数，它选取第2列。

```
mask = np.array([1, 0, 1, 0, 0, 1], dtype=np.bool)
a[mask, 2]
```

```
array([ 2, 22, 52])
```

注意，如果mask不是布尔数组而是整数数组、列表或元组，就按照以整数数组作为下标的方式进行运算：

```
mask1 = np.array([1,0,1,0,0,1])
mask2 = [True,False,True,False,False,True]
a[mask1, 2]          a[mask2, 2]
-----
[12,  2, 12,  2,  2, 12] [12,  2, 12,  2,  2, 12]
```

当下标的长度小于数组的维数时，剩余的各轴所对应的下标是“:”，即选取它们的所有数据：

```
          a[[1,2],:]          a[[1,2]]
-----
[[10, 11, 12, 13, 14, 15],  [[10, 11, 12, 13, 14, 15],
 [20, 21, 22, 23, 24, 25]]  [20, 21, 22, 23, 24, 25]]
```

当所有轴都用形状相同的整数数组作为下标时，得到的数组和下标数组的形状相同：

```
x = np.array([[0,1],[2,3]])
y = np.array([[-1,-2],[-3,-4]])
a[x,y]
array([[ 5, 14],
       [23, 32]])
```

效果和下面的程序相同：

```
a[(0,1,2,3),(-1,-2,-3,-4)].reshape(2,2)
array([[ 5, 14],
       [23, 32]])
```

当没有指定第1轴的下标时，使用“:”作为下标，因此得到了一个三维数组：

```

a[x]
array([[ 0,  1,  2, -3,  4,  5],
       [10, 11, 12, 13, 14, 15]],

      [[20, 21, 22, 23, 24, 25],
       [30, 31, 32, 33, 34, 35]])

```

可以使用这种以整数数组作为下标的方式快速替换数组中的每个元素，例如有一个表示索引图像的数组`image`，以及一个调色板数组`palette`，则`palette[image]`可以得到通过调色板着色之后的彩色图像：

```

palette = np.array( [ [0,0,0],
                      [255,0,0],
                      [0,255,0],
                      [0,0,255],
                      [255,255,255] ] )

image = np.array( [ [ 0, 1, 2, 0 ],
                   [ 0, 3, 4, 0 ] ] )

palette[image]
array([[ 0,  0,  0],
       [255,  0,  0],
       [  0, 255,  0],
       [  0,  0,  0]],

      [[ 0,  0,  0],
       [  0,  0, 255],
       [255, 255, 255],
       [  0,  0,  0]])

```

2.1.6 结构数组

在C语言中我们可以通过struct关键字定义结构类型，结构中的字段占据连续的内存空间。类型相同的两个结构所占用的内存大小相同，因此可以很容易定义结构数组。和C语言一样，在NumPy中也很容易对这种结构数组进行操作。只要NumPy中的结构定义和C语言中的结构定义相同，就可以很方便地读取C语言的结构数组的二进制数据，将其转换为NumPy的结构数组。

假设我们需要定义一个结构数组，它的每个元素都有name、age和weight字段。在NumPy中可以如下定义：

```
persontype = np.dtype({ ❶
    'names': ['name', 'age', 'weight'],
    'formats': ['S30', 'i', 'f']}, align=True)
a = np.array([("Zhang", 32, 75.5), ("Wang", 24, 65.2)], ❷
    dtype=persontype)
```

❶我们先创建一个dtype对象persontype，它的参数是一个描述结构类型的各个字段的字典。字典有两个键：'names'和'formats'。每个键对应的值都是一个列表。'names'定义结构中每个字段的名称，而'formats'则定义每个字段的类型。这里我们使用类型字符串定义字段类型：

- 'S30'：长度为30个字节的字符串类型，由于结构中的每个元素的大小必须固定，因此需要指定字符串的长度。
- 'i'：32位的整数类型，相当于np.int32。
- 'f'：32位的单精度浮点数类型，相当于np.float32。

❷然后调用array()以创建数组，通过dtype参数指定所创建的数组的元素类型为persontype。下面查看数组a的元素类型：

```
a.dtype
dtype({'names': ['name', 'age', 'weight'], 'formats': ['S30', '<i4', '<f4'],
    'offsets': [0, 32, 36], 'itemsize': 40}, align=True)
```

还可以用包含多个元组的列表来描述结构的类型：

```
dtype([('name', '|S30'), ('age', '<i4'), ('weight', '<f4')])
```

其中形如“(字段名, 类型描述)”的元组描述了结构中的每个字段。类型字符串前面的'|'、'<'、'>'等字符表示字段值的字节顺序:

- |: 忽视字节顺序。
- lt: 低位字节在前, 即小端模式(little endian)。
- >: 高位字节在前, 即大端模式(big endian)。

结构数组的存取方式和一般数组相同, 通过下标能够取得其中的元素, 注意元素的值看上去像是元组, 实际上是结构:

```
print a[0]
a[0].dtype
('Zhang', 32, 75.5)
dtype({'names':['name','age','weight'], 'formats':['S30','<i4','<f4'],
      'offsets':[0,32,36], 'itemsize':40}, align=True)
```

我们可以使用字段名作为下标获取对应的字段值:

```
a[0]["name"]
'Zhang'
```

a[0]是一个结构元素, 它和数组a共享内存数据, 因此可以通过修改它的字段来改变原始数组中对应元素的字段:

```
c = a[1]
c["name"] = "Li"
a[1]["name"]
'Li'
```

我们不但可以获得结构元素的某个字段, 而且可以直接获得结构数组的字段, 返回的是原始数组的视图, 因此可以通过修改b[0]来改变a[0]["age"]:

```
b=a["age"]
b[0] = 40
```



```
print a[0]["age"]  
40
```

通过[a.tostring\(\)](#)或[a.tofile\(\)](#)方法，可以将数组a以二进制的方式转换成字符串或写入文件：

```
a.tofile("test.bin")
```

利用下面的C语言程序可以将test.bin文件中的数据读取出来。%%file为IPython的魔法命令，它将该单元格中的文本保存成文件read_struct_array.c：

```
%%file read_struct_array.c  
#include <stdio.h>  
  
struct person  
{  
    char name[30];  
    int age;  
    float weight;  
};  
  
struct person p[3];  
  
void main ()  
{  
    FILE *fp;  
    int i;  
    fp=fopen("test.bin","rb");  
    fread(p, sizeof(struct person), 2, fp);  
    fclose(fp);  
    for(i=0;i<2;i++)  
    {  
        printf("%s %d %f\n", p[i].name, p[i].age, p[i].weight);  
    }  
}
```

```
}  
  
}
```

在IPython中可以通过!执行系统命令，下面调用gcc编译前面的C语言程序并执行：

```
!gcc read_struct_array.c -o read_struct_array.exe  
!  
!read_struct_array.exe  
Zhang 40 75.500000  
Li 24 65.199997
```

内存对齐

为了内存寻址方便，C语言的结构类型会自动添加一些填充用的字节，这叫做内存对齐。例如上面C语言中定义的结构体的name字段虽然是30个字节长，但是由于内存对齐问题，在name和age中间会填补两个字节。因此，如果数组中所配置的内存大小不符合C语言的对齐规范，将会出现数据错位。为了解决这个问题，在创建dtype对象时，可以传递参数align=True，这样结构数组的内存对齐就和C语言的结构类型一致了。在前面的例子中，由于创建person_type时指定align参数为True，因此它占用40个字节。

结构类型中可以包括其他的结构类型，下面的语句创建一个有一个字段f1的结构，f1的值是另一个结构，它有字段f2，类型为16位整数：

```
np.dtype([('f1', [ ('f2', np.int16)])])  
dtype([('f1', [ ('f2', '<i2')])])
```

当某个字段类型为数组时，用元组的第三个元素表示其形状。在下面的结构体中，f1字段是一个形状为(2, 3)的双精度浮点数组：

```
np.dtype([('f0', 'i4'), ('f1', 'f8', (2, 3))])  
dtype([('f0', '<i4'), ('f1', '<f8', (2, 3))])
```

用下面的字典参数也可以定义结构类型，字典的键为结构的字段名，值为字段的类型描述。但是由于字典的键是没有顺序的，因此字段的顺序需要在类型描述中给出。类型描述是一个元组，它的第二个值给出字段的以字节为单位的偏移量，例如下例中的age字段的偏移量为25个字节：

```
np.dtype({'surname': ('S25',0), 'age': (np.uint8,25)})
dtype([('surname', 'S25'), ('age', 'u1')])
```

2.1.7 内存结构

下面让我们看看数组对象是如何在内存中存储的。如图2-3所示，数组的描述信息保存在一个数据结构中，这个结构引用两个对象：用于保存数据的存储区域和用于描述元素类型的dtype对象。

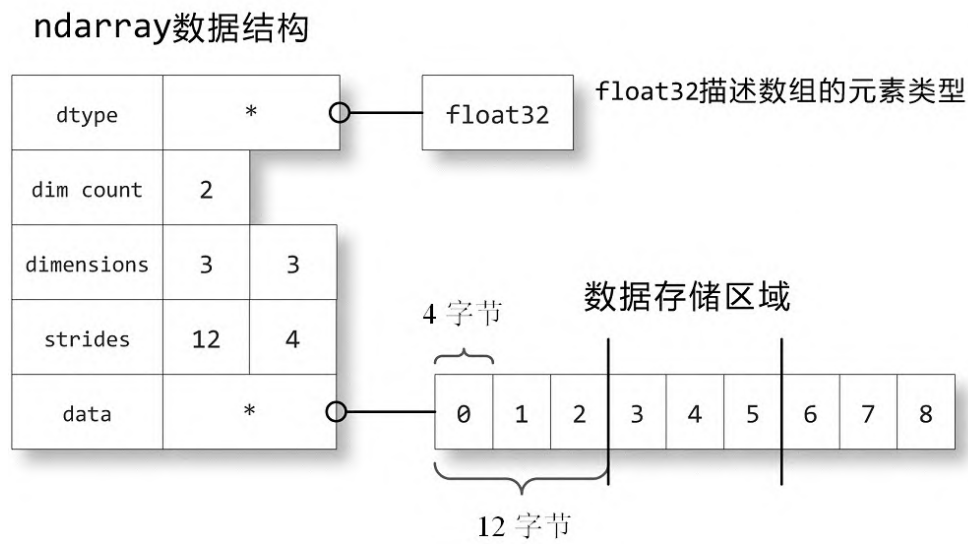


图2-3 ndarray数组对象在内存中的存储方式

数据存储区域保存着数组中所有元素的二进制数据，dtype对象则知道如何将元素的二进制数据转换为可用的值。数组的维数和形状等信息都保存在ndarray数组对象的数据结构中。图2-3中显示的是下面的数组a的内存结构：

```
a = np.array([[0,1,2],[3,4,5],[6,7,8]], dtype=np.float32)
```

数组对象使用strides属性保存每个轴上相邻两个元素的地址差，即当某个轴的下标增加1时，数据存储区中的指针所增加的字节数。例如图2-3中的strides为(12, 4)，即第0轴的下标增加1时，数据的地址增加12个字节。也就是a[1, 0]的地址比a[0, 0]的地址大12，正好是3个单精度浮点数的总字节数。第1轴的下标增加1时，数据的地址增加4个字节，正好是一个单精度浮点数的字节数。

如果strides属性中的数值正好和对应轴所占据的字节数相同，那么数据在内存中是连续存储的。通过切片下标得到的新数组是原始数组的视图，即它和原始数组共享数据存储区域，但是新数组的strides属性会发生变化：

```
b = a[:, ::2, ::2]

      b      b.strides
-----
[[ 0.,  2.], (24, 8)
 [ 6.,  8.]]
```

由于数组b和数组a共享数据存储区，而数组b中的第0轴和第1轴都是从a中隔一个元素取一个，因此数组b的strides变成了(24, 8)，正好都是数组a的两倍。对照前面的图2-3很容易看出数据0和2的地址相差8个字节，而数据0和6的地址相差24个字节。

元素在数据存储区中的排列格式有两种：C语言格式和Fortran语言格式。在C语言中，多维数组的第0轴是最上位的，即第0轴的下标增加1时，元素的地址增加的字节数最多；而Fortran语言中的多维数组的第0轴是最下位的，即第0轴的下标增加1时，地址只增加一个元素的字节数。在NumPy中默认以C语言格式存储数据，如果希望改为Fortran格式，只需要在创建数组时，设置order参数为“F”：

```
c = np.array([[0,1,2],[3,4,5],[6,7,8]], dtype=np.float32, order="F")
c.strides
(4, 12)
```

了解了数组的内存结构，就可以解释使用下标取得数据时的复制和引用问题：

- 当下标使用整数和切片时，所取得的数据在数据存储区域中是等间隔分布的。因为只需要修改图2-3所示的数据结构中的dim count、dimensions、stride等属性以及指向数据存储区域的指针data，就能实现整数和切片下标，所以新数组和原始数组能够共享数据存储区域。
- 当使用整数序列、整数数组和布尔数组时，不能保证所取得的数据在数据存储区域中是等间隔的，因此无法和原始数组共享数据，只能对数据进行复制。

数组的flags属性描述了数据存储区域的一些属性，直接查看flags属性将输出各个标志的值，也可以单独获得其中的某个标志值：

```
print a.flags
print "c_contiguous:", a.flags.c_contiguous

C_CONTIGUOUS : True
F_CONTIGUOUS : False
OWNDATA : True
WRITEABLE : True
ALIGNED : True
UPDATEIFCOPY : False
c_contiguous: True
```

下面是几个比较重要的标志：

- C_CONTIGUOUS：数据存储区域是否是C语言格式连续区域。
- F_CONTIGUOUS：数据存储区域是否是Fortran语言格式连续区域。
- OWNDATA：数组是否拥有此数据存储区域，当一个数组是其他数组的视图时，它不拥有数据存储区域。

由于数组a是通过array()直接创建的，因此它的数据存储区域是C语言格式连续区域，并且它拥有数据存储区域。下面我们看看数组a的转置标志，数组的转置可以通过其T属性获得，转置数组将其数据存储区域看作Fortran语言格式连续区域，并且它不拥有数据存储区域。

```

a.T.flags
C_CONTIGUOUS : False
F_CONTIGUOUS : True
OWNDATA : False
WRITEABLE : True
ALIGNED : True
UPDATEIFCOPY : False

```

下面查看数组b的标志，它不拥有数据存储区域，其数据也不是连续存储的。通过视图数组的base属性可以获得保存数据的原始数组：

```

b.flags
C_CONTIGUOUS : False
F_CONTIGUOUS : False
OWNDATA : False
WRITEABLE : True
ALIGNED : True
UPDATEIFCOPY : False

id(b.base)    id(a)
-----
119627272    119627272

```

我们还可以通过view()方法从同一块数据区创建不同的dtype的数组对象，也就是使用不同的数值类型查看同一段内存中的二进制数据：

```

a = np.array([[0, 1], [2, 3], [4, 5]], dtype=np.float32)
b = a.view(np.uint32)
c = a.view(np.uint8)

```

b	c
-----	-----
[[0, 1065353216],	[[0, 0, 0, 0, 0, 0, 128, 63],
[1073741824, 1077936128],	[0, 0, 0, 64, 0, 0, 64, 64],
[1082130432, 1084227584]]	[0, 0, 128, 64, 0, 0, 160, 64]]

由于数组a的元素类型是单精度浮点数，占用4个字节，通过a.view(np.uint32)，我们创建了一个新的数组，它和数组a使用同一段数据内存，

但是它将每4个字节的数据当作无符号32位整数处理。而`a.view(np.uint8)`将每个字节都当作一个单字节的无符号整数，因此得到一个形状为(3, 8)的数组。通过`view()`方法获得的新数组与原数组共享内存，当`a[0, 0]`被修改时，`b[0, 0]`和`c[0, :4]`都会改变：

```
a[0, 0] = 3.14
b[0, 0]          c[0, :4]
-----
1078523331  [195, 245, 72, 64]
```

下面我们看一个使用`view()`方法的有趣的例子。在《雷神之锤III：竞技场》的C语言源代码中有这样一个神奇的计算平方根倒数的函数`Q_rsqrt()`。代码中使用牛顿迭代法计算平方根倒数，这并没有任何神奇之处，但是其中包含了一个神奇的数字`0x5f3759df`，并将单精度浮点数当作32位的整数进行了一次令人毫无头绪的运算：



<http://zh.wikipedia.org/wiki/平方根倒数速算法>

维基百科关于《雷神之锤》中使用`0x5f3759df`计算平方根倒数算法的解释。

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                      // 对浮点数的邪恶位级
hack
    i  = 0x5f3759df - ( i >> 1 );              // 这到底是怎么回事？
```

```

    y = * ( float * ) &i;

    y = y * ( threehalfs - ( x2 * y * y ));           // 第一次牛顿迭代

    return y;
}

```

下面我们用NumPy实现同样的计算：

```

number = np.linspace(0.1, 10, 100)
y = number.astype(np.float32) ❶
x2 = y * 0.5
i = y.view(np.int32) ❷
i[:] = 0x5f3759df - (i >> 1) ❸
y = y * (1.5 - x2 * y * y) ❹
np.max(np.abs(1 / np.sqrt(number) - y)) ❺
0.0050456140410597428

```

❶由于linspace()创建的数组的类型为双精度浮点数，因此这里首先通过astype()方法将其转换成单精度浮点数数组y。❷通过view()方法创建一个与y共享内存的32位整数数组i。❸对整数数组i进行那段完全摸不着头脑的运算，并且将结果重新写入数组i中。由于i和y共享内存，此时y中的值也发生了变化。注意这里的赋值不能使用i = 0x5f3759df - (i >> 1)，如果这样写，那么数组i就是一个全新的数组了。❹进行一次牛顿迭代运算，这里由于使用y = ...的写法，因此y将变成一个全新的数组，和原来的i不再共享内存。在这段代码中有很多数组运算，关于这方面的内容将在下一节进行详细说明。❺最后输出真实值和近似值之间的最大误差。图2-4显示了绝对误差与自变量的关系，当number很小时绝对误差较大，但此时的函数值也较大，因此相对误差的变化并不大。

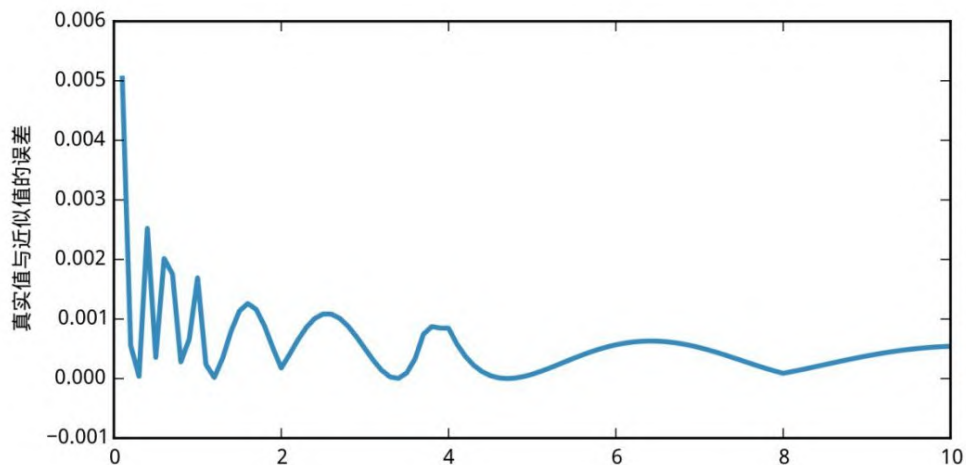


图2-4 《雷神之锤》中计算平方根倒数算法的绝对误差

除了使用切片从同一块数据区创建不同的shape和strides的数组对象之外，还可以直接设置这些属性，从而得到用切片实现不了的效果，例如：

```
from numpy.lib.stride_tricks import as_strided
a = np.arange(6)
b = as_strided(a, shape=(4, 3), strides=(4, 4))

      a              b
-----
[0, 1, 2, 3, 4, 5]  [[0, 1, 2],
[1, 2, 3],
[2, 3, 4],
[3, 4, 5]]
```

这个例子中，我们从NumPy的辅助模块中载入了as_strided()函数，并使用它从一个长度为6的一维数组a创建了一个shape为(4, 3)的二维数组b。由于通过strides参数直接指定了数组b的strides属性，因此不仅数组b和数组a共享数据区，而且b中的前后两行有两个元素是重合的。例如下面修改a[2]的值，b中的前三行中对应的元素也发生改变：

```
a[2] = 20
b
array([[ 0,  1, 20],
```

```
[ 1, 20,  3],  
[20,  3,  4],  
[ 3,  4,  5]])
```

在对数据进行处理时，可能经常需要对数据进行分块处理，而且为了保持平滑，每块数据之间需要有一定的重叠部分。这时可以使用上面介绍的方法对数据进行带重叠的分块。需要注意的是，使用`as_strided()`时NumPy不会进行内存越界检查，因此`shape`和`strides`设置不当可能会发生意想不到的错误。

2.2 ufunc函数



与本节内容对应的Notebook为：[02-numpy/numpy-200-ufunc.ipynb](#)。

ufunc是universal function的缩写，它是一种能对数组的每个元素进行运算的函数。NumPy内置的许多ufunc函数都是用C语言实现的，因此它们的计算速度非常快。让我们先看一个例子：

```
x = np.linspace(0, 2*np.pi, 10)  
y = np.sin(x)  
y  
array([ 0.00000000e+00,  6.42787610e-01,  9.84807753e-01,  
 8.66025404e-01,  3.42020143e-01, -3.42020143e-01,  
-8.66025404e-01, -9.84807753e-01, -6.42787610e-01,  
-2.44929360e-16])
```

先用`linspace()`产生一个从0到 2π 的等差数组，然后将其传递给`np.sin()`函数计算每个元素的正弦值。由于`np.sin()`是一个ufunc函数，因此在其内部对数组`x`的每个元素进行循环，分别计算它们的正弦值，并返回一个保存各个计算结果的数组。运算之后数组`x`中的值并没有改变，而是新创建了一个数组来保存结果。也可以

通过out参数指定保存计算结果的数组。因此如果希望直接在数组x中保存结果，可以将它传递给out参数：

```
t = np.sin(x, out=x)

t is x

True
```

ufunc函数的返回值仍然是计算的结果，只不过它就是数组x。下面比较np.sin()和Python标准库的math.sin()的计算速度：

```
import math

x = [i * 0.001 for i in xrange(1000000)]

def sin_math(x):
    for i, t in enumerate(x):
        x[i] = math.sin(t)

def sin_numpy(x):
    np.sin(x, x)

def sin_numpy_loop(x):
    for i, t in enumerate(x):
        x[i] = np.sin(t)

x1 = x[:]
%time sin_math(x)

xa = np.array(x)
%time sin_numpy(xa)

x1 = x[:]
%time sin_numpy_loop(x)
```

```
Wall time: 302 ms
Wall time: 30 ms
Wall time: 1.28 s
```

可以看出，`np.sin()` 比 `math.sin()` 快10倍多，这得益于 `np.sin()` 在C语言级别的循环计算。

列表推导式比循环更快

事实上，标准Python中有比for循环更快的方案：使用列表推导式 `x = [math.sin(t) for t in x]`。但是列表推导式将产生一个新的列表，而不是直接修改原列表中的元素。

`np.sin()` 同样也支持计算单个数值的正弦值。不过值得注意的是，对单个数值的计算，`math.sin()` 则比 `np.sin()` 快很多。在Python级别进行循环时，`np.sin()` 的计算速度只有 `math.sin()` 的1/6。这是因为：`np.sin()` 为了同时支持数组和单个数值的计算，其C语言的内部实现要比 `math.sin()` 复杂很多。此外，对于单个数值的计算，`np.sin()` 的返回值类型和 `math.sin()` 的不同，`math.sin()` 返回的是Python的标准float类型，而 `np.sin()` 返回float64类型：

```
type(math.sin(0.5))    type(np.sin(0.5))
-----
float                  numpy.float64
```

通过下标运算获取的数组元素的类型为NumPy中定义的类型，将其转换为Python的标准类型还需要花费额外的时间。为了解决这个问题，数组提供了 `item()` 方法，它用来获取数组中的单个元素，并且直接返回标准的Python数值类型：

```
a = np.arange(6.0).reshape(2, 3)
print a.item(1, 2), type(a.item(1, 2)), type(a[1, 2])
5.0 <type 'float'><type 'numpy.float64'>
```

通过上面的例子我们了解了如何最有效率地使用math模块和NumPy中的数学函数。由于它们各有优缺点，因此在导入时不建议使用import *全部载入，而是应该使用import numpy as np载入，这样可以根据需要选择合适的函数。

2.2.1 四则运算

NumPy提供了许多ufunc函数，例如计算两个数组之和的add()函数：

```
a = np.arange(0, 4)
b = np.arange(1, 5)
np.add(a, b)
array([1, 3, 5, 7])
```

add()返回一个数组，它的每个元素都是两个参数数组的对应元素之和。如果没有指定out参数，那么它将创建一个新的数组来保存计算结果。如果指定了第三个参数out，则不产生新的数组，而直接将结果保存进指定的数组。

```
np.add(a, b, a)
a
array([1, 3, 5, 7])
```

NumPy为数组定义了各种数学运算操作符，因此计算两个数组相加可以简单地写为a + b，而np.add(a, b, a)则可以用a += b来表示。表2-1列出了数组的运算符以及与之对应的ufunc函数，注意除号的意义根据是否激活__future__.division有所不同。

表2-1 数组的运算符以及对应的ufunc函数

表达式	对应的ufunc函数
y = x1 + x2	add(x1, x2 [, y])
y = x1 - x2	subtract(x1, x2 [, y])

<code>y = x1 * x2</code>	<code>multiply(x1, x2 [, y])</code>
<code>y = x1 / x2</code>	<code>divide(x1, x2 [, y])</code> , 如果两个数组的元素为整数, 那么用整数除法
<code>y = x1 / x2</code>	<code>true_divide(x1, x2 [, y])</code> , 总是返回精确的商
<code>y = x1 // x2</code>	<code>floor_divide(x1, x2 [, y])</code> , 总是对返回值取整
<code>y = -x</code>	<code>negative(x [, y])</code>
<code>y = x1**x2</code>	<code>power(x1, x2 [, y])</code>
<code>y = x1 % x2</code>	<code>remainder(x1, x2 [, y])</code> , <code>mod(x1, x2, [, y])</code>

数组对象支持操作符, 极大地简化了算式的编写, 不过要注意如果算式很复杂, 并且要运算的数组很大, 将会因为产生大量的中间结果而降低程序的运算速度。例如, 假设对a、b、c三个数组采用算式 $x=a*b+c$ 加以计算, 那么它相当于:

```
t = a * b
x = t + c
del t
```

也就是说, 需要产生一个临时数组t来保存乘法的运算结果, 然后再产生最后的结果数组x。可以将算式分解为下面的两行语句, 以减少一次内存分配:

```
x = a*b
x += c
```

2.2.2 比较运算和布尔运算

使用`==`、`>`等比较运算符对两个数组进行比较, 将返回一个布尔数组, 它的每个元素值都是两个数组对应元素的比较结果。例如:

```
np.array([1, 2, 3]) < np.array([3, 2, 1])
array([ True, False, False], dtype=bool)
```

每个比较运算符也与一个ufunc函数对应，表2-2是比较运算符与ufunc函数的对照表。

表2-2 比较运算符与相应的ufunc函数

表达式	对应的ufunc函数
y = x1 == x2	equal(x1, x2 [, y])
y = x1 != x2	not_equal(x1, x2 [, y])
y = x1 < x2	less(x1, x2, [, y])
y = x1 <= x2	less_equal(x1, x2, [, y])
y = x1 > x2	greater(x1, x2, [, y])
y = x1 >= x2	greater_equal(x1, x2, [, y])

由于Python中的布尔运算使用and、or和not等关键字，它们无法被重载，因此数组的布尔运算只能通过相应的ufunc函数进行。这些函数名都以logical_开头，在IPython中使用自动补全可以很容易地找到它们：

```
>>> np.logical # 按Tab键进行自动补全
np.logical_and np.logical_not np.logical_or np.logical_xor
```

下面是一个使用logical_or()进行“或运算”的例子：

```
a = np.arange(5)
b = np.arange(4, -1, -1)
print a == b
print a > b
print np.logical_or(a == b, a > b) # 和 a>=b 相同
[False False  True False False]
[False False False  True  True]
[False False  True  True  True]
```

对两个布尔数组使用and、or和not等进行布尔运算，将抛出ValueError异常。因为布尔数组中有True也有False，所以NumPy无法确定用户的运算目的：

```
a == b and a > b
-----
-----
ValueError                                Traceback (most recent call
last)
<ipython-input-13-99b8118687f0> in <module>()
----> 1 a == b and a > b

ValueError: The truth value of an array with more than one element is
ambiguous. Use a.any() or a.all()
```

错误信息告诉我们可以使用数组的any()或all()方法，在NumPy中同时也定义了any()和all()函数，它们的用法和Python内置的any()和all()类似。只要数组中有一个元素值为True，any()就返回True；而只有当数组的全部元素都为True时，all()才返回True。

```
np.any(a == b)    np.any(a == b) and np.any(a > b)
-----
True              True
```

以bitwise_开头的函数是位运算函数，包括bitwise_and、bitwise_not、bitwise_or和bitwise_xor等。也可以使用&、~、|和^等操作符进行计算。

对于布尔数组来说，位运算和布尔运算的结果相同。但在使用时要注意，位运算符的优先级比较运算符高，因此需要使用括号提高比较运算符的运算优先级。例如：

```
(a == b) | (a > b)
array([False, False,  True,  True,  True], dtype=bool)
```


整数数组的位运算和C语言的位运算相同，在使用时要注意元素类型的符号，例如下面的`arange()`所创建的数组的元素类型为32位符号整数，因此对正数按位取反将得到负数。以整数0为例，按位取反的结果是0xFFFFFFFF，在32位符号整数中，这个值表示-1。

```
~ np.arange(5)
array([-1, -2, -3, -4, -5])
```

而如果对8位无符号整数数组进行按位取反运算：

```
~ np.arange(5, dtype=np.uint8)
array([255, 254, 253, 252, 251], dtype=uint8)
```

同样的整数0，按位取反的结果是0xFF，当它是8位无符号整数时，它的值是255。

2.2.3 自定义ufunc函数

通过NumPy提供的标准ufunc函数，可以组合出复杂的表达式，在C语言级别对数组的每个元素进行计算。但有时这种表达式不易编写，而对每个元素进行计算的程序却很容易用Python实现，这时可以用`frompyfunc()`将计算单个元素的函数转换成ufunc函数，这样就可以方便地用所产生的ufunc函数对数组进行计算了。

例如，我们可以用一个分段函数描述三角波，三角波的形状如图2-5所示，它分为三段：上升段、下降段和平坦段。

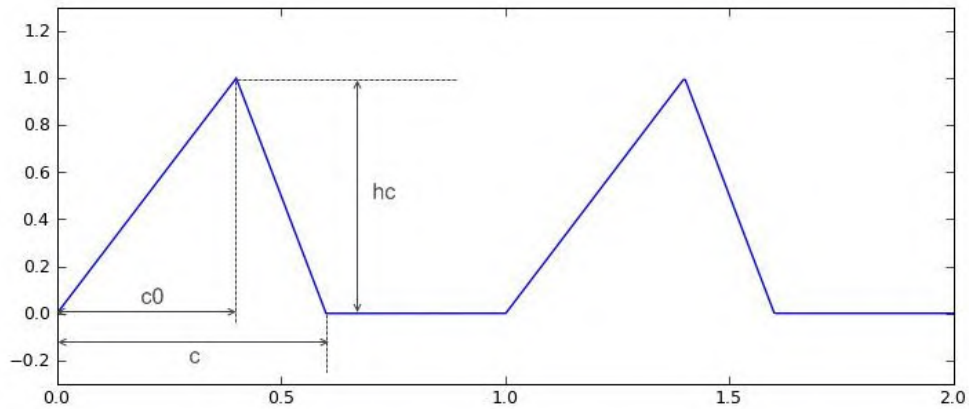


图2-5 三角波可以用分段函数进行计算

根据图2-5，我们很容易写出计算三角波上某点的Y坐标的函数。显然 `triangle_wave()` 只能计算单个数值，不能对数组直接进行处理。

```
def triangle_wave(x, c, c0, hc):
    x = x - int(x) # 三角波的周期为1，因此只取x坐标的小数部分进行计算
    if x >= c: r = 0.0
    elif x < c0: r = x / c0 * hc
    else: r = (c-x) / (c-c0) * hc
    return r
```

我们可以用下面的程序，先使用列表推导式计算出一个列表，然后用 `array()` 将列表转换为数组。这种做法每次都需要使用列表推导式语法调用函数，这对于多维数组很麻烦。

```
x = np.linspace(0, 2, 1000)
y1 = np.array([triangle_wave(t, 0.6, 0.4, 1.0) for t in x])
```

通过 `frompyfunc()` 可以将计算单个值的函数转换为能对数组的每个元素进行计算的ufunc函数。 `frompyfunc()` 的调用格式为：

```
frompyfunc(func, nin, nout)
```

其中：`func`是计算单个元素的函数，`nin`是`func`的输入参数的个数，`nout`是`func`的返回值的个数。下面的程序使用 `frompyfunc()` 将 `triangle_wave()` 转换为ufunc函

数对象triangle_ufunc1:

```
triangle_ufunc1 = np.frompyfunc(triangle_wave, 4, 1)
y2 = triangle_ufunc1(x, 0.6, 0.4, 1.0)
```

值得注意的是，triangle_ufunc1()所返回的数组的元素类型是object，因此还需要调用数组的astype()方法，以将其转换为双精度浮点数组：

```
y2.dtype    y2.astype(np.float).dtype
-----
dtype('O')  dtype('float64')
```

使用vectorize()也可以实现和frompyfunc()类似的功能，但它可以通过otypes参数指定返回的数组的元素类型。otypes参数可以是一个表示元素类型的字符串，也可以是一个类型列表，使用列表可以描述多个返回数组的元素类型。下面的程序使用vectorize()计算三角波：

```
triangle_ufunc2 = np.vectorize(triangle_wave, otypes=[np.float])
y3 = triangle_ufunc2(x, 0.6, 0.4, 1.0)
```

最后我们验证一下结果：

```
np.all(y1 == y2)  np.all(y2 == y3)
-----
True              True
```

2.2.4 广播

当使用ufunc函数对两个数组进行计算时，ufunc函数会对这两个数组的对应元素进行计算，因此它要求这两个数组的形状相同。如果形状不同，会进行如下广播(broadcasting)处理：

1) 让所有输入数组都向其中维数最多的数组看齐，shape属性中不足的部分都通过在前面加1补齐。

2) 输出数组的shape属性是输入数组的shape属性的各个轴上的最大值。

3) 如果输入数组的某个轴的长度为1或与输出数组的对应轴的长度相同，这个数组能够用来计算，否则出错。

4) 当输入数组的某个轴的长度为1时，沿着此轴运算时都用此轴上的第一组值。

上述4条规则理解起来可能比较费劲，下面让我们看一个实际的例子。

先创建一个二维数组a，其形状为(6, 1)：

```
a = np.arange(0, 60, 10).reshape(-1, 1)
a      a.shape
-----
[[ 0], (6, 1)
 [10],
 [20],
 [30],
 [40],
 [50]]
```

再创建一维数组b，其形状为(5,)：

```
b = np.arange(0, 5)
b      b.shape
-----
[0, 1, 2, 3, 4] (5,)
```

计算a与b的和，得到一个加法表，它相当于计算两个数组中所有元素对的和，得到一个形状为(6, 5)的数组：

```
c = a + b
c      c.shape
-----
```

```
[[ 0,  1,  2,  3,  4], (6, 5)
 [10, 11, 12, 13, 14],
 [20, 21, 22, 23, 24],
 [30, 31, 32, 33, 34],
 [40, 41, 42, 43, 44],
 [50, 51, 52, 53, 54]]
```

由于a和b的维数不同，根据规则1)，需要让b的shape属性向a对齐，于是在b的shape属性前加1，补齐为(1, 5)。相当于做了如下计算：

```
b.shape = 1, 5
      b      b.shape
-----
[[0, 1, 2, 3, 4]] (1, 5)
```

这样，加法运算的两个输入数组的shape属性分别为(6, 1)和(1, 5)，根据规则2)，输出数组的各个轴的长度为输入数组各个轴的长度的最大值，可知输出数组的shape属性为(6, 5)。

由于b的第0轴的长度为1，而a的第0轴的长度为6，为了让它们在第0轴上能够相加，需要将b的第0轴的长度扩展为6，这相当于：

```
b = b.repeat(6, axis=0)
      b      b.shape
-----
[[0, 1, 2, 3, 4], (6, 5)
 [0, 1, 2, 3, 4],
 [0, 1, 2, 3, 4],
 [0, 1, 2, 3, 4],
 [0, 1, 2, 3, 4],
 [0, 1, 2, 3, 4]]
```

这里的repeat()方法沿着axis参数指定的轴复制数组中各个元素的值。由于a的第1轴的长度为1，而b的第1轴的长度为5，为了让它们在第1轴上能够相加，需要将a

的第1轴的长度扩展为5，这相当于：

```
a = a.repeat(5, axis=1)

      a              a.shape
-----
[[ 0,  0,  0,  0,  0], (6, 5)
 [10, 10, 10, 10, 10],
 [20, 20, 20, 20, 20],
 [30, 30, 30, 30, 30],
 [40, 40, 40, 40, 40],
 [50, 50, 50, 50, 50]]
```

经过上述处理之后，a和b就可以按对应元素进行相加运算了。当然，在执行a + b运算时，NumPy内部并不会真正将长度为1的轴用repeat()进行扩展，这样太浪费内存空间了。由于这种广播计算很常用，因此NumPy提供了ogrid对象，用于创建广播运算用的数组。

```
x, y = np.ogrid[:5, :5]

      x              y
-----
[[0],  [[0, 1, 2, 3, 4]]
 [1],
 [2],
 [3],
 [4]]
```

此外，NumPy还提供了mgrid对象，它的用法和ogrid对象类似，但是它所返回的是进行广播之后的数组：

```
x, y = np.mgrid[:5, :5]

      x              y
-----
[[0, 0, 0, 0, 0],  [[0, 1, 2, 3, 4],
 [1, 1, 1, 1, 1],  [0, 1, 2, 3, 4],
```

```
[2, 2, 2, 2, 2],    [0, 1, 2, 3, 4],
[3, 3, 3, 3, 3],    [0, 1, 2, 3, 4],
[4, 4, 4, 4, 4]]    [0, 1, 2, 3, 4]]
```

ogrid是一个很有趣的对象，它像多维数组一样，用切片元组作为下标，返回的是一组可以用来广播计算的数组。其切片下标有两种形式：

- 开始值:结束值:步长，和np.arange(开始值,结束值,步长)类似。
- 开始值:结束值:长度j，当第三个参数为虚数时，它表示所返回的数组的长度，和np.linspace(开始值,结束值,长度)类似。

```
x, y = np.ogrid[:1:4j, :1:3j]

      x                      y
-----
[[ 0.          ], [[ 0. ,  0.5,  1. ]]
```

```
[ 0.33333333],
[ 0.66666667],
[ 1.          ]]
```

利用ogrid的返回值，我们很容易计算二元函数在等间距网格上的值。下面是绘制三维曲面 $(x, y) = xe^{x^2 - y^2}$ 的程序：

```
x, y = np.ogrid[-2:2:20j, -2:2:20j]
z = x * np.exp( - x**2 - y**2)
```

图2-6为使用ogrid计算的三维曲面。

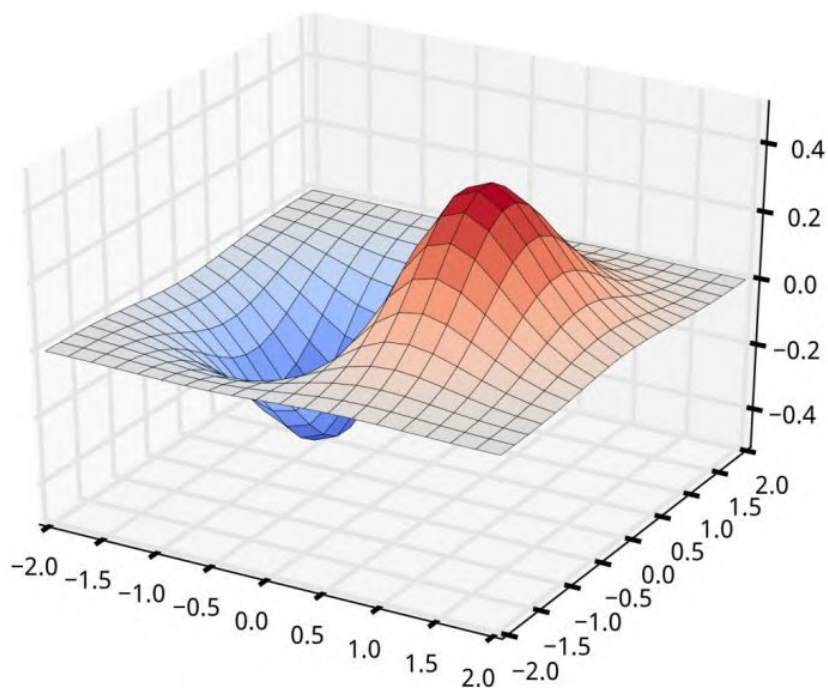


图2-6 使用ogrid计算二元函数的曲面

为了充分利用ufunc函数的广播功能，我们经常需要调整数组的形状，因此数组支持特殊的下标对象None，它表示在None对应的位置创建一个长度为1的新轴，例如对于一维数组a，a[None, :] 和 a.reshape(1, -1) 等效，而 a[:, None] 和 a.reshape(-1, 1) 等效：

```
a = np.arange(4)
a[None, :]    a[:, None]
-----
[[0, 1, 2, 3]] [[0],
[1],
[2],
[3]]
```

下面的例子利用None作为下标，实现广播运算：

```
x = np.array([0, 1, 4, 10])
y = np.array([2, 3, 8])
```



```
x[None, :] + y[:, None]
array([[ 2,  3,  6, 12],
       [ 3,  4,  7, 13],
       [ 8,  9, 12, 18]])
```

还可以使用`ix_()`将两个一维数组转换成可广播的二维数组：

```
gy, gx = np.ix_(y, x)

      gx          gy          gx + gy
-----
[[ 0,  1,  4, 10]] [[2],  [[ 2,  3,  6, 12],
[3],  [ 3,  4,  7, 13],
[8]] [ 8,  9, 12, 18]]
```

在上面的例子中，通过`ix_()`将数组`x`和`y`转换成能进行广播运算的二维数组。注意数组`y`对应广播运算结果中的第0轴，而数组`x`与第1轴对应。`ix_()`的参数可以是N个一维数组，它将这些数组转换成N维空间中可广播的N维数组。

2.2.5 ufunc的方法

ufunc函数对象本身还有一些方法函数，这些方法只对两个输入、一个输出的ufunc函数有效，其他的ufunc对象调用这些方法时会抛出`ValueError`异常。

`reduce()`方法和Python的`reduce()`函数类似，它沿着`axis`参数指定的轴对数组进行操作，相当于将`<op>`运算符插入到沿`axis`轴的所有元素之间：`<op>.reduce(array, axis=0, dtype=None)`。例如：

```
r1 = np.add.reduce([1, 2, 3]) # 1 + 2 + 3
r2 = np.add.reduce([[1, 2, 3], [4, 5, 6]], axis=1) # (1+2+3), (4+5+6)
r1      r2
--  -----
6      [ 6, 15]
```

`accumulate()` 方法和`reduce()` 类似，只是它返回的数组和输入数组的形状相同，保存所有的中间计算结果：

```
a1 = np.add.accumulate([1, 2, 3])
a2 = np.add.accumulate([1, 2, 3], [4, 5, 6], axis=1)

a1          a2
-----
[1, 3, 6]   [[ 1,  3,  6],
              [ 4,  9, 15]]
```

`reduceat()` 方法计算多组`reduce()` 的结果，通过`indices` 参数指定一系列的起始和终止位置。它的计算有些特别，让我们通过例子详细解释一下：

```
a = np.array([1, 2, 3, 4])
result = np.add.reduceat(a, indices=[0, 1, 0, 2, 0, 3, 0])
result
array([ 1,  2,  3,  3,  6,  4, 10])
```

对于`indices` 参数中的每个元素都会计算出一个值，因此最终的计算结果和`indices` 参数的长度相同。结果数组`result` 中除最后一个元素之外，都按照如下计算得出：

```
if indices[i] < indices[i+1]:
    result[i] = <op>.reduce(a[indices[i]:indices[i+1]])
else:
    result[i] = a[indices[i]]
```

而最后一个元素如下计算：

```
<op>.reduce(a[indices[-1]:])
```

因此上面的例子中，数组`result` 的每个元素按照如下计算得出：

```
1 : a[0] -> 1
2 : a[1] -> 2
```

```

3 : a[0] + a[1] -> 1 + 2
3 : a[2] -> 3
6 : a[0] + a[1] + a[2] -> 1 + 2 + 3 = 6
4 : a[3] -> 4
10: a[0] + a[1] + a[2] + a[4] -> 1 + 2 + 3 + 4 = 10

```

可以看出`result[:,2]`和`a`相等，而`result[1::2]`和`np.add.accumulate(a)`相等。

ufunc函数对象的`outer()`方法等同于如下程序：

```

a.shape += (1,)*b.ndim
<op>(a,b)
a = a.squeeze()

```

其中`squeeze()`方法剔除数组`a`中长度为1的轴。让我们看一个例子：

```

np.multiply.outer([1, 2, 3, 4, 5], [2, 3, 4])
array([[ 2,  3,  4],
       [ 4,  6,  8],
       [ 6,  9, 12],
       [ 8, 12, 16],
       [10, 15, 20]])

```

可以看出通过`outer()`计算的结果是如下乘法表：

```

*| 2  3  4
-----
1| 2  3  4
2| 4  6  8
3| 6  9 12
4| 8 12 16
5|10 15 20

```

如果将这两个数组按照等同程序一步一步地进行计算，就会发现乘法表最终是通过广播的方式计算出来的。

2.3 多维数组的下标存取



与本节内容对应的 Notebook 为：[02-numpy/numpy-300-multindex.ipynb](#)。

在前面的介绍中，我们通过一些实例介绍了如何对多维数组进行下标访问。实际上，NumPy提供的下标功能十分强大，在读者掌握了“广播”相关的知识之后，让我们再回过头来系统地学习数组的下标规则。

2.3.1 下标对象

首先，多维数组的下标应该是一个长度和数组的维数相同的元组。如果下标元组的长度比数组的维数大，就会出错；如果小，就会在下标元组的后面补“:”，使得它的长度与数组维数相同。

如果下标对象不是元组，则NumPy会首先把它转换为元组。这种转换可能会和用户所希望的不一致，因此为了避免出现问题，请“显式”地使用元组作为下标。例如数组a是一个三维数组，下面使用一个二维列表lidx和二维数组aidx作为下标，得到的结果就不一样。

```
a = np.arange(3 * 4 * 5).reshape(3, 4, 5)
lidx = [[0], [1]]
aidx = np.array(lidx)

a[lidx]          a[aidx]

-----
[[5, 6, 7, 8, 9]] [[[[ 0, 1, 2, 3, 4],
```

```

[ 5,  6,  7,  8,  9],
[10, 11, 12, 13, 14],
[15, 16, 17, 18, 19]],

[[[20, 21, 22, 23, 24],
  [25, 26, 27, 28, 29],
  [30, 31, 32, 33, 34],
  [35, 36, 37, 38, 39]]]]

```

这是因为NumPy将列表lidx转换成了([0],[1])，而将数组aidx转换成了(aidx, :, :)：

a[tuple(lidx)]	a[aidx, :, :]
-----	-----
[[5, 6, 7, 8, 9]]	[[[0, 1, 2, 3, 4], [5, 6, 7, 8, 9], [10, 11, 12, 13, 14], [15, 16, 17, 18, 19]]], [[20, 21, 22, 23, 24], [25, 26, 27, 28, 29], [30, 31, 32, 33, 34], [35, 36, 37, 38, 39]]]]

经过各种转换和添加“:”之后得到了一个标准的下标元组。它的各个元素有如下几种类型：切片、整数、整数数组和布尔数组。如果元素不是这些类型，如列表或元组，就将其转换成整数数组。

如果下标元组的所有元素都是切片和整数，那么用它作为下标得到的是原始数组的一个视图，即它和原始数组共享数据存储空间。

2.3.2 整数数组作为下标

下面看看下标元组中的元素由切片和整数数组构成的情况。假设整数数组有 N_t 个，而切片有 N_s 个。 $N_t + N_s$ 为数组的维数 D 。

首先，这 N_t 个整数数组必须满足广播条件，假设它们进行广播之后的维数为 M ，形状为 $(d_0, d_1, \dots, d_{M-1})$ 。

如果 N_s 为0，即没有切片元素时，则下标所得到的结果数组`result`的形状和整数数组广播之后的形状相同。它的每个元素值按照下面的公式获得：

$$\text{result}[i_0, i_1, \dots, i_{M-1}] = X[\text{ind}_0[i_0, i_1, \dots, i_{M-1}], \dots, \text{ind}_{N_t-1}[i_0, i_1, \dots, i_{M-1}]]$$

其中， ind_0 到 ind_{n_t-1} 为进行广播之后的整数数组。让我们看一个例子，从而加深对此公式的理解：



若只需要沿着指定轴通过整数数组获取元素，可以使用`numpy.take()`函数，其运算速度比整数数组的下标运算略快，并且支持下标越界处理。

```
i0 = np.array([[1, 2, 1], [0, 1, 0]])
i1 = np.array([[0], [1]])
i2 = np.array([[2, 3, 2]])
b = a[i0, i1, i2]
b
array([[22, 43, 22],
       [ 2, 23,  2]],

      [[27, 48, 27],
       [ 7, 28,  7]])
```

首先，i0、i1、i2三个整数数组的shape属性分别为(2,3)、(2,1,1)、(1,1,3)。根据广播规则，先在长度不足3的shape属性前面补1，使得它们的维数相同，广播之后的shape属性为各个轴的最大值：

```
(1, 2, 3)
(2, 1, 1)
(1, 1, 3)
-----
2  2  3
```

即三个整数数组广播之后的shape属性为(2,2,3)，这也就是下标运算所得到的结果数组的维数：

```
b.shape
(2, 2, 3)
```

我们可以使用broadcast_arrays() 查看广播之后的数组：

ind0, ind1, ind2 = np.broadcast_arrays(i0, i1, i2)		
ind0	ind1	ind2

[[[1, 2, 1], [0, 1, 0]]]	[[[0, 0, 0], [0, 0, 0]]]	[[[2, 3, 2], [2, 3, 2]]]
[[[1, 2, 1], [0, 1, 0]]]	[[[1, 1, 1], [1, 1, 1]]]	[[[2, 3, 2], [2, 3, 2]]]

对于b中的任意一个元素b[i, j, k]，它是数组a中经过ind0、ind1和ind2进行下标转换之后的值：

```
i, j, k = 0, 1, 2
print b[i, j, k], a[ind0[i, j, k], ind1[i, j, k], ind2[i, j, k]]

i, j, k = 1, 1, 1
print b[i, j, k], a[ind0[i, j, k], ind1[i, j, k], ind2[i, j, k]]

2 2
28 28
```

下面考虑 N_s 不为0的情况。当存在切片下标时，情况就变得更加复杂了。可以细分为两种情况：下标元组中的整数数组之间没有切片，即整数数组只有一个或连续的多个整数数组。这时结果数组的shape属性为：将原始数组的shape属性中整数数组所占据的部分替换为它们广播之后的shape属性。例如假设原始数组a的shape属性为(3,4,5)， i_0 和 i_1 广播之后的形状为(2,2,3)，则 $a[1:3, i_0, i_1]$ 的形状为(2,2,2,3)：

```
c = a[1:3, i0, i1]
c.shape
(2, 2, 2, 3)
```

其中，c的shape属性中的第一个2是切片“1:3”的长度，后面的(2,2,3)则是 i_0 和 i_1 广播之后的数组的形状：

```
ind0, ind1 = np.broadcast_arrays(i0, i1)
ind0.shape
(2, 2, 3)
```

```
i, j, k = 1, 1, 2
print c[:, i, j, k]
print a[1:3, ind0[i, j, k], ind1[i, j, k]] # 和c[:,i,j,k]的值相同
[21 41]
[21 41]
```

当下标元组中的整数数组不连续时，结果数组的shape属性为整数数组广播之后的形状后面添加上切片元素所对应的形状。例如 $a[i_0, :, i_1]$ 的shape属性为(2,2,3,4)，其中(2,2,3)是 i_0 和 i_1 广播之后的形状，而4是数组a的第1轴的长度：

```
d = a[i0, :, i1]
d.shape
(2, 2, 3, 4)
```



```
i, j, k = 1, 1, 2
d[i,j,k,:]      a[ind0[i,j,k],:,ind1[i,j,k]]
-----
[ 1,   6, 11, 16] [ 1,   6, 11, 16]
```

2.3.3 一个复杂的例子

下面让我们用所学的下标存取的知识，解决在NumPy邮件列表中提出的一个比较经典的问题。



<http://mail.scipy.org/pipermail/numpy-discussion/2008-July/035764.html>

NumPy邮件列表中原文的链接。

我们对问题进行一些简化，提问者想要实现的下标运算是：有一个形状为(I, J, K)的三维数组v和一个形状为(I, J)的二维数组idx，idx的每个值都是0到K-L的整数。他想通过下标运算得到一个数组r，对于第0轴和第1轴的每个下标i和j都满足下面的条件：

```
r[i,j,:] = v[i,j,idx[i,j]:idx[i,j]+L]
```

如图2-7所示，左图中不透明的方块是我们希望获取的部分，通过下标运算之后将得到右侧所示的数组。

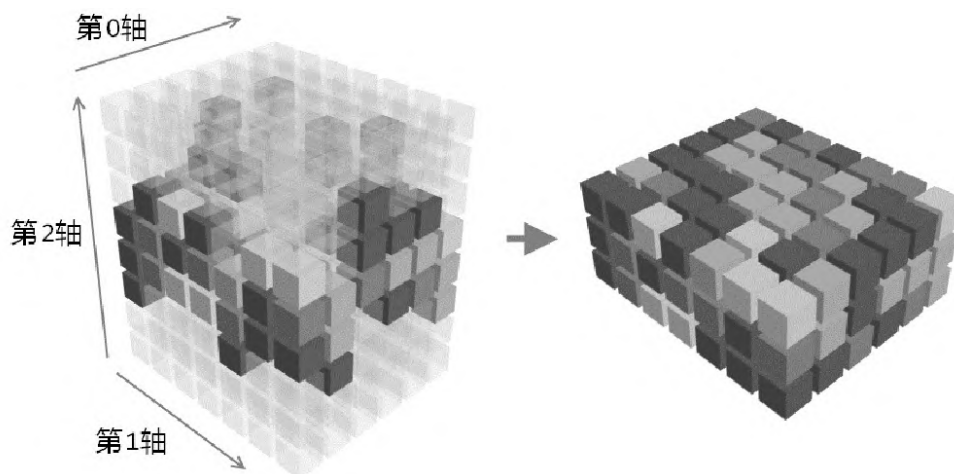


图2-7 三维数组下标运算问题的示意图

首先创建一个方便调试的数组 v ，它在第2轴上每一层的值就是该层的高度，即 $v[:, :, i]$ 的所有元素值都为 i 。然后随机产生数组 idx ，它的每个元素的取值都在0到 $K-L$ 之间：

```
I, J, K, L = 6, 7, 8, 3
_, _, v = np.mgrid[:I, :J, :K]
idx = np.random.randint(0, K - L, size=(I, J))
```

然后用数组 idx 创建第2轴的下标数组 idx_k ，它是一个形状为 (I, J, L) 的三维数组。它的第2轴上的每一层的值都等于 idx 数组加上层的高度，即 $idx_k[:, :, i] = idx[:, :] + i$ ：

```
idx_k = idx[:, :, None] + np.arange(3)
idx_k.shape
(6, 7, 3)
```

然后分别创建第0轴和第1轴的下标数组，它们的shape分别为 $(I, 1, 1)$ 和 $(1, J, 1)$ ：

```
idx_i, idx_j, _ = np.ogrid[:I, :J, :K]
```

使用 idx_i , idx_j , idx_k 对数组 v 进行下标运算即可得到结果：

```

r = v[idx_i, idx_j, idx_k]
i, j = 2, 3 # 验证结果, 读者可以将之修改为使用循环验证所有的元素
r[i,j,:] = v[i,j,idx[i,j]:idx[i,j]+L]
-----
[0, 1, 2] [0, 1, 2]

```

2.3.4 布尔数组作为下标

当使用布尔数组直接作为下标对象或者元组下标对象中有布尔数组时, 都相当于用`nonzero()`将布尔数组转换成一组整数数组, 然后使用整数数组进行下标运算。

`nonzero(a)`返回数组`a`中值不为零的元素的下标, 它的返回值是一个长度为`a.ndim`(数组`a`的轴数)的元组, 元组的每个元素都是一个整数数组, 其值为非零元素的下标在对应轴上的值。例如对于一维布尔数组`b1`, `nonzero(a)`所得到的是一个长度为1的元组, 它表示`b1[0]`和`b1[2]`的值不为0。



若只需要沿着指定轴通过布尔数组获取元素, 可以使用`numpy.compress()`函数。

```

b1 = np.array([True, False, True, False])
np.nonzero(b1)
(array([0, 2]),)

```

对于二维数组`b2`, `nonzero(a)`所得到的是一个长度为2的元组。它的第0个元素是数组`a`中值不为0的元素的第0轴的下标, 第1个元素则是第1轴的下标, 因此从下面的结果可知`b2[0,0]`、`b2[0,2]`和`b2[1,0]`的值不为0:

```

b2 = np.array([[True, False, True], [True, False, False]])
np.nonzero(b2)
(array([0, 0, 1]), array([0, 2, 0]))

```

当布尔数组直接作为下标时，相当于使用由`nonzero()`转换之后的元组作为下标对象：

```
a = np.arange(3 * 4 * 5).reshape(3, 4, 5)

a[b2]          a[np.nonzero(b2)]

-----

[[ 0,  1,  2,  3,  4], [[ 0,  1,  2,  3,  4],
 [10, 11, 12, 13, 14], [10, 11, 12, 13, 14],
 [20, 21, 22, 23, 24]] [20, 21, 22, 23, 24]]
```

当下标对象是元组，并且其中有布尔数组时，相当于将布尔数组展开为由`nonzeros()`转换之后的各个整数数组：

```
a[1:3, b2]      a[1:3, np.nonzero(b2)[0], np.nonzero(b2)[1]]

-----

[[20, 22, 25],  [[20, 22, 25],
 [40, 42, 45]]  [40, 42, 45]]
```

2.4 庞大的函数库



与本节内容对应的 Notebook 为：[02-numpy/numpy-400-functions.ipynb](#)。

除了前面介绍的`ndarray`数组对象和`ufunc`函数之外，NumPy还提供了大量对数组进行处理的函数。充分利用这些函数，能够简化程序的逻辑，提高运算速度。本节通过一些较常用的例子，说明它们的一些使用技巧和注意事项。

2.4.1 随机数

本节介绍的函数如表2-3所示。

表2-3 本节要介绍的函数

函数名	功能	函数名	功能
rand	0到1之间的随机数	randn	标准正态分布的随机数
randint	指定范围内的随机整数	normal	正态分布
uniform	均匀分布	poisson	泊松分布
permutation	随机排列	shuffle	随机打乱顺序
choice	随机抽取样本	seed	设置随机数种子

numpy.random模块中提供了大量的随机数相关的函数，为了方便后面用随机数测试各种运算函数，让我们首先来看看如何产生随机数：

- rand() 产生0到1之间的随机浮点数，它的所有参数用于指定所产生的数组的形状。
- randn() 产生标准正态分布的随机数，参数的含义与rand() 相同。
- randint() 产生指定范围的随机整数，包括起始值，但是不包括终值，在下面的例子中，产生0到9的随机数，它的第三个参数用于指定数组的形状：

```
from numpy import random as nr
np.set_printoptions(precision=2) # 为了节省篇幅，只显示小数点后两位数字
r1 = nr.rand(4, 3)
r2 = nr.randn(4, 3)
r3 = nr.randint(0, 10, (4, 3))
```

r1	r2	r3
-----	-----	-----
[[0.87, 0.42, 0.34],	[[-1.32, -0.03, -0.05],	[[5, 9, 1],
[0.25, 0.87, 0.42],	[0.34, -0.42, -0.41],	[2, 9, 8],
[0.49, 0.18, 0.44],	[0.59, -0.49, -0.01],	[2, 6, 6],
[0.53, 0.23, 0.81]]	[-1.92, -0.13, -1.34]]	[3, 8, 1]]

random模块提供了许多产生符合特定随机分布的随机数的函数，它们的最后一个参数size都用于指定输出数组的形状，而其他参数都是分布函数的参数。例如：

- normal(): 正态分布，前两个参数分别为期望值和标准差。
- uniform(): 均匀分布，前两个参数分别为区间的起始值和终值。
- poisson(): 泊松分布，第一个参数指定 λ 系数，它表示单位时间(或单位面积)内随机事件的平均发生率。由于泊松分布是一个离散分布，因此它输出的数组是一个整数数组。

```
r1 = nr.normal(100, 10, (4, 3))
r2 = nr.uniform(10, 20, (4, 3))
r3 = nr.poisson(2.0, (4, 3))
```

r1	r2	r3
[102.89, 103.56, 111.46],	[19. , 18.69, 14.38],	[[3, 1, 5],
[83.54, 122.36, 98.31],	[17.97, 10.16, 12.47],	[2, 2, 3],
[87.95, 106.89, 99.28],	[19.36, 10.91, 19.65],	[2, 4, 4],
[92.66, 103.13, 106.28]]	[16.79, 16.46, 16.32]]	[2, 2, 3]]

permutation() 可以用于产生一个乱序数组，当参数为整数n时，它返回[0,n)这n个整数的随机排列；当参数为一个序列时，它返回一个随机排列之后的序列：

```
a = np.array([1, 10, 20, 30, 40])
print nr.permutation(10)
print nr.permutation(a)
[2 4 3 5 6 8 0 1 9 7]
[40  1 10 20 30]
```

permutation() 返回一个新数组，而shuffle() 则直接将参数数组的顺序打乱：

```
nr.shuffle(a)
a
array([ 1, 20, 30, 10, 40])
```

choice() 从指定的样本中随机进行抽取：

- size参数用于指定输出数组的形状。

- replace参数为True时，进行可重复抽取，而为False时进行不重复抽取，默认值为True。所以在下面的例子中，c1中可能有重复数值，而c2中的每个数值都是不同的。
- p参数指定每个元素对应的抽取概率，如果不指定，所有的元素被抽取到的概率相同。在下面的例子中，值越大的元素被抽到的概率越大，因此c3中数值较大的元素比较多。

```

a = np.arange(10, 25, dtype=float)
c1 = nr.choice(a, size=(4, 3))
c2 = nr.choice(a, size=(4, 3), replace=False)
c3 = nr.choice(a, size=(4, 3), p=a / np.sum(a))

```

c1	c2	c3
-----	-----	-----
[[12., 22., 17.],	[[10., 14., 23.],	[[21., 24., 23.],
[24., 13., 14.],	[24., 13., 19.],	[19., 18., 19.],
[19., 23., 23.],	[11., 22., 20.],	[24., 21., 22.],
[17., 19., 22.]]	[15., 17., 18.]]	[22., 21., 21.]]

为了保证每次运行时能重现相同的随机数，可以通过seed()函数指定随机数的种子。在下面的例子中，计算r3和r4之前，都使用42作为种子，因此得到的随机数组是相同的：

```

r1 = nr.randint(0, 100, 3)
r2 = nr.randint(0, 100, 3)
nr.seed(42)
r3 = nr.randint(0, 100, 3)
nr.seed(42)
r4 = nr.randint(0, 100, 3)

```

r1	r2	r3	r4
-----	-----	-----	-----
[84, 14, 46]	[23, 20, 66]	[51, 92, 14]	[51, 92, 14]

2.4.2 求和、平均值、方差

本节介绍的函数如表2-4所示。

表2-4 本节要介绍的函数

--	--	--	--

函数名	功能	函数名	功能
sum	求和	mean	求期望
average	加权平均数	std	标准差
var	方差	product	连乘积

sum() 计算数组元素之和，也可以对列表、元组等与数组类似的序列进行求和。当数组是多维时，它计算数组中所有元素的和。这里我们使用random.randint() 模块中的函数创建一个随机整数数组。

```

np.random.seed(42)
a = np.random.randint(0,10,size=(4,5))

      a      np.sum(a)
-----
[[6, 3, 7, 4, 6], 96
 [9, 2, 6, 7, 4],
 [3, 7, 7, 2, 5],
 [4, 1, 7, 5, 1]]

```

如果指定axis参数，则求和运算沿着指定的轴进行。在上面的例子中，数组a的第0轴的长度为4，第1轴的长度为5。如果axis参数为1，则对每行上的5个数求和，所得的结果是长度为4的一维数组。如果参数axis为0，则对每列上的4个数求和，结果是长度为5的一维数组。即结果数组的形状是原始数组的形状除去其第axis个元素：

```

np.sum(a, axis=1)    np.sum(a, axis=0)
-----
[26, 28, 24, 18]    [22, 13, 27, 18, 16]

```

当axis参数是一个轴的序列时，对指定的所有轴进行求和运算。例如下面的程序对一个形状为(2, 3, 4)的三维数组的第0和第2轴求和，得到的结果为一个形状为(3,)的数组。由于数组的所有元素都为1，因此求和的结果都是8：


```
np.sum(np.ones((2, 3, 4)), axis=(0, 2))  
array([ 8.,  8.,  8.])
```

有时我们希望能够保持原数组的维数，这时可以设置keepdims参数为True：

```
np.sum(a, 1, keepdims=True)  np.sum(a, 0, keepdims=True)  
-----  
[[26],                        [[22, 13, 27, 18, 16]]  
 [28],  
 [24],  
 [18]]
```

sum()默认使用和数组的元素类型相同的累加变量进行计算，如果元素类型为整数，则使用系统的默认整数类型作为累加变量，例如在32位系统中使用32位整数作为累加变量。因此对整数数组进行累加时可能会出现溢出问题，即数组元素的总和超过了累加变量的取值范围。下面的程序计算数组a中每个元素占其所在行总和的百分比。在调用sum()函数时：

- 设置dtype参数为float，这样得到的结果是浮点数组，能避免整数的整除运算。
- 设置keepdims参数为True，这样sum()得到的结果的形状为(4, 1)，能够和原始数组进行广播运算。

```
pa = a / np.sum(a, 1, dtype=float, keepdims=True) * 100  
          pa          pa.sum(1, keepdims=True)  
-----  
[[ 23.08,  11.54,  26.92,  15.38,  23.08], [[ 100.],  
 [ 32.14,   7.14,  21.43,  25.  ,  14.29], [ 100.],  
 [ 12.5 ,  29.17,  29.17,   8.33,  20.83], [ 100.],  
 [ 22.22,   5.56,  38.89,  27.78,   5.56]] [ 100.]]
```

对很大的单精度浮点数类型的数组进行计算时，也可能出现精度不够的现象，这时也可以通过dtype参数指定累加变量的类型。在下面的例子中，我们对一个元素

都为1.1的单精度数组进行求和，比较单精度累加变量和双精度累加变量的计算结果：

```
np.set_printoptions(precision=8)
b = np.full(1000000, 1.1, dtype=np.float32) # 创建一个很大的单精度浮点数数组
b # 1.1无法使用浮点数精确表示，存在一些误差
array([ 1.10000002,  1.10000002,  1.10000002, ...,  1.10000002,
        1.10000002,  1.10000002], dtype=float32)
```

使用单精度累加变量进行累加计算，误差将越来越大，而使用双精度浮点数则能够得到较精确的结果：

```
np.sum(b)    np.sum(b, dtype=np.double)
-----
1099999.3    1100000.0238418579
```

上面的例子将产生一个新的数组来保存求和的结果，如果希望将结果直接保存到另一个数组中，可以和ufunc函数一样使用out参数指定输出数组，它的形状必须和结果数组的形状相同。

mean() 求数组的平均值，它的参数与sum()相同。和sum()不同的是：对于整数数组它使用双精度浮点数进行计算，而对于其他类型的数组，则使用和数组元素类型相同的累加变量进行计算：

```
np.mean(a, axis=1) # 整数数组使用双精度浮点数进行计算
array([ 5.2,  5.6,  4.8,  3.6])

np.mean(b)    np.mean(b, dtype=np.double)
-----
1.0999993     1.1000000238418579
```

此外，average()也可以对数组进行平均计算。它没有out和dtype参数，但有一个指定每个元素权值的weights参数，可以用于计算加权平均数。例如有三个班级，

number数组中保存每个班级的人数，score数组中保存每个班级的平均分，下面计算所有班级的加权平均分，得到整个年级的平均分：

```
score = np.array([83, 72, 79])
number = np.array([20, 15, 30])
print np.average(score, weights=number)

78.6153846154
```

相当于进行如下计算：

```
print np.sum(score * number) / np.sum(number, dtype=float)

78.6153846154
```

std() 和 var() 分别计算数组的标准差和方差，有 axis、out、dtype 以及 keepdims 等参数。方差有两种定义：偏样本方差(biased sample variance)和无偏样本方差(unbiased sample variance)。偏样本方差的计算公式为：

$$s_n^2 = \frac{1}{n} \sum_{i=1}^n (y_i - \bar{y})^2$$

而无偏样本方差的公式为：

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (y_i - \bar{y})^2$$

当ddof参数为0时，计算偏样本方差；当ddof为1时，计算无偏样本方差，默认值为0。下面我们用程序演示这两种方差的差别。

首先产生一个标准差为2.0、方差为4.0的正态分布的随机数组。我们可以认为总体样本的方差为4.0。假设从总体样本中随机抽取10个样本，我们分别计算这10个样本的两种方差，这里我们用一个二维数组重复上述操作100000次，然后计算所有这些方差的期望值：

```
a = nr.normal(0, 2.0, (100000, 10))
v1 = np.var(a, axis=1, ddof=0) #可以省略ddof=0
v2 = np.var(a, axis=1, ddof=1)

np.mean(v1)          np.mean(v2)
```

```
-----
3.6008566906846693  4.0009518785385216
```

可以看到无偏样本方差的期望值接近于总体方差4.0，而偏样本方差比4.0小一些。

偏样本方差是正态分布随机变量的最大似然估计。如果有一个样本包含n个随机数，并且知道它们符合正态分布，通过该样本可以估算出正态分布的概率密度函数的参数。所估算的那组正态分布参数最符合给定的样本，就称为最大似然估计。

正态分布的概率密度函数的定义如下，其中 μ 表示期望， σ^2 表示方差：

$$f(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

所谓最大似然估计，就是找到一组参数使得下面的乘积最大，其中 x_1 为样本中的值：

$$f(x_1) f(x_2) \cdots f(x_n)$$

专业术语总是很难理解，下面我们还是用程序来验证：

```
def normal_pdf(mean, var, x):
    return 1 / np.sqrt(2 * np.pi * var) * np.exp(-(x - mean) ** 2 / (2 *
var))

nr.seed(42)
data = nr.normal(0, 2.0, size=10) ❶
mean, var = np.mean(data), np.var(data) ❷
var_range = np.linspace(max(var - 4, 0.1), var + 4, 100) ❸

p = normal_pdf(mean, var_range[:, None], data) ❹
p = np.product(p, axis=1) ❺
```

`normal_pdf()` 为计算正态分布的概率密度的函数。❶产生10个正态分布的随机数。❷计算其最大似然估计的参数。❸以最大似然估计的方差为中心，产生一组方差值。❹用正态分布的概率密度函数计算每个样本、每个方差所对应的概率密度。由于使用了广播运算，得到的结果`p`是一个二维数组，它的第0轴对应`var_range`中的各个方差，第1轴对应`data`中的每个元素。❺沿着`p`的第1轴求所有概率密度的乘积。`product()`和`sum()`的用法类似，用于计算数组所有元素的乘积。

下面绘制`var_range`中各个方差对应的似然估计值，并用一条竖线表示偏样本方差。由图2-8可以看到偏样本方差位于似然估计曲线的最大值处。

```
import pylab as pl
pl.plot(var_range, p)
pl.axvline(var, 0, 1, c="r")
pl.show()
```

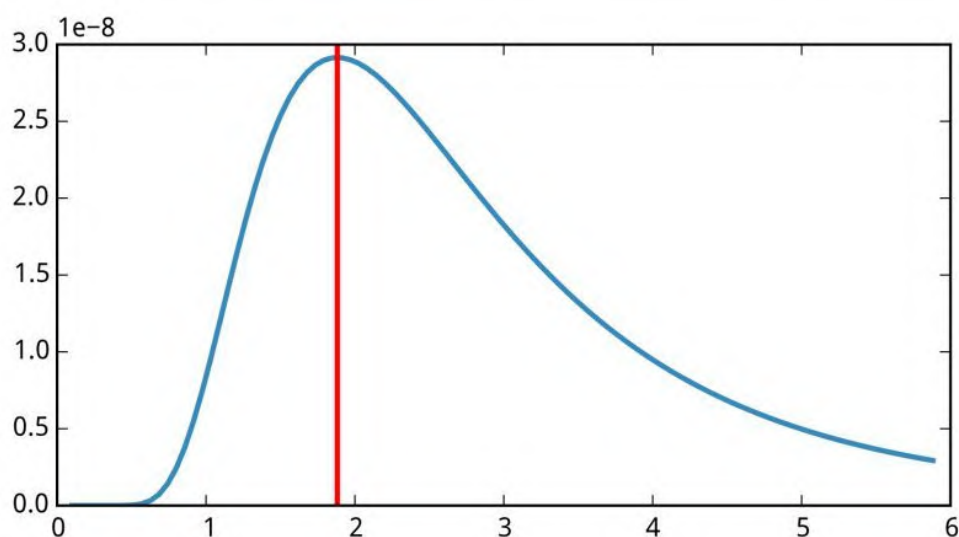


图2-8 偏样本方差位于似然估计曲线的最大值处

2.4.3 大小与排序



与本节内容对应的Notebook为：02-numpy/numpy-410-functions-sort.ipynb。

本节介绍的函数如表2-5所示。

表2-5 本节要介绍的函数

函数名	功能	函数名	功能
min	最小值	max	最大值
minimum	二元最小值	maximum	二元最大值
ptp	最大值与最小值的差	argmin	最小值的下标
argmax	最大值的下标	unravel_index	一维下标转换成多维下标
sort	数组排序	argsort	计算数组排序的下标
lexsort	多列排序	partition	快速计算前k位
argpartition	前k位的下标	median	中位数
percentile	百分位数	searchsorted	二分查找

用min()和max()可以计算数组的最小值和最大值，它们都有axis、out、keepdims等参数。这些参数的用法和sum()基本相同，但是axis参数不支持序列。此外，ptp()计算最大值和最小值之间的差，有axis和out参数。这里就不再多举例了，请读者自行查看函数的文档。minimum()和maximum()用于比较两个数组对应下标的元素，返回数组的形状为两参数数组广播之后的形状。

```
a = np.array([1, 3, 5, 7])
b = np.array([2, 4, 6])
```

```
np.maximum(a[None, :], b[:, None])
array([[2, 3, 5, 7],
       [4, 4, 5, 7],
       [6, 6, 6, 7]])
```

用`argmax()`和`argmin()`可以求最大值和最小值的下标。如果不指定`axis`参数，则返回平坦化之后的数组下标，例如下面的程序找到`a`中最大值的下标，有多个最值时得到第一个最值的下标：

```
np.random.seed(42)
a = np.random.randint(0, 10, size=(4, 5))
max_pos = np.argmax(a)
max_pos
5
```

下面查看`a`平坦化之后的数组中下标为`max_pos`的元素：

```
a.ravel()[max_pos]  np.max(a)
-----
9                  9
```

可以通过`unravel_index()`将一维数组的下标转换为多维数组的下标，它的一个参数为一维数组的下标，第二个参数是多维数组的形状：

```
idx = np.unravel_index(max_pos, a.shape)
idx  a[idx]
-----
(1, 0)  9
```

当使用`axis`参数时，可以沿着指定轴计算最大值的下标。例如下面的结果表示，在数组`a`中第0行的最大值的下标为2，第1行的最大值的下标为0：

```
idx = np.argmax(a, axis=1)
idx
array([2, 0, 1, 2])
```

使用下面的语句可以通过idx选出每行的最大值：

```
a[np.arange(a.shape[0]), idx]
array([7, 9, 7, 7])
```

数组的`sort()`方法对数组进行排序，它会改变数组的内容；而`sort()`函数则返回一个新数组，不改变原始数组。它们的`axis`默认值都为-1，即沿着数组的最终轴进行排序。`sort()`函数的`axis`参数可以设置为None，此时它将得到平坦化之后进行排序的新数组。在下面的例子中，`np.sort(a)`对a中每行的数值进行排序，而`np.sort(a, axis=0)`对数组a每列上的数值进行排序。

<code>np.sort(a)</code>	<code>np.sort(a, axis=0)</code>
-----	-----
<code>[[3, 4, 6, 6, 7],</code>	<code>[[3, 1, 6, 2, 1],</code>
<code> [2, 4, 6, 7, 9],</code>	<code> [4, 2, 7, 4, 4],</code>
<code> [2, 3, 5, 7, 7],</code>	<code> [6, 3, 7, 5, 5],</code>
<code> [1, 1, 4, 5, 7]]</code>	<code> [9, 7, 7, 7, 6]]</code>

`argsort()`返回数组的排序下标，参数`axis`的默认值为-1：

<code>sort_axis1</code>	<code>sort_axis0</code>
-----	-----
<code>[[1, 3, 0, 4, 2],</code>	<code>[[2, 3, 1, 2, 3],</code>
<code> [1, 4, 2, 3, 0],</code>	<code> [3, 1, 0, 0, 1],</code>
<code> [3, 0, 4, 1, 2],</code>	<code> [0, 0, 2, 3, 2],</code>
<code> [1, 4, 0, 3, 2]]</code>	<code> [1, 2, 3, 1, 0]]</code>

为了使用`sort_axis0`和`sort_axis1`计算排序之后的数组，即`np.sort(a)`的结果，需要产生非排序轴的下标。下面使用`ogrid`对象产生第0轴和第1轴的下标`axis0`和`axis1`：

```
axis0, axis1 = np.ogrid[:a.shape[0], :a.shape[1]]
```


然后使用这些下标数组得到排序之后的数组：

```
a[axis0, sort_axis1]  a[sort_axis0, axis1]
-----
[[3, 4, 6, 6, 7],      [[3, 1, 6, 2, 1],
 [2, 4, 6, 7, 9],      [4, 2, 7, 4, 4],
 [2, 3, 5, 7, 7],      [6, 3, 7, 5, 5],
 [1, 1, 4, 5, 7]]      [9, 7, 7, 7, 6]]
```

使用这种方法可以对两个相关联的数组进行排序，即从数组a产生排序下标数组，然后使用它对数组b进行排序。排序相关的函数或方法还可以通过kind参数指定排序算法，对于结构数组可以通过order参数指定排序所使用的字段。

lexsort()类似于Excel中的多列排序。它的参数是一个形状为(k, N)的数组，或者包含k个长度为N的数组的序列，可以把它理解为Excel中N行k列的表格。lexsort()返回排序下标，注意数组中最后的列为排序的主键。在下面的例子中，按照“姓名-年龄”的顺序对数据排序：

```
names = ["zhang", "wang", "li", "wang", "zhang"]
ages = [37, 33, 32, 31, 36]
idx = np.lexsort([ages, names])
sorted_data = np.array(zip(names, ages), "O")[idx]

      idx      sorted_data
-----
[2, 3, 1, 4, 0][['li', 32],
 ['wang', 31],
 ['wang', 33],
 ['zhang', 36],
 ['zhang', 37]]
```

如果需要对一个N行k列的数组以第一列为主键进行排序，可以先通过切片下标::-1反转数组的第1轴，然后对其转置进行lexsort()排序：

```

b = np.random.randint(0, 10, (5, 3))

      b      b[np.lexsort(b[:, ::-1].T)]
-----
[[4, 0, 9],  [[3, 8, 2],
 [5, 8, 0],  [4, 0, 9],
 [9, 2, 6],  [4, 2, 6],
 [3, 8, 2],  [5, 8, 0],
 [4, 2, 6]]  [9, 2, 6]]

```

`partition()` 和 `argpartition()` 对数组进行分割，可以很快地找出排序之后的前 k 个元素，由于它不需要对整个数组进行完整排序，因此速度比调用 `sort()` 之后再取前 k 个元素要快许多。下面从10万个随机数中找出前5个最小的数，注意 `partition()` 得到的前5个数值没有按照从小到大排序，如果需要，可以再调用 `sort()` 对这5个数进行排序即可：

```

r = np.random.randint(10, 1000000, 100000)

np.sort(r)[:5]      np.partition(r, 5)[:5]
-----
[15, 23, 25, 37, 47] [15, 47, 25, 37, 23]

```

下面用 `%timeit` 测试 `sort()` 和 `partition()` 的运行速度：

```

%timeit np.sort(r)[:5]
%timeit np.sort(np.partition(r, 5)[:5])
100 loops, best of 3: 6.02 ms per loop
1000 loops, best of 3: 348 µs per loop

```

用 `median()` 可以获得数组的中值，即对数组进行排序之后，位于数组中间位置的值。当长度是偶数时，则得到正中间两个数的平均值。它也可以指定 `axis` 和 `out` 参数：

```

np.median(a, axis=1)
array([ 6.,  6.,  5.,  4.])

```

`percentile()` 用于计算百分位数，即将数值从小到大排列，计算处于 $p\%$ 位置上的值。下面的程序计算标准正态分布随机数的绝对值在68.3%、95.4%以及99.7%处的百分位数，它们应该约等于1倍、2倍和3倍的标准差：

```
r = np.abs(np.random.randn(100000))
np.percentile(r, [68.3, 95.4, 99.7])
array([ 1.00029686,  1.99473003,  2.9614485 ])
```

当数组中的元素按照从小到大的顺序排列时，可以使用`searchsorted()`在数组中进行二分搜索。在下面的例子中，`a`是一个已经排好序的列表，`v`是需要搜索的数值列表。`searchsorted()`返回一个下标数组，将`v`中对应的元素插入到`a`中的位置，能够保持数据的升序排列。当`v`中的元素在`a`中出现时，通过`side`参数指定返回最左端的下标还是最右端的下标。在下面的例子中，16放到`a`的下标为3、4、5的位置都能保持升序排列，`side`参数为默认值“left”时返回3，而为“right”时返回5。

```
a = [2, 4, 8, 16, 16, 32]
v = [1, 5, 33, 16]
np.searchsorted(a, v)  np.searchsorted(a, v, side="right")
-----
[0, 2, 6, 3]          [0, 2, 6, 5]
```

`searchsorted()`可以用于在两个数组中查找相同的元素。下面看一个比较复杂的例子：有`x`和`y`两个一维数组，找到`y`中每个元素在`x`中的下标。若不存在，将下标设置为-1。

```
x = np.array([3, 5, 7, 1, 9, 8, 6, 10])
y = np.array([2, 1, 5, 10, 100, 6])

def get_index_searchsorted(x, y):
    index = np.argsort(x) ❶
    sorted_x = x[index] ❷
    sorted_index = np.searchsorted(sorted_x, y) ❸
    yindex = np.take(index, sorted_index, mode="clip") ❹
    mask = x[yindex] != y ❺
```

```

    yindex[mask] = -1
    return yindex

get_index_searchsorted(x, y)
array([-1,  3,  1,  7, -1,  6])

```

❶由于x并不是按照升序排列，因此先调用`argsort()`获得升序排序的下标`index`。❷使用`index`获得将x排序之后的`sorted_x`。❸使用`searchsorted()`在`sorted_x`中搜索y中每个元素对应的下标`sorted_index`。

❹如果搜索的值大于x的最大值，那么下标会越界，因此这里调用`take()`函数，`take(index, sorted_index)`与`index[sorted_index]`的含义相同，但是能处理下标越界的情况。通过设置`mode`参数为“clip”，将下标限定在0到`len(x)-1`之间。

❺使用`yindex`获取x中的元素并和y比较，若值相同则表示该元素确实存在于x之中，否则表示不存在。

这段算法有些复杂，但由于利用了NumPy提供的数组操作函数，它的运算速度比使用字典的纯Python程序要快。下面我们用两个较大的数组测试运算速度。为了比较的公平性，我们调用`tolist()`方法将数组转换成列表：

```

x = np.random.permutation(1000)[:100]
y = np.random.randint(0, 1000, 2000)
x1, y1 = x.tolist(), y.tolist()

def get_index_dict(x, y):
    idx_map = {v:i for i,v in enumerate(x)}
    yindex = [idx_map.get(v, -1) for v in y]
    return yindex

yindex1 = get_index_searchsorted(x, y)
yindex2 = get_index_dict(x1, y1)
print np.all(yindex1 == yindex2)

```

```
%timeit get_index_searchsorted(x, y)
%timeit get_index_dict(x1, y1)

True

10000 loops, best of 3: 122 µs per loop
1000 loops, best of 3: 368 µs per loop
```

2.4.4 统计函数



与本节内容对应的Notebook为：[02-numpy/numpy-420-functions-count.ipynb](#)。

本节介绍的函数如表2-6所示。

表2-6 本节要介绍的函数

函数名	功能	函数名	功能
unique	去除重复元素	bincount	对整数数组的元素计数
histogram	一维直方图统计	digitze	离散化

`unique()` 返回其参数数组中所有不同的值，并且按照从小到大的顺序排列。它有两个可选参数：

- `return_index`: True表示同时返回原始数组中的下标。
- `return_inverse`: True表示返回重建原始数组用的下标数组。

下面通过几个例子介绍`unique()`的用法。首先用`randint()`创建有10个元素、值在0到9范围内的随机整数数组，通过`unique(a)`可以找到数组a中所有的整数，并按照升序排列：

```
np.random.seed(42)
a = np.random.randint(0, 8, 10)

a                                np.unique(a)
-----
[6, 3, 4, 6, 2, 7, 4, 4, 6, 1]  [1, 2, 3, 4, 6, 7]
```

如果参数`return_index`为`True`，则返回两个数组，第二个数组是第一个数组在原始数组中的下标。在下面的例子中，数组`index`保存的是数组`x`中每个元素在数组`a`中的下标：

```
x, index = np.unique(a, return_index=True)

x                                index                                a[index]
-----
[1, 2, 3, 4, 6, 7]  [9, 4, 1, 2, 0, 5]  [1, 2, 3, 4, 6, 7]
```

如果参数`return_inverse`为`True`，则返回的第二个数组是原始数组`a`的每个元素在数组`x`中的下标：

```
x, rindex = np.unique(a, return_inverse=True)

rindex                                x[rindex]
-----
[4, 2, 3, 4, 1, 5, 3, 3, 4, 0]  [6, 3, 4, 6, 2, 7, 4, 4, 6, 1]
```

`bincount()`对整数数组中各个元素所出现的次数进行统计，它要求数组中的所有元素都是非负的。其返回数组中第`i`个元素的值表示整数`i`出现的次数。

```
np.bincount(a)
array([0, 1, 1, 1, 3, 0, 3, 1])
```

由上面的结果可知，在数组`a`中有1个1、1个2、1个3、3个4、3个6和1个7，而0、5等数没有在数组`a`中出现。

通过`weights`参数可以指定每个数所对应的权值。当指定`weights`参数时，`bincount(x, weights=w)`返回数组`x`中的每个整数所对应的`w`中的权值之和。用文字解释比较难以理解，下面我们看一个实例：

```
x = np.array([0, 1, 2, 2, 1, 1, 0])
w = np.array([0.1, 0.3, 0.2, 0.4, 0.5, 0.8, 1.2])
np.bincount(x, w)
array([ 1.3,  1.6,  0.6])
```

在上面的结果中，1.3是数组x中0所对应的w中的元素(0.1和1.2)之和，1.6是1所对应的w中的元素(0.3、0.5和0.8)之和，而0.6是2所对应的w中的元素(0.2和0.4)之和。如果要求平均值，可以用求和的结果与次数相除：

```
np.bincount(x, w) / np.bincount(x)
array([ 0.65,  0.53333333,  0.3])
```

histogram()对一维数组进行直方图统计。其参数列表如下：

```
histogram(a, bins=10, range=None, weights=None, density=False)
```

其中a是保存待统计数据的数组，bins指定统计的区间个数，即对统计范围的等分数。range是一个长度为2的元组，表示统计范围的最小值和最大值，默认值为None，表示范围由数据的范围决定，即(a.min(), a.max())。当density参数为False时，函数返回a中的数据在每个区间的个数，参数为True则返回每个区间的概率密度。weights参数和bincount()的类似。

histogram()返回两个一维数组——hist和bin_edges，第一个数组是每个区间的统计结果，第二个数组的长度为len(hist) + 1，每两个相邻的数值构成一个统计区间。下面我们看一个例子：

```
a = np.random.rand(100)
np.histogram(a, bins=5, range=(0, 1))
(array([28, 18, 17, 19, 18]), array([ 0.,  0.2,  0.4,  0.6,  0.8,  1.]))
```

首先创建了一个有100个元素的一维随机数组a，取值范围在0到1之间。然后用histogram()对数组a中的数据进行直方图统计。结果显示有28个元素的值在0到0.2之间，18个元素的值在0.2到0.4之间。读者可以尝试用rand()创建更大的随机数

组，由统计结果可知每个区间出现的次数近似相等，因此rand()所创建的随机数在0到1范围之间是平均分布的。

如果需要统计的区间的长度不等，可以将表示区间分隔位置的数组传递给bins参数，例如：

```
np.histogram(a, bins=[0, 0.4, 0.8, 1.0])  
(array([46, 36, 18]), array([ 0. ,  0.4,  0.8,  1. ]))
```

结果表示0到0.4之间有46个值，0.4到0.8之间有36个值。

如果用weights参数指定了数组a中每个元素所对应的权值，则histogram()对区间中数值所对应的权值进行求和。下面看一个使用histogram()统计男性青少年年龄和身高的例子。“height.csv”文件是100名年龄在7到20岁之间的男性青少年的身高统计数据。

首先用loadtxt()从数据文件载入数据。在数组d中，第0列是年龄，第1列是身高。可以看到年龄的范围在7到20之间：

```
d = np.loadtxt("height.csv", delimiter=",")  
d.shape      np.min(d[:, 0])      np.max(d[:, 0])  
-----  
(100, 2)  7.0999999999999996  19.899999999999999
```

下面对数据进行统计，sums是每个年龄段的身高总和，cnts是每个年龄段的数据个数，因此很容易计算出每个年龄段的平均身高：

```
sums = np.histogram(d[:, 0], bins=range(7, 21), weights=d[:, 1])[0]  
cnts = np.histogram(d[:, 0], bins=range(7, 21))[0]  
sums / cnts  
array([ 125.96      , 132.06666667, 137.82857143, 143.8      ,  
        148.14      , 153.44      , 162.15555556, 166.86666667,  
        172.83636364, 173.3      , 175.275      , 174.19166667, 175.075      ])
```

histogram2d()和histogramdd()对二维和N维数据进行直方图统计，我们将在第9章介绍OpenCV时对histogram2d()进行详细介绍。

2.4.5 分段函数

本节介绍的函数如表2-7所示。

表2-7 本节要介绍的函数

函数名	功能
where	矢量化判断表达式
piecewise	分段函数
select	多分支判断选择

在前面的小节中介绍过如何使用`frompyfunc()`函数计算三角波形。由于三角波形是分段函数，需要根据自变量的取值范围决定计算函数值的公式，因此无法直接通过`ufunc`函数计算。NumPy提供了一些计算分段函数的方法。

在Python 2.6中新增加了如下判断表达式语法，当`condition`条件为`True`时，表达式的值为`y`，否则为`z`：

```
x = y if condition else z
```

在NumPy中，`where()`函数可以看作判断表达式的数组版本：

```
x = where(condition, y, z)
```

其中`condition`、`y`和`z`都是数组，它的返回值是一个形状与`condition`相同的数组。当`condition`中的某个元素为`True`时，`x`中对应下标的值从数组`y`获取，否则从数组`z`获取：

```
x = np.arange(10)
np.where(x < 5, 9 - x, x)
array([9, 8, 7, 6, 5, 5, 6, 7, 8, 9])
```

如果y和z是单个数值或者它们的形状与condition的不同，将先通过广播运算使其形状一致：

```
np.where(x > 6, 2 *  
  
x, 0)  
array([ 0,  0,  0,  0,  0,  0,  0, 14, 16, 18])
```

使用where()很容易计算前面介绍过的三角波形：

```
def triangle_wavel(x, c, c0, hc):  
    x = x - x.astype(np.int) # 三角波的周期为1，因此只取x坐标的小数部分进行计算  
    return np.where(x >= c,  
                    0,  
                    np.where(x < c0,  
                               x / c0 *  
  
hc,  
  
                    (c - x) / (c - c0) *  
  
hc))
```

由于三角波形分为三段，因此需要两个嵌套的where()进行计算。由于所有的运算和循环都在C语言级别完成，因此它的计算效率比frompyfunc()高。

随着分段函数的分段数量的增加，需要嵌套更多层where()。这样不便于程序的编写和阅读。可以用select()解决这个问题，它的调用形式如下：

```
select(condlist, choicelist, default=0)
```

其中condlist是一个长度为N的布尔数组列表，choicelist是一个长度为N的存储候选值的数组列表，所有数组的长度都为M。如果列表元素不是数组而是单个数值，那么它相当于元素值都相同、长度为M的数组。

对于从0到M-1的数组下标i，从布尔数组列表中找出满足条件condlist[j][i]==True的j的最小值，则out[i]=choicelist[j][i]，其中out是select()的返回数组。我们可以使用select()计算三角波形：

```
def triangle_wave2(x, c, c0, hc):
    x = x - x.astype(np.int)
    return np.select([x >= c, x < c0, True],
                     [0, x/c0,
                      hc, (c-x)/(c-c0)*
                      hc])
```

由于分段函数分为三段，因此每个列表都有三个元素。choicelist的最后一个元素为True，表示前面的所有条件都不满足时，将使用choicelist的最后一个数组中的值。也可以用default参数指定条件都不满足时的候选值数组：

```
return np.select([x>= c, x < c0],
                 [0, x / c0 *
                  hc],
                 default=(c-x)/(c-c0)*
                 hc)
```

但是where()和select()的所有参数都需要在调用它们之前完成计算，因此NumPy会计算下面4个数组：

```
x >= c, x < c0, x / c0 *
hc, (c - x) / (c - c0) *
hc
```

在计算时还会产生许多保存中间结果的数组，因此如果输入的数组x很大，将会发生大量内存分配和释放。

为了解决这个问题，NumPy提供了`piecewise()`专门用于计算分段函数，它的调用参数如下：

```
piecewise(x, condlist, funclist)
```

参数x是一个保存自变量值的数组，condlist是一个长度为M的布尔数组列表，其中的每个布尔数组的长度都和数组x相同。funclist是一个长度为M或M+1的函数列表，这些函数的输入和输出都是数组。它们计算分段函数中的每个片段。如果不是函数而是数值，则相当于返回此数值的函数。每个函数与condlist中下标相同的布尔数组对应，如果funclist的长度为M+1，则最后一个函数计算所有条件都为False时的值。下面是使用`piecewise()`计算三角波形的程序：

```
def triangle_wave3(x, c, c0, hc):
    x = x - x.astype(np.int)
    return np.piecewise(x,
        [x >= c, x < c0],
        [0, # x>=c
         lambda x: x / c0 *
         hc, # x<c0
         lambda x: (c - x) / (c - c0) *
         hc]) # else
```

使用`piecewise()`的好处在于它只计算需要计算的值。因此在上面的例子中，表达式 $x/c0 * hc$ 和 $(c-x)/(c-c0) * hc$ 只对输入数组x中满足条件的部分进行计算。下面运行前面定义的两个分段函数，并使用`%timeit`命令比较这三个函数的运行时间：

```
x = np.linspace(0, 2, 10000)
y1 = triangle_wave1(x, 0.6, 0.4, 1.0)
y2 = triangle_wave2(x, 0.6, 0.4, 1.0)
```

```
y3 = triangle_wave3(x, 0.6, 0.4, 1.0)
np.all(y1 == y2), np.all(y1 == y3)

(True, True)
```

```
%timeit triangle_wave1(x, 0.6, 0.4, 1.0)
%timeit triangle_wave2(x, 0.6, 0.4, 1.0)
%timeit triangle_wave3(x, 0.6, 0.4, 1.0)

1000 loops, best of 3: 614 µs per loop
1000 loops, best of 3: 736 µs per loop
1000 loops, best of 3: 311 µs per loop
```

2.4.6 操作多维数组



与本节内容对应的Notebook为：[02-numpy/numpy-430-functions-array-op.ipynb](#)。

本节介绍的函数如表2-8所示。

表2-8 本节要介绍的函数

函数名	功能	函数名	功能
concatenate	连接多个数组	vstack	沿第0轴连接数组
hstack	沿第1轴连接数组	column_stack	按列连接多个一维数组
split、array_split	将数组分为多段	transpose	重新设置轴的顺序
swapaxes	交换两个轴的顺序		

`concatenate()`是连接多个数组的最基本的函数，其他函数都是它的快捷版本。它的第一个参数是包含多个数组的序列，它将沿着`axis`参数指定的轴（默认为第0轴）连接数组。所有这些数组的形状除了第`axis`轴之外都相同。

`vstack()`沿着第0轴连接数组，当被连接的数组是长度为N的一维数组时，将其形状改为(1, N)。

`hstack()`沿着第1轴连接数组。当所有数组都是一维时，沿着第0轴连接数组，因此结果数组仍然为一维的。

`column_stack()`和`hstack()`类似，沿着第1轴连接数组，但是当数组为一维时，将其形状改为(N, 1)，经常用于按列连接多个一维数组。

```
a = np.arange(3)
b = np.arange(10, 13)
v = np.vstack((a, b))
h = np.hstack((a, b))
c = np.column_stack((a, b))
```

v	h	c

[[0, 1, 2],	[0, 1, 2, 10, 11, 12]	[[0, 10],
[10, 11, 12]]		[1, 11],
[2, 12]]		

此外，`c_[ ]`对象也可以用于按列连接数组：

```
np.c_[a, b, a+b]
array([[ 0, 10, 10],
       [ 1, 11, 12],
       [ 2, 12, 14]])
```

`split()`和`array_split()`的用法基本相同，将一个数组沿着指定轴分成多个数组，可以直接指定切分轴上的切分点下标。下面的代码把随机数组`a`切分为多个数组，保证每个数组中的元素都是升序排列的。注意通过`diff()`和`nonzero()`获得的下标是每个升序片段中最后一个元素的下标，而切分点为每个片段第一个元素的下标，因此需要+1。

```

np.random.seed(42)
a = np.random.randint(0, 10, 12)
idx = np.nonzero(np.diff(a) < 0)[0] + 1

a          idx      np.split(a, idx)
-----
[6, 3, 7, 4, 6, 9, 2, 6, 7, 4, 3, 7] [ 1, 3, 6, 9, 10] [array([6]),
                    array([3, 7]),
                    array([4, 6, 9]),
                    array([2, 6, 7]),
                    array([4]),
                    array([3, 7])]

```

当第二个参数为整数时，表示分组个数。split() 只能平均分组，而 array_split() 能尽量平均分组：

```

np.split(a, 6)  np.array_split(a, 5)
-----
[array([6, 3]), [array([6, 3, 7]),
array([7, 4]), array([4, 6, 9]),
array([6, 9]), array([2, 6]),
array([2, 6]), array([7, 4]),
array([7, 4]), array([3, 7])]
array([3, 7])]

```

transpose() 和 swapaxes() 用于修改轴的顺序，它们得到的是原数组的视图。transpose() 通过其第二个参数 axes 指定轴的顺序，默认时表示将整个形状翻转。而 swapaxes() 通过两个整数指定调换顺序的轴。在下面的例子中：

- transpose() 的结果数组的形状为 (3, 4, 2, 5)，它们分别位于原数组形状 (2, 3, 4, 5) 的 (1, 2, 0, 3) 下标位置处。
- swapaxes() 的结果数组的形状为 (2, 4, 3, 5)，它是通过将原数组形状的中间两个轴对调得到的。

```

a = np.random.randint(0, 10, (2, 3, 4, 5))
print u"原数组形状:", a.shape
print u"transpose:", np.transpose(a, (1, 2, 0, 3)).shape
print u"swapaxes:", np.swapaxes(a, 1, 2).shape

```

```
原数组形状: (2, 3, 4, 5)
transpose: (3, 4, 2, 5)
swapaxes: (2, 4, 3, 5)
```

下面以将多个缩略图拼成一幅大图为例，帮助读者理解多维数组中变换轴的顺序。在data/thumbnails目录之下有30个160×90像素的PNG图标图像，需要将这些图像拼成一幅6行5列的大图像。首先调用glob和cv2模块中的函数，获得一个数组列表imgs。cv2库将在第9章介绍OpenCV时进行详细介绍。

```
import glob
import numpy as np
from cv2 import imread, imwrite

imgs = [ ]

for fn in glob.glob("thumbnails/*
.png"):
    imgs.append(imread(fn, -1))

print imgs[0].shape
(90, 160, 3)
```

imgs中每个元素都是一个多维数组，它的形状为(90, 160, 3)，其中第0轴的长度为图像的高度，第1轴的长度为图像的宽度，第2轴为图像的通道数，彩色图像包含红、绿、蓝三个通道，所以第2轴的长度为3。

调用concatenate()将这些数组沿第0轴拼成一个大数组，结果img是一个宽为160像素、高为2700像素的图像：

```
img = np.concatenate(imgs, 0)
img.shape
(2700, 160, 3)
```


由于我们的最终目标是把它们拼成一幅如图2-9（左）所示的6行5列的缩略图，因此需要将img的第0轴分解为3个轴，长度分别为(6, 5, 90)。下面使用reshape()完成这个工作。使用img1[i, j]可以获取第i行、第j列上的图像：

```
img1 = img.reshape(6, 5, 90, 160, 3)
img1[0, 1].shape
(90, 160, 3)
```

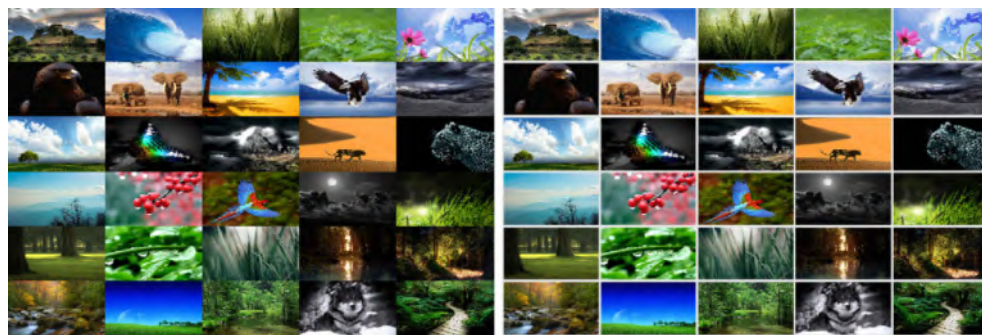


图2-9 使用操作多维数组的函数拼接多幅缩略图

根据目标图像的大小，可以算出目标数组的形状为(540, 800, 3)，即(6* 90, 5* 160, 3)，也可以把它看作形状为(6, 90, 5, 160, 3)的多维数组。与img1的形状相比，可以看出需要交换img1的第1轴和第2轴。这个操作可以通过img1.swapaxes()或img1.transpose()完成。然后再通过reshape()将数组的形状改为(540, 800, 3)。

```
img2 = img1.swapaxes(1, 2).reshape(540, 800, 3)
```

请读者思考下面的img3会得到怎样的图像：

```
img = np.concatenate(imgs, 0)
img3 = img.reshape(5, 6, 90, 160, 3) \
        .transpose(1, 2, 0, 3, 4) \
        .reshape(540, 800, 3)
```

下面的程序将每幅缩略图的边沿上的两个像素填充为白色，效果如图2-9（右）所示。❶这里使用一个形状与img1的前4个轴相同的mask布尔数组，该数组的初始值

为True。❷通过切片将mask中除去边框的部分设置为False。❸将img1中与mask为True的对应像素填充为白色。

```
img = np.concatenate(imgs, 0)
img1 = img.reshape(6, 5, 90, 160, 3)
mask = np.ones(img1.shape[:-1], dtype=bool) ❶
mask[:, :, 2:-2, 2:-2] = False ❷
img1[mask] = 230 ❸
img4 = img1.swapaxes(1, 2).reshape(540, 800, 3)
```

2.4.7 多项式函数



与本节内容对应的Notebook为：02-numpy/numpy-450-functions-poly.ipynb。

多项式函数是变量的整数次幂与系数的乘积之和，可以用下面的数学公式表示：

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0$$

由于多项式函数只包含加法和乘法运算，因此它很容易计算，可用于计算其他数学函数的近似值。多项式函数的应用非常广泛，例如在嵌入式系统中经常会用它计算正弦、余弦等函数。在NumPy中，多项式函数的系数可以用一维数组表示，例如可以用下面的数组表示，其中a[0]是最高次的系数，a[-1]是常数项，注意 x^2 的系数为0。

```
a = np.array([1.0, 0, -2, 1])
```

我们可以用poly1d()将系数转换为poly1d（一元多项式）对象，此对象可以像函数一样调用，它返回多项式函数的值：

```

p = np.poly1d(a)
print type(p)
p(np.linspace(0, 1, 5))
<class 'numpy.lib.polynomial.poly1d'>
array([ 1.          ,  0.515625,   0.125 , -0.078125,  0. ])

```

对poly1d对象进行加减乘除运算相当于对相应的多项式函数进行计算。例如：

```

p + [-2, 1] # 和 p + np.poly1d([-2, 1]) 相同
poly1d([ 1., 0., -4., 2.])

```

```

p *

p # 两个3次多项式相乘得到一个6次多项式
poly1d([ 1., 0., -4., 2., 4., -4., 1.])

```

```

p / [1, 1] # 除法返回两个多项式，分别为商式和余式
(poly1d([ 1., -1., -1.]), poly1d([ 2.]))

```

由于多项式的除法不一定能正好整除，因此它返回除法所得到的商式和余式。在上面的例子中，商式为 $x^2 - x - 1$ ，余式为2。因此将商式和被除式相乘，再加上余式就等于原来的p：

```

p == np.poly1d([ 1., -1., -1.]) *

[1,1] + 2

True

```

多项式对象的deriv()和integ()方法分别计算多项式函数的微分和积分：

```

p.deriv()
poly1d([ 3., 0., -2.])

```

```
p.integ( )
poly1d([ 0.25, 0. , -1. , 1. , 0. ])
```

```
p.integ( ).deriv( ) == p
True
```

多项式函数的根可以使用`roots()`函数计算：

```
r = np.roots(p)
r
array([-1.61803399, 1. , 0.61803399])
```

```
p(r) # 将根带入多项式计算，得到的值近似为0
array([ 2.33146835e-15, 1.33226763e-15, 1.11022302e-16])
```

而`poly()`函数可以将根转换回多项式的系数：

```
np.poly(r)
array([ 1.00000000e+00, -1.66533454e-15, -2.00000000e+00,
        1.00000000e+00])
```

除了使用多项式对象之外，也可以直接使用NumPy提供的多项式函数对表示多项式系数的数组进行运算。可以在IPython中使用自动补全查看函数名：

```
>>> np.poly # 按Tab键
np.poly np.polyadd np.polydiv np.polyint np.polysub
np.polyld np.polyder np.polyfit np.polymul np.polyval
```

其中的`polyfit()`函数可以对一组数据使用多项式函数进行拟合，找到与这组数据的误差平方和最小的多项式的系数。下面的程序用它计算 $-\pi/2 \sim \pi/2$ 区间与 $\sin(x)$ 函数最接近的多项式的系数：

```
np.set_printoptions(suppress=True, precision=4)
```

```

x = np.linspace(-np.pi / 2, np.pi / 2, 1000) ❶
y = np.sin(x) ❷

for deg in [3, 5, 7]:
    a = np.polyfit(x, y, deg) ❸
    error = np.abs(np.polyval(a, x) - y) ❹
    print "degree {}: {}".format(deg, a)
    print "max error of order %d:" % deg, np.max(error)
degree 3: [-0.145 -0.          0.9887 0. ]
max error of order 3: 0.00894699376707
degree 5: [ 0.0076 -0.         -0.1658 -0.         0.9998 -0. ]
max error of order 5: 0.000157408614187
degree 7: [-0.0002 -0.          0.0083 0.         -0.1667 -0.          1.         0. ]
max error of order 7: 1.52682558063e-06

```

❶首先通过`linspace()`将 $-\pi/2 \sim \pi/2$ 区间等分为(1000-1)等份。❷计算拟合目标函数 $\sin(x)$ 的值。❸将表示目标函数的数组传递给`polyfit()`进行拟合，第三个参数`deg`为多项式函数的最高阶数。`polyfit()`所得到的多项式和目标函数在给定的1000个点之间的误差最小，`polyfit()`返回多项式的系数数组。❹使用`polyval()`计算多项式函数的值，并计算与目标函数的差的绝对值。

从程序的输出可以看到，由于正弦函数是一个奇函数，因此拟合的多项式系数中偶数次项的系数接近于0。图2-10显示了各阶多项式与正弦函数之间的误差，请注意图中Y轴为对数坐标。

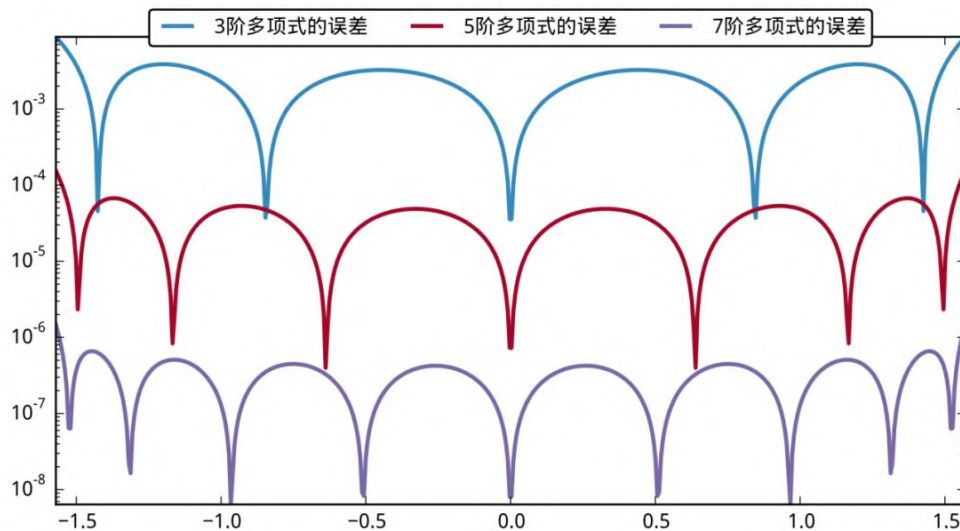


图2-10 各阶多项式近似正弦函数的误差

2.4.8 多项式函数类

numpy.polynomial模块中提供了更丰富的多项式函数类，例如Polynomial、Chebyshev、Legendre等。它们和前面介绍的numpy.poly1d相反，多项式各项的系数按照幂从小到大的顺序排列，下面使用Polynomial类表示多项式 $x^3 - 2x + 1$ ，并计算 $x=2$ 处的值：

```
from numpy.polynomial import Polynomial, Chebyshev
p = Polynomial([1, -2, 0, 1])
print p(2.0)
5.0
```

Polynomial对象提供了众多的方法对多项式进行操作，例如deriv()计算导函数：

```
p.deriv()
Polynomial([-2., 0., 3.], [-1., 1.], [-1., 1.])
```

切比雪夫多项式是一个正交多项式序列 $T_i(x)$ ，一个 n 次多项式可以表示为多个切比雪夫多项式的加权和。在NumPy中，使用Chebyshev类表示由切比雪夫多项式组成的多项式 $p(x)$ ：

$$p(x) = \sum_{i=0}^n c_i T_i(x)$$

$T_i(x)$ 多项式可以通过`Chebyshev.basis(i)`获得，图2-11显示了0到4次切比雪夫多项式。通过多项式类的`convert()`方法可以在不同类型的多项式之间相互转换，转换的目标类型由`kind`参数指定。例如下面将 $T_4(x)$ 转换成Polynomial类。由结果可知： $T_4(x) = 1 - 8x^2 + 8x^4$ 。

```
Chebyshev.basis(4).convert(kind=Polynomial)
Polynomial([ 1., 0., -8., 0., 8.], [-1., 1.], [-1., 1.])
```

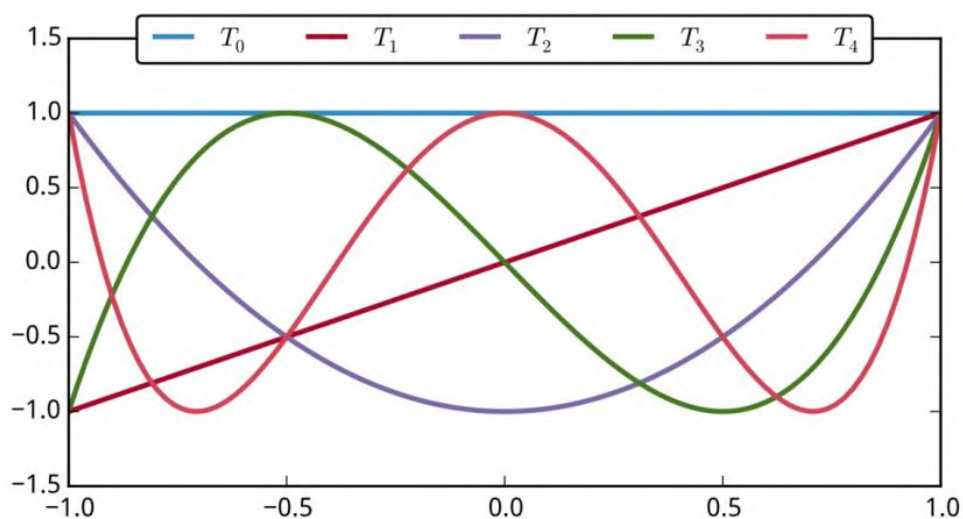


图2-11 0到4次切比雪夫多项式

切比雪夫多项式的根被称为切比雪夫节点，可以用于多项式插值。相应的插值多项式能最大限度地降低龙格现象，并且提供多项式在连续函数的最佳一致逼近。

下面以 $f(x) = \frac{1}{1+25x^2}$ 函数插值为例演示切比雪夫节点与龙格现象。

❶在 $[-1, 1]$ 区间上等距离取 n 个取样点。❷使用 n 阶切比雪夫多项式的根作为取样点。❸使用两种取样点分别对 $f(x)$ 进行多项式插值，即计算一个多项式经过所有的插值点。图2-12显示了两种插值点所得到的插值多项式，由左图可知等距离插值多项式在两端有非常大的振荡，这种现象被称为龙格现象， n 越大振荡也越大；而右图采用切比雪夫节点作为插值点，插值多项式的振荡明显减小，并且 n 越大振荡越小。

插值与拟合

所谓多项式插值就是找到一个多项式经过所有的插值点。一个 n 阶多项式有 $n+1$ 个系数，因此可以通过解方程求解经过 $n+1$ 个插值点的 n 阶多项式的系数。`fit()`方法虽然计算与目标点拟合的多项式系数，但是当使用 n 阶多项式拟合 $n+1$ 的目标点时，多项式将经过所有目标点，因此其结果与多项式插值相同。

```
def f(x):  
    return 1.0 / ( 1 + 25 *  
  
x**  
2)  
  
n = 11  
x1 = np.linspace(-1, 1, n) ❶  
x2 = Chebyshev.basis(n).roots( ) ❷  
xd = np.linspace(-1, 1, 200)  
  
c1 = Chebyshev.fit(x1, f(x1), n - 1, domain=[-1, 1]) ❸
```



```

c2 = Chebyshev.fit(x2, f(x2), n - 1, domain=[-1, 1])

print u"插值多项式的最大误差: ",
print u"等距离取样点: ", abs(c1(xd) - f(xd)).max( ),
print u"切比雪夫节点: ", abs(c2(xd) - f(xd)).max( )
插值多项式的最大误差: 等距离取样点:  1.91556933029 切比雪夫节点:  0.109149825014

```

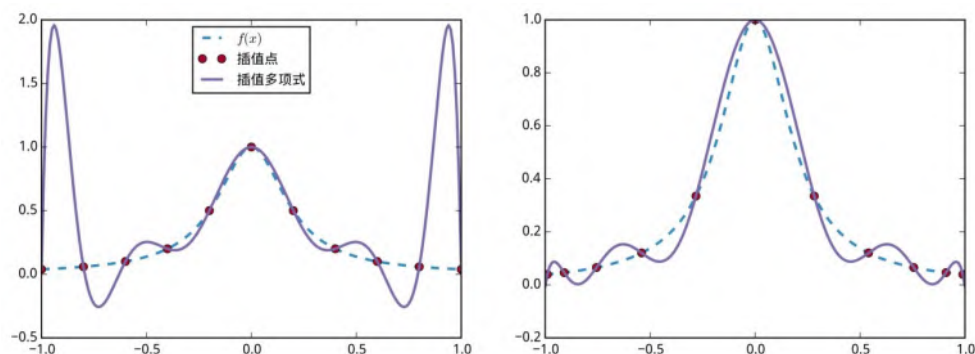


图2-12 等距离插值点（左）、切比雪夫插值点（右）

在使用多项式逼近函数时，使用切比雪夫多项式进行插值的误差比一般多项式要小许多。在下面的例子中，对 $g(x)$ 在100个切比雪夫节点之上分别使用Polynomial和Chebyshev进行插值，结果如图2-13所示。在使用Polynomial.fit()插值时，产生了RankWarning: The fit may be poorly conditioned警告，因此其结果多项式未能经过所有插值点。

```

def g(x):
    x = (x - 1) *

5
    return np.sin(x**)

2) + np.sin(x)**

2

n = 100

```

```

x = Chebyshev.basis(n).roots( )
xd = np.linspace(-1, 1, 1000)

p_g = Polynomial.fit(x, g(x), n - 1, domain=[-1, 1])
c_g = Chebyshev.fit(x, g(x), n - 1, domain=[-1, 1])

print "Max Polynomial Error:", abs(g(xd) - p_g(xd)).max( )
print "Max Chebyshev Error:", abs(g(xd) - c_g(xd)).max( )

Max Polynomial Error: 1.19560558744
Max Chebyshev Error: 6.47575726376e-09

```

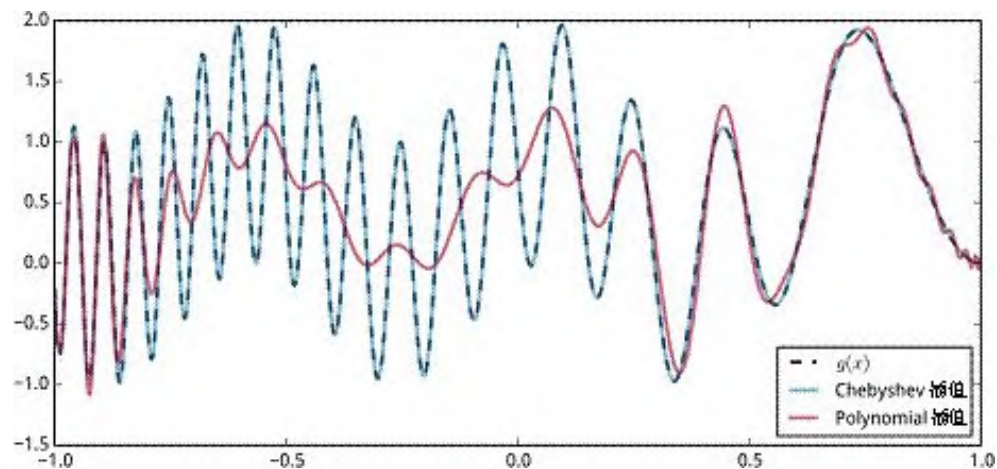


图2-13 Chebyshev插值与Polynomial插值比较

trim()方法可以降低多项式的次数，将尾部绝对值小于参数tol的高次系数截断。下面使用trim()方法获取c中前68个系数，得到一个新的Chebyshev对象c_trimed，其最大误差上升到0.09左右。

```

c_trimed = c_g.trim(tol=0.05)
print "degree:", c_trimed.degree( )
print "error:", abs(g(xd) - c_trimed(xd)).max( )

degree: 68
error: 0.0912094835458

```

下面用同样的方法对函数h(x)进行19阶的切比雪夫多项式插值，得到插值多项式c_h:

```

def h(x):
    x = 5 *

x

    return np.exp(-x**

2 / 10)

n = 20
x = Chebyshev.basis(n).roots( )
c_h = Chebyshev.fit(x, h(x), n - 1, domain=[-1, 1])

print "Max Chebyshev Error:", abs(h(xd) - c_h(xd)).max( )
Max Chebyshev Error: 1.66544267266e-09

```

多项式类支持四则运算，下面将`c_g`和`c_h`相减得到`c_diff`，并调用其`roots( )`计算其所有根。然后找出其中所有的实数根`real_roots`，它们就是 $g(x)$ 与 $h(x)$ 交点的横坐标。图2-14显示了这两条函数曲线以及通过插值多项式计算的交点：

```

c_diff = c_g - c_h
roots = c_diff.roots( )
real_roots = roots[roots.imag == 0].real
print np.allclose(c_diff(real_roots), 0)
True

```

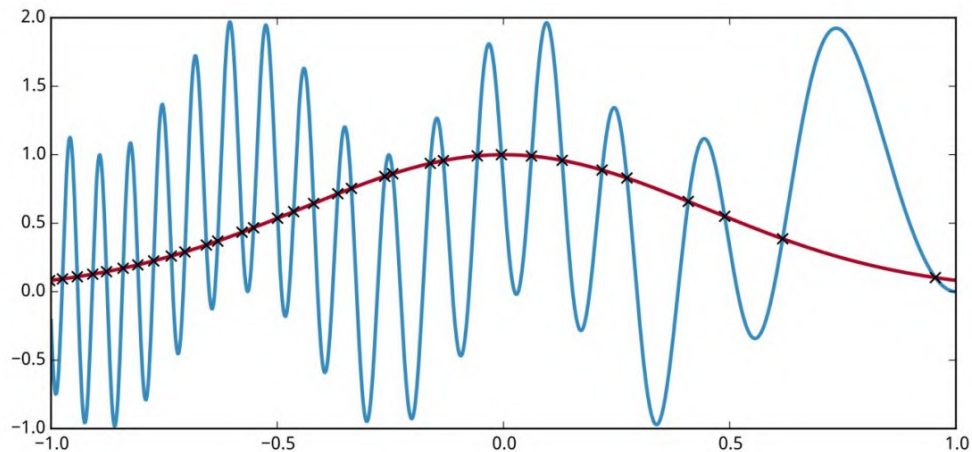


图2-14 使用Chebyshev插值计算两条曲线在 $[-1, 1]$ 之间的所有交点

切比雪夫多项式在区间 $[-1, 1]$ 上为正交多项式，因此只有在该区间才能对目标函数正确插值。为了对任何区域的目标函数进行插值，需要对自变量的区间进行缩放和平移变换。可以通过domain参数指定拟合点的区间。在下面的例子中，对 $g_2(x)$ 在区间 $[-10, 0]$ 之内使用切比雪夫多项式进行插值。❶为了产生目标区间的切比雪夫节点，在通过basis()方法创建 $T_n(x)$ 时，通过domain参数指定目标区间。❷在调用fit()方法进行拟合时，通过domain参数指定同样的区间。❸最后输出拟合得到的c_g2多项式在 $[-10, 0]$ 中与目标函数的最大误差。

```
def g2(x):
    return np.sin(x)**2) + np.sin(x)**2

n = 100
x = Chebyshev.basis(n, domain=[-10, 0]).roots( ) ❶
xd = np.linspace(-10, 0, 1000)

c_g2 = Chebyshev.fit(x, g2(x), n - 1, domain=[-10, 0]) ❷
```

```
print "Max Chebyshev Error:", abs(g2(xd) - c_g2(xd)).max( ) ❸
Max Chebyshev Error: 6.47574571744e-09
```

2.4.9 各种乘积运算



与本节内容对应的Notebook为：[02-numpy/numpy-460-functions-dot.ipynb](#)。

本节介绍的函数如表2-9所示。

表2-9 本节要介绍的函数

函数名	功能	函数名	功能
dot	矩阵乘积	inner	内积
outter	外积	tensordot	张量乘积

矩阵的乘积可以使用`dot()`计算。对于二维数组，它计算的是矩阵乘积；对于一维数组，它计算的是内积。当需要将一维数组当作列矢量或行矢量进行矩阵运算时，先将一维数组转换为二维数组：

```
a = np.array([1, 2, 3])
a[:, None] a[None, :]
-----
[[1], [[1, 2, 3]]
 [2],
 [3]]
```

对于多维数组，`dot()`的通用计算公式如下，即结果数组中的每个元素都是：数组a的最后轴上的所有元素与数组b的倒数第二轴上的所有元素的乘积和：

```
dot(a, b)[i,j,k,m] = sum(a[i,j,:] *
b[k,:,m])
```

下面以两个三维数组的乘积演示dot()的计算结果。首先创建两个三维数组，这两个数组的最后两轴满足矩阵乘积的条件：

```
a = np.arange(12).reshape(2, 3, 2)
b = np.arange(12, 24).reshape(2, 2, 3)
c = np.dot(a, b)
c.shape
(2, 3, 2, 3)
```

c是数组a和b的多个子矩阵的乘积。我们可以把数组a看作两个形状为(3, 2)的矩阵，而把数组b看作两个形状为(2, 3)的矩阵。a中的两个矩阵分别与b中的两个矩阵进行矩阵乘积，就得到数组c，c[i, :, j, :]是a中第i个矩阵与b中第j个矩阵的乘积。

```
for i, j in np.ndindex(2, 2):
    assert np.alltrue( c[i, :, j, :] == np.dot(a[i], b[j]) )
```

对于两个一维数组，inner()和dot()一样，计算两个数组对应下标元素的乘积和。而对于多维数组，它计算的结果数组中的每个元素都是：数组a和b的最后轴的内积。因此数组a和b的最后轴的长度必须相同：

```
inner(a, b)[i,j,k,m] = sum(a[i,j,:] *
b[k,m,:])
```

下面是对inner()的演示：

```
a = np.arange(12).reshape(2, 3, 2)
b = np.arange(12, 24).reshape(2, 3, 2)
c = np.inner(a, b)
```

```
c.shape
(2, 3, 2, 3)
```

```
for i, j, k, l in np.ndindex(2, 3, 2, 3):
    assert c[i, j, k, l] == np.inner(a[i, j], b[k, l])
```

`outer()` 只对一维数组进行计算，如果传入的是多维数组，则先将此数组展平为一维数组之后再行运算。它计算列向量和行向量的矩阵乘积：

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6, 7])
np.outer(a, b)      np.dot(a[:, None], b[None, :])
-----
[[ 4, 5, 6, 7],      [[ 4, 5, 6, 7],
 [ 8, 10, 12, 14],    [ 8, 10, 12, 14],
 [12, 15, 18, 21]]    [12, 15, 18, 21]]
```

`tensordot()` 将两个多维数组 `a` 和 `b` 指定轴上的对应元素相乘并求和，它是最一般化的乘积运算函数。下面通过一些例子逐步介绍其用法。下面计算两个矩阵的乘积：❶ `axes` 参数有两个元素，第一个元素表示 `a` 中的轴，第二个元素表示 `b` 中的轴，这两个轴上对应的元素相乘之后求和。❷ `axes` 也可以是一个整数，它表示把 `a` 中的后 `axes` 个轴和 `b` 中的前 `axes` 个轴进行乘积和运算，而对于乘积和之外的轴则保持不变。

```
a = np.random.rand(3, 4)
b = np.random.rand(4, 5)

c1 = np.tensordot(a, b, axes=[[1], [0]]) ❶
c2 = np.tensordot(a, b, axes=1)           ❷
c3 = np.dot(a, b)
assert np.allclose(c1, c3)
assert np.allclose(c2, c3)
```

对于多维数组的`dot()`乘积，可以用`tensordot(a, b, axes=[[-1], [-2]])`表示，即将a的最后轴和b中的倒数第二轴求乘积和：

```
a = np.arange(12).reshape(2, 3, 2)
b = np.arange(12, 24).reshape(2, 2, 3)
c1 = np.tensordot(a, b, axes=[[-1], [-2]])
c2 = np.dot(a, b)
assert np.alltrue(c1 == c2)
```

在下面的例子中，将a的第1、第2轴与b的第1、第0轴求乘积和，因此c中的每个元素都是按照如下表达式计算的：

```
c[i, j, k, l] = np.sum(a[i, :, :, j] *
b[:, :, k, l].T)
```

注意由于b对应的axes中的轴是倒序的，因此需要做转置操作。

```
a = np.random.rand(4, 5, 6, 7)
b = np.random.rand(6, 5, 2, 3)
c = np.tensordot(a, b, axes=[[1, 2], [1, 0]])

for i, j, k, l in np.ndindex(4, 7, 2, 3):
    assert np.allclose(c[i, j, k, l], np.sum(a[i, :, :, j] *
b[:, :, k, l].T))

c.shape
(4, 7, 2, 3)
```

2.4.10 广义ufunc函数



与本节内容对应的Notebook为：02-numpy/numpy-470-gufuncs.ipynb。

从NumPy 1.8开始正式支持广义ufunc函数(generalized ufunc，以下简称gufunc)。gufunc是对ufunc的推广，所谓ufunc就是将对单个数值的运算通过广播运用到整个数组中的所有元素之上，而gufunc则是将对单个矩阵的运算通过广播运用到整个数组之上。例如numpy.linalg.inv()是求逆矩阵的gufunc函数。在其文档中描述其输入输出数组的形状如下：

```
ainv = inv(a)
a : (... , M, M)
ainv : (... , M, M)
```

输入数组a的形状中带有“...”，它表示0到任意多个轴。当它为空时，就是对单个矩阵求逆，gufunc函数将对这些轴进行广播运算。最后两个轴的长度为M，表示任意大小的方形矩阵。



NumPy中的线性代数模块linalg中提供的函数大都为广义ufunc函数。在SciPy中也提供了线性代数模块linalg，但其中的函数都是一般函数，只能对单个矩阵进行计算。关于线性代数函数库的用法将在下一章进行详细介绍。

在输出数组ainv中，由于逆矩阵的形状与原矩阵相同，因此ainv的最后两轴的形状也是(M, M)。“...”表示广播运算之后的形状，而由于矩阵求逆只对一个矩阵进行运算，因此“...”的形状和a中的“...”的形状相同。

在下面的例子中，a的形状为(10, 20, 3, 3)，其中(10, 20)与“...”对应，3与M对应。而inv()通过广播运算对10×20个形状为(3, 3)的矩阵求逆。得到的结果ainv的形状与a相同，也是(10, 20, 3, 3)。

```
a = np.random.rand(10, 20, 3, 3)
ainv = np.linalg.inv(a)
```

```
ainv.shape  
(10, 20, 3, 3)
```

下面的程序验证第*i*行、第*j*列的矩阵及其逆矩阵的乘积，应该近似等于3阶单位矩阵：

```
i, j = 3, 4  
np.allclose(np.dot(a[i, j], ainv[i, j]), np.eye(3))  
True
```

`numpy.linalg.det()` 计算矩阵的行列式，它也是一个gufunc函数。它的输入输出的形状为：

```
adet = det(a)  
a : (... , M, M)  
adet : (...)
```

由于矩阵的行列式是将一个*M*×*M*的矩阵映射到一个标量，因此输出*adet*的形状中只包含“...”。

```
adet = np.linalg.det(a)  
adet.shape  
(10, 20)
```

下面以多个二次函数的数据拟合为例，介绍如何使用gufunc函数提高运算效率。首先通过随机数函数创建测试用的数据*x*和*y*，这两个数组的形状都为(*n*, 10)。其中的每行数据(*x*[*i*]和*y*[*i*])构成一个曲线拟合的数据集，它们的关系为： $y = \beta_2 + \beta_1 x + \beta_0 x^2$ 。现在需要计算每对数据所对应的系数 β 。

```
n = 10000  
np.random.seed(0)  
beta = np.random.rand(n, 3)  
x = np.random.rand(n, 10)  
y = beta[:, 2, None] + x*
```

```

beta[:, 1, None] + x**
2*
beta[:, 0, None]

```

显然使用前面介绍过的`numpy.polyfit()`可以很方便地完成这个任务，下面的程序输出第42组的实际系数以及拟合的结果：

```

print beta[42]
print np.polyfit(x[42], y[42], 2)
[ 0.0191932  0.30157482  0.66017354]
[ 0.0191932  0.30157482  0.66017354]

```

只需要循环调用`n`次`numpy.polyfit()`即可得到所需的结果，但是它的运算速度有些慢：

```

%time beta2 = np.vstack([np.polyfit(x[i], y[i], 2) for i in range(n)])
Wall time: 1.52 s

```

```

np.allclose(beta, beta2)
True

```

在`numpy.polyfit()`内部实际上是通过调用最小二乘法函数`numpy.linalg.lstsq()`来实现多项式拟合的，我们也可以直接调用`lstsq()`计算系数：

```

xx = np.column_stack(([x[42]**
2, x[42], np.ones_like(x[42])]))
print np.linalg.lstsq(xx, y[42])[0]
[ 0.0191932  0.30157482  0.66017354]

```

但遗憾的是，目前`numpy.linalg.lstsq()`还不是gufunc函数，因此无法直接使用它计算所有数据组的拟合系数。然而`numpy.linalg`中对线性方程组求解的函数`solve()`是一个gufunc函数。并且根据最小二乘法的公式：

$$\beta = (X^T X)^{-1} X^T y$$

只需要求出 $X^T X$ 和 $X^T y$ ，就可以使用`numpy.linalg.solve()`计算出 $\beta = \text{solve}(X^T X, X^T y)$ 。为了实现这个运算，还需要计算矩阵乘积的gufunc函数。然而`dot()`并不是一个gufunc函数，因为它不遵循广播规则。NumPy中目前还没有正式提供计算矩阵乘积的gufunc函数，不过在`umath_tests`模块中提供了一个测试用的函数：`matrix_multiply()`。下面的程序使用它和`solve()`实现高速多项式拟合运算，它所需的时间约为`polyfit()`版本的五十分之一。

```
%%time

X = np.dstack([x**
2, x, np.ones_like(x)])
Xt = X.swapaxes(-1, -2)

import numpy.core.umath_tests as umath
A = umath.matrix_multiply(Xt, X)
b = umath.matrix_multiply(Xt, y[..., None]).squeeze()

beta3 = np.linalg.solve(A, b)

print np.allclose(beta3, beta2)
True
Wall time: 30 ms
```

在上面的运算中， X 的形状为(10000, 10, 3)， Xt 的形状为(10000, 3, 10)。`matrix_multiply()`的各个参数和返回值的形状如下：

```
c = matrix_multiply(a, b)
a : (... , M, N)
```

```
b : (... , N, K)
c : (... , M, K)
```

调用`matrix_multiply()`对`Xt`和`X`中的每对矩阵进行乘积运算，得到的结果`A`的形状为`(10000, 3, 3)`。而为了计算 $X^T y$ ，需要通过`y[..., None]`将`y`变成形状为`(10000, 10, 1)`的数组。`matrix_multiply(Xt, y[..., None])`所得到的形状为`(10000, 3, 1)`。调用其`squeeze()`，删除长度为1的轴。这样`b`的形状为`(10000, 3)`。

`solve()`的参数`b`支持两种形状，其中第一种情况的形状如下：

```
x = solve(a, b)
a : (... , M, M)
b : (... , M)
x : (... , M)
```

因此`solve()`的返回值`beta3`的形状为`(10000, 3)`。

前面的例子中，使用的都是最简单的广播规则。实际上`gufunc`函数支持所有的`ufunc`函数的广播规则。因此形状分别为`(a, m, n)`和`(b, 1, n, k)`的两个数组通过`matrix_multiply()`乘积之后得到的数组的形状为`(b, a, m, k)`。下面看一个使用`gufunc`函数广播运算的例子：

在二维平面上的旋转矩阵为：

$$M(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

它可将平面上的某点的坐标围绕原点旋转 θ 。对于形状为`(N, 2)`的矩阵`P`，可以表示平面上`N`个点的坐标。而矩阵乘积得到的则是将这`N`个点绕坐标原点旋转 θ 之后的坐标。下面的程序使用`matrix_multiply()`将3条曲线上的坐标点分别旋转4个角度，得到12条曲线。

调用`matrix_multiply()`时两个参数数组的形状分别为`(3, 100, 2)`和`(4, 1, 2, 2)`，其中广播轴的形状分别为`(3,)`和`(4, 1)`，运算轴的形状分别为`(100, 2)`和

(2, 2)。广播轴进行广播之后的形状为(4, 3)，而运算轴进行矩阵乘积之后的形状为(100, 2)，因此结果rpoints的形状为(4, 3, 100, 2)。

```
M = np.array([[[np.cos(t), -np.sin(t)],
                [np.sin(t), np.cos(t)]]
               for t in np.linspace(0, np.pi, 4, endpoint=False)])

x = np.linspace(-1, 1, 100)
points = np.array((np.c_[x, x], np.c_[x, x**
3], np.c_[x**
3, x]))

rpoints = umath.matrix_multiply(points, M[:, None, ...])

print points.shape, M.shape, rpoints.shape
(3, 100, 2) (4, 2, 2) (4, 3, 100, 2)
```

将这12条曲线绘制成图表之后的效果如图2-15所示。

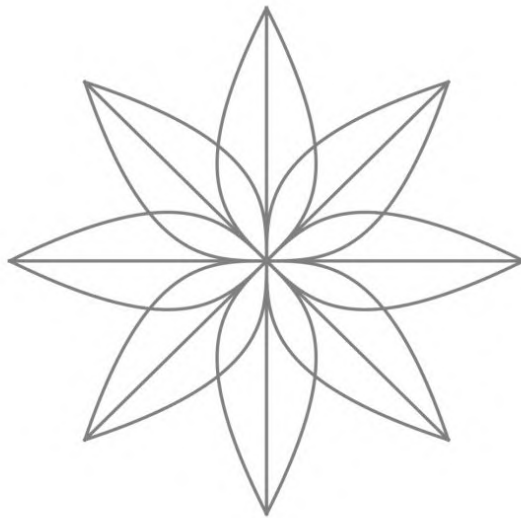


图2-15 使用矩阵乘积的广播运算将3条曲线分别旋转4个角度

2.5 实用技巧



与本节内容对应的Notebook为：[02-numpy/numpy-900-tips.ipynb](#)。

在编写应用程序时，可能经常会与其他的程序库交换大量数据。本节介绍一些通过NumPy数组共享内存的方法。

2.5.1 动态数组

NumPy的数组对象不能像列表一样动态地改变其大小，在做数据采集的时候，需要频繁地往数组中添加数据时很不方便。而Python标准库中的array数组提供了动态分配内存的功能，而且它和NumPy数组一样直接将数值的二进制数据保存在一块内存中，因此我们可以先用array数组收集数据，然后通过`np.frombuffer()`将array数组的数据内存直接转换为NumPy数组。下面是一个例子：

```
import numpy as np
from array import array
a = array("d", [1,2,3,4])    # 创建一个array数组
# 通过np.frombuffer()创建一个和a共享内存的NumPy数组
na = np.frombuffer(a, dtype=np.float)
print a
print na
na[1] = 20    # 修改NumPy数组中下标为1的元素
print a
array('d', [1.0, 2.0, 3.0, 4.0])
[ 1.  2.  3.  4.]
array('d', [1.0, 20.0, 3.0, 4.0])
```

array数组只支持一维，如果我们需要采集多个通道的数据，可以将这些数据依次添加进array数组，然后通过`reshape()`方法将`np.frombuffer()`所创建的NumPy数组改为二维数组。在下面的例子中，我们通过array数组buf采集两个通道的数据，数据采集完毕之后，通过`np.frombuffer()`将其转换为NumPy数组，并通过`reshape()`将其形状改为二维数组：

```
import math
buf = array("d")
for i in range(5):
    buf.append(math.sin(i*
0.1))
    buf.append(math.cos(i*
0.1))

data = np.frombuffer(buf, dtype=np.float).reshape(-1, 2)
print data
[[ 0.          1.          ]
 [ 0.09983342  0.99500417]
 [ 0.19866933  0.98006658]
 [ 0.29552021  0.95533649]
 [ 0.38941834  0.92106099]]
```

下面是Python中实现array对象动态添加元素的算法：

- array对象拥有一块用于保存数据的内存，其长度通常比数组中的所有数据的字节数要长。
- 当往array中添加数据时，如果数据内存中还有空余位置，则直接写入空余位置。
- 当数据内存中无空余位置时，则重新分配一块更大的数据内存，并将当前的数据都复制到这块新的数据内存中，而旧的数据内存则被释放掉。

根据上述算法可知，只要往array中添加元素，其数据内存的地址就可能发生改变。在此之前通过np.frombuffer()创建的数组仍然引用旧的数据内存，从而成为“野指针”。下面的代码演示了这个过程。其中array.buffer_info()获得数据内存的地址以及其中有效数据的个数。

```
a = array("d")
for i in range(10):
    a.append(i)
    if i == 2:
        na = np.frombuffer(a, dtype=float)
        print a.buffer_info(),
    if i == 4:
        print
(83088512, 1) (83088512, 2) (83088512, 3) (83088512, 4) (31531848, 5)
(31531848, 6) (31531848, 7) (31531848, 8) (34405776, 9) (34405776, 10)
```

由上面的结果可知，当数组a的长度为5和9时，数据内存被重新分配了。而na数组是在a的长度为3时通过np.frombuffer()得到的，因此它的数据指针已经成为野指针。ndarray.ctypes.data可以获得数组的数据内存的地址，可以看出na的数据内存地址仍然是a在重新分配之前的地址，而na中的数据也变成了随机的无效数据。

```
print na.ctypes.data
print na
83088512
[ 2.11777767e+161  6.24020631e-085  8.82069697e+199]
```

由上面的分析可知，每次动态数组的长度改变时，我们都需要重新调用np.frombuffer()以创建一个新的ndarray数组对象来访问其中的数据。

当每个通道的数据类型不同时，就不能采用array.array对象了。这时可以使用bytearray收集数据。bytearray是字节数组，因此首先需要通过struct模块将Python的数值转换成其字节表示形式。如果数据来自二进制文件或硬件，那么很可能得到的已经是字节数据了，这个步骤可以省略。下面是使用bytearray进行数据采集的例子：



bytearray对象的+=运算与其extend()方法的功能相同，但+=的运行速度要比extend()快许多，读者可以使用%timeit自行验证。

```
import struct
buf = bytearray( )
for i in range(5):
    buf += struct.pack("=hdd", i, math.sin(i*
0.1), math.cos(i*
0.1)) ❶

    dtype = np.dtype({"names":["id","sin","cos"], "formats":["h", "d",
"d"]}) ❷
data = np.frombuffer(buf, dtype=dtype) ❸
print data
[(0, 0.0, 1.0) (1, 0.09983341664682815, 0.9950041652780258)
(2, 0.19866933079506122, 0.9800665778412416)
(3, 0.2955202066613396, 0.955336489125606)
(4, 0.3894183423086505, 0.9210609940028851)]
```

❶采集三个通道的数据，其中通道1是短整型数，其类型符号为“h”，通道2和3为双精度浮点数，其类型符号为“d”。类型格式字符串中的“=”表示输出的字节数据不进行内存对齐。即一条数据的字节数为 $2+8+8=18$ ，如果没有“=”，那么一条数据的字节数为 $8+8+8=24$ 。

❷定义一个dtype对象来表示一条数据的结构，dtype对象默认不进行内存对齐。如果采集数据用的bytearray中的数据是内存对齐的话，只需要设置dtype()的align参数为True即可。

③最后通过`np.frombuffer()`将`bytearray`转换为NumPy的结构数组。然后就可以通过`data["id"]`、`data["sin"]`和`data["cos"]`访问这三个通道的数据了。

2.5.2 和其他对象共享内存

在前面的章节中介绍过，当其他对象提供了获取其内部数据存取区的接口时，可以用`numpy.frombuffer()`创建一个数组与此对象共享数据内存。如果对象没有提供该接口，但是能够获取数据存储区的地址，可以通过`ctypes`和`numpy.ctypeslib`模块中提供的函数，创建与对象共享内存的数组。下面以PyQt4中的`QImage`对象为例，介绍如何创建一个与`QImage`对象共享内存的数组。

首先创建一个`QImage`对象，并载入“lena.png”文件中的内容。然后输出与图像相关的一些信息，为了创建与该图像共享内存的数组，我们需要使用这些信息。

```
from PyQt4.QtGui import QImage, qRgb
img = QImage("lena.png")
print "width & height:", img.width(), img.height()
print "depth:", img.depth() #每个像素的比特数
print "format:", img.format(), QImage.Format_RGB32
print "byteCount:", img.byteCount() #图像的总字节数
print "bytesPerLine:", img.bytesPerLine() #每行的字节数
print "bits:", int(img.bits()) #图像第一个字节的地址
width & height: 512 393
depth: 32
format: 4 4
byteCount: 804864
bytesPerLine: 2048
bits: 156041248
```

①由于我们只知道数据的地址，首先需要使用`ctypes.cast()`将整数转换为一个指向单字节类型的指针。②然后使用`numpy.ctypeslib.as_array()`将`ctypes`的指

针指向的内存转换成NumPy的数组。as_array()的第二个参数是该数组的形状，注意数组的第0轴为图像的高，第1轴为图像的宽，第2轴为每个像素的字节数。

```
import ctypes
addr = int(img.bits( ))
pointer = ctypes.cast(addr, ctypes.POINTER(ctypes.c_uint8)) ❶
arr = np.ctypeslib.as_array(pointer, (img.height( ), img.width( ),
img.depth( )//8)) ❷
```

下面通过arr数组和img对象查看位于像素坐标(50, 100)处的像素颜色值，可以看到二者是完全相同的：

```
x, y = 100, 50
b, g, r, a = arr[y, x]
print qRgb(r, g, b)
print img.pixel(x, y)
4289282380
4289282380
```

下面通过arr数组修改颜色值，并通过img对象查看修改的结果，由结果可知二者的确共享着同一块内存：

```
arr[y, x, :3] = 0x12, 0x34, 0x56
print hex(img.pixel(x, y))
0xff563412L
```

使用上述方法共享内存时需注意必须保持目标对象处于可访问状态。例如在上例中，如果执行del img语句引起img对象被垃圾回收，则通过arr数组将访问被释放掉的内存区域。为了解决这个问题，可以让数组的base属性引用目标对象，这样只要数组不被释放，则目标对象也不会被释放。为了能正确设置base属性，需要使用数组的__array_interface__接口。

❶ 在调用 array() 将目标对象转换成数组时，如果目标对象拥有__array_interface__属性，则根据该属性的描述创建数组。它是一个具有特定键值

的字典，参见表2-10。

表2-10 键值及含义

键值	含义
shape	所创建数组的形状
data	数据存储区的首地址，以及是否只读
strides	数组的strides属性
typestr	元素类型描述符
descr	如果创建结构数组，该键描述结构体各个字段名以及对应的数据类型
version	固定为3

❷设置copy参数为False，这样所创建的数组与目标对象共享内存，否则将复制目标对象的内存。❸在创建完数组之后，可以删除__array_interface__属性。❹所得到的数组arr2与arr相同，并且其base属性为img对象。

```
interface = {
    'shape': (img.height( ), img.width( ), 4),
    'data': (int(img.bits( )), False),
    'strides': (img.bytesPerLine( ), 4, 1),
    'typestr': "|u1",
    'version': 3,
}

img.__array_interface__ = interface ❶

arr2 = np.array(img, copy=False) ❷

del img.__array_interface__ ❸
```

```
print np.all(arr2 == arr), arr2.base is img ④
True True
```

如果目标对象只读，无法为其添加`__array_interface__`属性，可以创建一个代理用的`ArrayProxy`对象，在该代理对象中引用目标对象，使其不会被垃圾回收，同时提供`__array_interface__`属性，以供创建相应的数组。

```
class ArrayProxy(object):
    def __init__(self, base, interface):
        self.base = base
        self.__array_interface__ = interface

arr3 = np.array(ArrayProxy(img, interface), copy=False)
print np.all(arr3 == arr)
True
```

2.5.3 与结构数组共享内存

从结构数组获取某个字段时，得到的是原数组的视图，但是如果获取多个字段，将得到一个全新的数组，不与原数组共享内存。

```
persontype = np.dtype({
    'names': ['name', 'age', 'weight', 'height'],
    'formats': ['S30', 'i', 'f', 'f']}, align= True )
a = np.array([("Zhang", 32, 72.5, 167.0),
              ("Wang", 24, 65.2, 170.0)], dtype=persontype)

print a["age"].base is a #视图
print a[["age", "height"]].base is None #复制
True
True
```

为了创建结构数组的多字段视图，可以使用下面的`fields_view()`函数。它通过原数组的`dtype`属性创建视图数组的`dtype`对象。然后通过`ndarray()`创建视图数组。

```
def fields_view(arr, fields):
    dtype2 = np.dtype({name:arr.dtype.fields[name] for name in fields})
    return np.ndarray(arr.shape, dtype2, arr, 0, arr.strides)

v = fields_view(a, ["age", "weight"])
print v.base is a

v["age"] += 10
print a
True
[('Zhang', 42, 72.5, 167.0) ('Wang', 34, 65.19999694824219, 170.0)]
```

`dtype`对象的`fields`属性是一个以字段名为键、以字段类型和字节偏移量为值的字典，使用它创建新的`dtype`对象时，可以保持字段的偏移量：

```
print a.dtype.fields
print a.dtype
print v.dtype
{'age': (dtype('int32'), 32), 'name': (dtype('S30'), 0),
 'weight': (dtype('float32'), 36), 'height': (dtype('float32'), 40)}
      {'names': ['name', 'age', 'weight', 'height'],      'formats':
['S30', '<i4', '<f4', '<f4'],
      'offsets': [0, 32, 36, 40], 'itemsizes': 44, 'aligned': True}
      {'names': ['age', 'weight'],      'formats': ['<i4', '<f4'],      'offsets':
[32, 36], 'itemsizes': 40}
```

如果这两个`dtype`对象的`itemsizes`属性相同，那么可以使用数组的`view()`方法创建视图对象。但是从上面的输出可以看到两个`dtype`对象的字节数并不相同，一个

是44个字节，另一个是40个字节。遇到这种情况时，可以使用`ndarray()`创建数组的视图，它的调用参数如下：

```
ndarray(shape, dtype=float, buffer=None, offset=0, strides=None,
order=None)
```

- `shape`: 所创建数组的形状。
- `dtype`: 数组元素类型的`dtype`对象。
- `buffer`: 拥有`buffer`接口的对象，所创建的数组将与该对象共享内存。
- `offset`: `buffer`对象的数据内存中的起始地址的偏移量。
- `strides`: 所创建数组的`strides`属性，即每个轴上的下标增加1时的地址增量。
- `order`: C语言格式或Fortran语言格式。

在`fields_view()`中，我们所创建的数组视图与原数组拥有相同的`shape`、`data`和`strides`属性。而`dtype`属性中的字段与原数组拥有相同的偏移量，显然这样的新数组能够与原数组共享内存。

第3章 SciPy-数值计算库

SciPy在NumPy的基础上增加了众多的数学计算、科学计算以及工程计算中常用的模块，例如线性代数、常微分方程数值求解、信号处理、图像处理、稀疏矩阵等。在本章中，将通过实例介绍SciPy中常用的一些模块。为了方便读者理解，在实例程序中会使用matplotlib、TVTK以及Mayavi等扩展库绘制二维和三维的图表。在阅读实例的源程序时，读者可以忽略绘图部分，在后续的章节中，我们会对这些绘图库进行详细介绍。

本章的所有实例程序都在SciPy 0.15下调试通过，请读者在运行之前检查SciPy的版本：

```
import scipy
scipy.__version__
'0.15.0'
```

3.1 常数和特殊函数



与本节内容对应的Notebook为：03-scipy/scipy-100-intro.ipynb。

SciPy的constants模块包含了众多的物理常数：

```
from scipy import constants as C
print C.c # 真空中的光速
print C.h # 普朗克常数
```

```
299792458.0
6.62606957e-34
```

在字典`physical_constants`中，以物理常量名为键，对应的值是一个含有三个元素的元组，分别为常数值、单位以及误差，例如下面的程序用来查看电子的质量：

```
C.physical_constants["electron mass"]
(9.10938291e-31, 'kg', 4e-38)
```

除了物理常数之外，`constants`模块中还包括许多单位信息，它们是1单位的量转换成标准单位时的数值：

```
# 1英里等于多少米，1英寸等于多少米，1克等于多少千克，1磅等于多少千克
      C.mile      C.inch  C.gram      C.pound
-----
1609.3439999999998  0.0254  0.001    0.45359236999999997
```

SciPy的`special`模块是一个非常完整的函数库，其中包含了基本数学函数、特殊数学函数以及NumPy中出现的所有函数。由于函数数量众多，本节仅对其进行简要介绍。至于其具体所包含的函数列表，请读者参考SciPy的帮助文档。

伽玛(gamma)函数 Γ 是概率统计学中经常出现的一个特殊函数，它的计算公式如下：

$$\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t} dt$$

显然，要通过此公式计算 $\Gamma(z)$ 的值是比较麻烦的，可以用`special`模块中的`gamma()`进行计算：

```
import scipy.special as S
print S.gamma(4)
print S.gamma(0.5)
print S.gamma(1+1j) # gamma函数支持复数
print S.gamma(1000)

6.0
1.77245385091
(0.498015668118-0.154949828302j)
inf
```

$\Gamma(z)$ 函数是阶乘函数在实数和复数系上的扩展，它的增长速度非常快，1000的阶乘已经超过了双精度浮点数的表示范围，因此结果是无穷大。为了计算更大的范围，可以使用 `gammaaln()` 计算 $\ln(|\Gamma(x)|)$ 的值，它使用特殊的算法，能直接计算 Γ 函数的对数值，因此可以表示更大的范围。

```
S.gammaln(1000)
5905.2204232091808
```

`special` 模块中的某些函数并不是数学意义上的特殊函数，例如 `loglp(x)` 计算 $\log(1+x)$ 的值。这是由于浮点数的精度有限，无法很精确地表示非常接近1的实数。例如无法用浮点数表示 $1 + 1e-20$ 的值，因此 $\log(1+1e-20)$ 的值为0，而当使用 `loglp()` 时，则可以很精确地计算：

```
print 1 + 1e-20
print np.log(1+1e-20)
print S.loglp(1e-20)

1.0
0.0
1e-20
```

实际上当 x 非常小时， $\log_{1p}(x)$ 约等于 x ，这可以通过对 $\log(1+x)$ 进行泰勒级数展开证明。在后续章节我们会学习如何用符号计算库SymPy进行泰勒级数展开。

这些特殊函数与NumPy中的一般数学函数一样，都是ufunc函数，支持数组的广播运算。例如`ellipj(u, m)`计算雅可比椭圆函数，它有两个参数 u 和 m ，返回4个值 sn 、 cn 、 dn 和 ϕ ，其中 ϕ 满足下面的椭圆积分，而 $sn = \sin\phi$ 、 $cn = \cos\phi$ 、 $dn = \sqrt{1 - m\sin^2\phi}$ 。

$$u = \int_0^{\phi} \frac{d\theta}{\sqrt{1 - m\sin^2\theta}}$$

由于`ellipj()`支持广播运算，因此下面的程序在调用它时传递的两个参数的形状分别为 $(200, 1)$ 和 $(1, 5)$ ，于是得到的4个结果数组的形状都为 $(200, 5)$ ，图3-1显示了这些曲线。

```
m = np.linspace(0.1, 0.9, 4)
u = np.linspace(-10, 10, 200)
results = S.ellipj(u[:, None], m[None, :])

print [y.shape for y in results]

[(200, 4), (200, 4), (200, 4), (200, 4)]
```

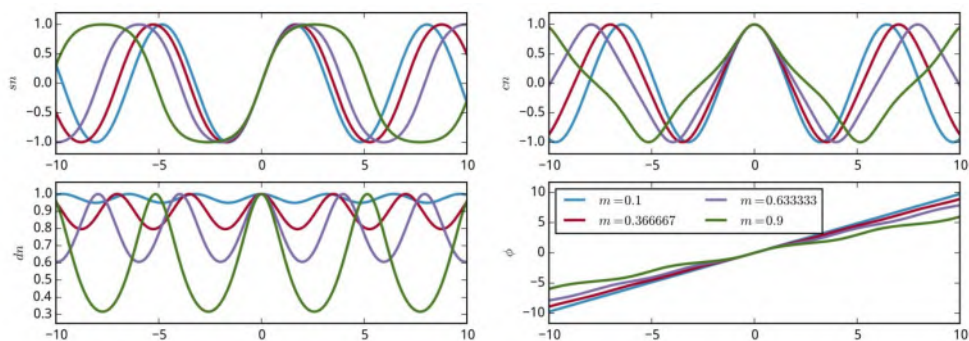


图3-1 使用广播计算得到的`ellipj()`返回值

3.2 拟合与优化-optimize



与本节内容对应的 Notebook 为：03-scipy/scipy-210-optimize.ipynb。

SciPy的optimize模块提供了许多数值优化算法，本节对其中的非线性方程组求解、数据拟合、函数最小值等进行简单介绍。

3.2.1 非线性方程组求解

`fsolve()`可以对非线性方程组进行求解，它的基本调用形式为`fsolve(func, x0)`。其中`func`是计算方程组误差的函数，它的参数`x`是一个数组，其值为方程组的一组可能的解。`func`返回将`x`代入方程组之后得到的每个方程的误差，`x0`为未知数的一组初始值。假设要对下面的方程组进行求解：

$$f_1(u_1, u_2, u_3)=0, \quad f_2(u_1, u_2, u_3)=0, \quad f_3(u_1, u_2, u_3)=0$$

那么`func`函数可以定义如下：

```
def func(x):  
    u1, u2, u3 = x  
    return [f1(u1, u2, u3), f2(u1, u2, u3), f3(u1, u2, u3)]
```

下面我们看一个对下列方程组求解的例子：

$$5x_1 + 3 = 0, \quad 4x_0^2 - 2\sin(x_1x_2) = 0, \quad x_1x_2 - 1.5 = 0$$

```

from math import sin, cos
from scipy import optimize

def f(x): ❶
    x0, x1, x2 = x.tolist() ❷
    return [
        5*
x1+3,
        4*
x0*
x0 - 2*
sin(x1*
x2),
        x1*
x2 - 1.5
    ]

# f计算方程组的误差, [1,1,1]是未知数的初始值
result = optimize.fsolve(f, [1,1,1]) ❸
print result
print f(result)

```

```
[-0.70622057 -0.6          -2.5          ]  
[0.0, -9.126033262418787e-14, 5.329070518200751e-15]
```

❶ `f()` 是计算方程组的误差的函数，`x` 参数是一组可能的解。`fsolve()` 在调用 `f()` 时，传递给 `f()` 的参数是一个数组。❷ 先调用数组的 `tolist()` 方法，将其转换为 Python 的标准浮点数列表，然后调用 `math` 模块中的函数进行运算。因为在进行单个数值的运算时，标准浮点类型比 NumPy 的浮点类型要快许多，所以把数值都转换成标准浮点类型，能缩短一些计算时间。❸ 调用 `fsolve()` 时，传递计算误差的函数 `f()` 以及未知数的初始值。

在对方程组进行求解时，`fsolve()` 会自动计算方程组在某点对各个未知数变量的偏导数，这些偏导数组成一个二维数组，数学上称之为雅可比矩阵。如果方程组中的未知数很多，而与每个方程有关联的未知数较少，即雅可比矩阵比较稀疏时，将计算雅可比矩阵的函数作为参数传递给 `fsolve()`，这能大幅度提高运算速度。笔者在一个模拟计算的程序中需要求解有 50 个未知数的非线性方程组。每个方程平均与 6 个未知数相关，通过传递计算雅可比矩阵的函数使 `fsolve()` 的计算速度提高了 4 倍。

雅可比矩阵

雅可比矩阵是一阶偏导数以一定方式排列的矩阵，它给出了可微分方程与给定点的最优线性逼近，因此类似于多元函数的导数。例如前面的函数 `f1`、`f2`、`f3` 和未知数 `u1`、`u2`、`u3` 的雅可比矩阵如下：

$$\begin{bmatrix} \frac{\partial f1}{\partial u1} & \frac{\partial f1}{\partial u2} & \frac{\partial f1}{\partial u3} \\ \frac{\partial f2}{\partial u1} & \frac{\partial f2}{\partial u2} & \frac{\partial f2}{\partial u3} \\ \frac{\partial f3}{\partial u1} & \frac{\partial f3}{\partial u2} & \frac{\partial f3}{\partial u3} \end{bmatrix}$$

下面使用雅可比矩阵对方程组进行求解。❶计算雅可比矩阵的函数j()和f()一样，其x参数是未知数的一组值，它计算非线性方程组在x处的雅可比矩阵。❷通过fprime参数将j()传递给fsolve()。由于本例中的未知数很少，因此计算雅可比矩阵并不能显著地提高计算速度。

```
def j(x): ❶
    x0, x1, x2 = x.tolist( )
    return [
        [0, 5, 0],
        [8*
x0, -2*
x2*
cos(x1*
```

```

x2), -2*

x1*

cos(x1*

x2)],

        [0, x2, x1]

    ]

result = optimize.fsolve(f, [1,1,1], fprime=j) ❷
print result
print f(result)

[-0.70622057 -0.6          -2.5          ]
[0.0, -9.126033262418787e-14, 5.329070518200751e-15]

```

3.2.2 最小二乘拟合

假设有一组实验数据 (x_i, y_i) ，我们事先知道它们之间应该满足某函数关系： $y_i = f(x_i)$ 。通过这些已知信息，需要确定函数 $f()$ 的一些参数。例如，如果函数 $f()$ 是线性函数 $f(x) = kx + b$ ，那么参数 k 和 b 就是需要确定的值。

如果用 $\mathbf{p}$ 表示函数中需要确定的参数，则目标是找到一组 $\mathbf{p}$ 使得函数 S 的值最小：

$$S(\mathbf{p}) = \sum_{i=1}^m [y_i - f(x_i, \mathbf{p})]^2$$

这种算法被称为最小二乘拟合(least-square fitting)。在optimize模块中,可以使用leastsq()对数据进行最小二乘拟合计算。leastsq()的用法很简单,只需要将计算误差的函数和待确定参数的初始值传递给它即可。下面是用leastsq()对线性函数进行拟合的程序:

```
import numpy as np
from scipy import optimize

X = np.array([ 8.19,  2.72,  6.39,  8.71,  4.7 ,  2.66,  3.78])
Y = np.array([ 7.01,  2.78,  6.47,  6.71,  4.1 ,  4.23,  4.05])

def residuals(p): ❶
    "计算以p为参数的直线和原始数据之间的误差"
    k, b = p
    return Y - (k*
X + b)

# leastsq使得residuals()的输出数组的平方和最小, 参数的初始值为[1,0]
r = optimize.leastsq(residuals, [1, 0]) ❷
k, b = r[0]
print "k =",k, "b =",b
k = 0.613495349193 b = 1.79409254326
```

图3-2(左)直观地显示了原始数据、拟合直线以及它们之间的误差。

❶residuals()的参数p是拟合直线的参数,函数返回的是原始数据和拟合直线之间的误差。图中用数据点到拟合直线在Y轴上的距离表示误差。

❷leastsq()使得这些误差的平方和最小,即图中所有正方形的面积之和最小。

由前面的函数S的公式可知，对于直线拟合来说，误差的平方和是直线参数k和b的二次多项式函数，因此可以用如图3-2（右）所示的曲面直观地显示误差平方和与两个参数之间的关系。图中用红色圆点表示曲面的最小点，它的X-Y轴的坐标就是leastsq()的拟合结果。

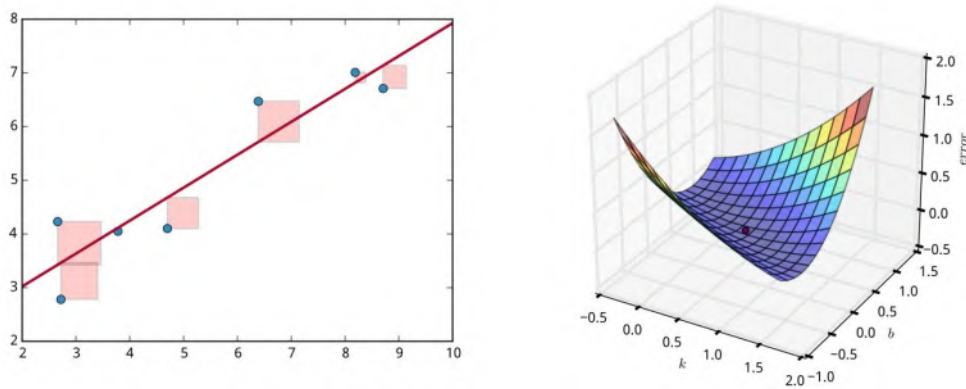


图3-2 最小化正方形面积之和（左），误差曲面（右）

接下来，让我们再看一个对正弦波数据进行拟合的例子：

```
def func(x, p): ❶
    """
    数据拟合所用的函数：A *
    sin(2 *
    pi *
    k *
    x + theta)
    """
    A, k, theta = p
```

```

        return A*

np.sin(2*

np.pi*

k*

x+theta)

def residuals(p, y, x): ❷
    """
    实验数据x, y和拟合函数之间的差, p为拟合需要找到的系数
    """
    return y - func(x, p)

x = np.linspace(0, 2*

np.pi, 100)

A, k, theta = 10, 0.34, np.pi/6 # 真实数据的函数参数
y0 = func(x, [A, k, theta]) # 真实数据
# 加入噪声之后的实验数据
np.random.seed(0)

y1 = y0 + 2 *

np.random.randn(len(x)) ❸

p0 = [7, 0.40, 0] # 第一次猜测的函数拟合参数

```

```

# 调用leastsq进行数据拟合
# residuals为计算误差的函数
# p0为拟合参数的初始值
# args为需要拟合的实验数据

plsq = optimize.leastsq(residuals, p0, args=(y1, x)) ④

print u"真实参数:", [A, k, theta]
print u"拟合参数", plsq[0] # 实验数据拟合后的参数

pl.plot(x, y1, "o", label=u"带噪声的实验数据")
pl.plot(x, y0, label=u"真实数据")
pl.plot(x, func(x, plsq[0]), label=u"拟合数据")
pl.legend(loc="best")

真实参数: [10, 0.34, 0.5235987755982988]
拟合参数 [ 10.25218748    0.3423992    0.50817424]

```

图3-3显示了带噪声的正弦波拟合。

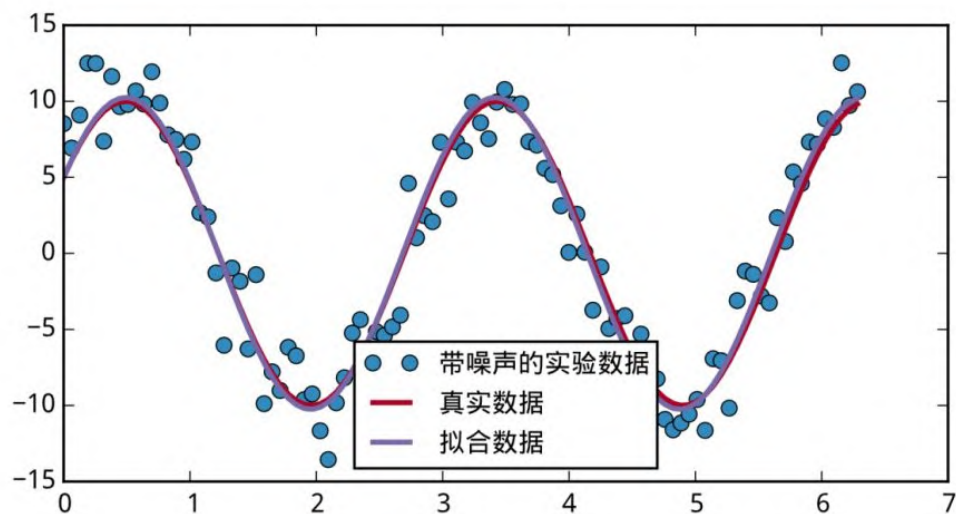


图3-3 带噪声的正弦波拟合

程序中，❶要拟合的目标函数`func()`是一个正弦函数，它的参数`p`是一个数组，包含决定正弦波的三个参数`A`、`k`、`theta`，分别对应正弦函数的振幅、频率和相角。❷待拟合的实验数据是一组包含噪声的数据`(x, y1)`，其中数组`y1`为标准正弦波数据`y0`加上随机噪声。

❸用`leastsq()`对带噪声的实验数据`(x, y1)`进行数据拟合，它可以找到数组`x`和`y0`之间的正弦关系，即确定`A`、`k`、`theta`等参数。和前面的直线拟合程序不同的是，这里我们将`(y1, x)`传递给`args`参数。`leastsq()`会将这两个额外的参数传递给`residuals()`。❹因此`residuals()`有三个参数，`p`是正弦函数的参数，`y`和`x`是表示实验数据的数组。

对于这种一维曲线拟合，`optimize`库还提供了一个`curve_fit()`函数，下面使用此函数对正弦波数据进行拟合。它的目标函数与`leastsq()`稍有不同，各个待优化参数直接作为函数的参数传入。

```
def func2(x, A, k, theta):  
    return A *  
  
np.sin(2 *  
  
np.pi *  
  
k *  
  
x + theta)  
  
popt, _ = optimize.curve_fit(func2, x, y1, p0=p0)  
print popt  
[ 10.25218748  0.3423992  0.50817424]
```

如果频率的初值和真实值的差别较大，拟合结果中的频率参数可能无法收敛于实际的频率。在下面的例子中，由于频率初值的选择不当，导致 `curve_fit()` 未能收敛到真实的参数。这时可以通过其他方法先估算一个频率的近似值，或者使用全局优化算法。在后面的例子中，我们会使用全局优化算法重新对正弦波数据进行拟合。

```
popt, _ = optimize.curve_fit(func2, x, y1, p0=[10, 1, 0])
print u"真实参数:", [A, k, theta]
print u"拟合参数", popt
真实参数: [10, 0.34, 0.5235987755982988]
拟合参数 [ 0.71093473  1.02074599 -0.1277666 ]
```

3.2.3 计算函数局域最小值

`optimize` 库还提供了许多求函数最小值的算法：Nelder-Mead、Powell、CG、BFGS、Newton-CG、L-BFGS-B等。下面我们用一个实例观察这些优化函数是如何找到函数的最小值的。在本例中，要计算最小值的函数 $f(x, y)$ 为：

$$f(x, y) = (1-x)^2 + 100(y-x^2)^2$$

这个函数叫作Rosenbrock函数，它经常用来测试最小化算法的收敛速度。它有一个十分平坦的山谷区域，收敛到此山谷区域比较容易，但是在山谷区域搜索到最小点则比较困难。根据函数的计算公式不难看出此函数的最小值是0，在(1, 1)处。

为了提高运算速度和精度，有些算法带有一个 `fprime` 参数，它是计算目标函数 $f()$ 对各个自变量的偏导数的函数。 $f(x, y)$ 对变量 x 和 y 的偏导函数为：

$$\frac{\partial f}{\partial x} = -2 + 2x - 400x(y - x^2)$$

$$\frac{\partial f}{\partial y} = 200y - 200x^2$$

而Newton-CG算法则需要计算海森矩阵，它是一个由自变量为向量的实值函数的二阶偏导数构成的方块矩阵，对于函数 $f(x_1, x_2, \dots, x_n)$ ，其海森矩阵的定义如下：

$$\begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

对于本例来说，海森矩阵为一个二阶矩阵：

$$\begin{bmatrix} 2(600x^2 - 200y + 1) & -400x \\ -400x & 200 \end{bmatrix}$$

下面使用各种最小值优化算法计算 $f(x, y)$ 的最小值，根据其输出可知有些算法需要较长的收敛时间，而有些算法则利用导数信息更快地找到最小点。

```
def target_function(x, y):
    return (1-x)**
```

2 + 100*

$(y-x^{**$

$2)^{**$

2

```
class TargetFunction(object):
```

```
    def __init__(self):
```

```
        self.f_points = [ ]
```

```
        self.fprime_points = [ ]
```

```
        self.fhess_points = [ ]
```

```
    def f(self, p):
```

```
        x, y = p.tolist( )
```

```
        z = target_function(x, y)
```

```
        self.f_points.append((x, y))
```

```
        return z
```

```
    def fprime(self, p):
```

```
        x, y = p.tolist( )
```

```
        self.fprime_points.append((x, y))
```

```
        dx = -2 + 2*
```

$x - 400^{*}$

x^{*}

```

(y - x**

2)

dy = 200*

y - 200*

x**

2

return np.array([dx, dy])

def fhess(self, p):
    x, y = p.tolist( )
    self.fhess_points.append((x, y))
    return np.array([[2*

(600*

x**

2 - 200*

y + 1), -400*

x],

```

```

[-400*
x, 200]])

def fmin_demo(method):
    target = TargetFunction( )
    init_point =(-1, -1)
    res = optimize.minimize(target.f, init_point,
                            method=method,
                            jac=target.fprime,
                            hess=target.fhess)
    return res, [np.array(points) for points in
                  (target.f_points, target.fprime_points,
target.fhess_points)]

methods = ("Nelder-Mead", "Powell", "CG", "BFGS", "Newton-CG",
"L-BFGS-B")
for method in methods:
    res, (f_points, fprime_points, fhess_points) =
fmin_demo(method)
    print "{:12s}: min={:12g}, f count={:3d}, fprime count=
{:3d}, "\
    "fhess count={:3d}".format(
method, float(res["fun"]), len(f_points),
len(fprime_points), len(fhess_points))
    Nelder-Mead : min= 5.30934e-10, f count=125, fprime count= 0,
fhess count= 0
    Powell      : min=          0, f count= 52, fprime count= 0,

```

```

fhess count= 0
    CG          : min= 7.6345e-15, f count= 34, fprime count= 34,
fhess count= 0
    BFGS        : min= 2.31605e-16, f count= 40, fprime count= 40,
fhess count= 0
    Newton-CG   : min= 5.22666e-10, f count= 60, fprime count= 97,
fhess count= 38
    L-BFGS-B    : min= 6.5215e-15, f count= 33, fprime count= 33,
fhess count= 0

```

图3-4显示了各种优化算法的搜索路径，图中用圆点表示调用 $f()$ 时的坐标点，圆点的颜色表示调用顺序；叉点表示调用 $fprime()$ 时的坐标点。图中用图像表示二维函数的值，值越大则颜色越浅，值越小则颜色越深。为了更清晰地显示函数的山谷区域，图中显示的实际上是通过函数 $\log_{10}()$ 对 $f(x, y)$ 进行处理之后的结果。

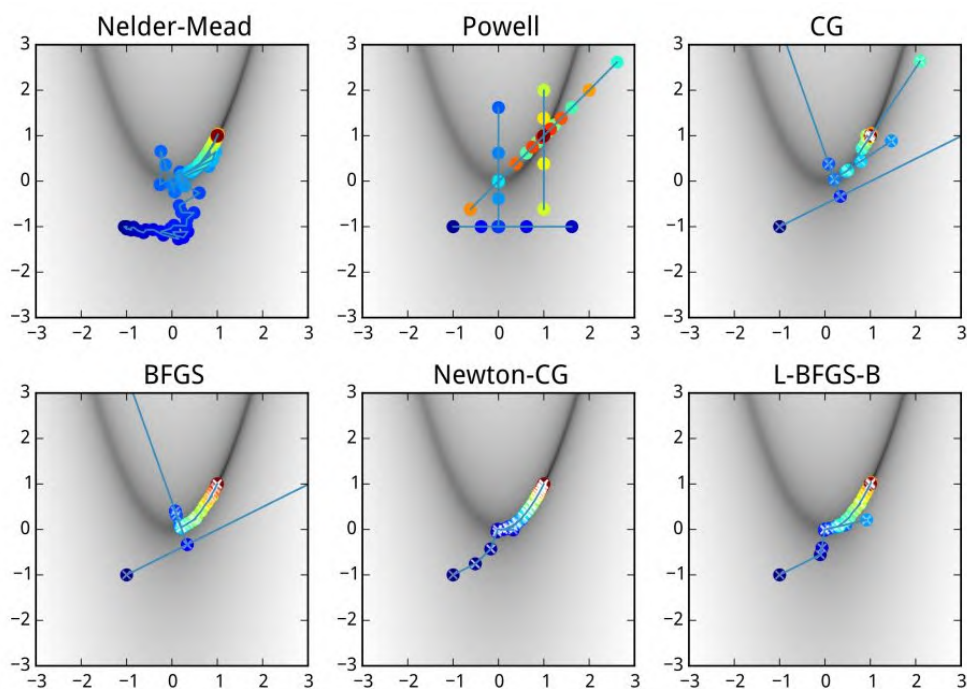


图3-4 各种优化算法的搜索路径

3.2.4 计算全域最小值

前面介绍的几种最小值优化算法都只能计算局域的最小值，optimize库还提供了几种能进行全局优化的算法，下面以前面的正弦波拟合为例，演示全局优化函数的用法。在使用leastsq()对正弦波进行拟合时，误差函数residuals()返回一个数组，表示各个取样点的误差。而函数最小值算法则只能对一个标量值进行最小化，因此最小化的目标函数func_error()返回所有取样点的误差的平方和。

```
def func(x, p):  
    A, k, theta = p  
    return A *  
  
np.sin(2 *  
  
np.pi *  
  
k *  
  
x + theta)  
  
def func_error(p, y, x):  
    return np.sum((y - func(x, p)) **  
  
2)  
  
x = np.linspace(0, 2 *
```

```

np.pi, 100)

    A, k, theta = 10, 0.34, np.pi/6

    y0 = func(x, [A, k, theta])

    np.random.seed(0)

    y1 = y0 + 2 *

np.random.randn(len(x))

```

我们使用`optimize.basinhopping()`全域优化函数找出正弦波的两个参数。它的前两个参数和其他求最小值的函数一样：目标函数和初始值。由于它是全局优化函数，因此初始值的选择并不是太重要。`niter`参数是全域优化算法的迭代次数，迭代的次数越多，就越有可能找到全域最优解。

在`basinhopping()`内部需要调用局域最小值函数，其`minimizer_kwargs`参数决定了所采用的局域最小值算法以及传递给此函数的参数。下面的程序指定使用L-BFGS-B算法搜索局域最小值，并且将两个对象`y1`和`x`传递给该局域最小值求解函数的`args`参数，而该函数会将这两个参数传递给`func_error()`。

```

result = optimize.basinhopping(func_error, (1, 1, 1),
                               niter = 10,
                               minimizer_kwargs={"method": "L-BFGS-B",
                                                  "args": (y1, x)})

print result.x

[ 10.25218694 -0.34239909  2.63341582]

```

虽然频率和相位与原系数不同，但是由于正弦函数的周期性，其拟合曲线是和原始数据重合的，如图3-5所示。

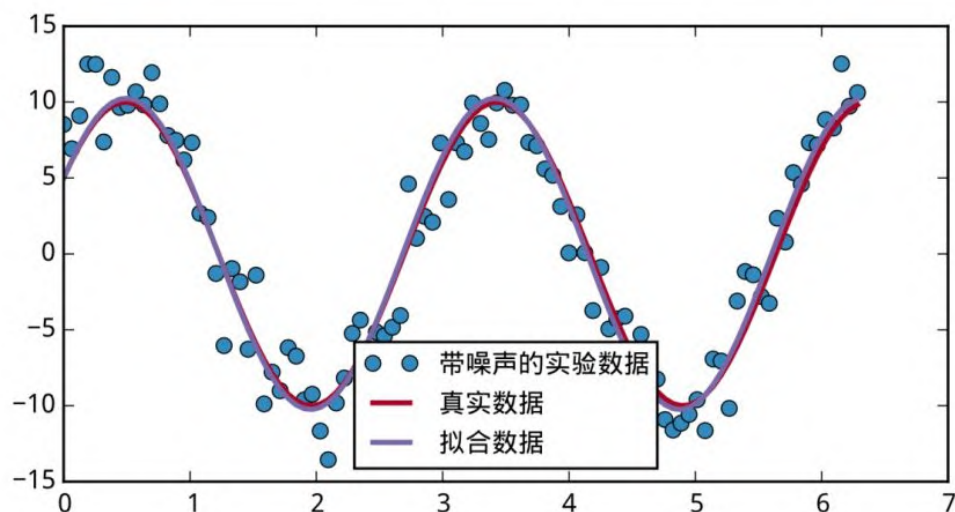


图3-5 用basinhopping()拟合正弦波

3.3 线性代数-linalg



与本节内容对应的Notebook为：`03-scipy/scipy-310-linalg.ipynb`。

NumPy和SciPy都提供了线性代数函数库linalg，SciPy的线性代数库比NumPy更加全面。

3.3.1 解线性方程组

`numpy.linalg.solve(A, b)`和`scipy.linalg.solve(A, b)`可以用来解线性方程组 $Ax=b$ ，也就是计算 $x=A^{-1}b$ 。这里， A 是 $m \times n$ 的方形矩阵， x 和 b 是长为 m 的向量。有时候 A 是固定的，需要对多组 b 进行求解，因此第二

个参数也可以是 $m \times n$ 的矩阵 B 。这样计算出来的 X 也是 $m \times n$ 的矩阵，相当于计算 $A^{-1} B$ 。

在一些矩阵公式中经常会出现类似于 $A^{-1} B$ 的运算，它们都可以用`solve(A,B)`计算，这要比直接计算逆矩阵然后做矩阵乘法更快捷一些，下面的程序比较`solve()`和逆矩阵的运算速度：

```
import numpy as np
from scipy import linalg

m, n = 500, 50
A = np.random.rand(m, m)
B = np.random.rand(m, n)
X1 = linalg.solve(A, B)
X2 = np.dot(linalg.inv(A), B)
print np.allclose(X1, X2)
%timeit linalg.solve(A, B)
%timeit np.dot(linalg.inv(A), B)

True
100 loops, best of 3: 10.1 ms per loop
10 loops, best of 3: 20 ms per loop
```

若需要对多组 B 进行求解，但是又不好将它们合并成一个矩阵，例如某些矩阵公式中可能会有 $A^{-1} B$ 、 $A^{-1} C$ 、 $A^{-1} D$ 等乘法，而 B 、 C 、 D 是通过某种方式逐次计算的。这时可以采用`lu_factor()`和`lu_solve()`。先调用`lu_factor(A)`对矩阵 A 进行LU分解，得到一个元组：（LU矩阵，排序数组）。将这个元组传递给`lu_solve()`，即可对不同的 B 进行求解。由于已经对 A 进行了LU分解，`lu_solve()`能够很快得出结果。

```
luf = linalg.lu_factor(A)
X3 = linalg.lu_solve(luf, B)
np.allclose(X1, X3)

True
```

除了使用`lu_factor()`和`lu_solve()`之外，可以先通过`inv()`计算逆矩阵，然后通过`dot()`计算矩阵乘积。下面比较二者的速度，可以看出`lu_factor()`比`inv()`要快很多，而`lu_solve()`和`dot()`的运算速度几乎相同：

```
M, N = 1000, 100
np.random.seed(0)
A = np.random.rand(M, M)
B = np.random.rand(M, N)
Ai = linalg.inv(A)
luf = linalg.lu_factor(A)
%timeit linalg.inv(A)
%timeit np.dot(Ai, B)
%timeit linalg.lu_factor(A)
%timeit linalg.lu_solve(luf, B)

10 loops, best of 3: 131 ms per loop
100 loops, best of 3: 9.65 ms per loop
10 loops, best of 3: 52.6 ms per loop
100 loops, best of 3: 13.8 ms per loop
```

3.3.2 最小二乘解

lstsq()比solve()更一般化，它不要求矩阵A是正方形的，也就是说方程的个数可以少于、等于或多于未知数的个数。它找到一组解x，使得 $\|b - Ax\|$ 最小。我们称得到的结果为最小二乘解，即它使得所有等式的误差的平方和最小。下面以求解离散卷积的逆运算为例，介绍lstsq()的用法。

首先简单介绍一下离散卷积的相关知识和计算方法。对于离散的线性时不变系统h，如果它的输入是x，那么其输出y可以用x和h的卷积表示：
 $y = x * h$ 。

离散卷积的计算公式如下：

$$y[n] = \sum h[m] x[n-m]$$

假设h的长度为n，x的长度为m，则卷积计算所得到的y的长度将为n+m-1。它的每个值都是按照下面的公式计算得到的：

$$y[0] = h[0] * x[0]$$

$$x[0]$$

$$y[1] = h[0] * x[1] + h[1] * x[0]$$

$$x[1] + h[1] * x[0]$$

$$x[0]$$

$$y[2] = h[0] * x[2] + h[1] * x[1] + h[2] * x[0]$$

$$x[2] + h[1] * x[1] + h[2] * x[0]$$

$x[1] + h[2]^*$

$x[0]$

$y[3] = h[0]^*$

$x[3] + h[1]^*$

$x[2] + h[2]^*$

$x[1]$

$\dots$

$y[n+m-1] = h[n-1]^*$

$x[m-1]$

所谓卷积的逆运算就是指：假设已知 x 和 y ，要求解 h 。由于 h 的长度为 n ，于是有 n 个未知数，而由于 y 的长度为 $n+m-1$ ，因此这 n 个未知数需要满足 $n+m-1$ 个线性方程。由于方程数比未知数多，卷积的逆运算不一定有精确解，因此问题就变成了找到一组 h ，使得 $x * h$ 与 y 之间的误差最小，显然它就是最小二乘解。下面的程序演示了如何使用`lstsq()`计算卷积的逆运算：

❶ 首先 `make_data()` 创建所需的数据，它使用随机数函数 `standard_normal()` 初始化数组 x 和 h 。在实际的系统中 h 通常是未知的，并且值会逐渐衰减。`make_data()`返回系统的输入信号 x 以及添加了随机噪声的输出信号 y_n 。为了和最小二乘法的结果相比较，我们同时也输出了系统的系数 h 。

❷`solve_h()`使用最小二乘法计算系统的参数`h`，因为通常我们不知道未知系统的系数的长度，因此这里用`N`表示所求系数的长度。

观察前面的卷积方程组可知，在 $n+m-1$ 个方程中，中间的 $n-m+1$ 个方程使用了`h`的所有系数。为了程序计算方便，我们对这 $m-n+1$ 个方程进行最小二乘运算。❸根据`h`的长度，需要将一维数组`x`变换成一个形状为 $(m-n+1, n)$ 的二维数组`X`，它的每行相对于上一行都左移了一个元素。这个二维数组可以很容易地使用第2章中介绍过的`as_strided()`得到。❹我们取出输出数组`y`中与数组`X`每行对应的部分，❺然后调用`lstsq()`对这 $m-n+1$ 个方程进行最小二乘运算。❻`lstsq()`返回一个元组，它的第0个元素是最小二乘解，注意得到的结果顺序是颠倒的，因此还需要对其进行翻转。

```
from numpy.lib.stride_tricks import as_strided

def make_data(m, n, noise_scale): ❶
    np.random.seed(42)
    x = np.random.standard_normal(m)
    h = np.random.standard_normal(n)
    y = np.convolve(x, h)
    yn = y + np.random.standard_normal(len(y)) *
    noise_scale *
    np.max(y)
    return x, yn, h

def solve_h(x, y, n): ❷
    X = as_strided(x, shape=(len(x)-n+1, n), strides=
(x.itemsize, x.itemsize)) ❸
```

```

Y = y[n-1:len(x)]      ❷
h = linalg.lstsq(X, Y)  ❸
return h[0][::-1]       ❹

```

接下来对长度为100的未知系统系数 h ，分别计算长度为80和120的最小二乘解。由于我们对系统的输出添加了一些噪声信号，因此二者并不完全吻合。图3-6比较了这两个解与真实系数。

```

x, yn, h = make_data(1000, 100, 0.4)
H1 = solve_h(x, yn, 120)
H2 = solve_h(x, yn, 80)

print "Average error of H1:", np.mean(np.abs(H[:100] - h))
print "Average error of H2:", np.mean(np.abs(h[:80] - H2))

Average error of H1: 0.301548854044
Average error of H2: 0.295842215834

```

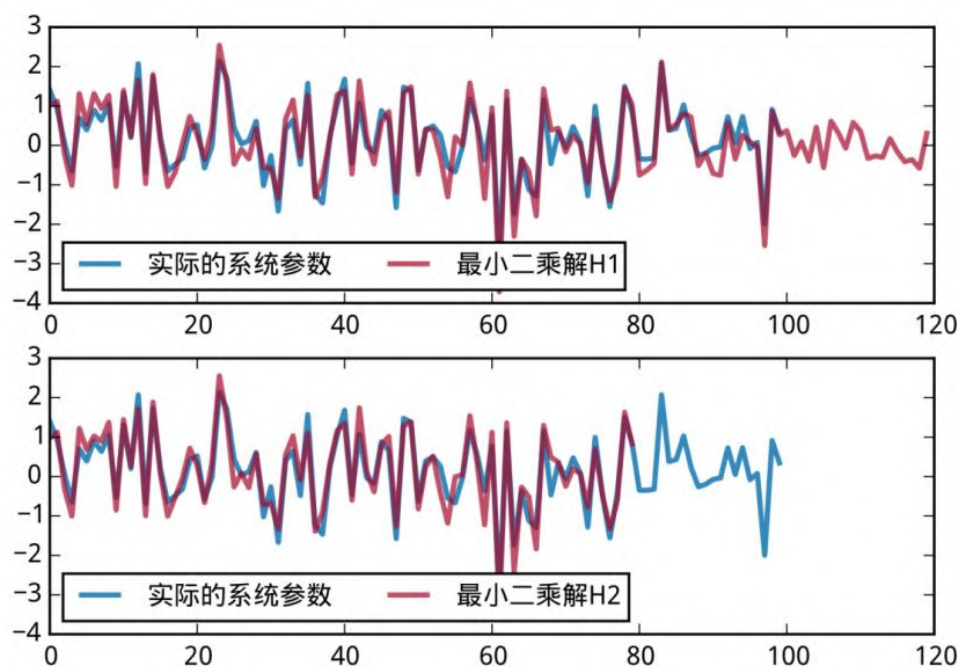


图3-6 实际的系统参数与最小二乘解的比较

3.3.3 特征值和特征向量

$n \times n$ 的矩阵A 可以看作 n 维空间中的线性变换。若 $\mathbf{x}$ 为 n 维空间中的一个向量，那么A 与 $\mathbf{x}$ 的矩阵乘积就是对 $\mathbf{x}$ 进行线性变换之后的向量。如果 $\mathbf{x}$ 是线性变换的特征向量，那么经过这个线性变换之后，得到的新向量仍然与原来的 $\mathbf{x}$ 保持在同一方向上，但其长度也许会改变。特征向量的长度在该线性变换下缩放的比例称为其特征值。即特征向量 $\mathbf{x}$ 满足如下等式， λ 的值可以是一个任意复数：

$$A\mathbf{x} = \lambda \mathbf{x}$$

下面以二维平面上的线性变换矩阵为例，演示特征值和特征变量的几何含义。通过`linalg.eig(A)` 计算矩阵A 的两个特征值`evalues`和特征向量`eectors`，在`eectors`中，每一列是一个特征向量。

```
A = np.array([[1, -0.3], [-0.1, 0.9]])
evalues, eectors = linalg.eig(A)

              evalues                      eectors
-----
[ 1.13027756+0.j,  0.76972244+0.j]  [[ 0.91724574,  0.79325185],
[-0.3983218 ,  0.60889368]]
```

图3-7显示了变换前后的向量。在图中，蓝色箭头为变换之前的向量，红色箭头为变换之后的向量。粗箭头为变换前后的特征向量。可以看出特征向量变换前后方向没有改变，只是长度发生了变化。长度的变化倍数由特征值决定：一个变为原来的1.13倍长，一个变为原来的0.77倍长。

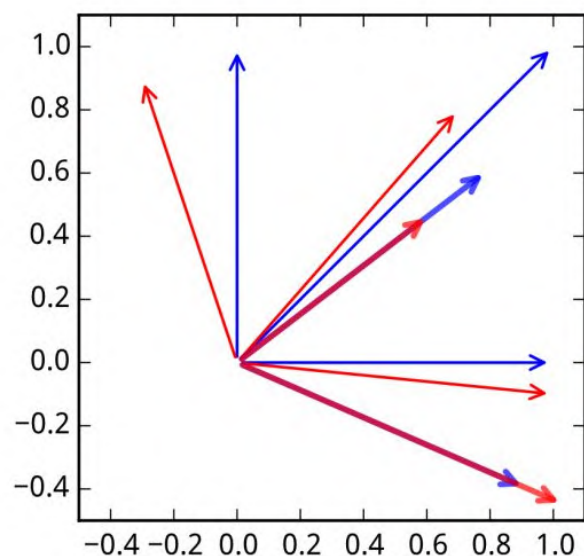


图3-7 线性变换将蓝色箭头变换为红色箭头

numpy.linalg模块中也有eig()函数，与之不同的是，scipy.linalg模块中的eig()函数支持计算广义特征值和广义特征向量，它们满足如下等式，其中B 是一个 $n \times n$ 的矩阵：

$$Ax = \lambda Bx$$

广义特征向量可以用于椭圆拟合，椭圆拟合的公式与原理可以参考下面的论文：



pami.pdf

<http://research.microsoft.com/pubs/67845/ellipse->

用广义特征向量计算椭圆拟合。

椭圆上的点满足如下方程，其中a, b, c, d, e, f为椭圆的参数，(x, y)为平面上的坐标点：

$$f(x, y) = ax^2 + bxy + cy^2 + dx + ey + f = 0$$

所谓椭圆拟合，就是指给出一组平面上的点 (x_i, y_i) ，找到一组椭圆参数，使得 $\sum f(x_i, y_i)^2$ 最小。显然这是一个最小化问题，可以使用上节介绍的优化算法`optimize.leastsq()`求解。为了避免参数全为0的平凡解，需要一点小技巧，请读者自行演练一下。下面给出论文中用广义特征向量计算椭圆拟合的方法：

首先定义 $\mathbf{x}_i = [x_i^2, x_i y_i, y_i^2, x_i, y_i, 1]^T$ ， $\mathbf{D} = [\mathbf{x}_1, \dots, \mathbf{x}_n]$ 。D 是一个 $n \times 6$ 的矩阵，其中n为点的个数，D 中的每一行与一个坐标点相对应。a 为拟合椭圆的系数： $\mathbf{a} = [a, b, c, d, e, f]^T$ 。则a 满足如下方程：

$$\mathbf{D}^T \mathbf{D} \mathbf{a} = \lambda \mathbf{C} \mathbf{a}$$

其中C 是一个 6×6 的矩阵：

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

显然上式符合广义特征向量的等式，因此可以用`linalg.eig()`求解。下面首先使用椭圆的参数方程计算某个椭圆上随机的60个点，并引入一些随机噪声：

$$\begin{aligned} X(t) &= X_c + a \cos t \cos \varphi - b \sin t \sin \varphi \\ Y(t) &= Y_c + a \cos t \sin \varphi + b \sin t \cos \varphi \end{aligned}$$

```
np.random.seed(42)

t = np.random.uniform(0, 2*

np.pi, 60)

alpha = 0.4
a = 0.5
b = 1.0
x = 1.0 + a*

np.cos(t)*

np.cos(alpha) - b*

np.sin(t)*

np.sin(alpha)

y = 1.0 + a*

np.cos(t)*

np.sin(alpha) - b*

np.sin(t)*

np.cos(alpha)
```

```
x += np.random.normal(0, 0.05, size=len(x))
y += np.random.normal(0, 0.05, size=len(y))
```

❶当传递第二个参数时，`eig()`计算广义特征值和向量。`evecs`中共有6个特征向量，❷将这6个特征向量代入椭圆方程中，计算平均误差，❸并挑选误差最小的特征向量作为椭圆的参数`p`。图3-8显示了参数`p`所表示的椭圆以及数据点。

```
D = np.c_[x**
2, x*
y, y**
2, x, y, np.ones_like(x)]
A = np.dot(D.T, D)
C = np.zeros((6, 6))
C[[0, 1, 2], [2, 1, 0]] = 2, -1, 2
evals, evecs = linalg.eig(A, C) ❶
evecs = np.real(evecs)
err = np.mean(np.dot(D, evecs)**
2, 0) ❷
p = evecs[:, np.argmin(err)] ❸
print p
[-0.55214278  0.5580915 -0.23809922  0.54584559 -0.08350449
-0.14852803]
```

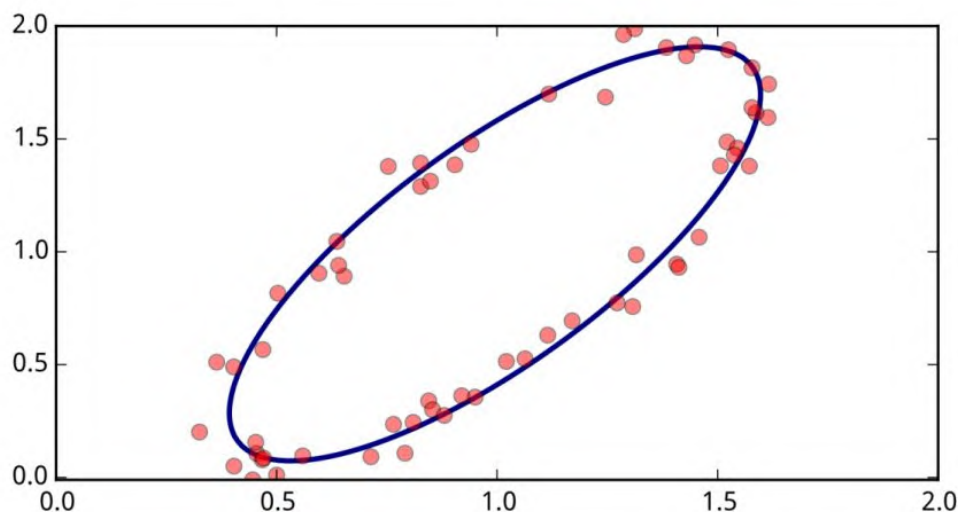


图3-8 用广义特征向量计算的拟合椭圆

3.3.4 奇异值分解-SVD

奇异值分解是线性代数中一种重要的矩阵分解，在信号处理、统计学等领域都有重要应用。假设 M 是一个 $m \times n$ 阶矩阵，存在一个分解使得： $M = U \Sigma V^*$ 。其中 U 是 $m \times m$ 阶酉矩阵； Σ 是半正定 $m \times n$ 阶对角矩阵；而 V^* ，即 V 的共轭转置，是 $n \times n$ 阶酉矩阵。这样的分解就称作 M 的奇异值分解。 Σ 对角线上的元素为 M 的奇异值。通常奇异值按照从大到小的顺序排列。

奇异值的数学描述读起来有些难懂，让我们通过一个实例说明奇异值分解的用途。下面的例子对一幅灰度图像进行奇异值分解，然后从三个分解矩阵中提取奇异值较大的部分数据还原原始图像。首先读入一幅图像，并通过其红绿蓝三个通道计算灰度图像，图像的宽为375个像素、高为505个像素。

```
r, g, b = np.rollaxis(pl.imread("vinci_target.png"),
2).astype(float)
```

```

img = 0.2989 *
r + 0.5870 *
g + 0.1140 *
b
img.shape
(505, 375)

```

调用`scipy.linalg.svd()`对图像矩阵进行奇异值分解，得到三个分解部分：

- U：对应于公式中的 U 。
- s：对应于公式中的 Σ ，由于它是一个对角矩阵，只有对角线上的元素非零，因此s是一个一维数组，保存对角线上的非零元素。
- Vh：对应于公式中的 V^* 。

下面的程序查看这三个数组的形状：

```

U, s, Vh = linalg.svd(img)
U.shape      s.shape      Vh.shape
-----
(505, 505)   (375,)   (375, 375)

```

s中的每个值与Vh的行向量以及U中的列向量对应，默认按照从大到小的顺序排列，它表示与其对应的向量的重要性。由图3-9可知s中的奇异值大小差别很大，注意Y轴是对数坐标系。

```

pl.semilogy(s, lw=3)

```

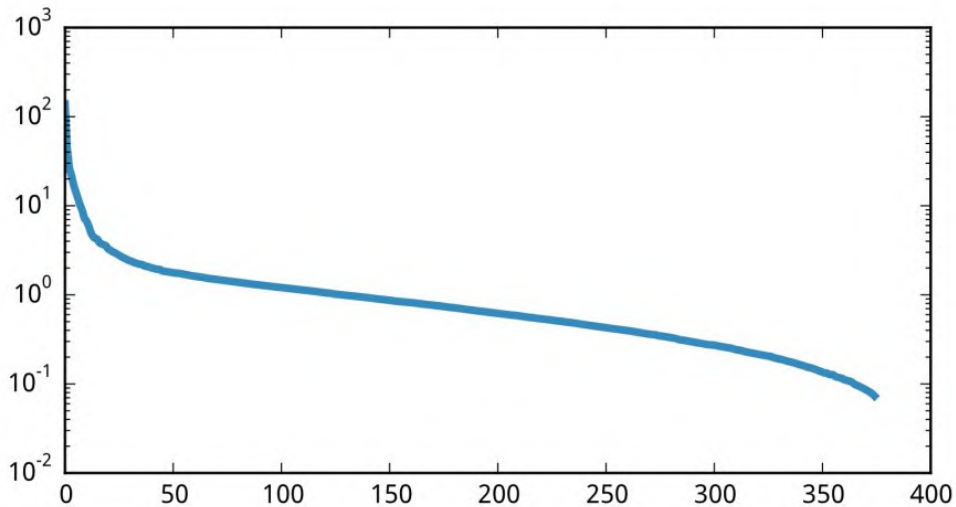


图3-9 按从大到小排列的奇异值

下面的`composite()`选择`U`和`Vh`中的前`n`个向量重新合成矩阵，当用上所有向量时，重新合成的矩阵和原始矩阵相同：

```
def composite(U, s, Vh, n):  
    return np.dot(U[:, :n], s[:n, np.newaxis] *  
  
Vh[:n, :])  
  
print np.allclose(img, composite(U, s, Vh, len(s)))  
True
```

下面演示选择前10个、20个以及50个向量合成时的效果，如图3-10所示，可以看到使用的向量越多，结果就越接近原始图像：

```
img10 = composite(U, s, Vh, 10)  
img20 = composite(U, s, Vh, 20)  
img50 = composite(U, s, Vh, 50)  
  
%array_image img; img10; img20; img50
```

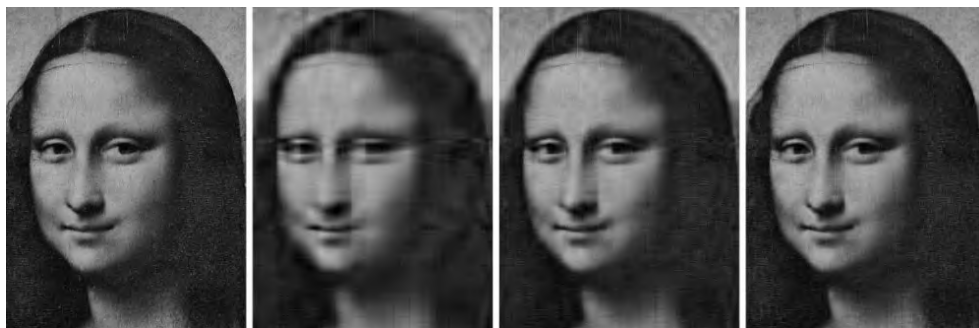


图3-10 原始图像以及使用10个、20个、50个向量合成的图像（从左到右）

3.4 统计-stats



与本节内容对应的 Notebook 为：`03-scipy/scipy-400-stats.ipynb`。

SciPy的stats模块包含了多种概率分布的随机变量（随机变量是指概率论中的概念，不是Python中的变量），随机变量分为连续和离散两种。所有的连续随机变量都是rv_continuous的派生类的对象，而所有的离散随机变量都是rv_discrete的派生类的对象。

3.4.1 连续概率分布

可以使用下面的语句获得stats模块中所有的连续随机变量：

```

from scipy import stats
[k for k, v in stats.__dict__.items() if isinstance(v, stats.rv_continuous)]
['genhalflogistic', 'triang', 'rayleigh', 'betaprime',
 'levy', 'foldnorm', 'genlogistic', 'gilbrat',
 'lognorm', 'anglit', 'truncnorm', 'erlang',
 'norm', 'nakagami', 'weibull_min', 'cosine',
 'logistic', 'fisk', 'genpareto', 'tukeylambda',
 'dgamma', 'pareto', 'halflogistic', 'semicircular',
 'ksone', 'mielke', 'ncx2', 'gengamma',
 'johnsonsu', 'powernorm', 'powerlaw', 'burr',
 'johnsonsb', 'beta', 'gamma', 'wald',
 'arcsine', 'maxwell', 'invgauss', 'gausshyper',
 'rice', 'vonmises_line', 'loglaplace', 'levy_stable',
 'exponweib', 'pearson3', 'chi', 't',
 'cauchy', 'truncexpon', 'kstwobign', 'recipinvgauss',
 'frechet_l', 'foldcauchy', 'wrapcauchy', 'ncf',
 'genexpon', 'expon', 'reciprocal', 'f',
 'lomax', 'loggamma', 'invgamma', 'powerlognorm',
 'laplace', 'vonmises', 'frechet_r', 'dweibull',
 'rdist', 'gumbel_r', 'gompertz', 'halfcauchy',
 'invweibull', 'exponpow', 'weibull_max', 'gumbel_l',
 'halfnorm', 'fatiguelife', 'chi2', 'nct',
 'uniform', 'genextreme', 'alpha', 'hypsecant',
 'bradford', 'levy_l']

```

连续随机变量对象都有如下方法：

- rvs: 对随机变量进行随机取值，可以通过size参数指定输出的数组的大小。
- pdf: 随机变量的概率密度函数。
- cdf: 随机变量的累积分布函数，它是概率密度函数的积分。
- sf: 随机变量的生存函数，它的值是 $1 - \text{cdf}(t)$ 。
- ppf: 累积分布函数的反函数。
- stat: 计算随机变量的期望值和方差。
- fit: 对一组随机取样进行拟合，找出最适合取样数据的概率密度函数的系数。

下面以正规分布为例，简单地介绍随机变量的用法。下面的语句获得默认正规分布的随机变量的期望值和方差，我们看到默认情况下它是一个均值为0、方差为1的随机变量：


```
stats.norm.stats( )  
  
(array(0.0), array(1.0))
```

norm可以像函数一样调用，通过loc和scale参数可以指定随机变量的偏移和缩放参数。对于正态分布的随机变量来说，这两个参数相当于指定其期望值和标准差，标准差是方差的算术平方根，因此标准差为2.0时，方差为4.0：

```
X = stats.norm(loc=1.0, scale=2.0)  
  
X.stats( )  
  
(array(1.0), array(4.0))
```

下面调用随机变量X的rvs()方法，得到包含一万次随机取样值的数组x，然后调用NumPy的mean()和var()计算此数组的均值和方差，其结果符合随机变量X的特性：

```
x = X.rvs(size=10000) # 对随机变量取10000个值  
  
np.mean(x), np.var(x) # 期望值和方差  
  
(1.0043406567303883, 3.8899572813426553)
```

也可以使用fit()方法对随机取样序列x进行拟合，它返回的是与随机取样值最吻合的随机变量的参数：

```
stats.norm.fit(x) # 得到随机序列的期望值和标准差  
  
(1.0043406567303883, 1.9722974626923433)
```

在下面的例子中，计算取样值x的直方图统计以及累积分布，并与随机变量的概率密度函数和累积分布函数进行比较。

❶其中histogram()对数组x进行直方图统计，它将数组x的取值范围分为100个区间，并统计x中的每个值落入各个区间的次数。histogram()

返回两个数组pdf和t，其中pdf表示各个区间的取样值出现的频数，由于normed参数为True，因此pdf的值是正规化之后的结果，其结果应与随机变量的概率密度函数一致。

❷t表示区间，由于其中包括区间的起点和终点，因此t的长度为101。计算每个区间的中间值，然后调用X.pdf(t)和X.cdf(t)计算随机变量的概率密度函数和累积分布函数的值，并与统计值比较。❸计算样本的累积分布时，需要与区间的大小相乘，这样才能保证其结果与累积分布函数相同。

```
pdf, t = np.histogram(x, bins=100, normed=True) ❶

t = (t[:-1] + t[1:]) *

0.5 ❷

cdf = np.cumsum(pdf) *

(t[1] - t[0]) ❸

p_error = pdf - X.pdf(t)
c_error = cdf - X.cdf(t)
print "max pdf error: {}, max cdf error: {}".format(
    np.abs(p_error).max( ), np.abs(c_error).max( ))
max pdf error: 0.0217211429624, max cdf error: 0.0209887986472
```

图3-11（左）显示了概率密度函数和直方图统计的结果，可以看出二者是一致的。右图显示了随机变量X的累积分布函数和数组pdf的累加结果。

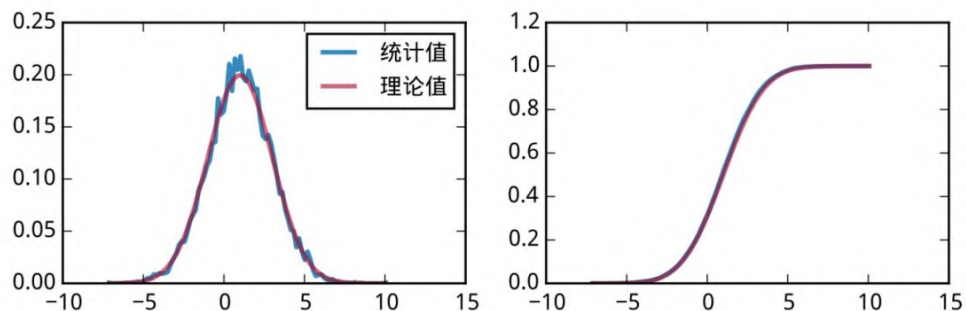


图3-11 正态分布的概率密度函数（左）和累积分布函数（右）

有些随机分布除了loc和scale参数之外，还需要额外的形状参数。例如伽玛分布可用于描述等待k个独立随机事件发生所需的时间，k就是伽玛分布的形状参数。下面计算形状参数k为1和2时的伽玛分布的期望值和方差：

```
print stats.gamma.stats(1.0)
print stats.gamma.stats(2.0)
(array(1.0), array(1.0))
(array(2.0), array(2.0))
```

伽玛分布的尺度参数 θ 和随机事件发生的频率相关，由scale参数指定：

```
stats.gamma.stats(2.0, scale=2)
(array(4.0), array(8.0))
```

根据伽玛分布的数学定义可知其期望值为 $k\theta$ ，方差为 $k\theta^2$ 。上面的程序验证了这两个公式。

当随机分布有额外的形状参数时，它所对应的`rvs()`、`pdf()`等方法都会增加额外的参数来接收形状参数。例如下面的程序调用`rvs()`对 $k=2$ 、 $\theta=2$ 的伽玛分布取4个随机值：

```
x = stats.gamma.rvs(2, scale=2, size=4)

x

array([ 2.47613445,  1.93667652,  0.85723572,  9.49088092])
```

接下来调用pdf()查看与上面4个随机值对应的概率密度:

```
stats.gamma.pdf(x, 2, scale=2)

array([ 0.17948513,  0.18384555,  0.13960273,  0.02062186])
```

也可以先创建将形状参数和尺度参数固定的随机变量,然后再调用其pdf()计算概率密度:

```
X = stats.gamma(2, scale=2)

X.pdf(x)

array([ 0.17948513,  0.18384555,  0.13960273,  0.02062186])
```

3.4.2 离散概率分布

当分布函数的值域为离散时我们称之为离散概率分布。例如投掷有六个面的骰子时,只能获得1到6的整数,因此所得到的概率分布为离散的。对于离散随机分布,通常使用概率质量函数(PMF)描述其分布情况。

在stats模块中所有描述离散分布的随机变量都从rv_discrete类继承,也可以直接用rv_discrete类自定义离散概率分布。例如假设有一个不均匀的骰子,它的各点出现的概率不相等。我们可以用下面的数组x保存骰子的所有可能值,数组p保存每个值出现的概率:

```
x = range(1, 7)

p = (0.4, 0.2, 0.1, 0.1, 0.1, 0.1)
```

然后创建表示这个特殊骰子的随机变量dice，并调用其rvs()方法投掷此骰子20次，获得符合概率p的随机数：

```
dice = stats.rv_discrete(values=(x, p))
dice.rvs(size=20)
array([1, 6, 3, 1, 2, 2, 4, 1, 1, 1, 2, 5, 6, 2, 4, 2, 5, 2, 1,
4])
```

下面我们用程序验证概率论中的中心极限定理：大量相互独立的随机变量，其均值的分布以正态分布为极限。我们计算上面那个特殊骰子投掷50次的平均值，由于每次投掷骰子都可以看作一个独立的随机事件，因此投掷50次的平均值可以看作“大量相互独立的随机变量”，其平均值的分布应该十分接近正态分布。仍然通过rvs()获得取样值，其结果是一个形状为(20000, 50)的数组，沿着第一轴计算每行的平均值，得到samples_mean：

```
np.random.seed(42)
samples = dice.rvs(size=(20000, 50))
samples_mean = np.mean(samples, axis=1)
```

3.4.3 核密度估计

在上面的例子中，由于每个取样都是离散的，因此其平均值也是离散的，对这样的数据进行直方图统计很容易出现许多离散点恰巧聚集到同一区间的现象。为了更平滑地显示样本的概率密度，可以使用kde.gaussian_kde()进行核密度估计。在图3-12中，直方图统计的结果有很大的起伏，而核密度估计与拟合的正态分布十分接近，因此验证了中心极限定理。

```
_, bins, step = pl.hist(
    samples_mean, bins=100, normed=True, histtype="step",
    label=u"直方图统计")

kde = stats.kde.gaussian_kde(samples_mean)
x = np.linspace(bins[0], bins[-1], 100)
pl.plot(x, kde(x), label=u"核密度估计")

mean, std = stats.norm.fit(samples_mean)
pl.plot(x, stats.norm(mean, std).pdf(x), alpha=0.8, label=u"正态
分布拟合")

pl.legend( )
```

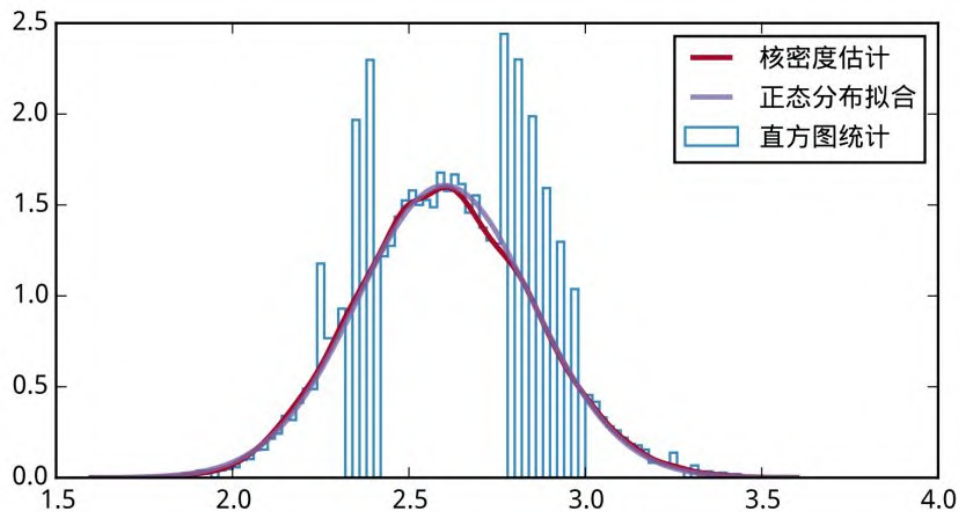


图3-12 核密度估计能更准确地表示随机变量的概率密度函数

核密度估计算法在每个数据点处放置一条核函数曲线，最终的核密度估计就是所有这些核函数曲线的叠加。`gaussian_kde()`的核函数为高斯曲线，其`bw_method`参数决定核函数的宽度，即高斯曲线的方差。`bw_method`参数可以是如下几种情况：

- 当为'`scott`'、'`silverman`'时将采用相应的公式根据数据个数和维数决定核函数的宽度系数。

- 当为函数时将调用此函数计算曲线宽度系数，函数的参数为gaussian_kde对象。
- 当为数值时，将直接使用此数值作为宽度系数。

核函数的方差由数据的方差和宽度系数决定。

下面的程序比较宽度系数对核密度估计的影响。当宽度系数较小时，可以看到在三个数据点处的高斯曲线的峰值，而当宽度逐渐变大时，这些峰值就合并成一个统一的峰值了。

```
for bw in [0.2, 0.3, 0.6, 1.0]:  
    kde = stats.gaussian_kde([-1, 0, 1], bw_method=bw)  
    x = np.linspace(-5, 5, 1000)  
    y = kde(x)  
    pl.plot(x, y, lw=2, label="bw={}".format(bw), alpha=0.6)  
pl.legend(loc="best")
```

图3-13显示bw_method参数越大，核密度估计曲线越平滑。

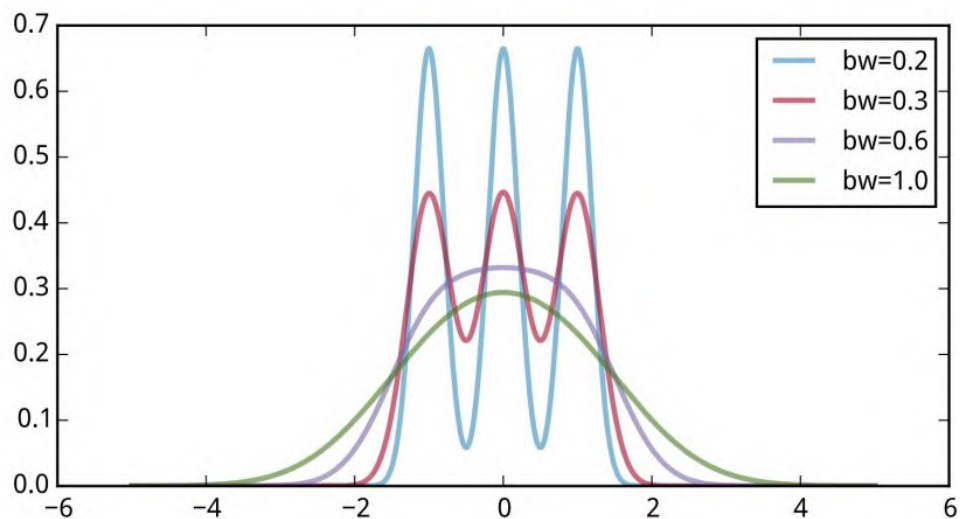


图3-13 bw_method参数越大，核密度估计曲线越平滑

3.4.4 二项分布、泊松分布、伽玛分布

本节用几个实例程序对概率论中的二项分布、泊松分布以及伽玛分布进行一些实验和讨论。

二项分布是最重要的离散概率分布之一。假设有一种只有两个结果的试验，其成功概率为 p ，那么二项分布描述了进行 n 次这样的独立试验，成功 k 次的概率。二项分布的概率质量函数公式如下：

$$f(k; n, p) = \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k}$$

例如，可以通过二项分布的概率质量公式计算投掷5次骰子出现3次6点的概率。投掷一次骰子，点数为6的概率（即试验成功的概率）为 $p=1/6$ ，试验次数为 $n=5$ 。使用二项分布的概率质量函数`pmf()`可以很容易计算出出现 k 次6点的概率。和概率密度函数`pdf()`类似，`pmf()`的第一个参数为随机变量的取值，后面的参数为描述随机分布所需的参数。对于二项分布来说，参数分别为 n 和 p ，而取值范围则为0到 n 之间的整数。下面的程序计算 k 为0到6时对应的概率：

```
stats.binom.pmf(range(6), 5, 1/6.0)
array([ 4.01877572e-01,  4.01877572e-01,  1.60751029e-01,
        3.21502058e-02,  3.21502058e-03,  1.28600823e-04])
```

由结果可知：出现0或1次6点的概率为40.2%，而出现3次6点的概率为3.215%。

在二项分布中，如果试验次数 n 很大，而每次试验成功的概率 p 很小，乘积 np 比较适中，那么试验成功次数的概率可以用泊松分布近似描述。

在泊松分布中使用 λ 描述单位时间（或单位面积）中随机事件的平均发生率。如果将二项分布中的试验次数 n 看作单位时间中所做的试验次数，那么它和事件出现的概率 p 的乘积就是事件的平均发生率 λ ，即 $\lambda = n \cdot p$ 。泊松分布的概率质量函数公式如下：

$$f(k; \lambda) = \frac{e^{-\lambda} \lambda^k}{k!}$$

下面的程序分别计算二项分布和泊松分布的概率质量函数，结果如图3-14所示。可以看出当 n 足够大时，二者是十分接近的。程序中的事件平均发生率 λ 恒等于10。根据二项分布的试验次数 n ，计算每次事件出现的概率 $p = \lambda / n$ 。

```
lambda_ = 10.0
x = np.arange(20)

n1, n2 = 100, 1000

y_binom_n1 = stats.binom.pmf(x, n1, lambda_ / n1)
y_binom_n2 = stats.binom.pmf(x, n2, lambda_ / n2)
y_poisson = stats.poisson.pmf(x, lambda_)
print np.max(np.abs(y_binom_n1 - poisson))
print np.max(np.abs(y_binom_n2 - poisson))

0.00675531110335
0.000630175404978
```

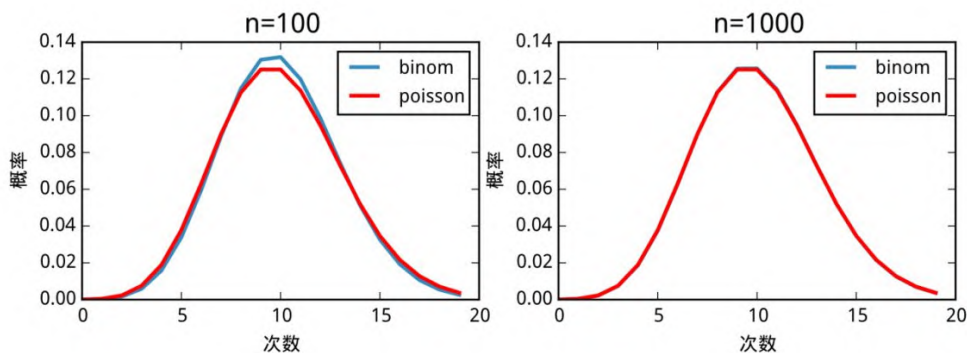


图3-14 当n足够大时二项分布和泊松分布近似相等

泊松分布适合描述单位时间内随机事件发生的次数的分布情况。例如某个设施在一定时间内的使用次数、机器出现故障的次数、自然灾害发生的次数等。

为了加深读者对泊松分布概念的理解，下面我们使用随机数模拟泊松分布，并与概率质量函数进行比较，结果如图3-15所示。图中，每秒内事件的平均发生次数为10，即 $\lambda = 10$ 。其中左图的观察时间为1000秒，而右图的观察时间为50000秒。可以看出观察时间越长，每秒内事件发生的次数越符合泊松分布。

```
np.random.seed(42)

def sim_poisson(lambda_, time):
    t = np.random.uniform(0, time, size=lambda_ *
time) ❶
    count, time_edges = np.histogram(t, bins=time, range=(0,
time)) ❷
    dist, count_edges = np.histogram(count, bins=20, range=(0,
20), density=True) ❸
    x = count_edges[:-1]
```

```

    poisson = stats.poisson.pmf(x, lambda_)
    return x, poisson, dist

lambda_ = 10
times = 1000, 50000
x1, poisson1, dist1 = sim_poisson(lambda_, times[0])
x2, poisson2, dist2 = sim_poisson(lambda_, times[1])
max_error1 = np.max(np.abs(dist1 - poisson1))
max_error2 = np.max(np.abs(dist2 - poisson2))
print "time={}, max_error={}".format(times[0], max_error1)
print "time={}, max_error={}".format(times[1], max_error2)
time=1000, max_error=0.019642302016
time=50000, max_error=0.00179801289496

```

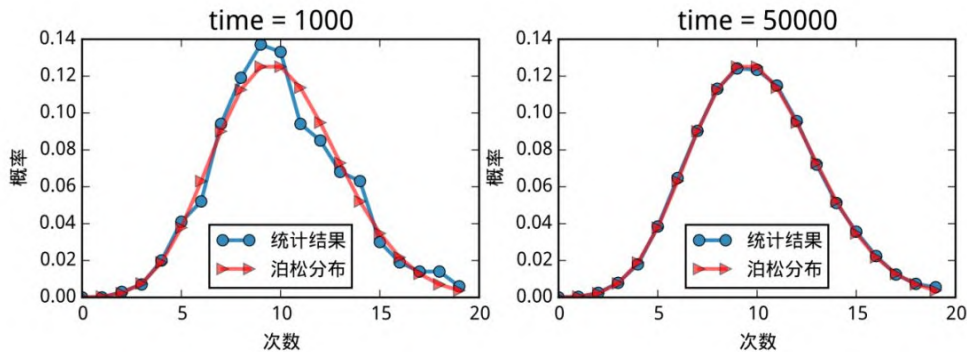


图3-15 模拟泊松分布

❶ 可以用NumPy的随机数生成函数`uniform()`产生平均分布于0到`time`之间的`lambda_* time`个事件所发生的时刻。❷ 用`histogram()`可以统计数组`t`中每秒之内的事件发生的次数`count`，根据泊松分布的定义，`count`数组中的数值的分布情况应该符合泊松分布。❸ 接下来统计事件次数在0到20区间内的概率分布。当`histogram()`的`density`参数为`True`时，结果和概率质量函数相等。

还可以换个角度看随机事件的分布问题。我们可以观察相邻两个事件之间的时间间隔的分布情况，或者隔k个事件的时间间隔的分布情况。根据概率论，事件之间的时间间隔应符合伽玛分布，由于时间间隔可以是任意数值，因此伽玛分布是连续概率分布。伽玛分布的概率密度函数公式如下，它描述第k个事件发生所需的等待时间的概率分布。 $\Gamma(k)$ 是伽玛函数，当k为整数时，它的值和k的阶乘k!相等。

$$f(X; k, \lambda) = \frac{\lambda^k e^{-\lambda X} X^{k-1}}{\Gamma(k)}$$

下面的程序模拟了事件的时间间隔的伽玛分布，结果如图3-16所示。图中的观察时间为1000秒，平均每秒产生10个事件。左图中k=1，它表示相邻两个事件间的间隔的分布，而k=2则表示相隔一个事件的两个事件间的间隔的分布，可以看出它们都符合伽玛分布。

```
def sim_gamma(lambda_, time, k):  
    t = np.random.uniform(0, time, size=lambda_ *  
time) ❶  
    t.sort() ❷  
    interval = t[k:] - t[:-k] ❸  
    dist, interval_edges = np.histogram(interval, bins=100,  
density=True) ❹  
    x = (interval_edges[1:] + interval_edges[:-1])/2 ❺  
    gamma = stats.gamma.pdf(x, k, scale=1.0/lambda_) ❻  
    return x, gamma, dist  
  
lambda_ = 10.0  
time = 1000  
ks = 1, 2
```

```
x1, gamma1, dist1 = sim_gamma(lambda_, time, ks[0])
x2, gamma2, dist2 = sim_gamma(lambda_, time, ks[1])
```

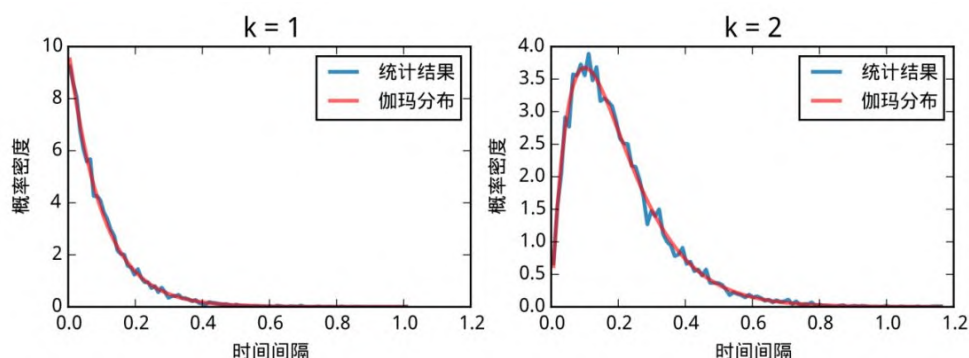


图3-16 模拟伽玛分布

❶首先在1000秒之内产生10000个随机事件发生的时刻。因此事件的平均发生次数为每秒10次。❷为了计算事件前后的时间间隔，需要先对随机时刻进行排序，❸然后再计算k个事件之间的时间间隔。❹对该时间间隔调用`histogram()`进行概率统计，设置`density`为`True`可以直接计算概率密度。`histogram()`返回的第二个值为统计区间的边界，❺接下来用`gamma.pdf()`计算伽玛分布的概率密度时，使用各个区间的中值进行计算。`pdf()`的第二个参数为k值，`scale`参数为 $1/\lambda$ 。

接下来我们看一道关于伽玛分布的概率题：有A和B两路公交车，平均发车间隔时间分别是5分钟和10分钟，某乘客在站点S可以任意选择两者之一乘坐，假设A和B到达S的时刻无法确定，计算该乘客的平均等待公交车的时间。

可以将“假设A和B到达S的时刻无法确定”理解为公交车到达S站点的时刻是完全随机的，因此单位时间之内到达S站点的公交车次数符合泊松分布，而前后两辆公交车的时间差符合 $k=1$ 的伽玛分布。下面我们先随机数模拟的方法求出近似解，然后推导出解的公式。

```

T = 100000
A_count = T / 5
B_count = T / 10

A_time = np.random.uniform(0, T, A_count) ❶
B_time = np.random.uniform(0, T, B_count)

bus_time = np.concatenate((A_time, B_time)) ❷
bus_time.sort( )

N = 200000
passenger_time = np.random.uniform(bus_time[0], bus_time[-1], N)
❸

idx = np.searchsorted(bus_time, passenger_time) ❹

np.mean(bus_time[idx] - passenger_time) *

60 ❺

199.12512768644049

```

模拟的总时间为T分钟，在这段之间之内，应该有A_count次A路公交车和B_count次B路公交车到达S站点。❶可以用均匀分布uniform()产生两路公交车到达S站点的时刻，❷将这两个保存时刻的数组连接起来，并进行排序。

❸在第一趟和最后一趟公交车的到达时间之间，产生乘客随机到达S站点的时刻数组。❹在已经排序的公交车到达时刻数组bus_time中使用二分法搜索每个乘客到达时刻所在的下标数组idx。❺bus_time[idx]就是乘客

到达车站之后第一个到达车站的公交车的时刻，因此只需要计算其差值，并求平均值即可。通过随机数模拟得出的平均等待时间约为200秒。

将A和B两路汽车一起考虑，前后两个车次的平均间隔也为200秒。这似乎有些不可思议，直觉上我们可能期待一个小于平均间隔的等待时间。

```
np.mean(np.diff(bus_time)) *  
  
60  
  
199.98208112933918
```

这是因为存在观察者偏差，即会有更多的乘客出现在时间间隔较长的时间段。我们可以想象如果公交车因为事故晚点很长时间，那么通常车站上会挤满等待的人。在图3-17（上）中，蓝色竖线代表公交车的到站时刻，红色竖线代表乘客的到站时刻。可以看出，两条蓝色竖线之间的距离越大，其间的红色竖线就会越多。图3-17（下）的横轴是前后两辆公交车的时间差，纵轴是这段时间差之内的等待人数，可以看出二者成正比关系。

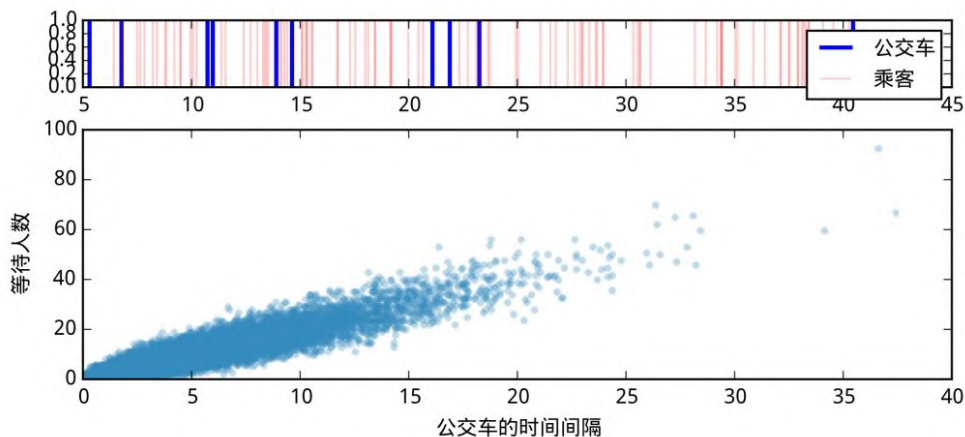


图3-17 观察者偏差

通过以上分析，不难写出计算平均等待时间的计算公式：

$$\frac{\int_0^{+\infty} \frac{x}{2} xf(x)dx}{\int_0^{+\infty} xf(x)dx}$$

在公式中， x 是两辆公交车之间的间隔时间， $f(x)dx$ 是时间间隔为 x 出现的概率。由于观察者效应，乘客出现在较长时间间隔的概率也较大，因此 $xf(x)dx$ 可以看作与乘客出现在时间间隔为 x 时段的概率成比例的量，分母的积分将其归一化。而分子中的 $x/2$ 是在该时间间隔段到达车站所需的平均等待时间。下面我们计算该公式，由图3-17可知，公交车的间隔几乎不会超过30分钟，因此虽然公式中的积分上限为 $+\infty$ ，但在实际计算时只需要指定一个较大的数即可。在本章后续的小节中会详细介绍数值积分`quad()`的用法。

```
from scipy import integrate
t = 10.0 / 3 # 两辆公交车之间的平均时间间隔
bus_interval = stats.gamma(1, scale=t)

n, _ = integrate.quad(lambda x: 0.5 *
x *
x *
bus_interval.pdf(x), 0, 1000)

d, _ = integrate.quad(lambda x: x *
bus_interval.pdf(x), 0, 1000)

n / d *
60
200.0
```


3.4.5 学生t-分布与t检验

从均值为 μ 的正态分布中，抽取有 n 个值的样本，计算样本均值 $\bar{x}$ 和样本方差 s ：

$$\bar{x} = \frac{x_1 + \cdots + x_n}{n}, \quad s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

则 $t = \frac{\bar{x} - \mu}{s/\sqrt{n}}$ 符合 $df=n-1$ 的学生 t -分布。 t 值是抽选的样本的平均值与整体样本的期望值之差经过正规化之后的数值，可以用来描述抽取的样本与整体样本之间的差异。

下面的程序模拟学生 t -分布（参见图3-18），❶创建一个形状为 $(100000, 10)$ 的正态分布的随机数数组，❷使用上面的公式计算 t 值，❸统计 t 值的分布情况并与 `stats.t` 的概率密度函数进行比较。如果我们使用10个样本计算 t 值，则它应该符合 $df=9$ 的学生 t -分布。

```
mu = 0.0
n = 10
samples = stats.norm(mu).rvs(size=(100000, n)) ❶
t_samples = (np.mean(samples, axis=1) - mu) / np.std(samples,
ddof=1, axis=1) *
n**
0.5 ❷
sample_dist, x = np.histogram(t_samples, bins=100, density=True)
```

3

```

x = 0.5 *

(x[:-1] + x[1:])

t_dist = stats.t(n-1).pdf(x)

print "max error:", np.max(np.abs(sample_dist - t_dist))

max error: 0.00658734287935

```

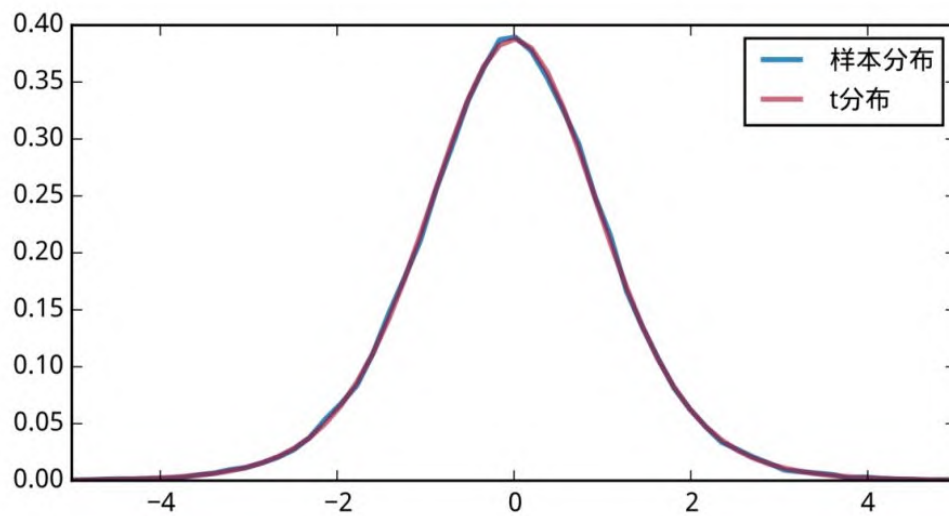


图3-18 模拟学生t-分布

图3-19绘制df为5和39的概率密度函数和生存函数，当df增大时，学生t-分布趋向于正态分布。

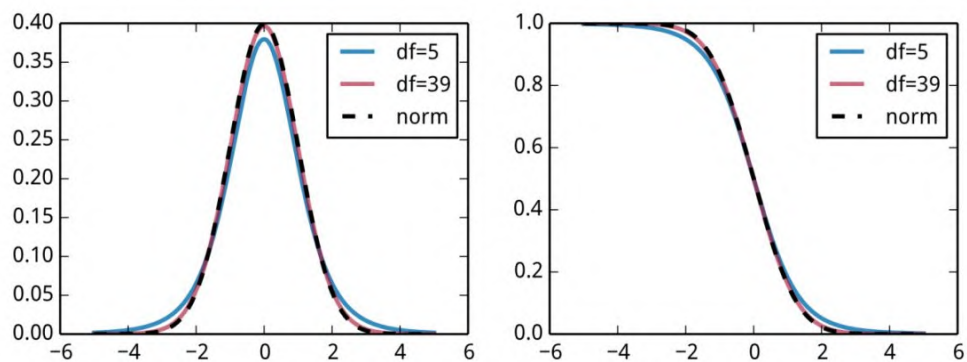


图3-19 当df增大时，学生t-分布趋向于正态分布

学生t-分布可以用于检测样本的平均值，下面我们从一个期望值为1的正态分布的随机变量中取30个数值：

```
n = 30
np.random.seed(42)
s = stats.norm.rvs(loc=1, scale=0.8, size=n)
```

我们建立整体样本的期望值为0.5的零假设，并用`stats.ttest_1samp()`检验零假设是否能够被推翻。`stats.ttest_1samp()`返回的第一个值为使用前述公式计算的t值，第二个值被称为p值。当p值小于0.05时，通常我们认为零假设不成立。因此下面的测试表明我们可以拒绝整体样本的期望值为0.5的假设。

零假设

在统计学中，零假设或虚无假设(null hypothesis)是做统计检验时的一类假设。零假设的内容一般是希望被证明为错误的假设或是需要着重考虑的假设。

```
t = (np.mean(s) - 0.5) / (np.std(s, ddof=1) / np.sqrt(n))
print t, stats.ttest_1samp(s, 0.5)
2.65858434088 (2.6585843408822241, 0.012637702257091229)
```

下面我们检验期望值是否为1.0，由于p值大于0.05，我们不能推翻期望值为1.0的零假设，但这并不意味着可以接受该假设，因为期望值为0.9的假设对应的p值也大于0.05，甚至比1.0的t值还大。

```
print (np.mean(s) - 1) / (np.std(s, ddof=1) / np.sqrt(n))
print stats.ttest_1samp(s, 1), stats.ttest_1samp(s, 0.9)
```

```
-1.14501736704
(-1.1450173670383303, 0.26156414618801477)
(-0.38429702545421962, 0.70356191034252025)
```

通过`ttest_1samp()`计算的p值就是图3-20中红色部分的面积。可以这样理解p值的含义：如果随机变量的期望值真的和假设的相同，那么从这个随机变量中随机抽取n个数值，其t值比测试样本的t值还极端（绝对值大）的可能性为p。因此当p很小时，我们可以推断假设不太可能成立。反过来当p值较大时，则不能推翻零假设，注意不能推翻假设并不代表能接受该假设。

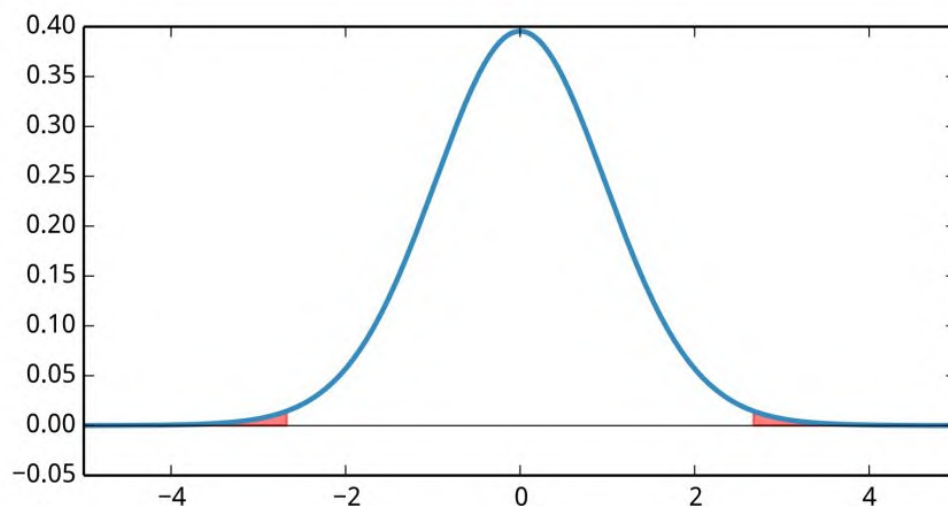


图3-20 红色部分为`ttest_1samp()`计算的p值

拿上面期望值为0.5的测试为例：如果整体样本的期望值真的是0.5，那么抽取到t值大于2.65858或小于-2.65858的样本的概率为0.0126377。由于这个概率太小，因此整体样本的期望值应该不是0.5。也可以这样理解：如果整体样本的期望值为0.5，那么随机抽取比s更极端的样本的概率为0.0126377。

```
x = np.linspace(-5, 5, 500)
y = stats.t(n-1).pdf(x)
```

```
plt.plot(x, y, lw=2)

t, p = stats.ttest_1samp(s, 0.5)

mask = x > np.abs(t)

plt.fill_between(x[mask], y[mask], color="red", alpha=0.5)

mask = x < -np.abs(t)

plt.fill_between(x[mask], y[mask], color="red", alpha=0.5)

plt.axhline(color="k", lw=0.5)

plt.xlim(-5, 5)
```

下面用`scipy.integrate.trapz()`积分验证p值，由于左右两块红色面积是相等的，下面的积分只需要计算其中一块的面积：

```
from scipy import integrate
x = np.linspace(-10, 10, 100000)
y = stats.t(n-1).pdf(x)
mask = x >= np.abs(t)

integrate.trapz(y[mask], x[mask])*
```

2

```
0.012633433707685974
```

下面我们用随机数验证前面计算的p值，我们创建m组随机数，每组都有n个数值，然后计算假设总体样本期望值为0.5时每组随机数对应的t值 t_r ，它是一个长度为m的数组。将 t_r 与样本的t值 t_s 进行绝对值大小比较，当 $|t_r| > |t_s|$ 时，就说明该组随机数比样本更极端，统计极端组出现的概率，可以看到它和p值是相同的。

```
m = 200000

mean = 0.5

r = stats.norm.rvs(loc=mean, scale=0.8, size=(m, n))
```

```

ts = (np.mean(s) - mean) / (np.std(s, ddof=1) / np.sqrt(n))

tr = (np.mean(r, axis=1) - mean) / (np.std(r, ddof=1, axis=1) /
np.sqrt(n))

np.mean(np.abs(tr) > np.abs(ts))

0.012695

```

如果s1和s2是两个独立的来自正态分布总体的样本，可以通过ttest_ind()检验这两个总体的均值是否存在差异。通过equal_var参数指定两个总体的方差是否相同。在下面的例子中，❶由于s1和s2样本来自不同方差的总体，因此equal_var参数为False。由于p<0.05，因此认为两个总体的均值存在差异。❷s2和s3来自相同方差的总体，因此equal_var参数为True，所得到的p值很大，因此无法推翻零假设，也就无法否定两个总体的均值相同的零假设。

```

np.random.seed(42)

s1 = stats.norm.rvs(loc=1, scale=1.0, size=20)
s2 = stats.norm.rvs(loc=1.5, scale=0.5, size=20)
s3 = stats.norm.rvs(loc=1.5, scale=0.5, size=25)

print stats.ttest_ind(s1, s2, equal_var=False) ❶
print stats.ttest_ind(s2, s3, equal_var=True) ❷

(-2.2391470627176755, 0.033250866086743665)
(-0.59466985218561719, 0.55518058758105393)

```

3.4.6 卡方分布和卡方检验

卡方分布 (χ^2) 是概率论与统计学中常用的一种概率分布。 k 个独立的标准正态分布变量的平方和服从自由度为 k 的卡方分布。下面通过随机数验证该分布，结果如图3-21所示：

```
a = np.random.normal(size=(300000, 4))

cs = np.sum(a**2, axis=1)

sample_dist, bins = np.histogram(cs, bins=100, range=(0, 20),
density=True)

x = 0.5 *

(bins[:-1] + bins[1:])

chi2_dist = stats.chi2.pdf(x, 4)

print "max error:", np.max(np.abs(sample_dist - chi2_dist))

max error: 0.00340194486328
```

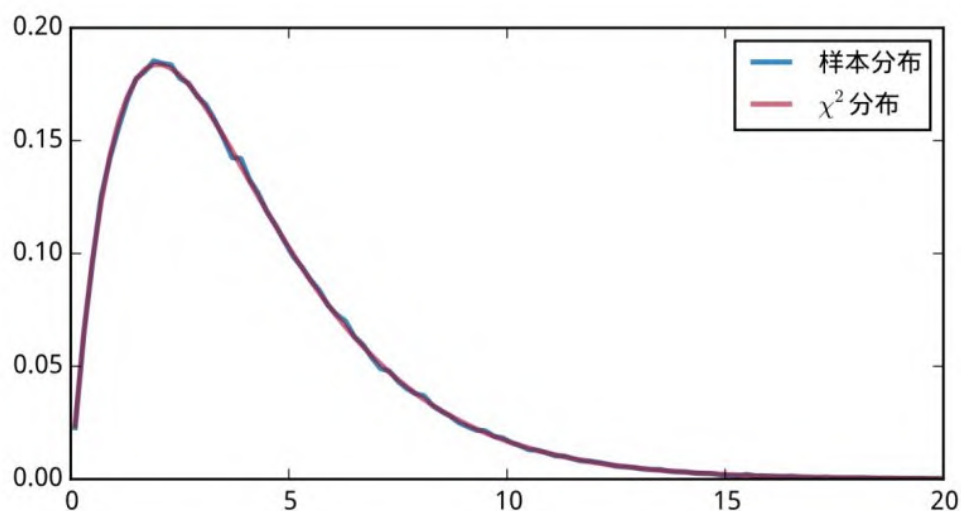


图3-21 使用随机数验证卡方分布

卡方分布可以用来描述这样的概率现象：袋子里有5种颜色的球，抽到每种球的概率相同，从中选N次，并统计每种颜色的次数 O_i 。则下面的 χ^2 符合自由度为4的卡方分布，其中 $E=N/5$ 为每种球被抽选的期望次数：

$$\chi^2 = \sum_{i=1}^k \frac{(O_i - E)^2}{E}$$

下面用程序模拟这个过程，结果如图3-22所示。❶首先调用`randint()`创建从0到5的随机数，其结果`ball_ids`的第0轴表示实验次数，第1轴为每次实验抽取的100个球的编号。❷使用`bincount()`统计每次实验中每个编号出现的次数，由于它不支持多维数组，因此这里使用`apply_along_axis()`对第1轴上的数据循环调用`bincount()`。为了保证每行的统计结果的长度相同，设置`minlength`参数为5，`apply_along_axis()`会将所有关键字参数传递给进行实际运算的函数。❸使用上面的公式计算 χ^2 统计量`cs2`，❹并用`gaussian_kde()`计算`cs2`的分布情况。

```
repeat_count = 60000
n, k = 100, 5

np.random.seed(42)

ball_ids = np.random.randint(0, k, size=(repeat_count, n)) ❶
counts = np.apply_along_axis(np.bincount, 1, ball_ids,
minlength=k) ❷

cs2 = np.sum((counts - n/k)**
2.0/(n/k), axis=1) ❸

k = stats.kde.gaussian_kde(cs2) ❹
x = np.linspace(0, 10, 200)
pl.plot(x, stats.chi2.pdf(x, 4), lw=2, label=u"$\chi ^{2}$分布")
```



```

pl.plot(x, k(x), lw=2, color="red", alpha=0.6, label=u"样本分布")
pl.legend(loc="best")
pl.xlim(0, 10)

```

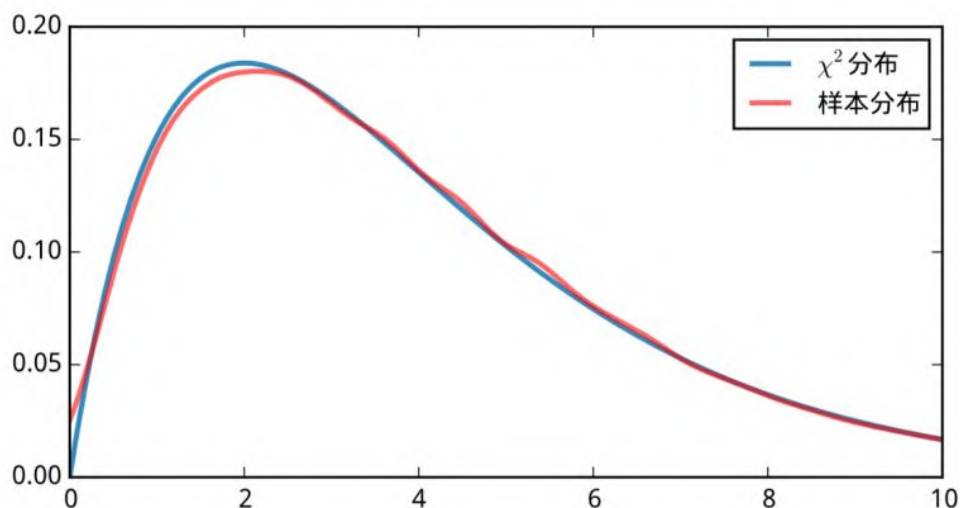


图3-22 模拟卡方分布

卡方检验可以用来评估观测值与理论值的差异是否只是因为随机误差造成的。在下面的例子中，袋子中各种颜色的球按照probabilities参数指定的概率分布，choose_balls(probabilities, size)从袋中选择size次并返回每种球被选中的次数。袋子1中的球的概率分布为：0.18、0.24、0.25、0.16、0.17。袋子2中各种颜色的球的个数一样多。通过调用choose_balls()得到两组数字：80 93 97 64 66和89 76 79 71 85。现在需要判断袋子中的球是否是平均分布的。

```

def choose_balls(probabilities, size):
    r = stats.rv_discrete(values=(range(len(probabilities)),
probabilities))

    s = r.rvs(size=size)

    counts = np.bincount(s)

    return counts

```

```

np.random.seed(42)

counts1 = choose_balls([0.18, 0.24, 0.25, 0.16, 0.17], 400)

counts2 = choose_balls([0.2]*
5, 400)

counts1          counts2
-----
[80, 93, 97, 64, 66] [89, 76, 79, 71, 85]

```

使用`chisquare()`进行卡方检验，它的参数为每种球被选中次数的列表，如果没有设置检验的目标概率，就测试它们是否符合平均分布。卡方检验的零假设为样本符合目标概率，由下面的检验结果可知，第一个袋子对应的p值只有0.02，也就是说如果第一个袋子中的球真的符合平均分布，那么得到的观测结果80 93 97 64 66的概率只有2%，因此可以推翻零假设，即袋子中的球不太可能是平均分布的。第二个袋子对应的p值为0.64，无法推翻零假设，即我们的结论是不能否定第二个袋子中的球是平均分布的。注意，和前面介绍的t检验一样，零假设只能用来否定，因此不能根据观测结果89 76 79 71 85得出袋子中的球是符合平均分布的结论。

```

chi1, p1 = stats.chisquare(counts1)
chi2, p2 = stats.chisquare(counts2)

print "chi1 =", chi1, "p1 =", p1
print "chi2 =", chi2, "p2 =", p2

chi1 = 11.375 p1 = 0.0226576012398
chi2 = 2.55 p2 = 0.635705452704

```

卡方检验是通过卡方分布进行计算的。图3-23显示自由度为4的卡方分布的概率密度函数，以及`chi1`和`chi2`对应的位置 χ_1^2 和 χ_2^2 。 p_1 是 χ_1^2 右侧部

分的面积，而 p_2 是 χ^2_2 右侧的面积。

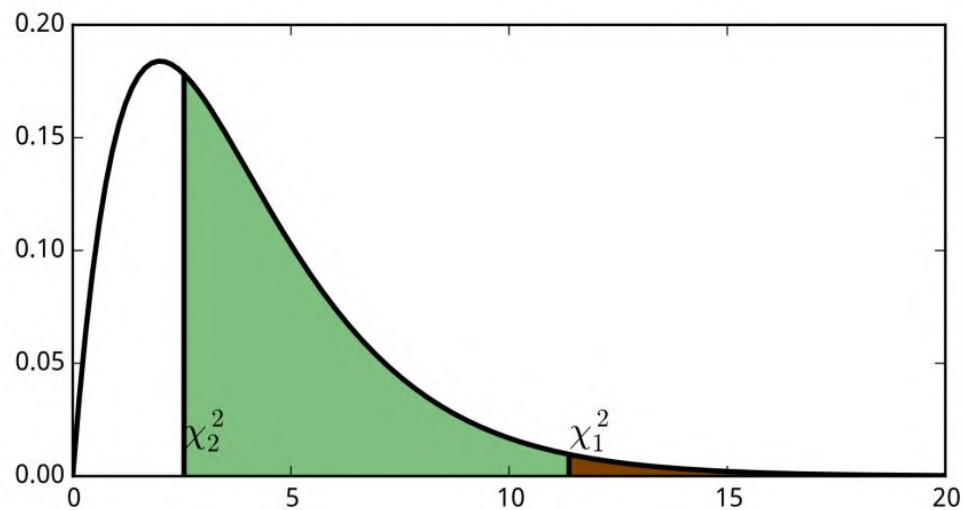


图3-23 卡方检验计算的概率为阴影部分的面积

卡方检验还可以用于二维数据。前面介绍的彩色球的例子中，只是按照球的颜色分组，而二维数据则按照样本的两个属性分组，统计学上称之为列联表。例如下面是性别与惯用手的列联表，我们希望知道这个统计结果能否说明性别与惯用手之间存在某种联系。

	右手	左手
男性	43	9
女性	44	4

下面使用`chi2_contingency()`对列联表进行卡方检验，零假设为性别与惯用手之间不存在联系，即男性与女性惯用左右手的概率相同。由于p值为0.3，因此不能推翻零假设，即该实验数据中没有明显证据表明男性和女性在使用左右手习惯上存在区别。

```
table = [[43, 9], [44, 4]]

chi2, p, dof, expected = stats.chi2_contingency(table)

        chi2                p
-----
1.0724852071005921  0.30038477039056899
```

对于上面的 2×2 的数值较小的列联表，可以使用`fisher_exact()`计算出精确的p值：

```
stats.fisher_exact(table)

(0.43434343434343436, 0.23915695682225618)
```

3.5 数值积分-integrate



与本节内容对应的 Notebook 为：`03-scipy/scipy-500-integrate.ipynb`。

SciPy的`integrate`模块提供了几种数值积分算法，其中包括对常微分方程组(ODE)的数值积分。

3.5.1 球的体积

数值积分是对定积分的数值求解，例如可以利用数值积分计算某个形状的面积。让我们先考虑一下如何计算半径为1的半圆的面积。根据圆的面积公式，其面积应该等于 $\pi/2$ 。单位半圆的曲线方程为 $y = \sqrt{1 - x^2}$ ，可以通过下面的half_circle()进行计算：

```
def half_circle(x):  
    return (1-x**  
  
2)**  
  
0.5
```

最简单的数值积分算法就是将要积分的面积分为许多小矩形，然后计算这些矩形的面积之和。下面使用这种方法，将X轴上-1到1的区间分为10000等份，然后计算面积和：

```
N = 10000  
x = np.linspace(-1, 1, N)  
dx = x[1] - x[0]  
y = half_circle(x)  
  
2 *  
  
dx *  
  
np.sum(y) # 面积的两倍  
3.1415893269307373
```

也可以用NumPy的trapz()计算半圆上的各点所构成的多边形的面积，trapz()计算的是以(x, y)为顶点坐标的折线与X轴所夹的面积：

```

np.trapz(y, x) *

2 # 面积的两倍

3.1415893269315975

```

如果使用`integrate.quad()`进行数值积分，就能得到非常精确的结果：

```

from scipy import integrate
pi_half, err = integrate.quad(half_circle, -1, 1)

pi_half *

2

3.141592653589797

```

计算多重定积分可以通过多次调用`quad()`实现，为了调用方便，`integrate`模块提供了`dblquad()`进行二重定积分，提供了`tplquad()`进行三重定积分。下面以计算单位半球体积为例，说明`dblquad()`的用法。

单位半球面上的点 (x, y, z) 满足方程 $x^2 + y^2 + z^2 = 1$ ，因此下面的`half_sphere()`可以通过X-Y轴坐标计算球面上的点的Z轴坐标值：

```

def half_sphere(x, y):

    return (1-x**

2-y**

2)**

```

X-Y轴平面与此球体的交线为一个单位圆，因此二重积分的计算区间为此单位圆。即对于X轴从-1到1进行积分，而对于Y轴则从-half_circle(x)到half_circle(x)进行积分。因此半球体积的二重积分公式为：

$$\int_{-1}^1 \int_{-\sqrt{1-x^2}}^{\sqrt{1-x^2}} \sqrt{1-x^2-y^2} dy dx$$

下面的程序使用dblquad()计算半球体积：

```
volume, error = integrate.dblquad(half_sphere, -1, 1,
                                   lambda x:-half_circle(x),
                                   lambda x:half_circle(x))

print volume, error, np.pi*
```

4/3/2

2.09439510239 2.32524566534e-14 2.09439510239

dblquad()的调用参数为：dblquad(func2d, a, b, gfun, hfun)。其中，func2d是需要进行二重积分的函数，它有两个参数，假设分别为x和y。a和b参数指定被积分函数的第一个变量x的积分区间，而gfun和hfun参数指定第二个变量y的积分区间。gfun和hfun是函数，它们通过变量x计算出变量y的积分区间，这样可以对X-Y平面上的任何区间对func2d进行积分。

图3-24是半球体积的积分的示意图。从此示意图可以看出，X轴的积分区间为-1.0到1.0，对于X轴上的某点x0，通过对Y轴的积分可以计算出图中

深色的垂直切面的面积，因此Y轴的积分区间如图中的点线所示。

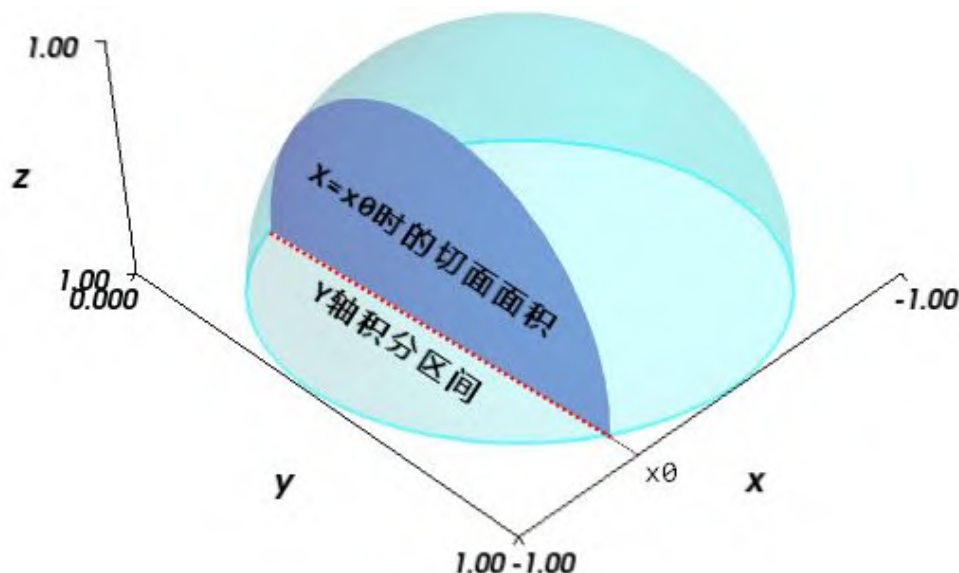


图3-24 半球体二重积分示意图

3.5.2 解常微分方程组

integrate模块还提供了对常微分方程组进行积分的函数odeint()。下面我们看看如何用它计算洛伦茨吸引子的轨迹。洛伦茨吸引子由下面的三个微分方程定义：

$$\frac{dx}{dt} = \sigma \cdot (y - x), \quad \frac{dy}{dt} = x \cdot (\rho - z) - y, \quad \frac{dz}{dt} = xy - \beta z$$

这三个方程定义了三维空间中各个坐标点上的速度矢量。从某个坐标开始沿着速度矢量进行积分，就可以计算出无质量点在此空间中的运动轨迹。其中 σ 、 ρ 、 β 为三个常数，不同的参数可以计算出不同的运动轨迹： $x(t)$ 、 $y(t)$ 、 $z(t)$ 。当参数为某些值时，轨迹出现混沌现象。即微小的初值差别也会显著地影响运动轨迹。下面是洛伦茨吸引子的轨迹计算和绘制程序。

❶程序中先定义一个函数`lorenz()`，它的任务是计算出某个坐标点的各个方向上的微分值，它可以直接根据洛伦茨吸引子的公式得出。

❷❸使用不同的位移初始值两次调用`odeint()`，对微分方程求解。`odeint()`有许多参数，这里用到的4个参数分别为：

- `lorenz`：它是计算某个位置上的各个方向的速度的函数。
- `(0.0, 1.0, 0.0)`：位置初始值，它是计算常微分方程所需的各个变量的初始值。
- `t`：表示时间的数组，`odeint()`对此数组中的每个时间点进行求解，得出所有时间点的位置。
- `args`：这些参数直接传递给`lorenz()`，因此它们在整个积分过程中都是常量。

图3-25显示了`odeint()`所得到的轨迹。由结果可知，即使初始值只相差0.01，两条运动轨迹也是完全不同的。

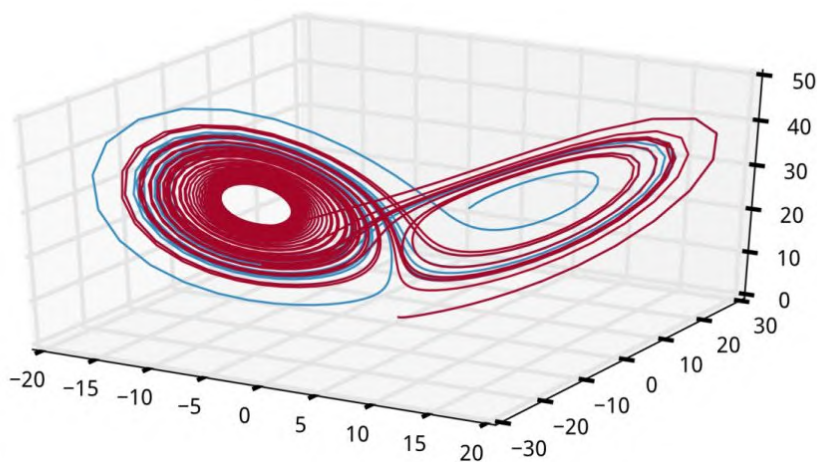


图3-25 洛伦茨吸引子：微小的初值差别也会显著地影响运动轨迹



http://bzhang.lamost.org/website/archives/lorenz_attactor

洛伦茨吸引子的详细介绍。

```
from scipy.integrate import odeint
import numpy as np

def lorenz(w, t, p, r, b): ❶
    # 给出位置矢量w和三个参数p、r、b
    # 计算出dx/dt、dy/dt、dz/dt的值
    x, y, z = w.tolist()
    # 直接与lorenz的计算公式对应
    return p*

(y-x), x*

(r-z)-y, x*

y-b*

z

t = np.arange(0, 30, 0.02) # 创建时间点
# 调用ode对lorenz进行求解，用两个不同的初始值
```

```

track1 = odeint(lorenz, (0.0, 1.00, 0.0), t, args=(10.0, 28.0,
3.0)) ❷

track2 = odeint(lorenz, (0.0, 1.01, 0.0), t, args=(10.0, 28.0,
3.0)) ❸

```

3.5.3 ode类

使用odeint()可以很方便地计算微分方程组的数值解，只需调用一次odeint()就能计算出一组时间点上的系统状态。但是有时我们希望一次只前进一个时间片段，从而对求解对象进行更精确的控制。在下面的例子中，我们通过ode类模拟如图3-26所示的弹簧系统每隔1ms周期的系统状态，并通过PID控制器控制滑块的位置。

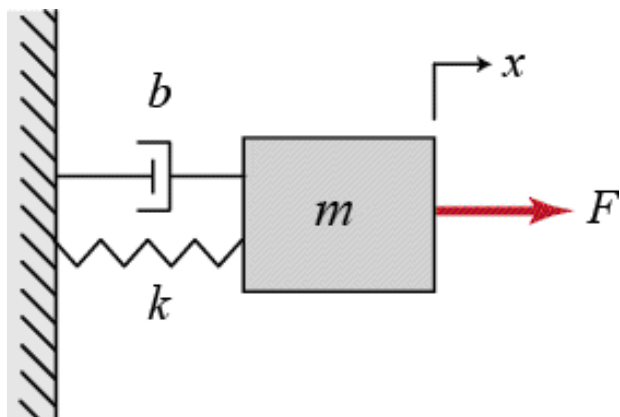


图3-26 质量-弹簧-阻尼系统

该系统的微分方程为： $m\ddot{x} + b\dot{x} + kx = F$ 。其中 x 为滑块的位移， $\ddot{x}$ 为位移对时间的二次导数，即滑块的加速度， $\dot{x}$ 为滑块的速度， m 为滑块的质量， b 为阻尼系数， k 为弹簧的系数， F 为外部施加于滑块的控制力。这是一个二次微分方程，为了使用ode对系统求解，需要将其转换成如下阶微分方程组：

$$\dot{x} = u, \quad \dot{u} = (F - kx - bu)/m$$

其中 x 为滑块的位移， u 为滑块的速度。这两个变量构成了系统的状态，它们对时间的导数可以通过这两个方程直接算出。

```
def mass_spring_damper(xu, t, m, k, b, F):
    x, u = xu.tolist()
    dx = u
    du = (F - k*
x - b*
u)/m
    return dx, du
```

下面使用`odeint()`对该系统进行求解，初值为滑块在位移0.0处，起始速度为0，外部控制力恒为1.0。如图3-27所示，系统经过约两秒钟，最终停在了位移0.05米处。

```
m, b, k, F = 1.0, 10.0, 20.0, 1.0
init_status = 0.0, 0.0
args = m, k, b, F
t = np.arange(0, 2, 0.01)
result = odeint(mass_spring_damper, init_status, t, args)
```

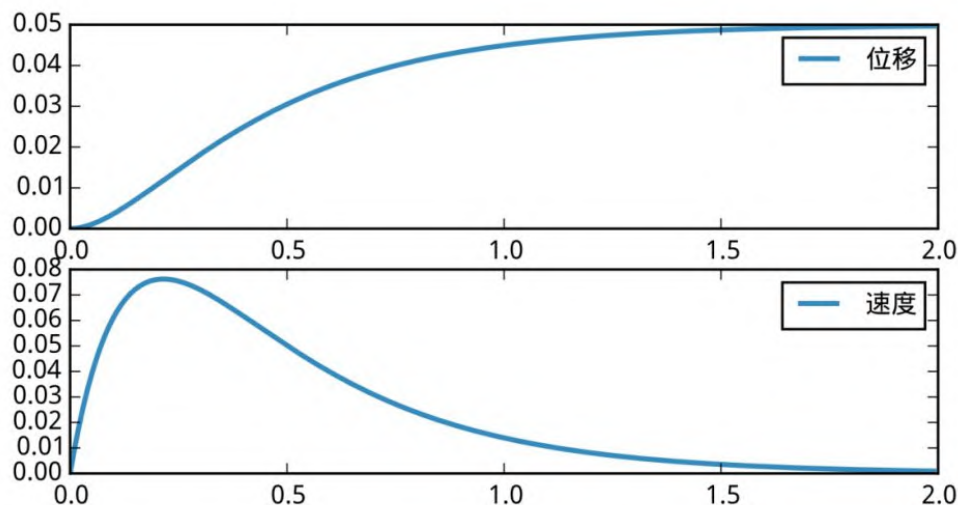


图3-27 滑块的速度和位移曲线

我们希望通过控制外力 F ，使得滑块更迅速地停在位移1.0处，这时可以使用PID控制器进行控制。在介绍PID控制器之前，首先让我们用ode类重写odeint()模拟的部分。ode类和odeint()一样，也需要一个计算各个状态的导数的函数。

❶这里使用MassSpringDamper类的方法f()计算状态点处的导数。注意该方法的参数顺序和odeint()所需的函数mass_spring_damper()不同，第一个参数为时间，第二个参数为系统状态。并且该方法必须返回一个列表来表示各个状态的导数，不能返回元组。这里使用MassSpringDamper类将系统的各个参数 M 、 k 、 b 以及 F 等包装在对象内部。

❷创建ode对象之后，通过set_integrator()设置积分器相关的参数。它的第一个参数为积分器的算法，其后的关键字参数设置该积分器算法的各个参数。关于各个参数的具体含义请读者参阅ode类的文档说明。然后调用set_initial_value()设置系统的初始状态和初始时间。

❸在while循环中，以 dt 为间隔对系统进行积分求解。ode对象的属性 $r.t$ 为当前的模拟时间，调用 $r.integrate(r.t + dt)$ 计算 $r.t + dt$ 处的状

态。系统的状态保存在`r.y`中。我们用两个列表`t`和`result`分别保存模拟时间和系统状态。

由`allclose()`的比较结果可知使用`ode`的结果与`odeint()`的完全相同。

```
from scipy.integrate import ode

class MassSpringDamper(object): ❶

    def __init__(self, m, k, b, F):
        self.m, self.k, self.b, self.F = m, k, b, F

    def f(self, t, xu):
        x, u = xu.tolist()

        dx = u

        du = (self.F - self.k*
x - self.b*

u)/self.m

        return [dx, du]

system = MassSpringDamper(m=m, k=k, b=b, F=F)
init_status = 0.0, 0.0
dt = 0.01

r = ode(system.f) ❷
r.set_integrator('vode', method='bdf')
```

```

r.set_initial_value(init_status, 0)

t = [ ]
result2 = [init_status]
while r.successful( ) and r.t + dt < 2: ❸
    r.integrate(r.t + dt)
    t.append(r.t)
    result2.append(r.y)

result2 = np.array(result2)
np.allclose(result, result2)

True

```

由于在while循环中我们逐点对系统的状态进行模拟，因此可以在其中增加控制代码，改变作用于滑块上的力，从而使滑块更迅速地停在目标位置。这里我们将采用控制理论中最常用的控制方法：PID控制。采用PID控制器的系统框图如图3-28所示。

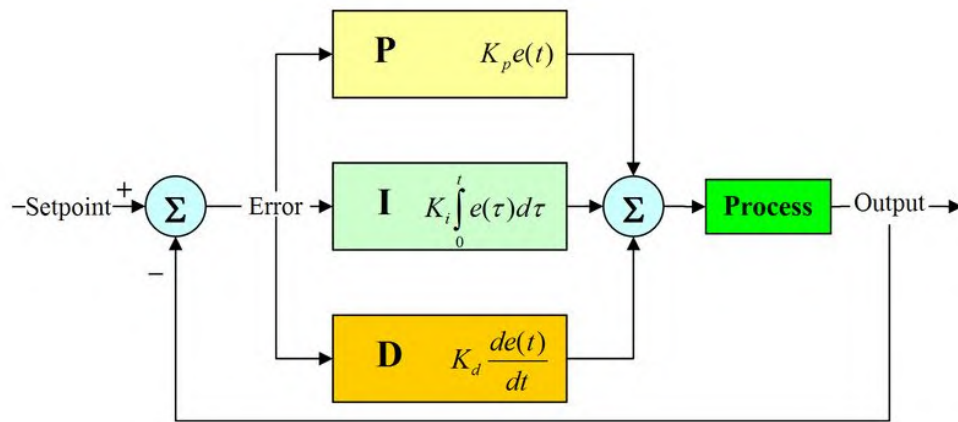


图3-28 PID控制系统

图3-28中，Setpoint为控制的目标状态，在本例中为滑块的目标位移，控制目标系统Process（即质量-弹簧-阻尼系统）的输出Output为系统

的输出，在本例中为滑块的实际位移，二者的误差Error作为PID系统的输入。PID控制器由三个独立的部分构成：

- P：比例项，输出和误差成正比。
- I：积分项，输出和误差的积分成正比。
- D：微分项，输出和误差的微分成正比。

三项之和作为PID控制器的输出，在本例中控制器的输出为作用在滑块之上的外力F。系统框图中使用积分和微分符号表示积分项和微分项，在实际的控制系统中，我们以一定的时间间隔计算控制器的输出，这时积分项可以用累加变量self.status表示，而微分项可以用当前的误差以及前次的误差self.last_error之间的差进行计算。

下面是PID控制器的程序，我们使用类把kp、ki、kd、dt等参数、累加变量status以及上一次的误差last_error等包装起来。

```
class PID(object):

    def __init__(self, kp, ki, kd, dt):
        self.kp, self.ki, self.kd, self.dt = kp, ki, kd, dt
        self.last_error = None
        self.status = 0.0

    def update(self, error):
        p = self.kp *

error

        i = self.ki *

self.status
```



```

        if self.last_error is None:
            d = 0.0
        else:
            d = self.kd *

(error - self.last_error) / self.dt

        self.status += error *

self.dt

        self.last_error = error

        return p + i + d

```

下面的程序使用PID控制器对系统进行控制，让滑块更快地停在位移1.0处，为了后续调用优化工具自动搜索合适的PID参数，这里将整个系统模拟用函数pid_control_system()封装起来，函数的参数为PID控制器的三个参数。

程序的基本构造与前面的无控制的程序相同，只是增加了PID控制器方面的运算：❶计算目标位置1.0与当前位置之间的误差，❷使用该误差更新PID控制器，获得控制器的输出F，❸更新目标系统中的控制力。

由程序的输出可知，由于PID控制器的控制，系统在两秒之内就已经停在了位移1.0处，如图3-29所示。

```

def pid_control_system(kp, ki, kd, dt, target=1.0):
    system = MassSpringDamper(m=m, k=k, b=b, F=0.0)
    pid = PID(kp, ki, kd, dt)
    init_status = 0.0, 0.0

    r = ode(system.f)

```

```

r.set_integrator('vode', method='bdf')
r.set_initial_value(init_status, 0)

t = [0]
result = [init_status]
F_arr = [0]

while r.successful( ) and r.t + dt < 2.0:
    r.integrate(r.t + dt)
    err = target - r.y[0] ❶
    F = pid.update(err) ❷
    system.F = F ❸
    t.append(r.t)
    result.append(r.y)
    F_arr.append(F)

result = np.array(result)
t = np.array(t)
F_arr = np.array(F_arr)
return t, F_arr, result

t, F_arr, result = pid_control_system(50.0, 100.0, 10.0, 0.001)
print u"控制力的终值:", F_arr[-1]
控制力的终值: 19.9434046839

```

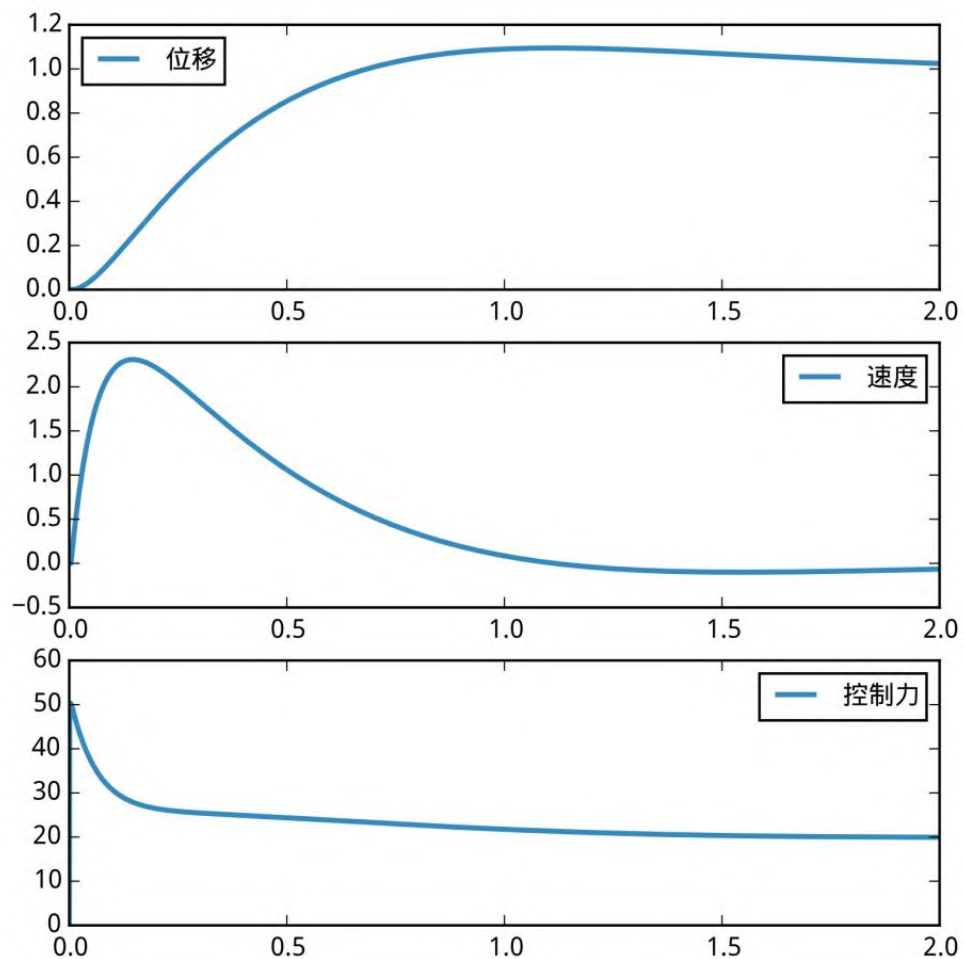


图3-29 使用PID控制器让滑块停在位移1.0处

通过调节PID控制器的三个参数可以获得最佳的控制效果，这里我们使用前面介绍过的optimize库中的函数自动寻找最优的PID参数。为了使用最优化函数，需要编写一个对控制结果进行评价的函数。由于我们的目标是让滑块尽快地停在位移1.0处，因此可以用前两秒钟滑块位移与目标位移差的绝对值之和作为控制结果的评价，该值越小表示控制得越好。为了让最优化运行得快一些，这里将控制器的时间间隔改为0.01秒。

```
%%time

from scipy import optimize

def eval_func(k):
```

```

    kp, ki, kd = k

    t, F_arr, result = pid_control_system(kp, ki, kd, 0.01)

    return np.sum(np.abs(result[:, 0] - 1.0))

kwargs = {"method": "L-BFGS-B",
          "bounds": [(10, 200), (10, 100), (1, 100)],
          "options": {"approx_grad": True}}

opt_k = optimize.basinhopping(eval_func, (10, 10, 10),
                              niter=10,
                              minimizer_kwargs=kwargs)

print opt_k.x
[ 199.81255771  100.                15.20382074]
Wall time: 1min 15s

```

下面使用优化器的输出作为PID的参数对系统进行模拟，可以看到控制开始0.5秒之后滑块已经基本上稳定在了位移1.0处，如图3-30所示。

```

kp, ki, kd = opt_k.x

t, F_arr, result = pid_control_system(kp, ki, kd, 0.01)

idx = np.argmin(np.abs(t - 0.5))

x, u = result[idx]

print "t={}, x={:g}, u={:g}".format(t[idx], x, u)

t=0.5, x=0.979592, u=0.00828481

```

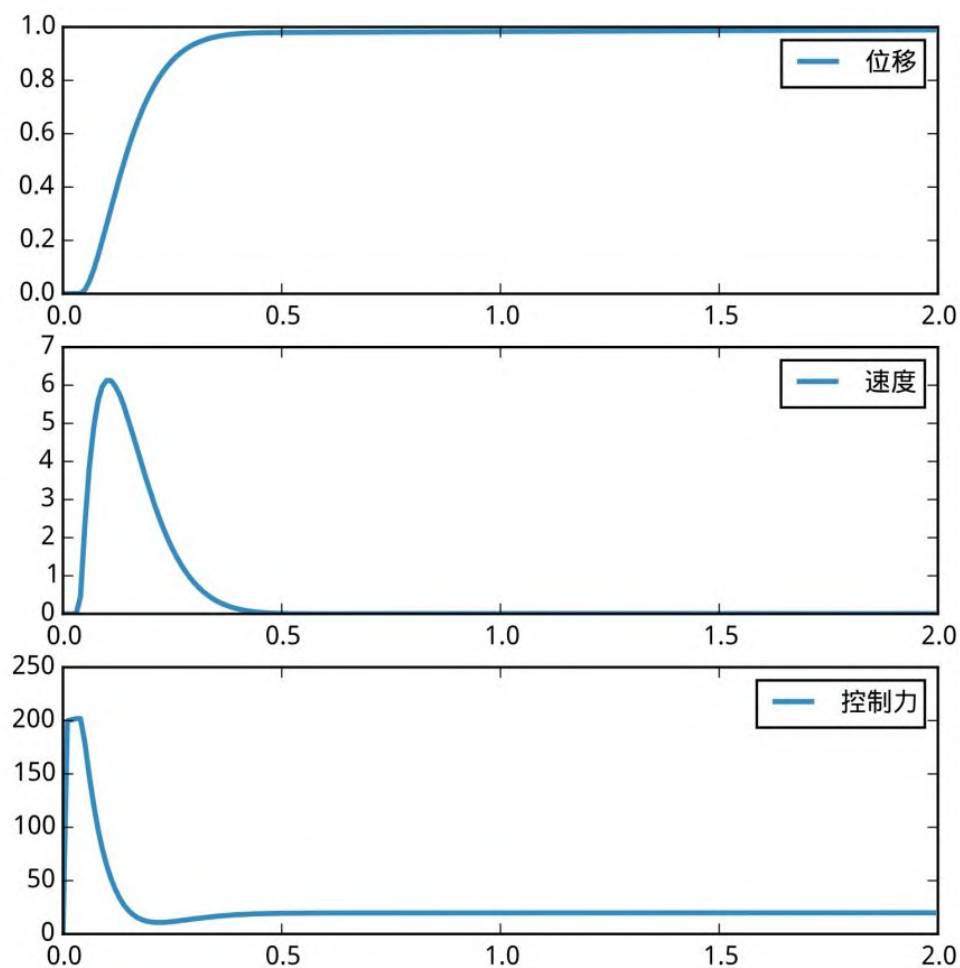


图3-30 优化PID的参数以降低控制响应时间

3.6 信号处理-signal



与本节内容对应的 Notebook 为：`03-scipy/scipy-600-signal.ipynb`。

SciPy的signal模块提供了信号处理方面的许多函数，包括卷积运算、B样条、滤波以及滤波器设计等方面的内容。

3.6.1 中值滤波

中值滤波能够比较有效地消除声音信号中的瞬间噪声或者图像中的斑点噪声。在signal模块中，`medfilt()`对一维信号进行中值滤波，而`medfilt2d()`对二维信号进行中值滤波。在`scipy.ndimage`模块中另有针对多维图像的中值滤波器，这里简单演示`medfilt()`的效果。

```
t = np.arange(0, 20, 0.1)
x = np.sin(t)
x[np.random.randint(0, len(t), 20)] +=
np.random.standard_normal(20)*
0.6 ❶
x2 = signal.medfilt(x, 5) ❷
x3 = signal.order_filter(x, np.ones(5), 2)
print np.all(x2 == x3)
pl.plot(t, x, label=u"带噪声的信号")
pl.plot(t, x2 + 0.5, alpha=0.6, label=u"中值滤波之后的信号")
pl.legend(loc="best")
True
```

❶首先创建一个带有随机的瞬间噪声的正弦波，❷然后调用`medfilt()`进行中值滤波，第二个参数为计算中值的窗口大小，它必须是一个奇数。`medfilt()`将信号中的每个元素都替换为其窗口内的中值。

最后绘制原始信号和滤波信号，为了便于比较，图中将滤波之后的信号统一向上偏移了0.5，结果如图3-31所示。中值滤波是排序滤波的一个特例。使用排序滤波可以将元素替换为其窗口内指定排序顺序的元素。其调用形式如下：

```
order_filter(a, domain, rank)
```

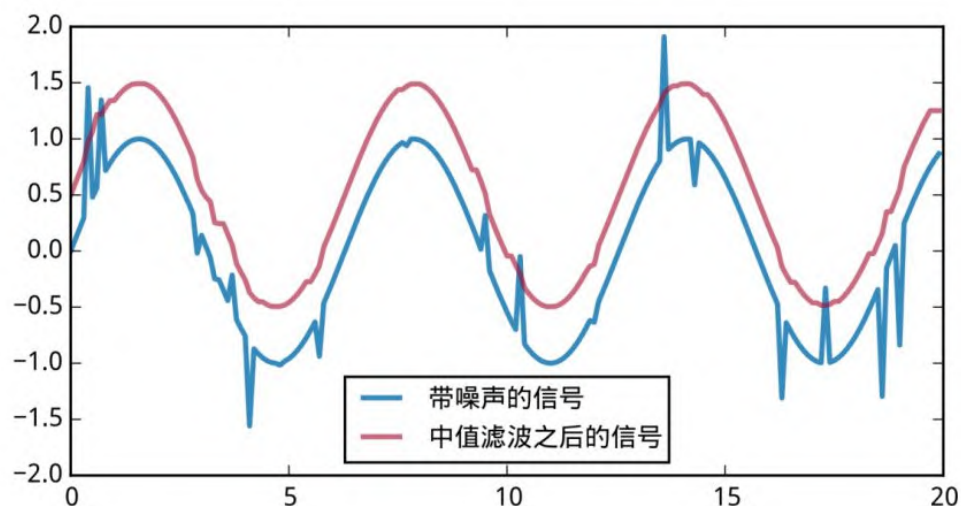


图3-31 使用中值滤波剔除瞬间噪声

其中a是一个多维数组，domain是维数和a相同的数组，它指定窗口的范围，rank是一个非负整数，用来选择窗口中元素排序后的值，0表示选择最小值，1表示选择第二小的值。中值滤波也可以用order_filter()计算，注意domain参数是一个长度为5、值全为1的数组。

3.6.2 滤波器设计

signal模块提供了许多滤波器设计的函数。在下面的实例中，我们设计一个IIR带通滤波器，并查看其频率响应，最后使用它对频率扫描信号进行滤波计算。

```
sampling_rate = 8000.0
```

```
# 设计一个带通滤波器：
```

```
# 通带为0.2*
```

```

4000 - 0.5*

4000

# 阻带为<0.1*

4000, >0.6*

4000

# 通带增益的最大衰减值为2dB
# 阻带的最小衰减值为40dB
b, a = signal.iirdesign([0.2, 0.5], [0.1, 0.6], 2, 40) ❶

# 使用freq计算滤波器的频率响应
w, h = signal.freqz(b, a) ❷

# 计算增益
power = 20*

np.log10(np.clip(np.abs(h), 1e-8, 1e100)) ❸

freq = w / np.pi *

sampling_rate / 2

```

❶首先用*iirdesign*()设计一个IIR带通滤波器。这个滤波器的通带为0.2*f*₀到0.5*f*₀，阻带为小于0.1*f*₀和大于0.6*f*₀，其中*f*₀为信号取样频率的一半。如果取样频率为8kHz，那么这个带通滤波器的通带为800Hz到2kHz。

通带的最大增益衰减为2dB，阻带的最小增益衰减为40dB，即通带的增益浮动在2dB之内，阻带至少有40dB的衰减。

`iirdesign()` 返回两个数组**b**和**a**，它们分别是IIR滤波器的分子和分母部分的系数。其中**a**[0]恒等于1。❷调用`freqz()`计算所得到的滤波器的频率响应。`freqz()`返回两个数组**w**和**h**，其中**w**是圆频率数组，通过 $\omega f_0/\pi$ 可以计算出与其对应的实际频率。**h**是**w**中对应频率点的响应，它是一个复数数组，其幅值表示滤波器的增益特性，相角表示滤波器的相位特性。

❸计算**h**的增益特性，并使用dB进行度量。由于**h**中存在幅值几乎为0的值，因此先用`clip()`对其裁剪之后，再调用对数函数，避免计算出错。

在实际运用中为了测量未知系统的频率特性，经常将频率扫描波输入到系统中，观察系统的输出，从而计算其频率特性。下面让我们模拟这一过程：

```
# 产生两秒钟的取样频率为sampling_rate Hz的频率扫描信号
# 开始频率为0，结束频率为sampling_rate/2
t = np.arange(0, 2, 1/sampling_rate) ❶
sweep = signal.chirp(t, f0=0, t1=2, f1=sampling_rate/2) ❷
# 对频率扫描信号进行滤波
out = signal.lfilter(b, a, sweep) ❸
# 将波形转换为能量
out = 20*

np.log10(np.abs(out)) ❹

# 找到所有局部最大值的下标
index = signal.argrelmax(out, order=3) ❺
# 绘制滤波之后的波形的增益
pl.figure(figsize=(8, 2.5))
```

```

pl.plot(freq, power, label=u"带通IIR滤波器的频率响应")

pl.plot(t[index]/2.0*

4000, out[index], label=u"频率扫描波测量的频谱", alpha=0.6) ⑥

pl.legend(loc="best")

```

❶为了调用chirp()产生频率扫描波形的数据，首先需要产生一个表示取样时间的等差数组，这里产生两秒的取样频率为8kHz的取样时间数组。❷然后调用chirp()得到两秒的频率扫描波形的数据。频率扫描波的开始频率f0为0Hz，结束频率f1为4kHz，到达4kHz的时间为两秒，使用数组t作为取样时间点。❸最后调用lfilter()计算频率扫描波形经过带通滤波器之后的结果。

❹为了和系统的增益特性图进行比较，需要获取输出波形的包络，因此先将输出波形数据转换为能量值。❺为了计算包络，调用argrelmax()找到out数组中所有局部最大值的下标，order参数指定局域最大值的范围，这里的值为3表示所有的局域最大值都是连续7个元素（前后各三个元素）中的最大值。❻最后将时间转换为对应的频率，绘制所有局部最大点的能量值。

图3-32显示了freqz()计算的频谱和频率扫描波得到的频率特性，可以看到结果是一致的。

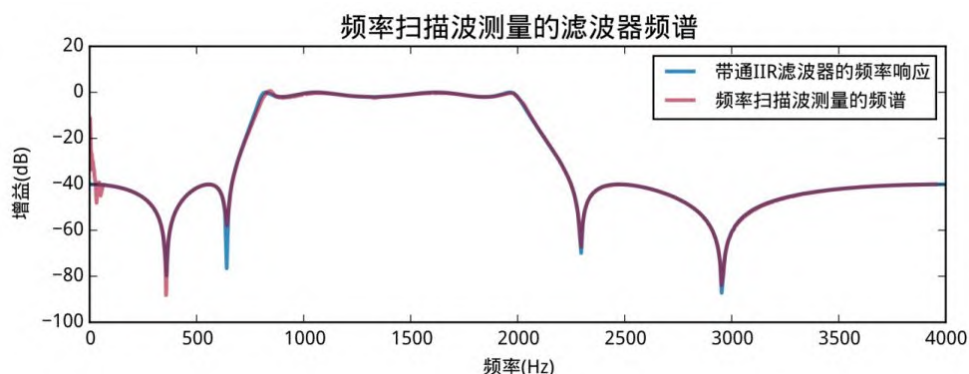


图3-32 用频率扫描波测量的频率响应

3.6.3 连续时间线性系统

在上一节中，我们使用`odeint()`对质量-弹簧-阻尼系统的微分方程组进行了数值积分，并且进行了PID控制模拟。该系统的微分方程为： $m\ddot{x} + b\dot{x} + kx = f$ 。通过拉普拉斯变换可以将微分方程化为容易求解的代数方程： $ms^2 X(s) + bsX(s) + kX(s) = F(s)$ 。其中 $F(s)$ 是 $f(t)$ 的拉普拉斯变换， $X(s)$ 是 $x(t)$ 的拉普拉斯变换，而 n 次微分变成了 s^n 。 $F(s)$ 是输入信号，而 $X(s)$ 是输出信号，将等式改写为输入除以输出的形式，就得到了系统的传递函数 $P(s)$ ：

$$P(s) = \frac{X(s)}{F(s)} = \frac{1}{ms^2 + bs + k}$$

连续时间系统的传递函数是两个 s 的多项式的商。通过连续时间系统的传递函数，很容易计算某输入信号对应的输出信号。在下面的例子中，使用`signal`模块计算质量-弹簧-阻尼系统对阶跃信号以及正弦波信号的响应输出。❶创建`lti`对象，可以使用控制理论中的多种形式表示连续时间线性系统，这里使用的是传递函数分子和分母多项式的系数。多项式的系数与`numpy.polyld`的约定相同，即下标为0的元素是最高次项的系数。❷调用`lti.step()`方法计算系统的阶跃响应。`T`参数为计算响应的时间数组。❸调用`signal.lsim()`计算系统对正弦波信号的响应，它的第一个参数为`lti`对象，也可以直接传递(numerator, denominator)。`U`参数为保存输入信号的数组。`step()`和`lsim()`计算结果中的第二项为系统的输出信号，这里忽略其余的输出。

图3-33显示阶跃响应最终稳定在 $x=0.05$ 处，这时的 $kx=1$ 。

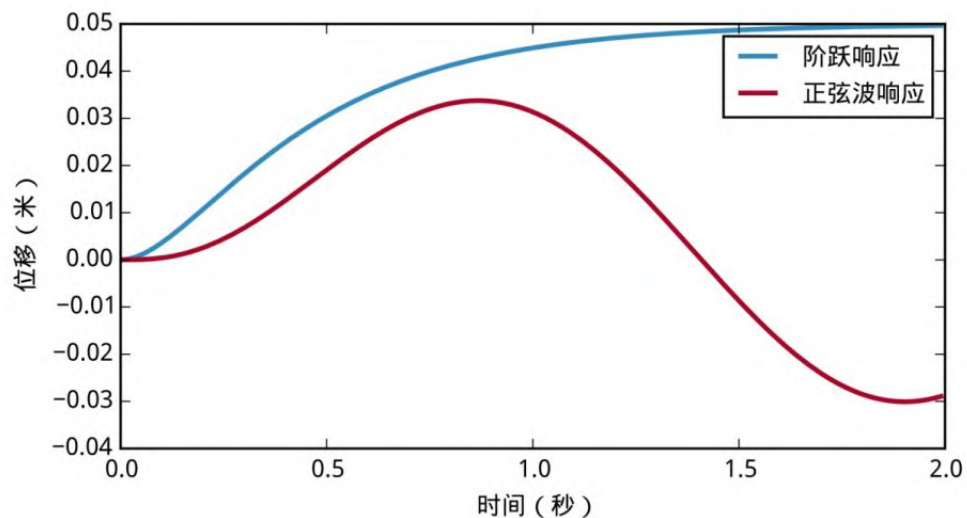


图3-33 系统的阶跃响应和正弦波响应

```
m, b, k = 1.0, 10, 20

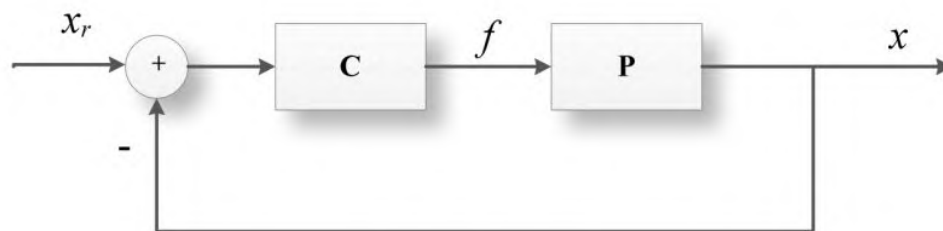
numerator = [1]
denominator = [m, b, k]

plant = signal.lti(numerator, denominator) ❶

t = np.arange(0, 2, 0.01)
_, x_step = plant.step(T=t) ❷

_, x_sin, _ = signal.lsim(plant, U=np.sin(np.pi*
t), T=t) ❸
```

传递函数的代数运算可以表示由多个连续时间系统组成的系统，例如两个系统的级联的传递函数是各个系统的传递函数的乘积。而传递函数由分子和分母两个多项式构成，因此传递函数的四则运算可以使用NumPy的poly1d相关的函数实现。下面的SYS类通过定义__mul__、__add__、__sadd__、__div__等魔法方法，让它支持四则运算。



反馈控制系统框图

图3-34 反馈控制系统框图

❶ `feedback()` 方法计算与之对应的反馈系统的传递函数。在图3-34中，P是被控制的系统，C是控制器，C的输入信号是目标信号与实际输入的差 $x_r - x$ 。我们从 x_r 到 x 的传递函数就是这个反馈系统的传递函数。根据图示可以列出如下拉普拉斯变换之后的代数方程：

$$X(s) = (X_r(s) - X(s)) \cdot C(s) \cdot P(s)$$

整理可得：

$$\frac{X(s)}{X_r(s)} = \frac{C(s) \cdot P(s)}{1 + C(s) \cdot P(s)}$$

如果将 $C(s) \cdot P(s)$ 看作系统 $Y(s)$ ，那么可以得出反馈系统的传递函数为： $\frac{Y(s)}{1+Y(s)}$ 。

❷ 为了让SYS对象能作为`step()`、`lsim()`等函数的第一个表示系统的参数，需要定义`__iter__()`魔法方法返回传递函数的分子与分母的多项式系数。

```
from numbers import Real

def as_sys(s):
```

```

if isinstance(s, Real):
    return SYS([s], [1])

return s

class SYS(object):
    def __init__(self, num, den):
        self.num = num
        self.den = den

    def feedback(self): ❶
        return self / (self + 1)

    def __mul__(self, s):
        s = as_sys(s)
        num = np.polymul(self.num, s.num)
        den = np.polymul(self.den, s.den)
        return SYS(num, den)

    def __add__(self, s):
        s = as_sys(s)
        den = np.polymul(self.den, s.den)
        num = np.polyadd(np.polymul(self.num, s.den),
                          np.polymul(s.num, self.den))
        return SYS(num, den)

    def __sadd__(self, s):
        return self + s

```

```

    def __div__(self, s):
        s = as_sys(s)
        return self *
        SYS(s.den, s.num)

    def __iter__(self): ❷
        return iter((self.num, self.den))

```

下面我们用SYS类计算使用PI控制器控制质量-弹簧-阻尼系统时的阶跃响应。PI控制器的传递函数为：

$$C = \frac{K_p s + K_i}{s}$$

注意上节中介绍的PI控制器是离散时间的，使用累加器近似计算积分器的输出，而本节采用连续时间系统的系统响应模拟控制系统。

❶质量-弹簧-阻尼系统的传递函数为plant，❷PI控制器的传递函数为pi_ctrl，为了step()不抛出LinAlgError异常，这里将PI控制器的传递函数的分母常数项设置为一个非常小的值。❸计算反馈系统的传递函数feedback。由图3-35可以看出 K_i 为0时，系统的输出位移与目标位移之间存在一定的差距， K_p 越大差距越小，但是会出现过冲现象。适当调节 K_p 与 K_i 可以减弱过冲现象，但是仍然会有超过目标位移的时刻。

```

M, b, k = 1.0, 10, 20
plant = SYS([1], [M, b, k]) ❶

pi_settings = [(10, 1e-10), (200, 1e-10),
               (200, 100), (50, 100)]

```

```

fig, ax = pl.subplots(figsize=(8, 3))

for pi_setting in pi_settings:
    pi_ctrl = SYS(pi_setting, [1, 1e-6]) ❷
    feedback = (pi_ctrl *
plant).feedback( ) ❸
    _, x = signal.step(feedback, T=t)
    label = "$K_p={:d}, K_i={:3.0f}$".format(*
pi_setting)
    ax.plot(t, x, label=label)

ax.legend(loc="best", ncol=2)
ax.set_xlabel(u"时间 (秒)")
ax.set_ylabel(u"位移 (米)")

```

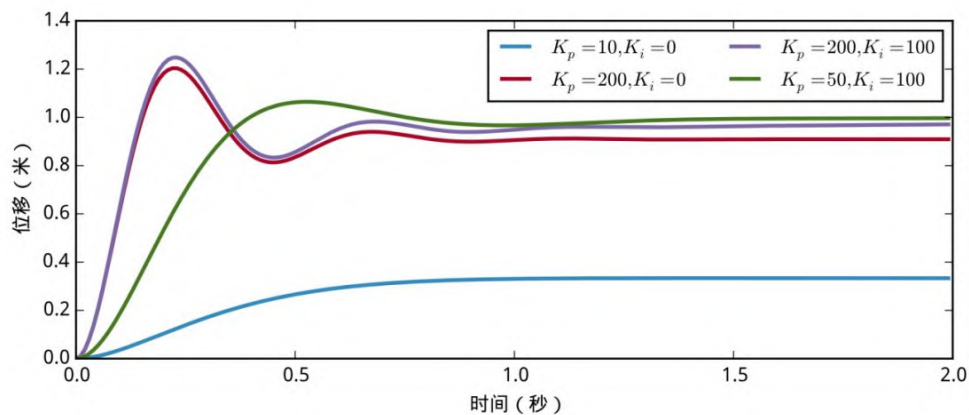


图3-35 使用PI控制器的控制系统的阶跃响应

为了计算施加于质量的控制力，可以将误差信号传递给`lsim()`计算控制器的输出：


```
_ , f, _ = signal.lsim(pi_ctrl, U=1-x, T=t)
```

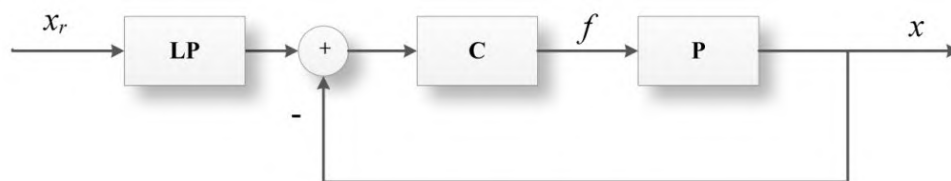
为了彻底消除过冲现象，需要使用PID控制，PID控制器的传递函数为：

$$C = \frac{K_d s^2 + K_p s + K_i}{s}$$

下面计算PID控制器构成的反馈系统的阶跃响应。由于PID控制器需要对输入信号进行微分，而阶跃输入信号会导致PID的输出中包含脉冲输出，即时间无限短、值无限大的信号。

```
kd, kp, ki = 30, 200, 400
pid_ctrl = SYS([kd, kp, ki], [1, 1e-6])
feedback = (pid_ctrl *
plant).feedback( )
_ , x2 = signal.step(feedback, T=t)
```

为了让PID控制器的输出在限定的范围之内，可以在反馈系统之前添加一个低通滤波器，一阶低通滤波器的传递函数为： $\frac{1}{a \cdot s + 1}$ 。添加低通滤波器之后，PID控制器的输入就是连续信号了，如图3-36所示。



带低通滤波器的反馈控制系统框图

图3-36 带低通滤波器的反馈控制系统框图

```
lp = SYS([1], [0.2, 1])
lp_feedback = lp *
```

```
(pid_ctrl *
plant).feedback( )
_, x3 = signal.step(lp_feedback, T=t)
```

由于PID控制器的传递函数的分子阶数高于分母阶数，因此无法使用lsim()计算。我们可以把x_r当作系统的输入，把f当作输出，通过下面的方程计算从x_r到f的传递函数：

$$F(s) = (X_r(s) \cdot LP(s) - F(s) \cdot P(s)) \cdot C(s)$$

得到的传递函数为：

$$\frac{F(s)}{X_r(s)} = \frac{C(s) \cdot LP(s)}{C(s) \cdot P(s) + 1}$$

下面根据上面的公式计算带低通滤波器的控制系统中控制器的输出：

```
pid_out = (pid_ctrl *
lp) / (pid_ctrl *
plant + 1)
_, f3 = signal.step(pid_out, T=t)
```

图3-37显示了上述PI控制、PID控制以及带低通滤波的PID控制等系统中滑块的位移以及控制力。由于PID控制的控制力存在脉冲信号，因此无法在图中正确显示。由位移曲线可以看出低通+PID控制可以有效抑制过冲现象。

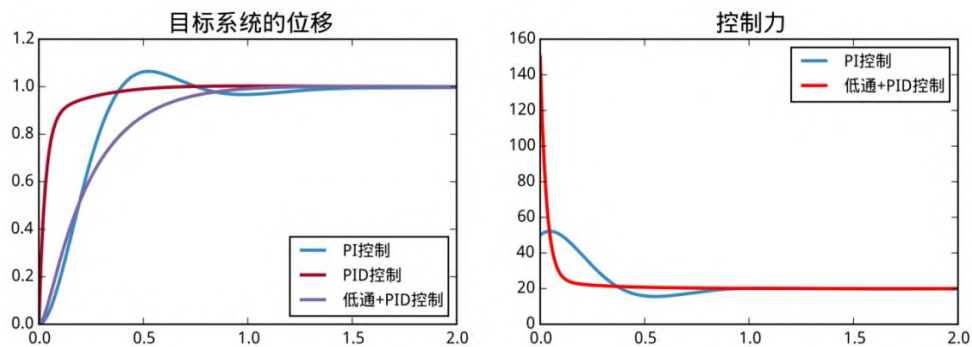


图3-37 滑块的位移以及控制力

3.7 插值-interpolate



与本节内容对应的 Notebook 为：03-scipy/scipy-700-interpolate.ipynb。

插值是通过已知的离散数据求未知数据的方法。与拟合不同的是，要求曲线通过所有的已知数据。SciPy的interpolate模块提供了许多对数据进行插值运算的函数。

3.7.1 一维插值

一维数据的插值运算可以通过`interp1d()`完成。其调用形式如下，它实际上不是函数而是一个类：

```
interp1d(x, y, kind='linear', ...)
```

其中，x和y参数是一系列已知的数据点，kind参数是插值类型，可以是字符串或整数，它给出插值的B样条曲线的阶数，可以有如下候选值：

- 'zero'、'nearest'：阶梯插值，相当于0阶B样条曲线。
- 'slinear'、'linear'：线性插值，用一条直线连接所有的取样点，相当于一阶B样条曲线，'slinear'使用扩展库中的相关函数计算，而'linear'则直接使用Python编写的函数进行运算，其结果一样。
- 'quadratic'、'cubic'：二阶和三阶B样条曲线，更高阶的曲线可以直接使用整数值指定。

interp1d对象可以计算x的取值范围之内任意点的函数值。它可以像函数一样直接调用，和NumPy的ufunc函数一样能对数组中的每个元素进行计算，并返回一个新的数组。

下面的程序演示了kind参数以及与其对应的插值曲线，结果如图3-38所示。程序中我们使用循环对相同的数据进行4种不同阶数的插值运算。❶首先使用数据点创建一个interp1d对象f，通过kind参数指定其阶数。❷调用f()计算出一系列的插值结果。本例中，决定插值曲线的数据点一共有11个，插值之后的曲线数据点有101个。

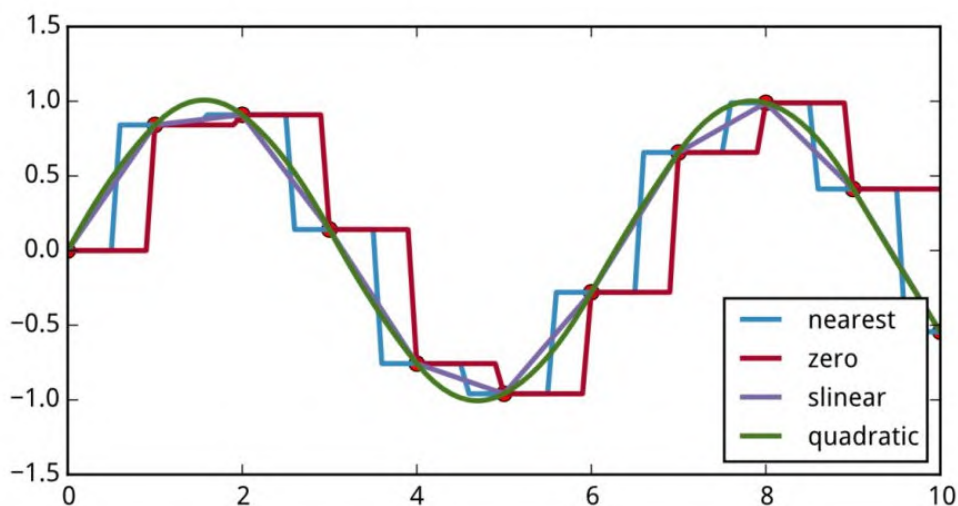


图3-38 interp1d的各阶插值



高次interp1d()插值的运算量很大，因此对于点数较多的数据，建议使用后面介绍的UnivariateSpline()。

```
from scipy import interpolate

x = np.linspace(0, 10, 11)
y = np.sin(x)

xnew = np.linspace(0, 10, 101)
pl.plot(x,y,'ro')
for kind in ['nearest', 'zero', 'slinear', 'quadratic']:
    f = interpolate.interp1d(x,y,kind=kind) ❶
    ynew = f(xnew) ❷
    pl.plot(xnew, ynew, label=str(kind))

pl.legend(loc='lower right')
```

1. 外推和Spline拟合

上节所介绍的interp1d类要求其参数x是一个递增的序列，并且只能在x的取值范围之内进行内插计算，不能用它进行外推运算，即计算x的取值范围之外的数据点。UnivariateSpline类的插值运算比interp1d更高级，它支持外推和拟合运算，其调用形式如下：

```
UnivariateSpline(x, y, w=None, bbox=[None, None], k=3, s=None)
```

- x、y是保存数据点的X-Y坐标的数组，其中x必须是递增序列。
- w是为每个数据点指定的权重值。
- k为样条曲线的阶数。
- s是平滑系数，它使得最终生成的样条曲线满足条件： $\sum_{i=1}^n (w_i \cdot (y_i - \text{spline}(x_i)))^2 \leq s$ 。即当s>0时，样条曲线并不一定通过各个数据点。为了让曲线通过所有数据点，必须将s参数设置为0。此外，还可以使用InterpolatedUnivariateSpline类，它与UnivariateSpline的唯一区别就是它通过所有的数据点，相当于将s设置为0。

下面的程序演示了使用UnivariateSpline对数据进行插值、外推以及样条曲线拟合：

❶如图3-39（上）所示，UnivariateSpline能够进行外推运算，虽然输入数据中没有X轴大于10的点，但是它能计算出X轴在0到12的插值结果。在X轴大于10的部分，样条曲线仍然呈现出和正弦波类似的形状，越远离输入数据范围，误差会越大，因此外推的范围是有限的。由于s参数为0，因此插值曲线经过所有的数据点。

❷图3-39（下）则显示了s参数不为零时的结果，对于带噪声的输入数据，选择合适的s参数能够使得样条曲线接近无噪声时的波形，可以把它看作使用样条曲线对数据进行拟合运算。

```
x1 = np.linspace(0, 10, 20)
y1 = np.sin(x1)
sx1 = np.linspace(0, 12, 100)
sy1 = interpolate.UnivariateSpline(x1, y1, s=0)(sx1) ❶
```

```
x2 = np.linspace(0, 20, 200)

y2 = np.sin(x2) + np.random.standard_normal(len(x2)) *
```

0.2

```
sx2 = np.linspace(0, 20, 2000)

spline2 = interpolate.UnivariateSpline(x2, y2, s=8) ②
sy2 = spline2(sx2)

pl.figure(figsize=(8, 5))
pl.subplot(211)
pl.plot(x1, y1, ".", label=u"数据点")
pl.plot(sx1, sy1, label=u"spline曲线")
pl.legend( )

pl.subplot(212)
pl.plot(x2, y2, ".", label=u"数据点")
pl.plot(sx2, sy2, linewidth=2, label=u"spline曲线")
pl.plot(x2, np.sin(x2), label=u"无噪声曲线")
pl.legend( )
```

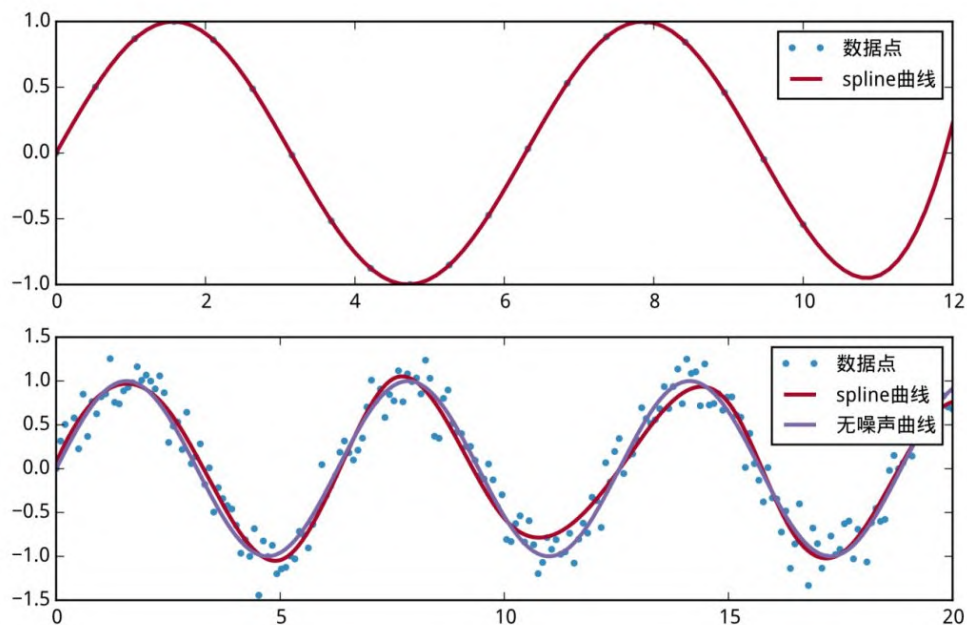


图3-39 使用UnivariateSpline进行插值：外推（上）和数据拟合（下）

当曲线为三阶曲线时，`UnivariateSpline.roots()`可以用于计算曲线与 $y = 0$ 的交点横坐标。下面显示了图3-39（下）中的曲线与X轴的6个交点的横坐标：

```
print np.array_str( spline2.roots( ), precision=3 )
[ 3.288  6.329  9.296 12.578 15.75 18.805]
```

如果需要计算曲线与任意横线的交点，可以事先将曲线的拟合数据在Y轴方向上进行平移。但若要计算与多条 $y=c$ 横线的交点，则需要对同样的数据进行平移和拟合多次。

❶ 下面的`root_at()`通过直接修改拟合曲线的参数，实现曲线的平移，从而可以计算与任意横线的交点。❷ 将`roots_at()`动态添加为`UnivariateSpline`类的方法。❸ 对多条横线求交点，并进行绘图，其结果如图3-40所示。


```

def roots_at(self, v): ❶
    coeff = self.get_coeffs( )
    coeff -= v
    try:
        root = self.roots( )
        return root
    finally:
        coeff += v

interpolate.UnivariateSpline.roots_at = roots_at ❷

pl.plot(sx2, sy2, linewidth=2, label=u"spline曲线")

ax = pl.gca( )
for level in [0.5, 0.75, -0.5, -0.75]:
    ax.axhline(level, ls=":", color="k")
    xr = spline2.roots_at(level) ❸
    pl.plot(xr, spline2(xr), "ro")

```

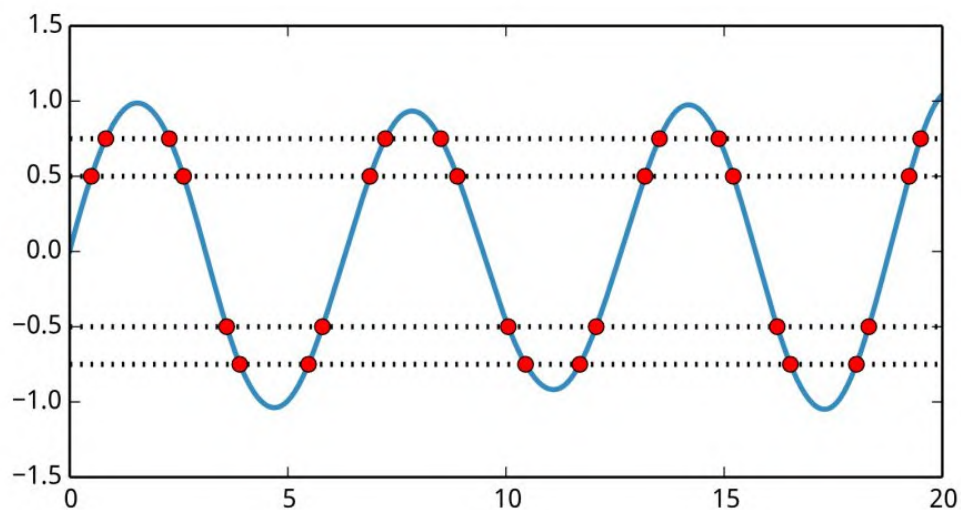


图3-40 计算Spline与水平线的交点

2. 参数插值

前面介绍的插值函数都需要X轴的数据是按照递增顺序排列的，就像一般的 $y=f(x)$ 函数曲线一样。数学上还有一种参数曲线，它使用参数 t 和两个函数 $x=f(t)$ 、 $y=g(t)$ 来定义二维平面上的一条曲线，例如圆形、心形等曲线都是参数曲线。参数曲线的插值可以通过`splprep()`和`splev()`实现，这组函数支持高维空间的曲线的插值，这里以二维曲线为例介绍其用法。

❶首先调用`splprep()`，其第一个参数为一组一维数组，每个数组是各点在对应轴上的坐标。`s`参数为平滑系数，与`UnivariateSpline`的含义相同。`splprep()`返回两个对象，其中`tck`是一个元组，它包含了插值曲线的所有信息。`t`是自动计算出的参数曲线的参数数组。

❷调用`splev()`进行插值运算，其第一个参数为一个新的参数数组，这里将`t`的取值范围等分200份，第二个参数为`splprep()`返回的第一个对象。实际上，参数数组`t`是正规化之后的各个线段长度的累计，因此`t`的范围为0到1。

其结果如图3-41所示，图中比较了平滑系数为0和 $1e-4$ 时的插值曲线。当平滑系数为0时，插值曲线通过所有的数据点。

```
x = [ 4.913, 4.913, 4.918, 4.938, 4.955, 4.949, 4.911,
      4.848, 4.864, 4.893, 4.935, 4.981, 5.01 , 5.021]

y = [ 5.2785, 5.2875, 5.291 , 5.289 , 5.28 , 5.26 ,
      5.245 ,
      5.245 , 5.2615, 5.278 , 5.2775, 5.261 , 5.245 ,
      5.241]
```

```

pl.plot(x, y, "o")

for s in (0, 1e-4):
    tck, t = interpolate.splprep([x, y], s=s) ❶
    xi, yi = interpolate.splev(np.linspace(t[0], t[-1], 200),
tck) ❷

    pl.plot(xi, yi, lw=2, label=u"s=%g" % s)

pl.legend( )

```

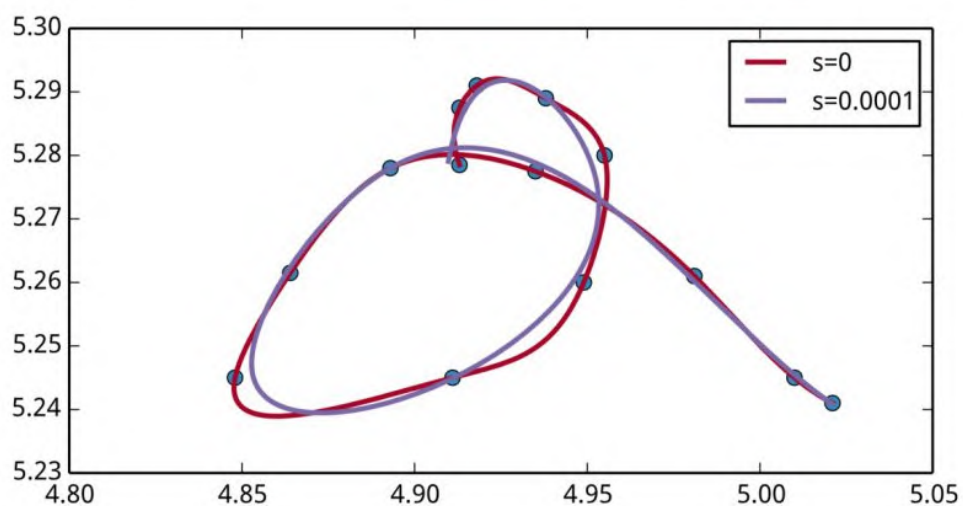


图3-41 使用参数插值连接二维平面上的点

3. 单调插值

前面介绍的几种插值方法不能保证数据点的单调性，即曲线的最值可能出现在数据点之外的地方。PchipInterpolator类（别名pchip）使用单调三次插值，能够保证曲线的所有最值都出现在数据点之上。下面的程序用pchip()对数据点进行插值，并绘制其一阶导数曲线，由图3-42的导数曲线可知，所有最值点处的导数都为0。

```

x = [0, 1, 2, 3, 4, 5]
y = [1, 2, 1.5, 2.5, 3, 2.5]
xs = np.linspace(x[0], x[-1], 100)
curve = interpolate.pchip(x, y)
ys = curve(xs)
dys = curve.derivative(xs)
pl.plot(xs, ys, label=u"pchip")
pl.plot(xs, dys, label=u"一阶导数")
pl.plot(x, y, "o")
pl.legend(loc="best")
pl.grid( )
pl.margins(0.1, 0.1)

```

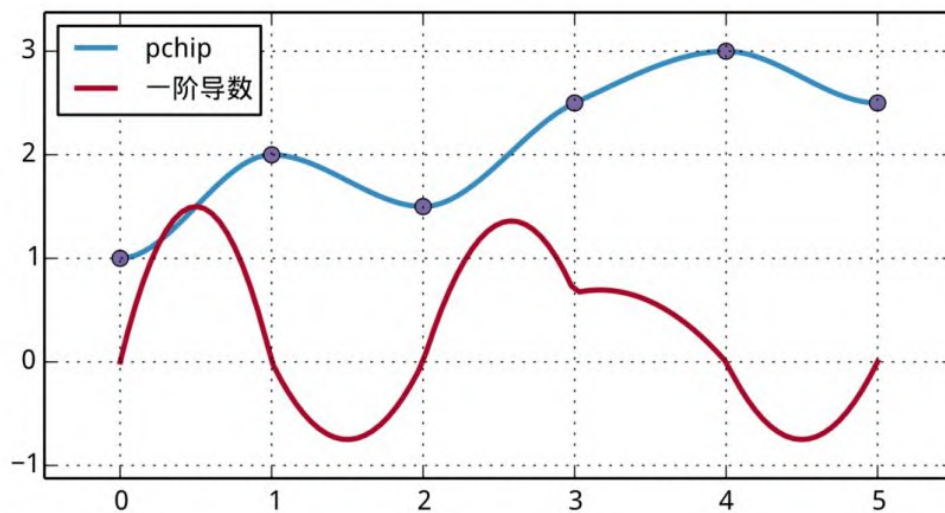


图3-42 单调插值能保证两个点之间的曲线为单调递增或递减

3.7.2 多维插值

使用`interp2d()`可以进行二维插值运算。它的调用形式如下：

```
interp2d(x, y, z, kind='linear', ...)
```

其中x、y、z都是一维数组，如果传入的是多维数组，则先将其转为一维数组。kind参数指定插值运算的阶数，可以为'linear'、'cubic'、'quintic'。

下面的例子对某个函数表面上的网格点进行二维插值，效果如图3-43所示。其中左图显示插值之前的数据，而右图显示插值运算后得到的结果。

```
def func(x, y): ❶
    return (x+y)*

np.exp(-5.0*

(x**

2 + y**

2))

# x-y轴分为15*

15的网格

y, x = np.mgrid[-1:1:15j, -1:1:15j] ❷
fvals = func(x, y) # 计算每个网格点上的函数值

# 二维插值
newfunc = interpolate.interp2d(x, y, fvals, kind='cubic') ❸
```

```
# 计算100*  
  
100的网格上的插值  
  
xnew = np.linspace(-1,1,100)  
ynew = np.linspace(-1,1,100)  
fnew = newfunc(xnew, ynew) ❷
```

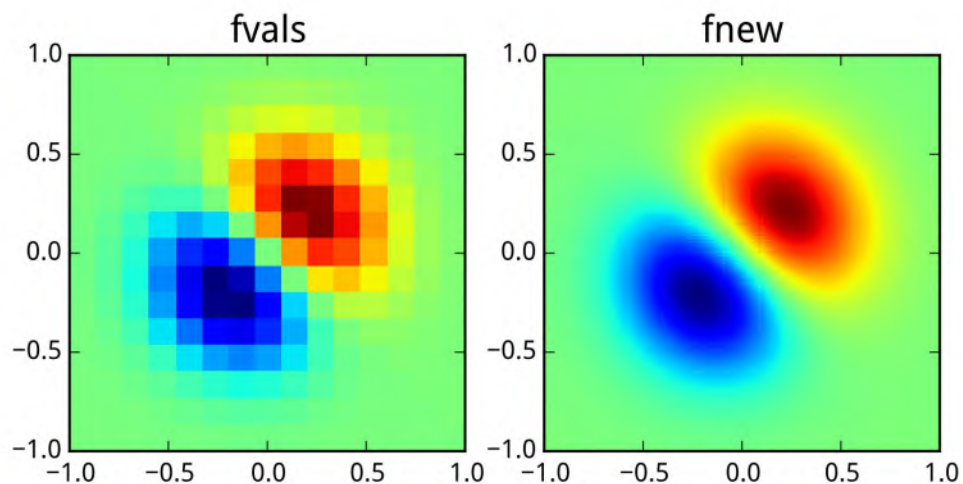


图3-43 使用interp2d类进行二维插值

❶func是计算曲面上各点高度的函数。❷计算X、Y轴在-1到1范围之内，大小为15×15的等间距网格上各点的高度。注意所得到的二维数组fvals的第0轴与Y轴对应，第一轴与X轴对应。❸使用网格上各点的X、Y和Z轴的坐标创建interp2d对象，这里我们使用二阶插值曲面。❹interp2d对象可以像函数一样调用，我们用它计算插值曲面在一个更密网格中的高度值。注意这里的参数是两个一维数组，分别指定网格的X-Y轴坐标，而不需要通过mgrid创建网格坐标数组。

1. griddata

interp2d类只能对网格形状的取样值进行插值运算，如果需要对随机散列的取样点进行插值，则可以使用griddata()。其调用形式如下：

```
griddata(points, values, xi, method='linear', fill_value=nan)
```

其中points表示K维空间中的坐标，它可以是形状为(N, k)的数组，也可以是一个有k个数组的序列，N为数据的点数。values是points中每个点对应的值。xi是需要进行插值运算的坐标，其形状为(M, k)。method有三个选项——'nearest'、'linear'、'cubic'，分别对应0阶、1阶以及3阶插值。

下面是griddata()的演示程序，其输出如图3-44所示。左图与'nearest'算法对应，平面上每个点都被填充为与它最近的采样点的数据，因此图中由许多相同颜色的色块构成。'linear'和'cubic'算法只对采样点构成的凸包区域进行插值，区域之外采用fill_value进行填充。中图和右图中的白色区域为插值的凸包区域之外。



griddata()使用欧几里得距离计算插值。如果K维空间中每个维度的取值范围相差较大，则应先将数据正规化，然后使用griddata()进行插值运算。

```
# 计算随机N个点的坐标，以及这些点对应的函数值
N = 200
np.random.seed(42)
x = np.random.uniform(-1, 1, N)
y = np.random.uniform(-1, 1, N)
z = func(x, y)
```

```

yg, xg = np.mgrid[-1:1:100j, -1:1:100j]
xi = np.c_[xg.ravel( ), yg.ravel( )]

methods = 'nearest', 'linear', 'cubic'

zgs = [interpolate.griddata((x, y), z, xi,
method=method).reshape(100, 100)
for method in methods]

```

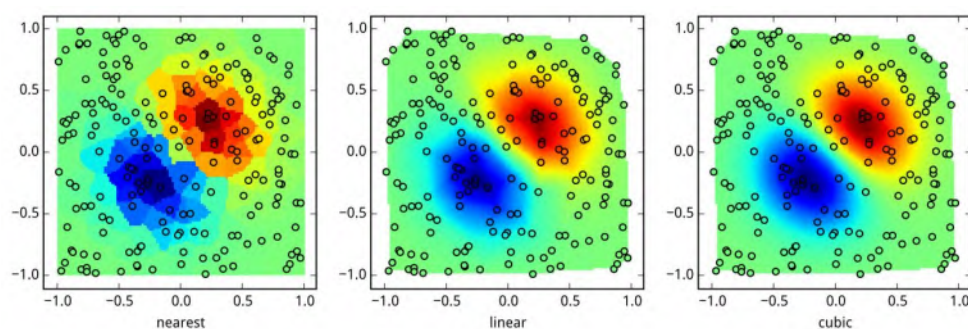


图3-44 使用griddata进行二维插值

2. 径向基函数插值

径向基函数(radial basis function)插值算法也可以用于高维随机散布点的插值。所谓径向基函数，是指函数值只与某特定点的距离相关的一类函数 $\phi(\|\mathbf{x} - \mathbf{x}_i\|)$ ，其中 $\mathbf{x}_i$ 是某个给定取样点的坐标。使用这些 ϕ 函数，可以近似表示N维空间中的函数：

$$y(\mathbf{x}) = \sum_{i=1}^N w_i \phi(\|\mathbf{x} - \mathbf{x}_i\|)$$

为了方便读者理解RBF，下面先看一个一维插值的例子，结果如图3-45所示。图中显示了三种 ϕ 函数对应的插值曲线：multiquadric、gaussian和linear。

```
from scipy.interpolate import Rbf

x1 = np.array([-1, 0, 2.0, 1.0])
y1 = np.array([1.0, 0.3, -0.5, 0.8])

funcs = ['multiquadric', 'gaussian', 'linear']
nx = np.linspace(-3, 4, 100)
rbfs = [Rbf(x1, y1, function=fname) for fname in funcs] ❶
rbf_ys = [rbf(nx) for rbf in rbfs] ❷
```

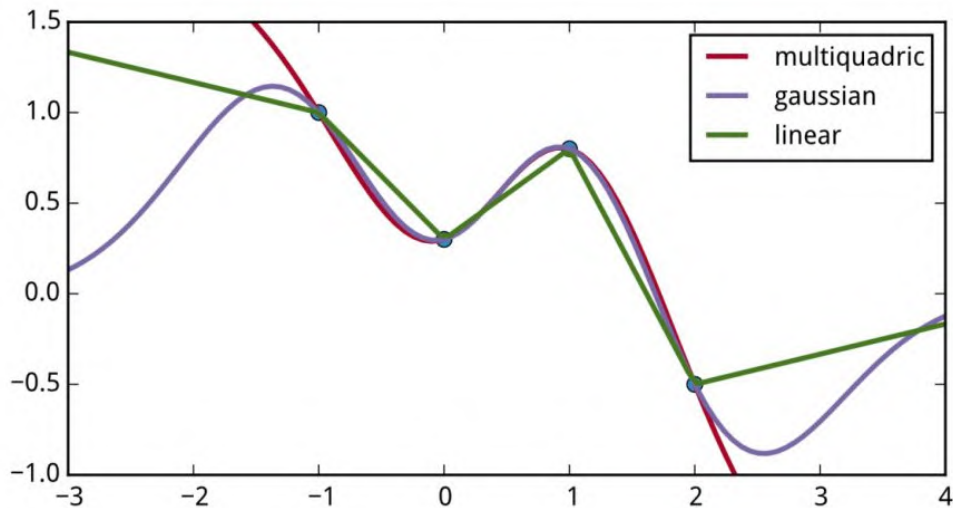


图3-45 一维RBF插值

❶使用表示取样点的x和y数组创建一个rbf对象，并通过function参数指定所使用的径向基函数。❷rbf对象可以像函数一样被调用，我们用它计算以nx为横坐标对应的插值曲线的值。

rbf对象的nodes属性保存wi系数：

```

for fname, rbf in zip(funcs, rbfs):
    print fname, rbf.nodes

    multiquadric [ -3.79570791      9.82703701      5.08190777
-11.13103777]

    gaussian [ 1.78016841 -1.83986382 -1.69565607  2.5266374 ]

    linear [-0.26666667  0.6      0.73333333 -0.9      ]

```

下面的程序演示二维径向基函数插值，效果如图3-46所示。

```

rbfs = [Rbf(x, y, z, function=fname) for fname in funcs]
rbf_zg = [rbf(xg, yg).reshape(xg.shape) for rbf in rbfs]

```

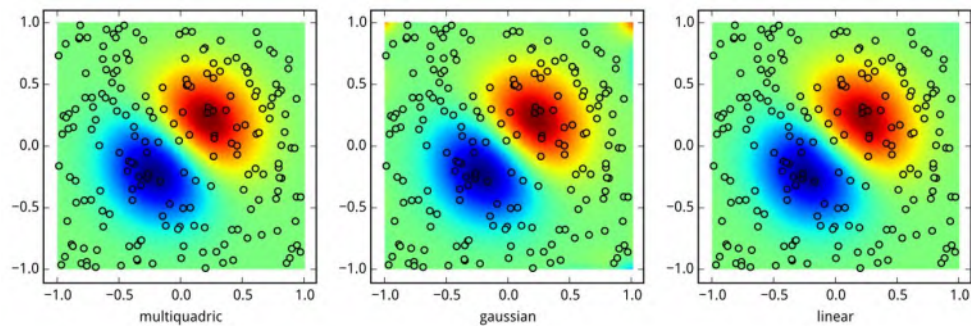


图3-46 二维径向基函数插值

某些径向基函数可以通过epsilon参数指定其作用范围，该值越大每个插值点的作用范围越广，所得到的曲面也就越平滑。下面的代码演示gaussian径向基函数的epsilon参数与插值结果的关系，效果如图3-47所示。

```

epsilons = 0.1, 0.15, 0.3

rbfs = [Rbf(x, y, z, function="gaussian", epsilon=eps) for eps
in epsilons]

zgs = [rbf(xg, yg).reshape(xg.shape) for rbf in rbfs]

```

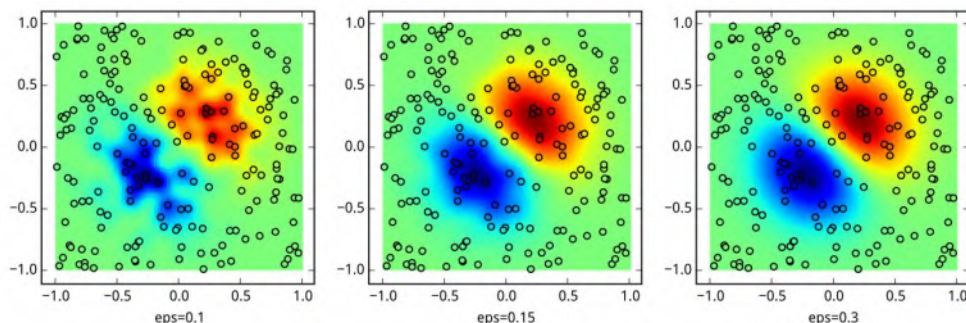


图3-47 epsilon参数指定径向基函数中数据点的作用范围

3.8 稀疏矩阵-sparse



与本节内容对应的 Notebook 为：[03-scipy/scipy-810-sparse.ipynb](#)。

在科学与工程领域求解线性模型时经常出现许多大型的矩阵，这些矩阵中大部分的元素都为0，被称为稀疏矩阵。用NumPy的ndarray数组保存这样的矩阵会很浪费内存，由于矩阵的稀疏特性，可以通过只保存非零元素的相关信息，从而节约内存的使用。此外，针对这种特殊结构的矩阵设计运算函数，也可以提高矩阵的运算速度。

scipy.sparse 中提供了多种表示稀疏矩阵的格式，scipy.sparse.linalg提供了对这些矩阵进行线性代数运算的函数，scipy.sparse.csgraph提供对用稀疏矩阵表示的图进行搜索的函数。本节首先介绍表示稀疏矩阵的各种格式，然后介绍如何使用csgraph中的函数搜索最佳路径，而在本书最后一章中会介绍使用稀疏矩阵的线性代数运算函数解决实际问题的例子。

3.8.1 稀疏矩阵的存储形式

`scipy.sparse`中提供了多种表示稀疏矩阵的格式，每种格式都有不同的用处，其中`dok_matrix`和`lil_matrix`适合逐渐添加元素。

`dok_matrix`从`dict`继承而来，它采用字典保存矩阵中的非零元素：字典的键是一个保存元素（行，列）信息的元组，其对应的值为矩阵中位于（行，列）中的元素值。显然字典格式的稀疏矩阵很适合单个元素的添加、删除和存取操作。通常用来逐个添加非零元素，然后转换成其他支持快速运算的格式。

```
from scipy import sparse
a = sparse.dok_matrix((10, 5))
a[2:5, 3] = 1.0, 2.0, 3.0
print a.keys( )
print a.values( )
[(2, 3), (3, 3), (4, 3)]
[1.0, 2.0, 3.0]
```

`lil_matrix`使用两个列表保存非零元素。`data`保存每行中的非零元素，`rows`保存非零元素所在的列。这种格式也很适合逐个添加元素，并且能快速获取行相关的数据。

```
b = sparse.lil_matrix((10, 5))
b[2, 3] = 1.0
b[3, 4] = 2.0
b[3, 2] = 3.0
print b.data
print b.rows
```

```
[[ ] [ ] [1.0] [3.0, 2.0] [ ] [ ] [ ] [ ] [ ] [ ]]  
[[ ] [ ] [3] [2, 4] [ ] [ ] [ ] [ ] [ ] [ ]]
```

`coo_matrix`采用三个数组`row`、`col`和`data`保存非零元素的信息。这三个数组的长度相同，`row`保存元素的行，`col`保存元素的列，`data`保存元素的值。`coo_matrix`不支持元素的存取和增删，一旦创建后，除了将之转换成其他格式的矩阵，几乎无法对其做任何操作和矩阵运算。

`coo_matrix`支持重复元素，即同一行列坐标可以出现多次，当转换为其他格式的矩阵时，将对同一行列坐标对应的多个值进行求和。在下面的例子中，(2, 3)对应两个值：1和10。在将其转换为`ndarray`数组时把这两个值加在一起，所以最终矩阵中(2, 3)坐标上的值为11。

许多稀疏矩阵的数据都是采用这种格式保存在文件中的，例如某个CSV文件中可能有这样三列：“用户ID，商品ID，评价值”。采用`numpy.loadtxt`或`pandas.read_csv`将数据读入之后，可以通过`coo_matrix`快速将其转换成稀疏矩阵：矩阵的每行对应一位用户，每列对应一件商品，而元素值为用户对商品的评价。

```
row = [2, 3, 3, 2]  
col = [3, 4, 2, 3]  
data = [1, 2, 3, 10]  
  
c = sparse.coo_matrix((data, (row, col)), shape=(5, 6))  
print c.col, c.row, c.data  
print c.toarray( )  
  
[3 4 2 3] [2 3 3 2] [ 1  2  3 10]  
  
[[ 0  0  0  0  0  0]  
 [ 0  0  0  0  0  0]  
 [ 0  0  0 11  0  0]
```

```
[ 0  0  3  0  2  0]
[ 0  0  0  0  0  0]]
```

3.8.2 最短路径

稀疏矩阵 w 可以用于表示图， $w[i, j]$ 保存图中节点 i 和节点 j 之间路径的权值。若节点 i 与 j 之间没有直接路径，则稀疏矩阵不包含该下标，因此使用稀疏矩阵可以表示权值为0的路径。

我们对图3-48中A、B、C、D这4个节点分别编号为0、1、2、3，于是可以构造如下稀疏矩阵。注意当将稀疏矩阵转换为数组显示时，矩阵中未设置的值默认为0，这并不表示图中有权值为0的边。当用稀疏矩阵表示无向图时，只需要设置 $w[i, j]$ 或 $w[j, i]$ 中的一个即可。

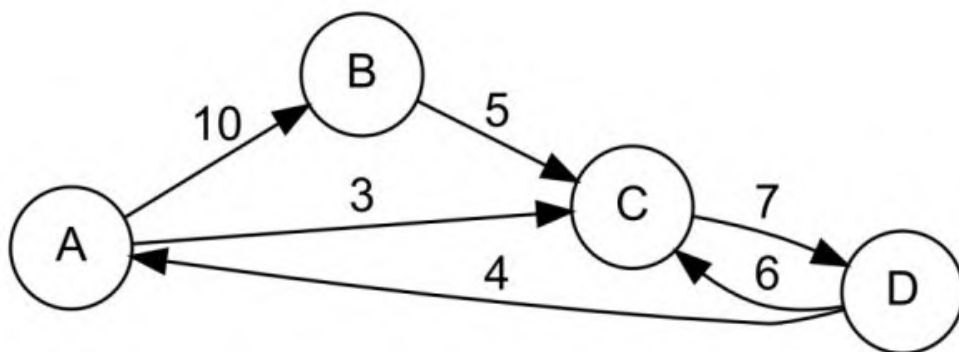


图3-48 使用稀疏矩阵可以表示有向图

```
w = sparse.dok_matrix((4, 4))

edges = [(0, 1, 10), (1, 2, 5), (0, 2, 3),
          (2, 3, 7), (3, 0, 4), (3, 2, 6)]

for i, j, v in edges:
```

```

w[i, j] = v

w.todense( )

matrix([[ 0., 10., 3., 0.],
        [ 0., 0., 5., 0.],
        [ 0., 0., 0., 7.],
        [ 4., 0., 6., 0.]])

```

使用`scipy.sparse.scgraph`模块可以在图中寻找最短路径，下面通过一个例子说明最短路径函数的用法。

图3-49（左）是一幅迷宫图像，其中的黑色曲线是用`scgraph`模块求得的从坐标 (sx, sy) 到 (ex, ey) 的最短路径。❶为了方便计算，下面先将彩色迷宫图像通过阈值转换为黑白二值图像，黑色表示墙壁，白色表示通路。❷为了避免将迷宫外部的余白当作通路，下面的程序在中部两侧添加了两条黑色线段，将余白分隔为上下两个部分。经过上述处理之后的迷宫如图3-49（右）所示。

```

img = pl.imread("maze.png")
sx, sy = (400, 979)
ex, ey = (398, 25)
bimg = np.all(img > 0.81, axis=2) ❶
H, W = bimg.shape

x0, x1 = np.where(bimg[H//2, :]==0)[0][[0, -1]] ❷
bimg[H//2, :x0] = 0
bimg[H//2, x1:] = 0

```

我们将迷宫中所有的像素都当作图中的节点，节点序号与像素坐标 (x, y) 的关系使用 $idx = y * W + x$ 计算。❶找到所有上下相邻、左右相邻的白

色像素对，将其对应的节点序号保存在形状为(N, 2)的edges数组中，N为图所包含的边数。❷通过coo_matrix()创建稀疏矩阵，所有边的权值均为1。

```
#上下相邻的白色像素
mask = (bimg[1:, :] & bimg[:-1, :])
idx = np.where(mask.ravel())[0]
vedge = np.c_[idx, idx + W]
pl.imsave("tmp.png", mask, cmap="gray")

#左右相邻的白色像素
mask = (bimg[:, 1:] & bimg[:, :-1])
y, x = np.where(mask)
idx = y *
W + x
hedge = np.c_[idx, idx + 1]

edges = np.vstack([vedge, hedge]) ❶

values = np.ones(edges.shape[0])
w = sparse.coo_matrix((values, (edges[:, 0], edges[:, 1])), ❷
                      shape=(bimg.size, bimg.size))
```

接下来导入csgraph模块，并调用dijkstra()计算从编号为startid的节点出发到达所有其他节点的最短路径。directed参数为False表示无向图，为了计算最短路径，需要设置return_predecessors参数为True。所返回的数组d和p的形状为(indices参数的长度，图的总节点数)。


```

from scipy.sparse import csgraph

startid = sy *

W + sx

        endid = ey *

W + ex

        d, p = csgraph.dijkstra(w, indices=[startid],
return_predecessors=True, directed=False)

        d.shape      p.shape
        -----
        (1, 801600)  (1, 801600)

```

$d[i, j]$ 保存从编号为 $indices[i]$ 的节点到编号为 j 的节点的距离。如果两个节点之间无路径联通，值为 inf 。下面计算从起点无法到达的节点数，这些节点包括迷宫中黑色像素表示的墙壁以及被黑色像素完全包围的区域。

```

np.isinf(d[0]).sum( )
322324

```

$p[i, j]$ 保存节点 $indices[i]$ 到节点 j 的路径中最后一个节点的编号。下面的代码从编号为 $endid$ 的节点开始回溯，直到找到 $startid$ 节点为止。将访问过的节点保存到 $path$ 中，将 $path$ 反转即可得到从起点到终点的路径。

```

path = [ ]
node_id = endid
while True:
    path.append(node_id)

```

```
if node_id == startid or node_id < 0:
    break
node_id = p[0, node_id]
path = np.array(path)
```

最后，在原来的迷宫图像中将path经过的像素涂黑，得到图3-49（左）中的路径。

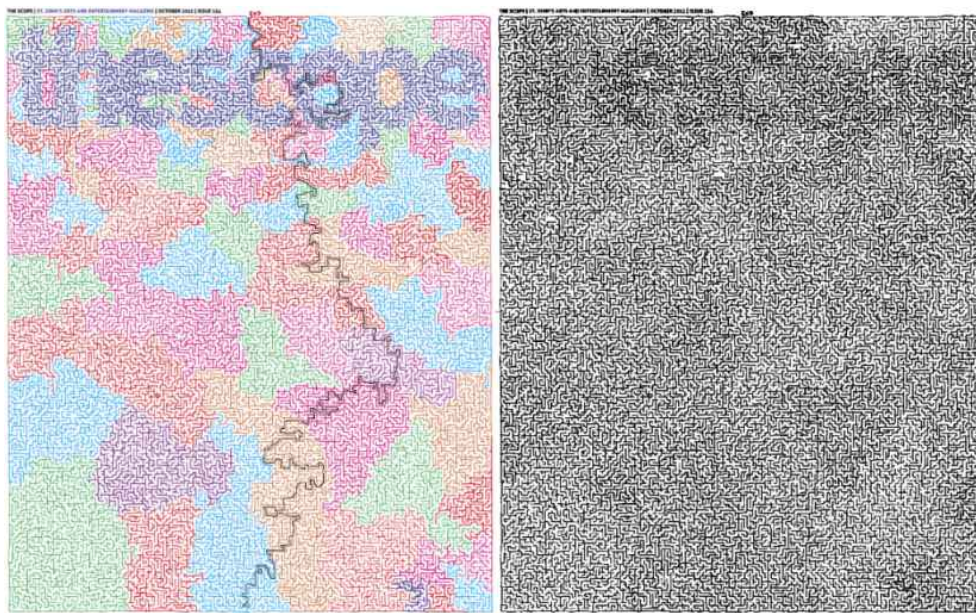


图3-49 用dijkstra计算最短路径



scipy2.scipy.hrd_solver使用csgraph计算华容道游戏“横刀立马”布局步数最少的解法。

在上面的迷宫中，两个相邻白色像素之间的路径权值均为1，因此搜索到的最佳路径为最短路径。而许多游戏地图中的路径搜索会考虑地形因素，这时可以根据不同的地形设置不同大小的路径权值，这样最佳路径就是使所有权值之和最小的路径。

3.9 图像处理-ndimage



与本节内容对应的 Notebook 为：`03-scipy/scipy-900-ndimage.ipynb`。

`scipy.ndimage`是一个处理多维图像的函数库，其中又包括以下几个模块：

- `filters`：图像滤波器。
- `fourier`：傅立叶变换。
- `interpolation`：图像的插值、旋转以及仿射变换等。
- `measurements`：图像相关信息的测量。
- `morphology`：形态学图像处理。

更强大的图像处理库

`scipy.ndimage`只提供了一些基础的图像处理功能，下面是一些更强大的图像处理库：

- `OpenCV`：它是使用C/C++开发的计算机视觉库，本书将用一整章的篇幅介绍OpenCV提供的Python调用接口的用法。
- `SimpleCV`：对多个计算机视觉库进行包装，提供了一套更方便、统一的Python调用接口。
- `scikit-image`：使用Python开发的图像处理库，高速运算部分多采用Cython编写。
- `Mahotas`：采用Python和C++开发的图像处理库。

3.9.1 形态学图像处理

本节介绍如何使用morphology模块实现二值图像处理。二值图像中每个像素的颜色只有两种：黑色和白色。在NumPy中可以用二维布尔数组表示：False表示黑色，True表示白色。也可以用无符号单字节整型(uint8)数组表示：0表示黑色，非0表示白色。

下面的两个函数用于显示形态学图像处理的结果：

```
import numpy as np

def expand_image(img, value, out=None, size = 10):
    if out is None:
        w, h = img.shape
        out = np.zeros((w*
size, h*
size), dtype=np.uint8)

    tmp = np.repeat(np.repeat(img, size, 0), size, 1)
    out[:, :] = np.where(tmp, value, out)
    out[:, :size, :] = 0
    out[:, :, :size] = 0
    return out
```

```

def show_image(*
imgs):
    for idx, img in enumerate(imgs, 1):
        ax = pl.subplot(1, len(imgs), idx)
        pl.imshow(img, cmap="gray")
        ax.set_axis_off( )
    pl.subplots_adjust(0.02, 0, 0.98, 1, 0.02, 0)

```

1. 膨胀和腐蚀

二值图像最基本的形态学运算是膨胀和腐蚀。膨胀运算是将与某物体（白色区域）接触的所有背景像素（黑色区域）合并到该物体中的过程。简单来说，就是对于原始图像中的每个白色像素进行处理，将其周围的黑色像素都设置为白色像素。这里的“周围”是一个模糊概念，在实际运算时，需要明确给出“周围”的定义。图3-50是膨胀运算的一个例子，其中左图是原始图像，中间的图是四连通定义的“周围”的膨胀效果，右图是八连通定义的“周围”的膨胀效果。图中用灰色方块表示由膨胀处理添加进物体的像素。

```

from scipy.ndimage import morphology

def dilation_demo(a, structure=None):
    b = morphology.binary_dilation(a, structure)
    img = expand_image(a, 255)
    return expand_image(np.logical_xor(a,b), 150, out=img)

a = pl.imread("scipy_morphology_demo.png")

```

```
[:, :, 0].astype(np.uint8)

img1 = expand_image(a, 255)

img2 = dilation_demo(a)

img3 = dilation_demo(a, [[1, 1, 1], [1, 1, 1], [1, 1, 1]])

show_image(img1, img2, img3)
```

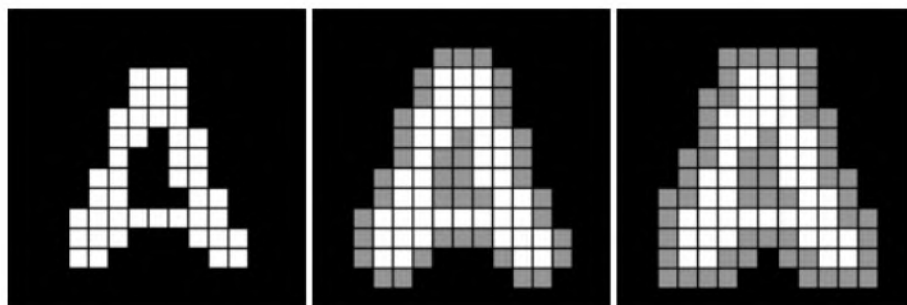


图3-50 四连通和八连通的膨胀运算

四连通包括上下左右4个像素，而八连通则还包括4个斜线方向上的邻接像素。它们都可以使用下面的正方形矩阵定义，其中正中心的元素表示当前要进行运算的像素，而其周围的1和0表示对应位置的像素是否算作其“周围”像素。这种矩阵描述了周围像素和当前像素之间的关系，被称作结构元素(structuring element)。

四连通 八连通

0 1 0	1 1 1
1 1 1	1 1 1
0 1 0	1 1 1

假设数组a是一个表示二值图像的数组，可以用如下语句对其进行膨胀运算：

```
binary_dilation(a)
```

`binary_dilation()` 默认使用四连通进行膨胀运算，通过 `structure` 参数可以指定其他的结构元素。下面是进行八连通膨胀运算的语句：

```
binary_dilation(a, structure=[[1,1,1],[1,1,1],[1,1,1]])
```

通过设置不同的结构元素，能够实现各种不同的效果，图3-51显示了三种不同结构元素的膨胀效果。图中的结构元素分别为：

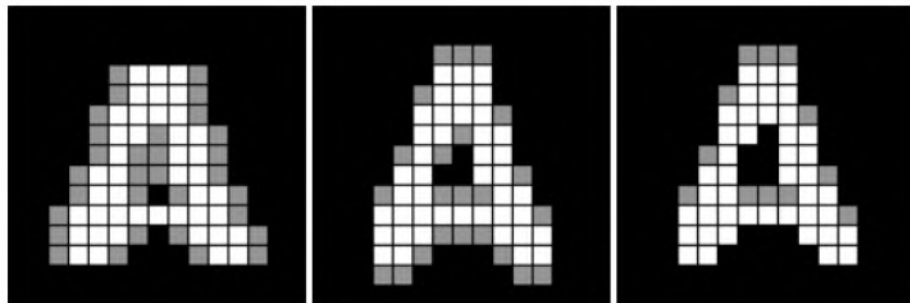


图3-51 不同结构元素的膨胀效果

左中右

```
0 0 0    0 1 0    0 1 0
1 1 1    0 1 0    0 1 0
0 0 0    0 1 0    0 0 0
```

```
img4 = dilation_demo(a, [[0,0,0],[1,1,1],[0,0,0]])
img5 = dilation_demo(a, [[0,1,0],[0,1,0],[0,1,0]])
img6 = dilation_demo(a, [[0,1,0],[0,1,0],[0,0,0]])
show_image(img4, img5, img6)
```

`binary_erosion()` 的腐蚀运算正好和膨胀相反，它将“周围”有黑色像素的白色像素设置为黑色。图3-52是四连通和八连通腐蚀的效果，图中用灰色方块表示被腐蚀的像素。

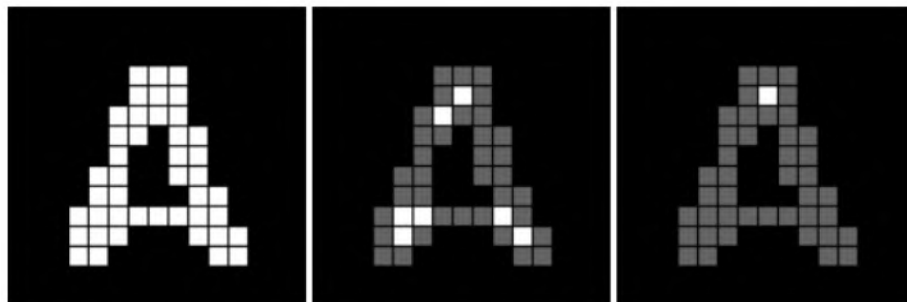


图3-52 四连通和八连通的腐蚀运算

```
def erosion_demo(a, structure=None):
    b = morphology.binary_erosion(a, structure)
    img = expand_image(a, 255)
    return expand_image(np.logical_xor(a,b), 100, out=img)

img1 = expand_image(a, 255)
img2 = erosion_demo(a)
img3 = erosion_demo(a, [[1,1,1],[1,1,1],[1,1,1]])
show_image(img1, img2, img3)
```

2. Hit和Miss

Hit和Miss是二值形态学图像处理中最基本的运算，因为几乎所有其他的运算都可以用Hit和Miss的组合推演出来。它对图像中的每个像素周围的像素进行模式判断，如果周围像素的黑白模式符合指定的模式，将此像素设为白色，否则设置为黑色。因为它需要同时对白色和黑色像素进行判断，因此需要指定两个结构元素。进行Hit和Miss运算的`binary_hit_or_miss()`的调用形式如下：

```
binary_hit_or_miss(input, structure1=None, structure2=None, ...)
```


其中structure1参数指定白色像素的结构元素，而structure2参数则指定黑色像素的结构元素。图3-53是binary_hit_or_miss()的运算结果。其中左图为原始图像，中图为使用下面两个结构元素进行运算的结果：

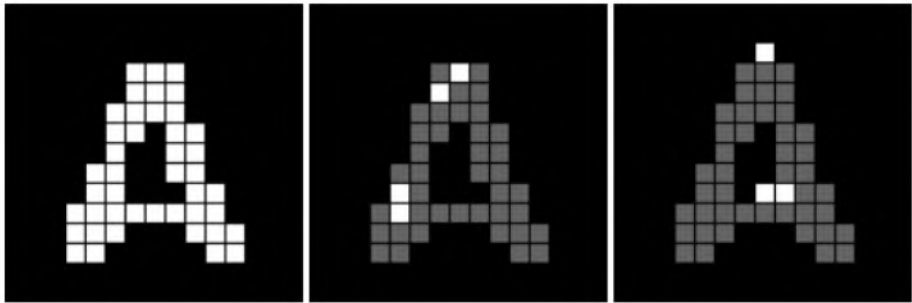


图3-53 Hit和Miss运算

白色结构元素	黑色结构元素
0 0 0	1 0 0
0 1 0	0 0 0
1 1 1	0 0 0

在这两个结构元素中，0表示不关心其对应位置的像素的颜色，1表示其对应位置的像素必须为结构元素所表示的颜色。因此通过这两个结构元素可以找到“下方三个像素为白色，并且左上像素为黑色的白色像素”。

与右图对应的结构元素如下，通过它可以找到“下方三个像素为白色、左上像素为黑色的黑色像素”。

白色结构元素	黑色结构元素
0 0 0	1 0 0
0 0 0	0 1 0
1 1 1	0 0 0

```
def hitmiss_demo(a, structure1, structure2):  
    b = morphology.binary_hit_or_miss(a, structure1, structure2)
```

```

img = expand_image(a, 100)
return expand_image(b, 255, out=img)

img1 = expand_image(a, 255)

img2 = hitmiss_demo(a, [[0,0,0],[0,1,0],[1,1,1]], [[1,0,0],
[0,0,0],[0,0,0]])
img3 = hitmiss_demo(a, [[0,0,0],[0,0,0],[1,1,1]], [[1,0,0],
[0,1,0],[0,0,0]])

show_image(img1, img2, img3)

```

使用Hit和Miss运算的组合，可以实现复杂的图像处理。例如文字识别中常用的细线化运算就可以用一系列的Hit和Miss运算实现。图3-54显示了细线化处理的效果，实现程序如下：

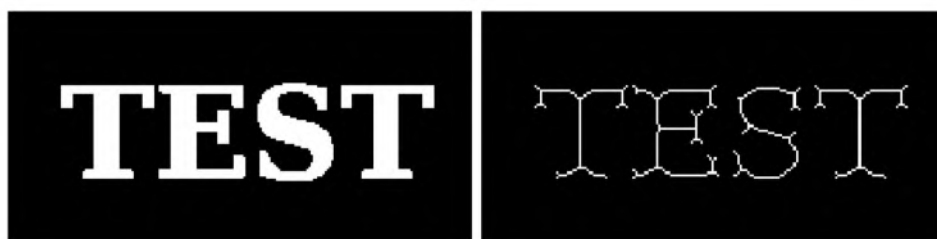


图3-54 使用Hit和Miss进行细线化运算

```

def skeletonize(img):
    h1 = np.array([[0, 0, 0],[0, 1, 0],[1, 1, 1]]) ❶
    m1 = np.array([[1, 1, 1],[0, 0, 0],[0, 0, 0]])
    h2 = np.array([[0, 0, 0],[1, 1, 0],[0, 1, 0]])
    m2 = np.array([[0, 1, 1],[0, 0, 1],[0, 0, 0]])
    hit_list = []
    miss_list = []

```

```

        for k in range(4): ❷
            hit_list.append(np.rot90(h1, k))
            hit_list.append(np.rot90(h2, k))
            miss_list.append(np.rot90(m1, k))
            miss_list.append(np.rot90(m2, k))

    img = img.copy()
    while True:
        last = img
        for hit, miss in zip(hit_list, miss_list):
            hm = morphology.binary_hit_or_miss(img, hit, miss)
❸
            # 从图像中删除hit_or_miss所得到的白色点
            img = np.logical_and(img, np.logical_not(hm)) ❹
            # 如果处理之后的图像和处理前的图像相同，则结束处理
            if np.all(img == last): ❺
                break
    return img

    a = pl.imread("scipy_morphology_demo2.png")
    [:,:,0].astype(np.uint8)
    b = skeletonize(a)

```

❶以图3-55所示的两个结构元素为基础，构造4个形状为(3, 3)的二维数组：h1、m1、h2、m2。其中h1和m1对应图中左边的结构元素，而h2和m2对应图中右边的结构元素，h1和h2对应白色结构元素，m1和m2对应黑色结构元素。❷将这些结构元素进行90°、180°、270°旋转之后一共得到8个结构元素。

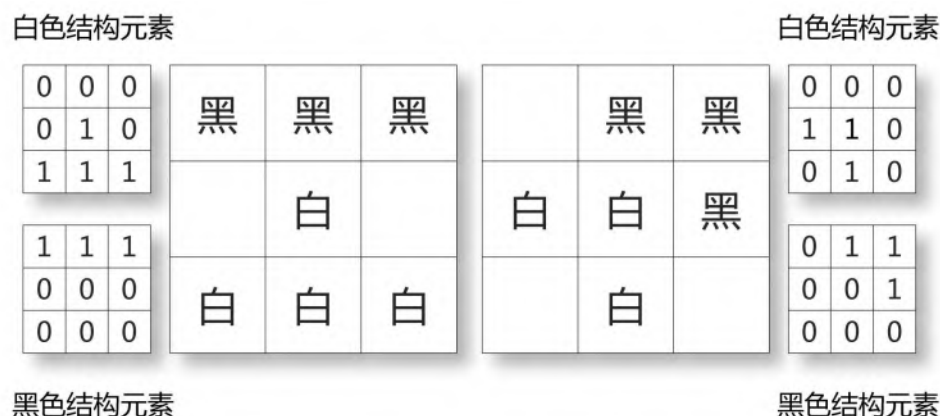


图3-55 细线化算法的4个结构元素

③依次使用这些结构元素进行Hit和Miss运算，④并从图像中删除运算所得到的白色像素，其效果就是依次从8个方向删除图像的边缘上的像素。
⑤重复运算直到没有像素可删除为止。

3.9.2 图像分割

下面以矩形区域识别为例，介绍如何使用measurements和morphology进行图像区域分割。我们要抽取矩形信息的图像如图3-56所示。这个图像是二值图像，其中矩形区域为白色，背景为黑色。但是由于它采用JPEG格式储存，因此用`pyplot.imread()`读取的是一个形状为(高，宽，3)的三通道图像。下面的程序使用其中的第0通道将其转换成二值数组`squares`，将矩形区域设置为1，将背景设置为0。结果如图3-56(左上)所示。

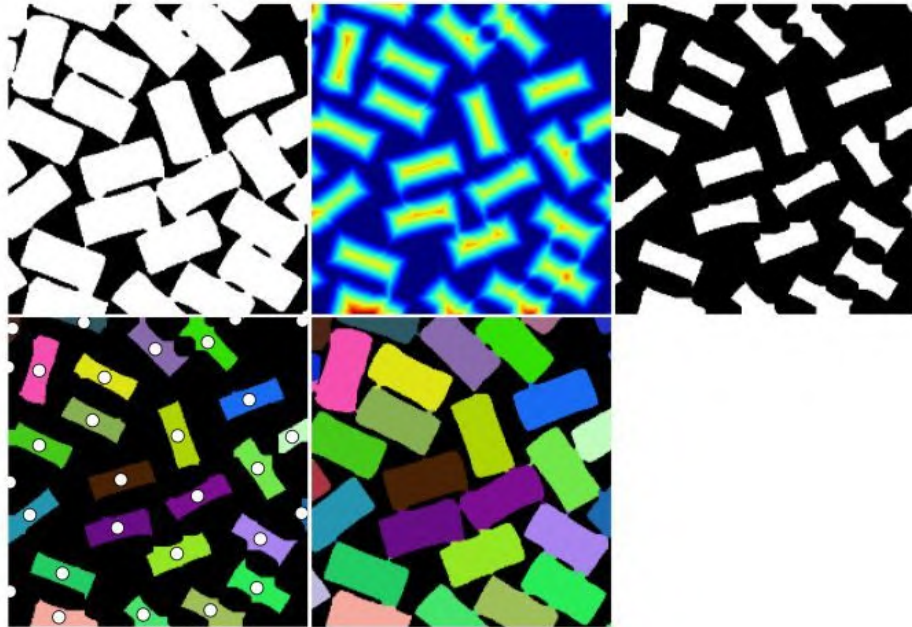


图3-56 矩形区域分割算法各个步骤的输出图像

```
squares = pl.imread("squares.jpg")
squares = (squares[:, :, 0] < 200).astype(np.uint8)
```

由于许多矩形都有一些小凸起与邻近的矩形连在一起，我们需要先将每块矩形与其周围的矩形分离出来。可以使用上节介绍的二值腐蚀函数 `morphology.binary_erosion()` 实现这一功能。不过这里我们采用另外的方法。

`morphology.distance_transform_cdt(image)` 计算二值图像中每个像素到最近的黑色像素的距离，返回一个保存所有距离的数组。图像上两点之间的距离有很多定义方式，此函数默认采用切比雪夫距离。两点之间的切比雪夫距离定义为其各坐标数值差的最大值，即 $D_{\text{Chess}} = \max(|x_2 - x_1|, |y_2 - y_1|)$ 。

下面调用 `distance_transform_cdt(squares)` 得到距离数组 `squares_dt`，并绘制成图。图中颜色越红的像素表示该点距离黑色背景越

远，而原图中值为0的像素对应的距离为0，离黑色背景最远的距离为27个像素。如果将距离数组当作图像输出，显示效果如图3-56(中上)所示。

```
from scipy.ndimage import morphology
squares_dt = morphology.distance_transform_cdt(squares)
print "各种距离值", np.unique(squares_dt)
各种距离值 [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
18 19 20 21 22 23 24 25 26 27]
```

只需要选择合适的阈值将距离数组squares_dt转换成二值数组，就可以实现矩形区域的分离，其效果和腐蚀算法类似。squares_core中每个矩形区域都缩得足够小，以至于没有任何两块区域之间有连通的路径。其效果如图3-56(右上)所示，可以看到所有的区域都和其相邻的区域分割开了。

```
squares_core = (squares_dt>8).astype(np.uint8)
```

下面调用measurements中的label()和center_of_mess()对各个独立的白色区域进行染色，并计算各个区域的重心坐标。

labels()对二值图像中每个独立的白色区域使用唯一的整数进行填充，这相当于将每块区域染上不同的“颜色”。这里所谓的“颜色”是指为每个区域指定的一个整数，而不是指图像中像素的真正颜色。

labels()返回的结果数组可以用于计算各个区域的一些统计信息。例如可以调用center_of_mass()计算每个区域的重心。其第一个参数为各个像素的权值(可以认为是每个像素的质量密度)，第二个参数为labels()输出的染色数组，第三个参数为需要计算重心的区域的标签列表。在下面的程序中，权值数组和染色数组相同，因此可以计算区域中所有白色像素的重心。

如果直接使用imshow()显示labels()输出的染色数组，将会得到一个区域颜色逐渐变化的图像，这样的图像不利于肉眼识别各个区域。因此下面的程序用random_palette()为每个整数分配一个随机颜色。其结果如图3-56(左下)所示，每个区域的重心使用白色小圆点表示。

```
from scipy.ndimage.measurements import label, center_of_mass

def random_palette(labels, count, seed=1):
    np.random.seed(seed)
    palette = np.random.rand(count+1, 3)
    palette[0,:] = 0
    return palette[labels]

labels, count = label(squares_core)
h, w = labels.shape
centers = np.array(center_of_mass(labels, labels, index=range(1,
count+1)), np.int)
cores = random_palette(labels, count)
```

我们几乎完成了区域识别的任务，但是每个矩形区域都比原始图像中的要小一圈。下面将染色之后的矩形区域恢复到原始图像中的大小。即对于原始图像中为白色、腐蚀之后的图像中为黑色的每个像素，将其颜色设置为染色数组中与之最近的区域的颜色。具体的计算步骤如下：

❶ 将腐蚀之后的图像 square_core 反转，并调用 distance_transform_cdt()。这样可以找到square_core中距离每个黑色像素最近的白色像素的坐标。为了让其返回坐标信息，需要设置参数 return_indices为True。由于这里不需要距离信息，因此可以设置参数 return_distances为False。其返回值index是一个形状为(2, 高, 宽)的三

维数组。index[0]为最近像素的第0轴(纵轴)坐标，index[1]为最近像素的第1轴(横轴)坐标。

❷ 使用index[0]和index[1]获取labels中对应坐标的颜色，得到near_labels。

❸ 创建一个布尔数组mask，其中的每个True值对应squares中为白色、squares_core中为黑色的像素。将labels复制一份到labels2，最后将mask中True对应的near_labels中的颜色值复制到labels2中。

图3-56(中下)是使用random_palette()随机着色之后的结果。

```
index = morphology.distance_transform_cdt(1-squares_core,
return_distances=False,
return_indices=True)
❶
near_labels = labels[index[0], index[1]] ❷

mask = (squares - squares_core).astype(bool)
labels2 = labels.copy()
labels2[mask] = near_labels[mask] ❸
separated = random_palette(labels2, count)
```

3.10 空间算法库-spatial



与本节内容对应的Notebook为：03-scipy/scipy-A00-spatial.ipynb。

本节介绍scipy.spatial模块中提供的K-d树、凸包、沃罗诺伊图、德劳内三角化等空间算法类的使用方法。

3.10.1 计算最近旁点

众所周知，对于一维的已排序的数值，可以使用二分法快速找到与指定数值最近的数值。在下面的例子中，在一个升序排序的随机数数组x中，使用numpy.searchsorted()搜索离0.5最近的数。排序算法的时间复杂度为 $O(N\log N)$ ，而每次二分搜索的时间复杂度为 $O(\log N)$ 。

```
x = np.sort(np.random.rand(100))
idx = np.searchsorted(x, 0.5)
print x[idx], x[idx - 1] #距离0.5最近的数是这两个数中的一个
0.542258714465 0.492205345391
```

类似的算法可以推广到N维空间，spatial模块提供的cKDTree类使用K-d树快速搜索N维空间中的最近点。在下面的例子中，用平面上随机的100个点创建cKDTree对象，并对其搜索与targets中每个点距离最近的3个点。cKDTree.query()返回两个数组dist和idx，dist[i, :]是距离targets[i]最近的3个点的距离，而idx[i, :]是这些最近点的下标：

```
from scipy import spatial
np.random.seed(42)
N = 100
points = np.random.uniform(-1, 1, (N, 2))
kd = spatial.cKDTree(points)
```

```

    targets = np.array([(0, 0), (0.5, 0.5), (-0.5, 0.5), (0.5,
-0.5), (-0.5, -0.5)])
    dist, idx = kd.query(targets, 3)

           dist                               idx
-----
[[ 0.15188266,  0.21919416,  0.27647793],  [[48, 73, 81],
 [ 0.09595807,  0.15745334,  0.22855398],  [37, 78, 43],
 [ 0.05009422,  0.17583445,  0.1807312 ],  [79, 22, 92],
 [ 0.11180181,  0.16618122,  0.18127473],  [35, 58,  6],
 [ 0.19015485,  0.19060739,  0.19361173]]  [83,  7, 42]]

```

`cKDTree.query_ball_point()` 搜索与指定点在一定距离之内的所有点，它只返回最近点的下标，由于每个目标点的近旁点数不一定相同，因此`idx2`数组中的每个元素都是一个列表：

```

r = 0.2
idx2 = kd.query_ball_point(targets, r)
idx2
array([[48], [37, 78], [79, 92, 22], [58, 35, 6], [7, 55, 83,
42]], dtype=object)

```

`cKDTree.query_pairs()` 找到`points`中距离小于指定值的每一对点，它返回的是一个下标对的集合对象。下面的程序使用集合差集运算找出所有距离在0.08到0.1之间的点对：

```

idx3 = kd.query_pairs(0.1) - kd.query_pairs(0.08)
idx3
{(1, 46),      (3, 21),      (3, 82),      (3, 95),      (5, 16),
(9, 30),
(10, 87),      (11, 42),      (11, 97),      (18, 41),      (29, 74),

```

```
(32, 51),  
      (37, 78),      (39, 61),      (41, 61),      (50, 84),      (55, 83),  
(73, 81) }
```

在图3-57中，与target中的每个点(用五角星表示)最近的点用与其相同的颜色标识：

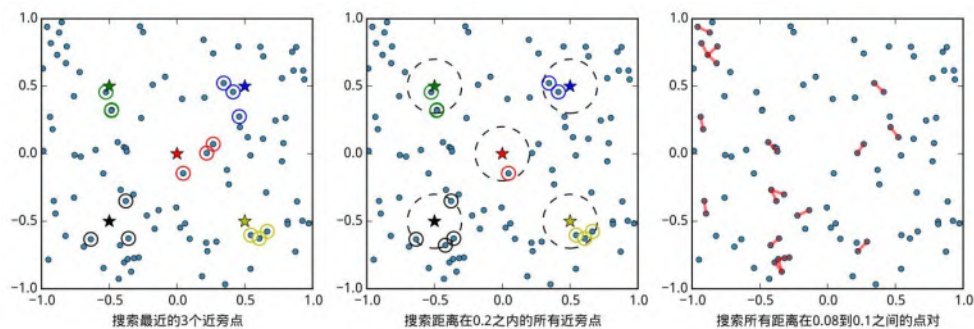


图3-57 用cKDTree寻找近旁点

cKDTree的所有搜索方法都有一个参数 p ，用于定义计算两点之间距离的函数。读者可以尝试使用不同的 p 参数，观察图3-57的变化：

- $p = 1$ ：绝对值之和作为距离
- $p = 2$ ：欧式距离
- $p = \text{np. inf}$ ：最大坐标差值作为距离

此外，`cKDTree.query_ball_tree()`可以在两棵K-d树之间搜索距离小于给定值的所有点对。

`distance`子模块中的`pdist()`计算一组点中每对点的距离，而`cdist()`计算两组点中每对点的距离。由于`pdist()`返回的是一个压缩之后的一维数组，需要用`squareform()`将其转换成二维数组。`dist1[i, j]`是`points`中下标为 i 和 j 的两个点的距离，`dist2[i, j]`是`points[i]`和`targets[j]`之间的距离。

```

from scipy.spatial import distance

dist1 = distance.squareform(distance.pdist(points))

dist2 = distance.cdist(points, targets)

dist1.shape  dist2.shape
-----
(100, 100)   (100, 5)

```

下面使用`np.min()`在`dist2`中搜索`points`中与`targets`距离最近的点，其结果与`cKDTree.query()`的结果相同：

```

print dist[:, 0] # cKDTree.query() 返回的与targets最近的距离
print np.min(dist2, axis=0)

[ 0.15188266  0.09595807  0.05009422  0.11180181  0.19015485]
[ 0.15188266  0.09595807  0.05009422  0.11180181  0.19015485]

```

为了找到`points`中最近的点对，需要将`dist1`对角线上的元素填充为无穷大：

```

dist1[np.diag_indices(len(points))] = np.inf

nearest_pair = np.unravel_index(np.argmin(dist1), dist1.shape)
print nearest_pair, dist1[nearest_pair]

(22, 92) 0.00534621024816

```

用`cKDTree.query()`可以快速找到这个距离最近的点对：在K-d树中搜索它自己包含的点，找到与每个点最近的两个点，其中距离最近的点就是它本身，距离为0，而距离第二近的点就是每个点的最近旁点，然后只需要找到这些距离中最小的那个即可：

```

dist, idx = kd.query(points, 2)

print idx[np.argmin(dist[:, 1])], np.min(dist[:, 1])

[22 92] 0.00534621024816

```

让我们看一个使用K-d树提高搜索速度的实例。下面的start和end数组保存用户登录和离开网站的时间，对于任意指定的时刻time，计算该时刻在线用户的数量。

```
N = 1000000
start = np.random.uniform(0, 100, N)
span = np.random.uniform(0.01, 1, N)
span = np.clip(span, 2, 100)
end = start + span
```

下面的naive_count_at()采用逐个比较的方法计算指定时间的在线用户数量：

```
def naive_count_at(start, end, time):
    mask = (start < time) & (end > time)
    return np.sum(mask)
```

图3-58显示了如何使用K-d树实现快速搜索。图中每点的横坐标为start，纵坐标为end。由于end大于start，因此所有的点都在y=x斜线的上方。图中阴影部分表示满足(start < time) & (end > time)条件的区域。该区域中的点数为时刻time时的在线人数。

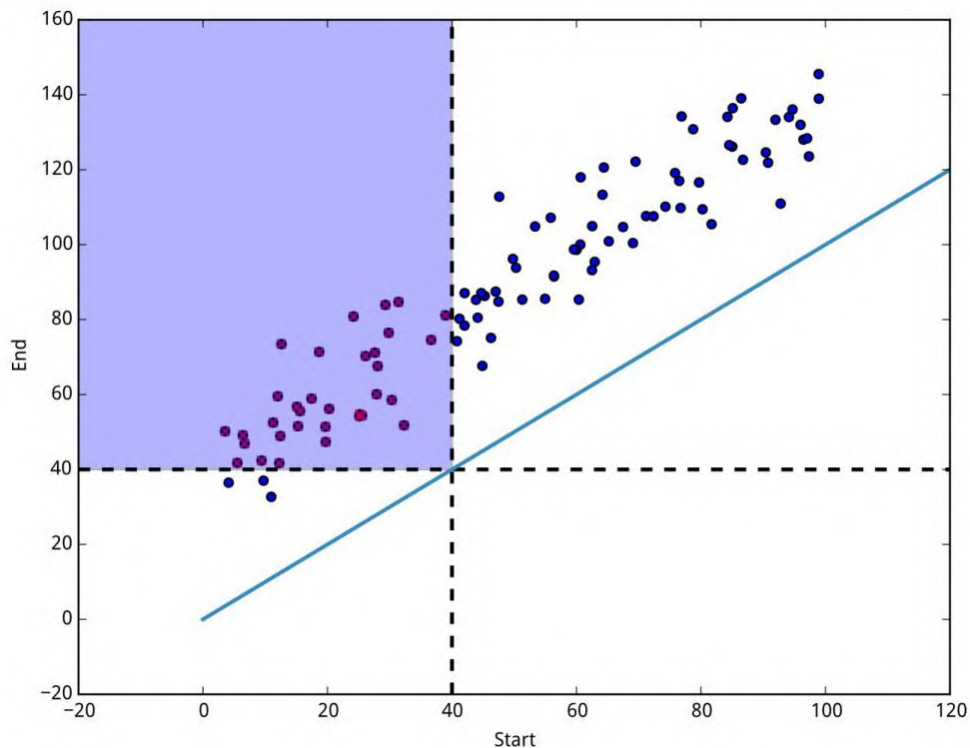


图3-58 使用二维K-d树搜索指定区间的在线用户

`tree.count_neighbors(other, r, p=2)`可以统计tree中到other的距离小于r的点数，其中p为计算距离的范数。距离按照如下公式计算：

$$N_p(x) = \|x\|_p = (|x_1|^p + |x_2|^p + \dots + |x_n|^p)^{\frac{1}{p}}$$

当p=2时，上面的公式就是欧几里得空间中的向量长度。当p=∞时，该距离变为各个轴上的最大绝对值：

$$N_{\infty}(x) = \|x\|_{\infty} = \max(|x_1|, |x_2|, \dots, |x_n|)$$

当使用p=∞时可以计算某正方形区域之内的点数。我们将该正方形的中心设置为(`time - max_time`, `time + max_time`)，正方形的边长为`2 * max_time`，即`r = max_time`。其中`max_time`为end中的最大值。下面的KdSearch类实现了该算法：

```

class KdSearch(object):
    def __init__(self, start, end, leafsize=10):
        self.tree = spatial.cKDTree(np.c_[start, end],
leafsize=leafsize)

        self.max_time = np.max(end)

    def count_at(self, time):
        max_time = self.max_time
        to_search = spatial.cKDTree([[time - max_time, time +
max_time]])

        return self.tree.count_neighbors(to_search, max_time,
p=np.inf)

naive_count_at(start, end, 40) == KdSearch(start,
end).count_at(40)

True

```

下面比较运算时间，由结果可知创建K-d树需要约0.5秒时间，K-d树的搜索速度则为线性搜索的17倍左右。



请读者研究点数N和leafsize参数与创建K-d树和搜索时间之间的关系。

```

%time ks = KdSearch(start, end, leafsize=100)

%timeit naive_search(start, end, 40)

%timeit ks.count_at(40)

Wall time: 484 ms

```

```
100 loops, best of 3: 3.85 ms per loop
```

```
1000 loops, best of 3: 221 µs per loop
```

3.10.2 凸包

所谓凸包是指N维空间中的一个区域，该区域中任意两点之间的线段都完全被包含在该区域之中，二维平面上的凸多边形就是典型的凸包。ConvexHull可以快速计算包含N维空间中点的集合的最小凸包。下面先看一个二维的例子：points2d是一组二维平面上的随机点，ch2d是这些点的凸包对象。ConvexHull.simplices是凸包的每条边线的两个顶点在points2d中的下标，由于它的形状为(5, 2)，因此凸包由5条线段构成。对于二维的情况，ConvexHull.vertices是凸多边形的每个顶点在points2d中的下标，按逆时针方向的顺序排列。

```
np.random.seed(42)
points2d = np.random.rand(10, 2)
ch2d = spatial.ConvexHull(points2d)
ch2d.simplices    ch2d.vertices
-----
[[2, 5],          [5, 2, 6, 1, 0]
 [2, 6],
 [0, 5],
 [1, 6],
 [1, 0]]
```

使用matplotlib中的Polygon对象可以绘制如图3-59所示的多边形。

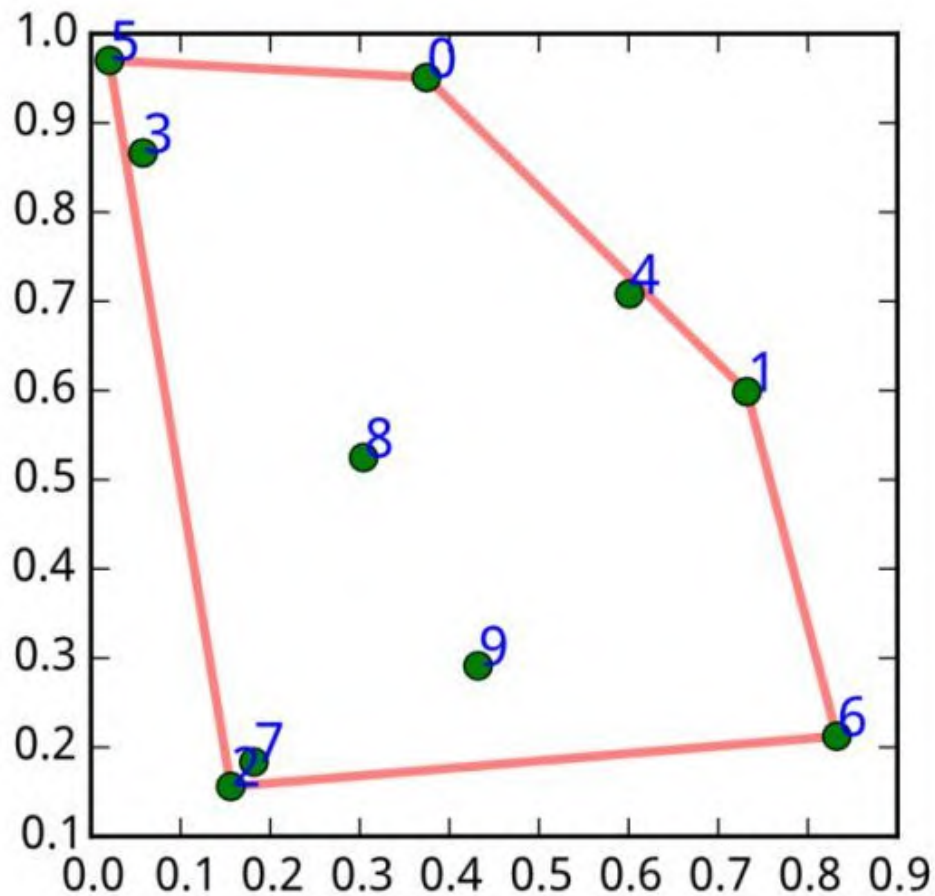


图3-59 二维平面上的凸包

```

poly = pl.Polygon(points2d[ch2d.vertices], fill=None, lw=2,
color="r", alpha=0.5)

ax = pl.subplot(aspect="equal")
pl.plot(points2d[:, 0], points2d[:, 1], "go")
for i, pos in enumerate(points2d):
    pl.text(pos[0], pos[1], str(i), color="blue")
ax.add_artist(poly)

```

三维空间中的凸包是一个凸多面体，每个面都是一个三角形。在下面的例子中，由simplices的形状可知，所得到的凸包由38个三角形面构成：

```
np.random.seed(42)
points3d = np.random.rand(40, 3)
ch3d = spatial.ConvexHull(points3d)
ch3d.simplices.shape
(38, 3)
```

下面的程序用TVTK直观地显示凸包，图3-60中所有的绿色圆球表示points3d中的点，由红色线段构成的三角形面表示凸多面体上的面。没有和红色线段相连的点就是凸包之内的点：

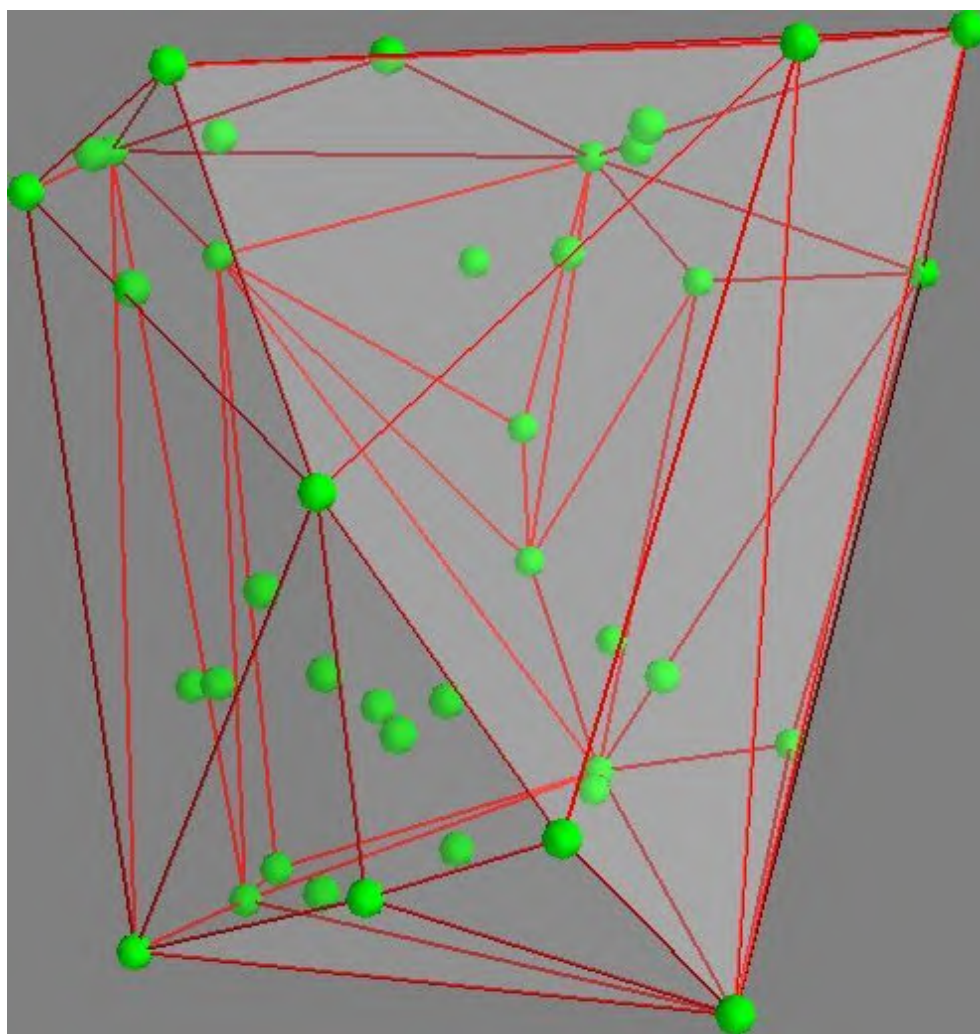


图3-60 三维空间中的凸包

```

from scipy2 import vtk_convexhull, vtk_scene, vtk_scene_to_array
actors = vtk_convexhull(ch3d)
scene = vtk_scene(actors, viewangle=22)
%array_image vtk_scene_to_array(scene)
scene.close()

```

如果读者希望采用三维交互界面，可以在Notebook中执行如下代码：

```

%gui qt
from scipy2 import ivtk_scene
ivtk_scene(actors)

```

3. 10. 3 沃罗诺伊图

沃罗诺伊图 (Voronoi Diagram) 是一种空间分割算法，它根据指定的N个胞点将空间分割为N个区域，每个区域中的所有坐标点都与该区域对应的胞点最近。

```

points2d = np.array([[0.2, 0.1], [0.5, 0.5], [0.8, 0.1],
[0.5, 0.8], [0.3, 0.6], [0.7, 0.6], [0.5, 0.35]])
vo = spatial.Voronoi(points2d)

```

vo.vertices	vo.regions	vo.ridge_vertices
-----	-----	-----
[[0.5 , 0.045],	[[-1, 0, 1],	[[-1, 0],
[0.245 , 0.351],	[-1, 0, 2],	[0, 1],
[0.755 , 0.351],	[],	[-1, 1],
[0.3375, 0.425],	[6, 4, 3, 5],	[0, 2],
[0.6625, 0.425],	[5, -1, 1, 3],	[-1, 2],
[0.45 , 0.65],	[4, 2, 0, 1, 3],	[3, 5],

```

[ 0.55 , 0.65 ]] [6, -1, 2, 4], [3, 4],
[6, -1, 5]] [4, 6],
[5, 6],
[1, 3],

[-1, 5],
[2, 4],
[-1, 6]]

```

使用`voronoi_plot_2d()`可以将沃罗诺伊图显示为图表，效果如图3-61(左)所示。图中蓝色小圆点为`points2d`指定的胞点，红色大圆点表示`Voronoi.vertices`中的点，图中为每个`vertices`点标注了下标。由虚线和实线将空间分为7个区域，以虚线为边的区域为无限大的区域，一直向外延伸，全部由实线构成的区域为有限区域。每个区域都以`vertices`中的点为顶点。

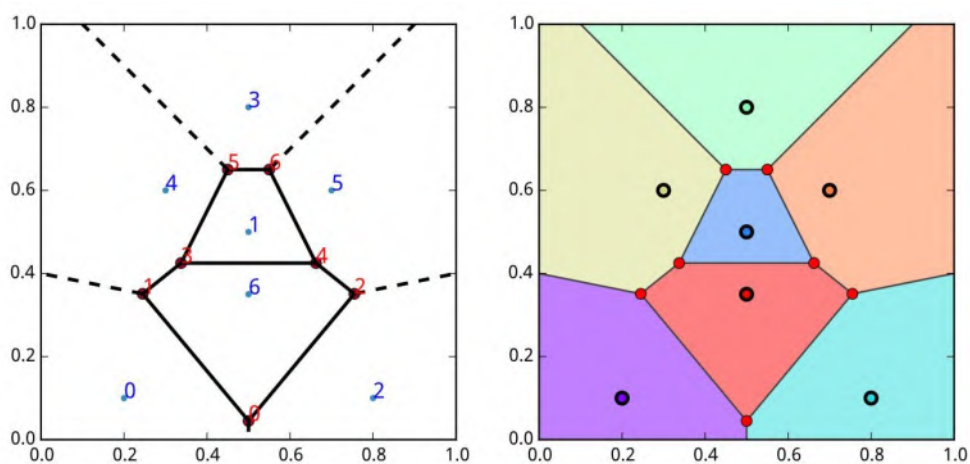


图3-61 沃罗诺伊图将空间分割为多个区域

`Voronoi.regions`是区域列表，其中每个区域由一个列表(忽略空列表)表示，列表中的整数为`vertices`中的序号，包含-1的区域为无限区域。例如`[6, 4, 3, 5]`为图中正中心的那块区域。

Voronoi.ridge_vertices是区域分割线列表，每条分割线由vertices中的两个序号构成，包含-1的分割线为图中的虚线，其长度为无限长。

如果希望将每块区域以不同颜色填充，但由于外围的区域是无限大的，因此无法使用matplotlib绘图，可以在外面添加4个点将整个区域围起来，这样每个points2d中的胞点都对应一个有限区域。在图3-61(右)中，黑圈点为points2d指定的胞点，将空间中与其最近的区域填充成胞点的颜色。

```
bound = np.array([[ -100, -100], [ -100,  100],
                  [ 100,  100], [ 100, -100]])
vo2 = spatial.Voronoi(np.vstack((points2d, bound)))
```

使用沃罗诺伊图可以解决最大空圆问题：找到一个半径最大的圆，使得其圆心在一组点的凸包区域之内，并且圆内没有任何点。根据图3-61可知最大空圆必定是以vertices为圆心，以与最近胞点的距离为半径的圆中的某一个。为了找到胞点与vertices之间的联系，可以使用Voronoi.point_region属性。point_region[i]是与第i个胞点(points2d[i])最近的区域的编号。例如由下面的输出可知，下标为5的蓝色点与下标为6的区域对应，而由Voronoi.regions[6]可知，该区域由编号为6、2、4的三个vertices点(图中红色的点)构成。因此vertices中与胞点points2d[5]最近的点的序号为[2, 4, 6]。

```
print vo.point_region
print vo.regions[6]

[0 3 1 7 4 6 5]

[6, -1, 2, 4]
```

下面是计算最大空圆的程序，效果如图3-62所示。程序中：❶使用pylab.Polygon.contains_point()判断用ConvexHull计算的凸包多边形是

否包含vertices中的点，用以在❷处剔除圆心在凸包之外的圆。

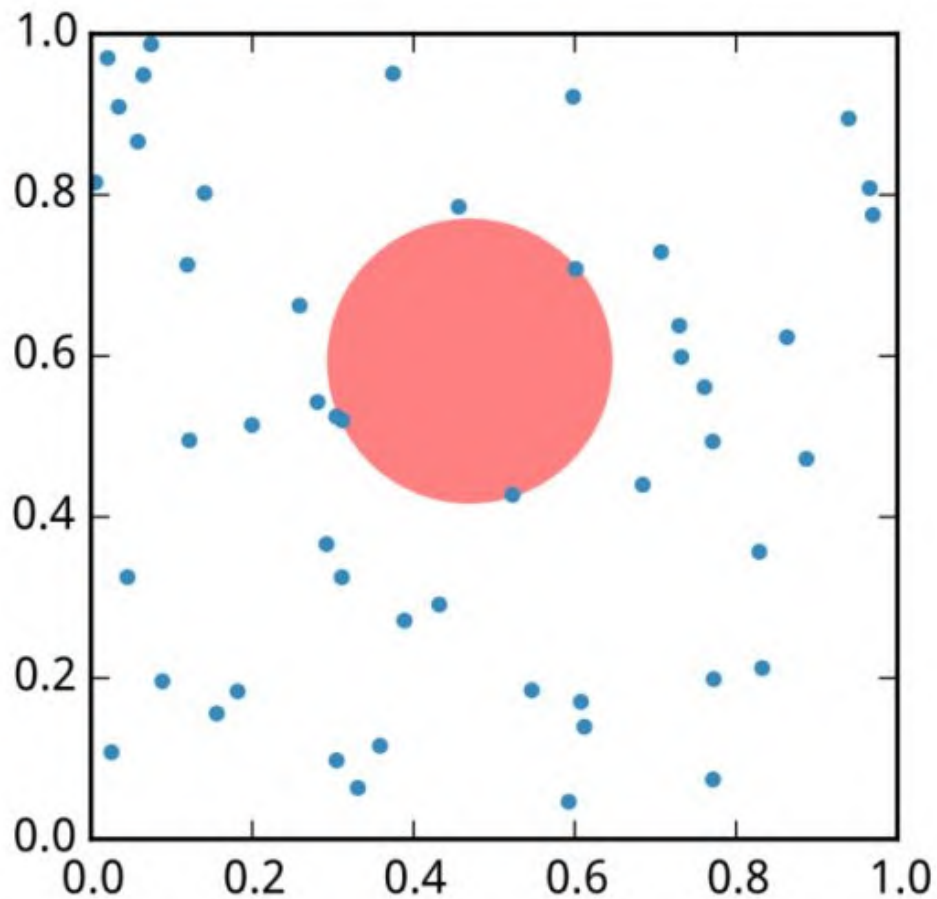


图3-62 使用沃罗诺伊图计算最大空圆

❸vertex_point_map是一个字典，它的键为vertices点的下标，值为与其最近的几个points2d点的序号。整个字典使用point_region和regions构建，注意这里剔除了所有在凸包之外的vertices点。

❹对于vertex_point_map中的每一对点，找到距离最大的那对点，即可得出圆心坐标和圆的半径。

```
from collections import defaultdict
```

```
n = 50
```

```

np.random.seed(42)
points2d = np.random.rand(n, 2)
vo = spatial.Voronoi(points2d)
ch = spatial.ConvexHull(points2d)
poly = pl.Polygon(points2d[ch.vertices]) ❶
vs = vo.vertices
convexhull_mask = [poly.contains_point(p, radius=0) for p in vs]

❷

vertex_point_map = defaultdict(list) ❸
for index_point, index_region in enumerate(vo.point_region):
    region = vo.regions[index_region]
    if -1 in region: continue
    for index_vertex in region:
        if convexhull_mask[index_vertex]:
            vertex_point_map[index_vertex].append(index_point)

def dist(p1, p2):
    return ((p1-p2)**2).sum()**0.5

max_cicle = max((dist(points2d[pidxs[0]], vs[vidx]), vs[vidx])

❹

                                for vidx, pidxs in
vertex_point_map.iteritems())
    r, center = max_cicle
    print "r = ", r, ", center = ", center
    r = 0.174278456762 , center = [ 0.46973363 0.59356531]

```

3.10.4 德劳内三角化

德劳内三角化算法对给定的点集合的凸包进行三角形分割，使得每个三角形的外接圆都不含任何点。下面的程序演示了德劳内三角化的效果：

Delaunay对象dy的simplices属性是每个三角形的顶点在points2d中的下标。可以通过三角形的顶点坐标计算其外接圆的圆心，不过这里使用同一点集合的沃罗诺伊图的vertices属性，vertices就是每个三角形的外接圆的圆心。

```
x = np.array([46.445, 263.251, 174.176, 280.899, 280.899,
              189.358, 135.521, 29.638, 101.907, 226.665])
y = np.array([287.865, 250.891, 287.865, 160.975, 54.252,
              160.975, 232.404, 179.187, 35.765, 71.361])
points2d = np.c_[x, y]
dy = spatial.Delaunay(points2d)
vo = spatial.Voronoi(points2d)
dy.simplices          vo.vertices
-----
[[8, 5, 7],          [[ 104.58977484,  127.03566055],
[1, 5, 3],           [ 235.1285      , 198.68143374],
[5, 6, 7],           [ 107.83960707, 155.53682482],
[6, 0, 7],           [  71.22104881, 228.39479887],
[0, 6, 2],           [ 110.3105      , 291.17642838],
[6, 1, 2],           [ 201.40695449, 227.68436282],
[1, 6, 5],           [ 201.61895891, 226.21958623],
[9, 5, 8],           [ 152.96231864,  93.25060083],
[4, 9, 8],           [ 205.40381294, -90.5480267 ],
```



```
[5, 9, 3],      [ 235.1285      , 127.45701644],  
[9, 4, 3]]     [ 267.91709907, 107.6135      ]]
```

下面是用`delaunay_plot_2d()`绘制德劳内三角化的结果，此外还绘制每个外接圆及其圆心。可以看到三角形的所有顶点都不在任何外接圆之内，效果如图3-63所示：

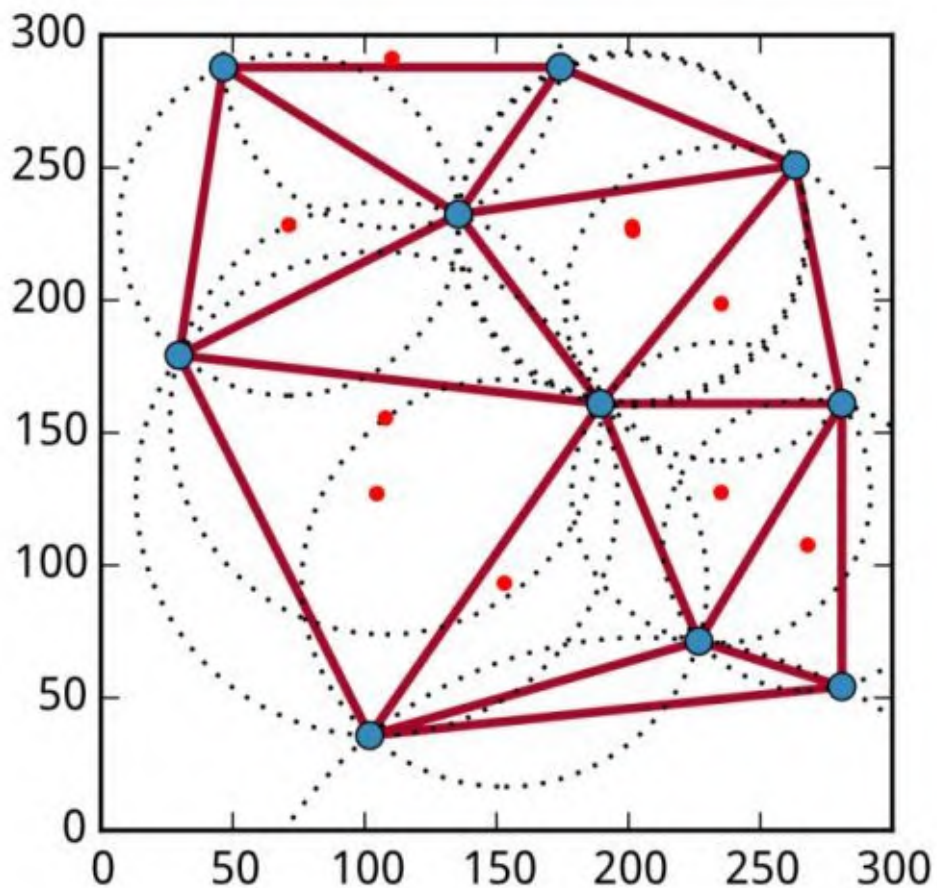


图3-63 德劳内三角形的外接圆与圆心

```
cx, cy = vo.vertices.T  
  
ax = pl.subplot(aspect="equal")  
spatial.delaunay_plot_2d(dy, ax=ax)  
ax.plot(cx, cy, "r.")
```

```
for i, (cx, cy) in enumerate(vo.vertices):
    px, py = points2d[dy.simplices[i, 0]]
    radius = np.hypot(cx - px, cy - py)
    circle = pl.Circle((cx, cy), radius, fill=False,
ls="dotted")

    ax.add_artist(circle)
ax.set_xlim(0, 300)
ax.set_ylim(0, 300)
```


第4章 matplotlib-绘制精美的图表

matplotlib是Python最著名的绘图库，它提供了一整套和MATLAB类似的绘图函数集，十分适合编写短小的脚本程序以进行快速绘图。matplotlib的文档十分完备，并且其展示页面中有上百幅图表的缩略图及其源程序。因此如果读者需要绘制某种类型的图表，只需要在<http://matplotlib.sourceforge.net/gallery.html>上“浏览/复制/粘贴”一下，基本上都能快速解决。

本章在简单介绍matplotlib的快速绘图功能之后，将较深入地挖掘几个实例，让读者从中学习和理解matplotlib绘图的一些基本概念。相信读者在了解本章的内容之后，应该能够根据官方的文档和演示程序使用matplotlib完美地展示数据。

4.1 快速绘图



与本节内容对应的 Notebook 为：04-matplotlib/matplotlib-100-fastdraw.ipynb。

matplotlib采用面向对象的技术来实现，因此组成图表的各个元素都是对象，在编写较大的应用程序时通过面向对象的方式使用matplotlib将更加有效。但是使用这种面向对象的调用接口进行绘图

比较烦琐，因此matplotlib还提供了快速绘图的pyplot模块。本节首先介绍该模块的使用方法。

为了将matplotlib绘制的图表嵌入Notebook中，需要执行下面的命令：

```
%matplotlib inline
```

使用inline模式在Notebook中绘制的图表会自动关闭，为了在Notebook的多个单元格内操作同一幅图表，需要运行下面的魔法命令：

```
%config InlineBackend.close_figures = False
```

4.1.1 使用pyplot模块绘图

matplotlib的pyplot模块提供了与MATLAB类似的绘图函数调用接口，方便用户快速绘制二维图表。我们先看一个简单的例子：

pylab模块

matplotlib还提供了一个名为pylab的模块，其中包括了许多NumPy和pyplot模块中常用的函数，方便用户快速进行计算和绘图，十分适合在IPython交互式环境中使用。本书使用import pylab as pl载入pylab模块。

```
import matplotlib.pyplot as plt ❶

x = np.linspace(0, 10, 1000)
y = np.sin(x)
z = np.cos(x**2)

plt.figure(figsize=(8,4)) ❷

plt.plot(x,y,label="$sin(x)$",color="red",linewidth=2) ❸
plt.plot(x,z,"b--",label="$cos(x^2)$") ❹

plt.xlabel("Time(s)") ❺
plt.ylabel("Volt")
plt.title("PyPlot First Example")
plt.ylim(-1.2,1.2)
plt.legend()

plt.show() ❻
```

程序的输出如图4-1所示。

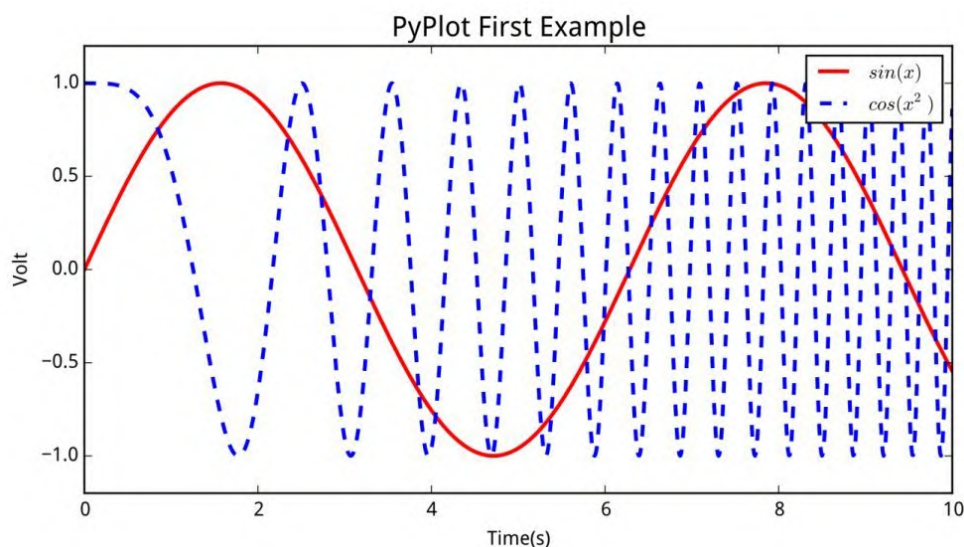


图4-1 使用pyplot模块快速将数据绘制成曲线

❶首先载入matplotlib的绘图模块pyplot，并且重命名为plt。❷调用figure() 创建一个Figure(图表)对象，并且它将成为当前Figure对象。也可以不创建Figure对象而直接调用接下来的plot() 进行绘图，这时matplotlib会自动创建一个Figure对象。figsize参数指定Figure对象的宽度和高度，单位为英寸。此外还可以用dpi参数指定Figure对象的分辨率，即每英寸所表示的像素数，这里使用默认值80。因此本例中所创建的Figure对象的宽度为 $8 \times 80 = 640$ 个像素。

❸创建Figure对象之后，接下来调用plot() 在当前的Figure对象中绘图。实际上plot() 是在Axes(子图)对象上绘图，如果当前的Figure对象中没有Axes对象，将会为之创建一个几乎充满整个图表的Axes对象，并且使此Axes对象成为当前的Axes对象。plot() 的前两个参数是分别表示X、Y轴数据的对象，这里使用的是NumPy数组。使用关键字参数可以指定所绘制曲线的各种属性：

- label: 给曲线指定一个标签，此标签将在图示中显示。如果标签字符串的前后有字符'\$'，matplotlib会使用内嵌的

LaTeX引擎将其显示为数学公式。

- `color`: 指定曲线的颜色，颜色可以用英文单词或以'#'字符开头的6位十六进制数表示，例如'#ff0000'表示红色。或者使用值在0到1范围内的三个元素的元组来表示，例如(1.0, 0.0, 0.0)也表示红色。
- `linewidth`: 指定曲线的宽度，可以不是整数，也可以使用缩写形式的参数名`lw`。



使用LaTeX语法绘制数学公式会极大地降低图表的描绘速度。

④直接通过第三个参数'`b--`'指定曲线的颜色和线型，它通过一些易记的符号指定曲线的样式。其中'`b`'表示蓝色，'`--`'表示线型为虚线。在IPython中输入`plt.plot?`可以查看格式化字符串以及各个参数的详细说明。

⑤接下来通过一系列函数设置当前Axes对象的各个属性：

- `xlabel`、`ylabel`: 分别设置X、Y轴的标题文字。
- `title`: 设置子图的标题。
- `xlim`、`ylim`: 分别设置X、Y轴的显示范围。
- `legend`: 显示图示，即图中表示每条曲线的标签(`label`)和样式的矩形区域。

⑥最后调用`plt.show()`显示绘图窗口，在Notebook中可以省略此步骤。在通常的运行情况下，`show()`将会阻塞程序的运行，直到用户

关闭绘图窗口。

还可以调用`plt.savefig()`将当前的Figure对象保存成图像文件，图像格式由图像文件的扩展名决定。下面的程序将当前的图表保存为`test.png`，并且通过`dpi`参数指定图像的分辨率为120，因此输出图像的宽度为 $8 \times 120 = 960$ 个像素。

```
plt.savefig("test.png", dpi=120)
```



如果关闭了图表窗口，则无法使用`savefig()`保存图像。实际上不需要调用`show()`显示图表，可以直接用`savefig()`将图表保存成图像文件。使用这种方法可以很容易编写批量输出图表的程序。

`savefig()`的第一个参数可以是文件名，也可以是和Python的文件对象有相同调用接口的对象。例如可以将图像保存到`io.BytesIO`对象中，这样就得到了一个表示图像内容的字符串。这里需要使用`fmt`参数指定保存的图像格式。

```
import io
buf = io.BytesIO() # 创建一个用来保存图像内容的BytesIO对象
plt.savefig(buf, fmt="png") # 将图像以png格式保存到buf中
buf.getvalue()[:20] # 显示图像内容的前20个字节
'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x03 '
```

4.1.2 面向对象方式绘图

matplotlib实际上是一套面向对象的绘图库，它所绘制的图表中的每个绘图元素，例如线条、文字、刻度等在内存中都有一个对象与之对应。为了方便快速绘图，matplotlib通过pyplot模块提供了一套和MATLAB类似的绘图API，将众多绘图对象所构成的复杂结构隐藏在这套API内部。我们只需要调用pyplot模块所提供的函数就可以实现快速绘图以及设置图表的各种细节。pyplot模块虽然用法简单，但不适合在较大的应用程序中使用，因此本章将着重介绍如何使用matplotlib的面向对象方式编写绘图程序。

为了将面向对象的绘图库包装成只使用函数的API，pyplot模块的内部保存了当前图表以及当前子图等信息。可以使用`gcf()`和`gca()`获得这两个对象，它们分别是“Get Current Figure”和“Get Current Axes”开头字母的缩写。`gcf()`获得的是表示图表的Figure对象，而`gca()`获得的则是表示子图的Axes对象。

```
fig = plt.gcf()
axes = plt.gca()
print fig, axes
Figure(640x320) Axes(0.125,0.1;0.775x0.8)
```

在pyplot模块中，许多函数都是对当前的Figure或Axes对象进行处理，例如前面介绍的`plot()`、`xlabel()`、`savefig()`等。我们可以在IPython中输入函数名并加“??”，查看这些函数的源代码以了解它们是如何调用各种对象的方法进行绘图处理的。例如下面的例子查看`plot()`函数的源程序，可以看到`plot()`函数实际上会通过`gca()`获得当前的Axes对象`ax`，然后再调用它的`plot()`方法来实现真正的绘图。请读者使用类似的方法查看pyplot模块的其他函数是如何对各种绘图对象进行包装的。

```

def plot(*args, **kwargs):
    ax = gca()
    ...
    try:
        ret = ax.plot(*args, **kwargs)
        ...
    finally:
        ax.hold(washold)

```

4.1.3 配置属性

matplotlib所绘制图表的每个组成部分都和一个对象对应，可以通过调用这些对象的属性设置方法`set_*`()或者pyplot模块的属性设置函数`setp()`来设置它们的属性值。例如`plot()`返回一个元素类型为`Line2D`的列表，下面的例子设置`Line2D`对象的属性：

```

plt.figure(figsize=(4, 3))
x = np.arange(0, 5, 0.1)
line = plt.plot(x, 0.05*x*x)[0] # plot返回一个列表
line.set_alpha(0.5) # 调用Line2D对象的set_*( )方法来设置属性值

```

上面的例子中，通过调用`Line2D`对象的`set_alpha()`，修改它在图中对应曲线的透明度。下面的语句同时绘制正弦和余弦两条曲线，`lines`是一个有两个`Line2D`对象的列表：

```

lines = plt.plot(x, np.sin(x), x, np.cos(x))

```

调用`setp()`可以同时配置多个对象的属性，这里同时设置两条曲线的颜色和线宽：

```
plt.setp(lines, color="r", linewidth=4.0)
```

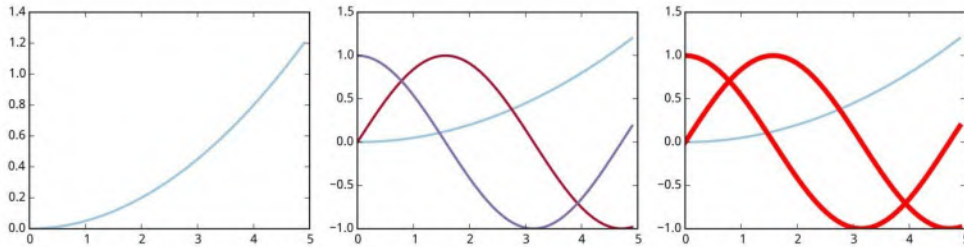


图4-2 配置绘图对象的属性

同样，可以通过调用`Line2D`对象的`get_*`()或者通过`plt.getp()`来获取对象的属性值：

```
print line.get_linewidth()
print plt.getp(lines[0], "color") # 返回color属性
2.0
r
```

注意`getp()`和`setp()`不同，它只能对一个对象进行操作，它有两种用法：

- 指定属性名：返回对象的某个属性的值。
- 不指定属性名：输出对象的所有属性和值。

下面通过`getp()`查看`Figure`对象的属性：

```
f = plt.gcf()
plt.getp(f)
```

```
agg_filter = None
alpha = None
animated = False
    axes = [<matplotlib.axes._subplots.AxesSubplot object
at ...
    ...
```

Figure对象的axes属性是一个列表，它保存图表中的所有子图对象。下面的程序查看当前图表的axes属性，它就是gca()所获得的当前子图对象：

```
print plt.getp(f, "axes"), plt.getp(f, "axes")[0] is
plt.gca()
[<matplotlib.axes._subplots.AxesSubplot object at
0x05DE5790>] True
```

用plt.getp()可以继续获取AxesSubplot对象的属性，例如它的lines属性为子图中的Line2D对象列表：

```
alllines = plt.getp(plt.gca(), "lines")
print alllines, alllines[0] is line # 其中的第一条曲线就是最开
始绘制的那条曲线
<a list of 3 Line2D objects> True
```

通过这种方法可以很容易查看对象的属性值以及各个对象之间的关系，找到需要配置的属性。因为matplotlib实际上是一套面向对象的绘图库，因此也可以直接获取对象的属性，例如：

```
print f.axes, len(f.axes[0].lines)
[<matplotlib.axes._subplots.AxesSubplot object at
0x05DE5790>] 3
```

4.1.4 绘制多子图

一个Figure对象可以包含多个子图(Axes)，在matplotlib中用Axes对象表示一个绘图区域，在本书中称之为子图。在前面的例子中，Figure对象只包括一个子图。可以使用subplot()快速绘制包含多个子图的图表，它的调用形式如下：

```
subplot(numRows, numCols, plotNum)
```

图表的整个绘图区域被等分为numRows行和numCols列，然后按照从左到右、从上到下的顺序对每个区域进行编号，左上区域的编号为1。plotNum参数指定所创建Axes对象的区域。如果numRows、numCols和plotNum三个参数都小于10，则可以把它们缩写成一个整数，例如subplot(323)和subplot(3,2,3)的含义相同。如果新创建的子图和之前创建的子图区域有重叠的部分，之前的子图将被删除。

下面的程序创建如图4-3(左)所示的3行2列共6个子图，并通过axisbg参数给每个子图设置不同的背景颜色。

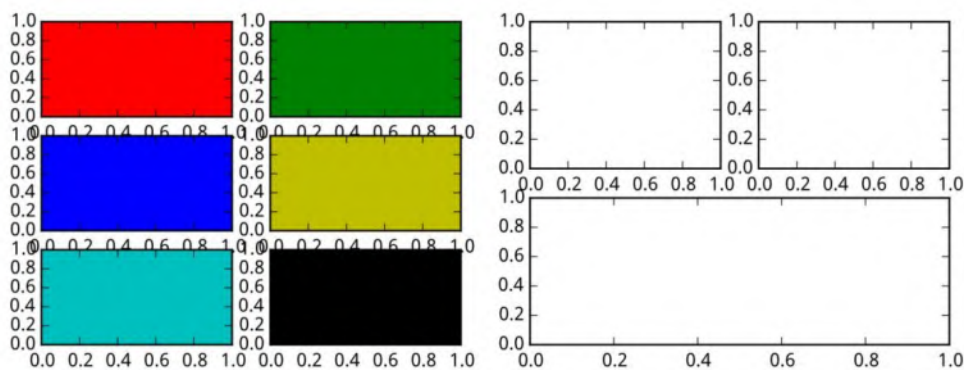


图4-3 在Figure对象中创建多个子图

```
for idx, color in enumerate("rgbyck"):
    plt.subplot(321+idx, axisbg=color)
```

如果希望某个子图占据整行或整列，可以如下调用`subplot()`，程序的输出如图4-3(右)所示。

```
plt.subplot(221) # 第一行的左图
plt.subplot(222) # 第一行的右图
plt.subplot(212) # 第二整行
```

在绘图窗口的工具栏中，有一个名为“Configure Subplots”的按钮，单击它会弹出调节子图间距和子图与图表边框距离的对话框。也可以在程序中调用`subplots_adjust()`调节这些参数，它有`left`、`right`、`bottom`、`top`、`wspace`和`hspace`共6个参数，这些参数与对话框中的各个控件对应。参数的取值范围为0到1，它们是以图表绘图区域的宽和高进行正规化之后的坐标或长度。

`subplot()`返回它所创建的Axes对象，我们可以将这些对象用变量保存起来，然后用`sca()`交替让它们成为当前Axes对象，并调用`plot()`在其中绘图。如果需要同时绘制多幅图表，可以给`figure()`传递一个

整数参数来指定Figure对象的序号。如果序号所指定的Figure对象已经存在，将不创建新的对象，而只是让它成为当前的Figure对象。下面的程序演示了依次在不同图表的不同子图中绘制曲线：

```
plt.figure(1) # 创建图表1
plt.figure(2) # 创建图表2
ax1 = plt.subplot(121) # 在图表2中创建子图1
ax2 = plt.subplot(122) # 在图表2中创建子图2

x = np.linspace(0, 3, 100)
for i in xrange(5):
    plt.figure(1) ❶选择图表1
    plt.plot(x, np.exp(i*x/3))
    plt.sca(ax1) ❷选择图表2的子图1
    plt.plot(x, np.sin(i*x))
    plt.sca(ax2) # 选择图表2的子图2
    plt.plot(x, np.cos(i*x))
```



也可以不调用sca()指定当前子图，而直接调用ax1和ax2的plot()方法来绘图。

首先通过figure()创建了两个图表，它们的序号分别为1和2。然后在图表2中创建了左右并排的两个子图，并用变量ax1和ax2保存。

在循环中，❶先调用figure(1)让图表1成为当前图表，并在其中绘图。❷然后调用sca(ax1)和sca(ax2)分别让子图ax1和ax2成为当前子图，并在其中绘图。当它们成为当前子图时，包含它们的图表2也自动成为当前图表，因此不需要调用figure(2)。这样依次在图表1和图表2的两个子图之间切换，逐步在其中添加新的曲线，效果如图4-4所示。

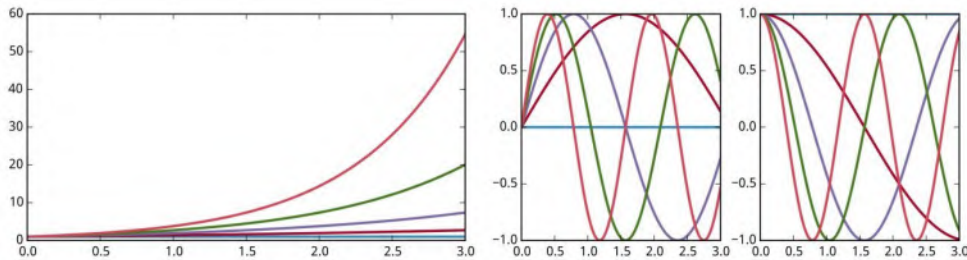


图4-4 同时多幅图表、多个子图中进行绘图

此外subplots()可以一次生成多个子图，并返回图表对象和保存子图对象的数组。在下面的例子中，axes是一个形状为(2, 3)的数组，每个元素都是一个子图对象，可以利用Python的赋值功能将这个数组中的每个元素用一个变量表示：

```
fig, axes = plt.subplots(2, 3)
[a, b, c], [d, e, f] = axes
print axes.shape
print b
(2, 3)
Axes(0.398529,0.536364;0.227941x0.363636)
```

还可以调用subplot2grid()进行更复杂的表格布局。表格布局和在Excel或Word中绘制表格十分类似，其调用参数如下：

```
subplot2grid(shape, loc, rowspan=1, colspan=1, **kwargs)
```

其中，`shape`为表示表格形状的元组：（行数，列数）。`loc`为子图左上角所在的坐标：（行，列）。`rowspan`和`colspan`分别为子图所占据的行数和列数。在下面的例子中，在 3×3 的网格上创建5个子图，在每个子图中间显示该子图对应的变量名，如图4-5所示：

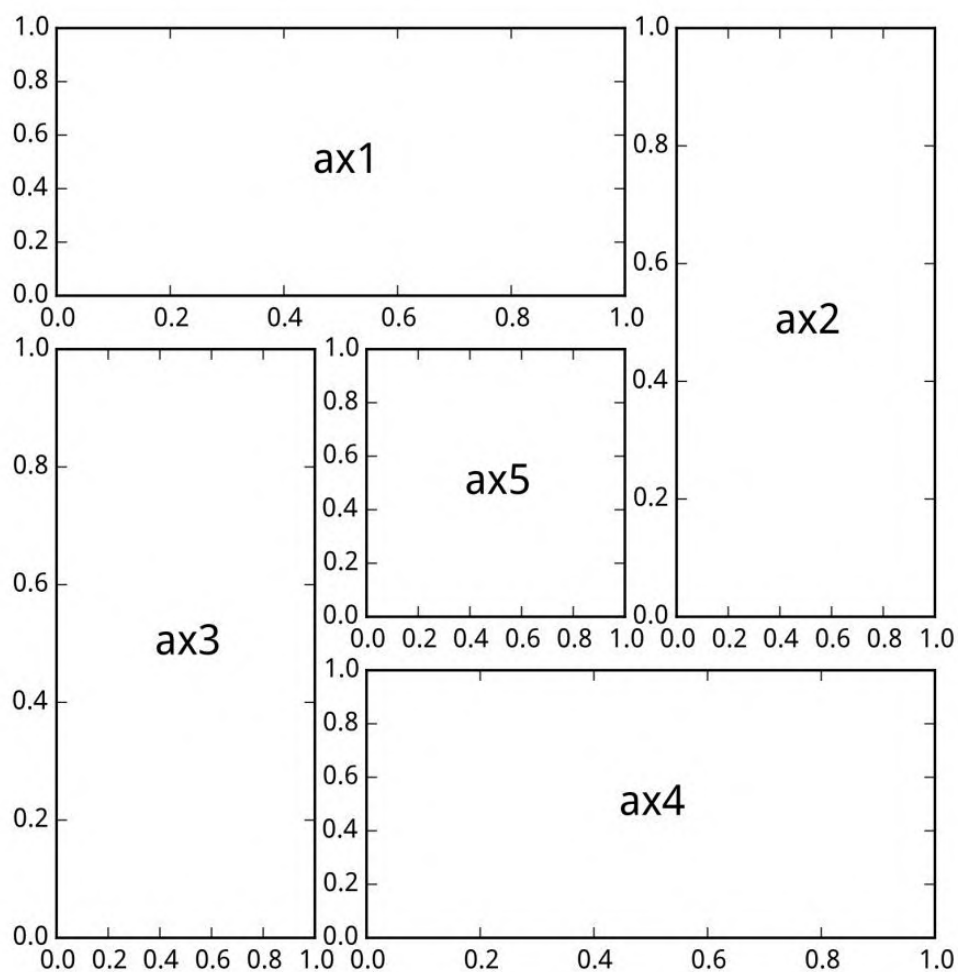


图4-5 使用`subplot2grid()`创建表格布局

```
fig = plt.figure(figsize=(6, 6))  
ax1 = plt.subplot2grid((3, 3), (0, 0), colspan=2)  
ax2 = plt.subplot2grid((3, 3), (0, 2), rowspan=2)
```

```
ax3 = plt.subplot2grid((3, 3), (1, 0), rowspan=2)
ax4 = plt.subplot2grid((3, 3), (2, 1), colspan=2)
ax5 = plt.subplot2grid((3, 3), (1, 1))
```

4.1.5 配置文件

绘制一幅图需要对许多对象的属性进行配置，例如颜色、字体、线型等。在前面的绘图程序中，并没有逐一对这些属性进行配置，而是直接采用matplotlib的默认配置。matplotlib将这些默认配置保存在一个名为matplotlibrc的配置文件中，通过修改配置文件，可以修改图表的默认样式。

在matplotlib中可以使用多个matplotlibrc配置文件，它们的搜索顺序如下：顺序靠前的配置文件将会被优先采用。

- 当前路径：程序的当前路径。
- 用户配置路径：通常在用户文件夹的.matplotlib目录下，可以通过环境变量MATPLOTLIBRC修改它的位置。
- 系统配置路径：保存在matplotlib的安装目录下的mpl-data中。

通过下面的语句可以获取用户配置路径：

```
from os import path
path.abspath(matplotlib.get_configdir())

u'C:\\Users\\RY\\Dropbox\\scipybook2\\settings\\.matplotlib'
```

通过下面的语句可以获得目前使用的配置文件的路径：

```
path.abspath(matplotlib.matplotlib_fname())  
  
u'C:\\Users\\RY\\Dropbox\\scipybook2\\settings\\.matplotlib\\matplotlibrc'
```

如果使用文本编辑器打开此配置文件，就会发现它实际上是一个字典。为了对众多的配置进行区分，字典的键根据配置的种类，用“.”分为多段。配置文件的读入可以使用`rc_params()`，它返回一个配置字典：

```
print(matplotlib.rc_params())  
agg.path.chunksize: 0  
animation.avconv_args: []  
animation.avconv_path: avconv  
animation.bitrate: -1  
...
```

在`matplotlib`模块载入时会调用`rc_params()`，并把得到的配置字典保存到`rcParams`变量中：

```
print(matplotlib.rcParams)  
agg.path.chunksize: 0  
animation.avconv_args: []  
animation.avconv_path: avconv  
animation.bitrate: -1  
...
```

matplotlib将使用rcParams字典中的配置进行绘图。用户可以直接修改此字典中的配置，所做的改变会反映到此后创建的绘图元素。例如下面的脚本所绘制的折线将带有圆形的点标识符：

```
matplotlib.rcParams["lines.marker"] = "o"
plt.plot([1,2,3,2])
```

为了方便对配置字典进行设置，可以使用rc()。下面的例子同时配置点标识符、线宽和颜色：

```
matplotlib.rc("lines", marker="x", linewidth=2,
color="red")
```

如果希望恢复到matplotlib载入时从配置文件读入的默认配置，可以调用rcdefaults()：

```
matplotlib.rcdefaults()
```

如果手工修改了配置文件，希望重新从配置文件载入最新的配置，可以调用：

```
matplotlib.rcParams.update( matplotlib.rc_params() )
```



通过pyplot模块也可以使用rcParams、rc和rcdefaults。

matplotlib.style模块提供绘图样式切换功能，所有可选样式可以通过available获得：

```
from matplotlib import style
print style.available
      [u'dark_background', u'bmh', u'grayscale', u'ggplot',
u'fivethirtyeight']
```

调用`use()`函数即可切换样式，例如下面使用`ggplot`样式绘图，效果如图4-6所示。

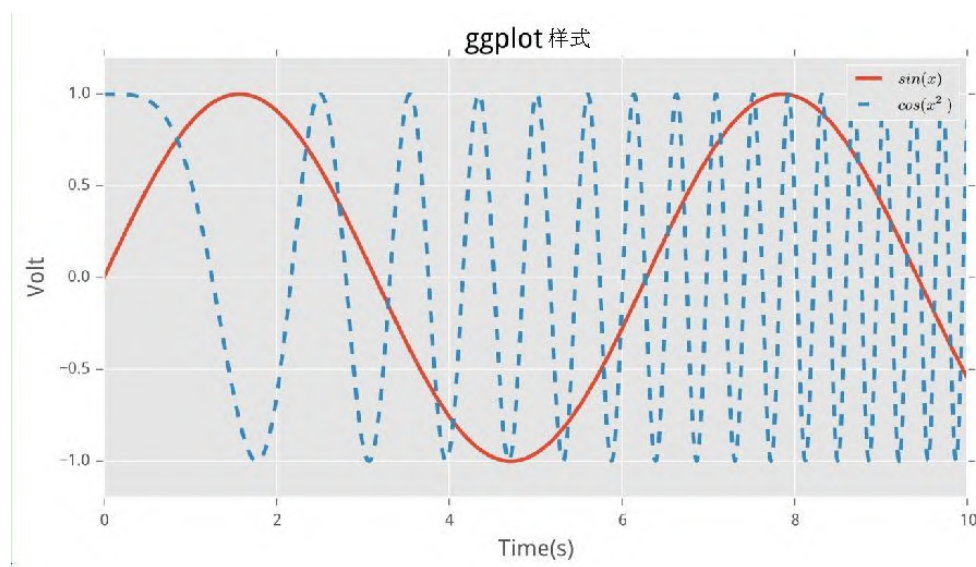


图4-6 使用`ggplot`样式绘图

```
style.use("ggplot")
```

4.1.6 在图表中显示中文

`matplotlib`的默认配置文件中所使用的字体无法正确显示中文，可以通过下面几种方法设置中文字体：

- 在程序中直接指定字体。

- 在程序开头修改配置字典rcParams。
- 修改配置文件。

在matplotlib中可以通过字体名指定字体，而每个字体名都与一个字体文件相对应。通过下面的程序可以获得所有可用字体的列表：

```
from matplotlib.font_manager import fontManager
fontManager.ttflist[:6]

[<Font 'cmss10' (cmss10.ttf) normal normal 400 normal>,
 <Font 'cmb10' (cmb10.ttf) normal normal 400 normal>,
 <Font 'cmex10' (cmex10.ttf) normal normal 400 normal>,
  <Font 'STIXSizeFourSym' (STIXSizFourSymBol.ttf) normal
normal 700 normal>,
  <Font 'Bitstream Vera Serif' (VeraSeBd.ttf) normal normal
700 normal>,
  <Font 'Bitstream Vera Sans' (VeraIt.ttf) oblique normal
400 normal>]
```

ttflist是matplotlib的系统字体列表。其中每个元素都是表示字体的Font对象，下面的程序显示了第一个字体文件的全路径和字体名，由路径可知它是matplotlib自带的字体：

```
print fontManager.ttflist[0].name
print fontManager.ttflist[0].fname
cmss10

C:\WinPython-32bit-2.7.9.2\python-2.7.9\lib\site-
packages\matplotlib\mpl-data\fonts\ttf\cmss10.ttf
```

下面的程序使用字体列表中的字体显示中文文字，效果图4-7所示。

中文字体	中文字体	中文字体	中文字体
LiSu	STXihei	FZShuTi	DengXian
中文字体	中文字体	????	????
Yu Gothic	YouYuan	Linux Biolinum G	Gabriola
中文字体	中文字体	中文字体	????
WenQuanYi Micro Hei	FangSong	Yu Gothic	Linux Libertine G
中文字体	中文字体	中文字体	中文字体
STCaiyun	Microsoft MHei	STXingkai	Arial Unicode MS
中文字体	????	????	????
STLiti	Linux Biolinum G	Linux Libertine G	Linux Libertine G
中文字体	????	中文字体	中文字体
WenQuanYi Micro Hei	Linux Biolinum G	STZhongsong	STCaiyun
中文字体	中文字体	????	????
Yu Gothic	FZYaoTi	Linux Libertine G	Linux Libertine G
中文字体	????	中文字体	中文字体
Microsoft MHei	Linux Libertine Display G	FZShuTi	DengXian
中文字体	????	中文字体	????
STZhongsong	Linux Biolinum G	STXinwei	Gabriola
????	中文字体	中文字体	????
Linux Libertine G	Microsoft MHei	Microsoft JhengHei	Linux Biolinum G
????	????	????	????
Linux Libertine G	Linux Biolinum G	Linux Libertine G	SimSun-ExtB
中文字体	????	????	????
STHupo	Linux Libertine G	Linux Libertine G	Malgun Gothic
中文字体	中文字体	中文字体	中文字体
STSong	Yu Gothic	SimHei	STHupo
中文字体	Yahei Mono	中文字体	中文字体
STSong	中文字体	中文字体	STFangsong
中文字体	STKaiti	Arial Unicode MS	中文字体
Microsoft JhengHei	中文字体	中文字体	LiSu
中文字体	中文字体	STLiti	中文字体
STXihei	STFangsong	????	中文字体
中文字体	中文字体	Linux Libertine G	Microsoft YaHei
Microsoft MHei	STKaiti	中文字体	中文字体
中文字体	????	DFKai-SB	YouYuan
Microsoft YaHei	Linux Libertine Display G	中文字体	中文字体
中文字体	????	STXinwei	FZYaoTi
KaiTi	Malgun Gothic	????	中文字体
		Linux Libertine G	STXingkai

图4-7 显示系统中所有的中文字体名



scipy2/matplotlib/chinese_fonts.py：显示系统中所有文件大于1MB的TTF字体，请读者使用该程序查询计算机中可使用的中文字体名。

```
import os

from os import path
```



```

fig = plt.figure(figsize=(8, 7))
ax = fig.add_subplot(111)
plt.subplots_adjust(0, 0, 1, 1, 0, 0)
plt.xticks([])
plt.yticks([])
x, y = 0.05, 0.05

fonts = [font.name for font in fontManager.ttflist if
          path.exists(font.fname) and
os.stat(font.fname).st_size>1e6] ❶

font = set(fonts)

dy = (1.0 - y) / (len(fonts) // 4 + (len(fonts)%4 != 0))

for font in fonts:
    t = ax.text(x, y + dy / 2, u"中文字体",
                {'fontname':font, 'fontsize':14},
transform=ax.transAxes) ❷
    ax.text(x, y, font, {'fontsize':12},
transform=ax.transAxes)

    x += 0.25

    if x >= 1.0:
        y += dy
        x = 0.05

plt.show()

```

❶利用os模块中的stat()获取字体文件的大小，并保留字体列表中所有大于1MB的字体文件。由于中文字体文件通常都很大，因此使用

这种方法可以粗略地找出所有的中文字体文件。

❷调用子图对象的`text()`在其中添加文字，注意文字必须是Unicode字符串。通过一个描述字体的字典指定文字的字体：'fontname'键对应的值就是字体名。

由于matplotlib只搜索TTF字体文件，因此无法通过上述方法使用系统中安装的许多复合TTC字体文件。可以直接创建使用字体文件的FontProperties对象，并使用此对象指定图表中的各种文字的字体。下面是一个例子：

```
from matplotlib.font_manager import FontProperties
font = FontProperties(fname=r"c:\windows\fonts\simsun.ttc",
size=14) ❶

t = np.linspace(0, 10, 1000)
y = np.sin(t)
plt.close("all")
plt.plot(t, y)
plt.xlabel(u"时间", fontproperties=font) ❷
plt.ylabel(u"振幅", fontproperties=font)
plt.title(u"正弦波", fontproperties=font)
plt.show()
```

❶创建一个描述字体属性的FontProperties对象，并设置其fname属性为字体文件的绝对路径。❷通过fontproperties参数将FontProperties对象传递给显示文字的函数。

还可以通过字体工具将TTC字体文件分解为多个TTF字体文件，并将其复制到系统的字体文件夹中。为了缩短启动时间，matplotlib不会每次启动时都重新扫描所有的字体文件并创建字体列表，因此在复制完字体文件之后，需要运行下面的语句重新创建字体列表：

```
from matplotlib.font_manager import _rebuild
_rebuild()
```

还可以直接修改配置字典，设置默认字体，这样就不需要在每次绘制文字时设置字体了。例如：

```
plt.rcParams["font.family"] = "SimHei"
plt.plot([1,2,3])
plt.xlabel(0.5 ,0.5, u"中文字体")
```

或者修改上节介绍的配置文件，修改其中的font.family配置为SimHei，注意SimHei是字体名，请读者运行前面的代码来查看系统中所有可用的中文字体名。

4.2 Artist对象



与 本 节 内 容 对 应 的 Notebook 为 : 04-matplotlib/matplotlib-200-artists.ipynb

matplotlib是一套面向对象的绘图库，它有三个层次：

- `backend_bases.FigureCanvas`: 绘图用的画布。
- `backend_bases.Renderer`: 知道如何在`FigureCanvas`对象上绘图。
- `artist.Artist`: 知道如何使用`Renderer`在`FigureCanvas`对象上绘图。

`FigureCanvas`和`Renderer`需要处理底层的绘图操作，例如在`wxPython`界面库所生成的界面上绘图，或者使用`PostScript`在PDF文件中绘图。`Artist`对象则处理所有的高层结构，例如处理图表、文字和曲线等各种绘图元素的绘制和布局。通常我们只和`Artist`对象打交道，而不需要关心底层是如何实现绘图细节的。

`Artist`对象分为简单类型和容器类型两种。简单类型的`Artist`对象是标准的绘图元件，例如`Line2D`、`Rectangle`、`Text`、`AxesImage`等。而容器类型则可以包含多个`Artist`对象，使它们组织成一个整体，例如`Axis`、`Axes`、`Figure`等。

直接创建`Artist`对象进行绘图的流程如下：

- (1) 创建`Figure`对象。
- (2) 为`Figure`对象创建一个或多个`Axes`对象。
- (3) 调用`Axes`对象的方法来创建各种简单类型的`Artist`对象。

在下面的程序中，首先调用`figure()`创建`Figure`对象，`figure()`是一个辅助函数，帮助我们创建`Figure`对象，它会进行许多初始化操作，因此不建议直接使用`Figure()`创建。然后调用`Figure`对象的`add_axes()`在其中创建一个`Axes`对象，`add_axes()`的参数是一个形如

[left, bottom, width, height]的列表，这些数值分别指定所创建的Axes对象在Figure对象中的位置和大小，各个值的取值范围都在0到1之间：

```
from matplotlib import pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0.15, 0.1, 0.7, 0.3])
```

然后调用Axes对象的plot()来绘制曲线，并且返回表示此曲线的Line2D对象。

```
line = ax.plot([1, 2, 3], [1, 2, 1])[0] # 返回的是只有一个元素的列表
print line is ax.lines[0]
True
```

Axes对象的lines属性是一个包含所有曲线的列表，如果继续运行ax.plot()，所创建的Line2D对象都会添加到此列表中。如果想删除某条曲线，直接从此列表中删除即可。

Axes对象还包括许多其他的Artists对象，例如可以通过set_xlabel()设置其X轴上的标题：

```
ax.set_xlabel("time")
```

如果查看set_xlabel()的源代码，就会发现它是通过下面的语句实现的：

```
self.xaxis.set_label_text(xlabel)
```

如果一直跟踪下去，就会发现Axes对象的xaxis属性是一个XAxis对象，其label属性是一个Text对象，而Text对象的_text属性为我们设置的值：

```
print ax.xaxis
print ax.xaxis.label
print ax.xaxis.label._text
XAxis(72.000000,24.000000)
Text(0.5,2.2,u'time')
time
```

Axes、XAxis和Text类都从Artist继承，也可以调用它们的get_*()以获得相应的属性值：

```
ax.get_xaxis().get_label().get_text()
u'time'
```

4.2.1 Artist的属性

通过前面的介绍我们已经知道，图表中的每个绘图元素都用一个Artist对象表示，而每个Artist对象都有许多属性控制其显示效果。例如Figure对象和Axes对象都有patch属性作为其背景，它是一个Rectangle对象。通过设置它的属性可以修改图表的背景色或透明度，下面的例子将图表的背景色设置为绿色：

```
fig = plt.figure()
fig.patch.set_color("g") # 设置背景色为绿色
```

注意当代码作为单独程序运行时，调用set_color()设置好背景色之后，并不会立即在界面上显示出来，还需要调用fig.canvas.draw()才能更新界面显示。

表4-1是所有Artist对象都拥有的一些属性。

表4-1 所有Artist对象都拥有的一些属性

属性	说明
alpha	透明度，值在0到1之间，0为完全透明，1为完全不透明
animated	布尔值，在绘制动画效果时使用
axes	拥有此Artist对象的Axes对象，可能为None
clip_box	对象的裁剪框
clip_on	是否裁剪
clip_path	裁剪的路径
contains	判断指定点是否在对象之上的函数
figure	拥有此Artist对象的Figure对象，可能为None
label	文本标签
picker	控制Artist对象选取
transform	控制偏移、旋转、缩放等坐标变换
visible	控制是否可见

Artist对象的所有属性都可以通过相应的`get_*`()和`set_*`()方法进行读写, 例如下面的语句将新绘制的曲线对象的`alpha`属性设置0.5, 使它变成半透明:

```
line = plt.plot([1, 2, 3, 2, 1], lw=4)[0]
line.set_alpha(0.5)
```

可以使用`set()`一次设置多个属性:

```
line.set(alpha=0.5, zorder=2)
```

使用前面介绍的`getp()`可以方便地输出Artist对象的所有属性名以及与之对应的值:

```
plt.getp(fig.patch)
aa = False
agg_filter = None
alpha = None
animated = False
...
```

4.2.2 Figure容器

现在我们知道如何观察和修改Artist对象的属性, 接下来要解决的问题是如何找到指定的Artist对象。前面介绍过Artist对象有容器类型和简单类型两种, 这一节让我们详细看看容器类型。

在构成图表的各种Artist对象中，最上层的Artist对象是Figure，它包含组成图表的所有元素。当调用add_subplot()或add_axes()方法往图表中添加子图时，这些子图都将添加到axes属性列表中，同时这两个方法也返回新创建的Axes对象。注意add_subplot()和add_axes()所返回对象的类型有所不同，分别为AxesSubplot和Axes，AxesSubplot是Axes的派生类。

```
fig = plt.figure()
ax1 = fig.add_subplot(211)
ax2 = fig.add_axes([0.1, 0.1, 0.7, 0.3])
print ax1 in fig.axes and ax2 in fig.axes
True
```

为了支持gca()等函数，Figure对象内部保存有当前轴的信息，因此不建议直接对axes属性进行列表操作，而应该使用add_subplot()、add_axes()、delaxes()等方法进行子图的添加和删除操作。但是使用for循环对axes属性中的每个元素进行操作是没有问题的，下面的语句打开所有子图的栅格显示。

```
for ax in fig.axes:
    ax.grid(True)
```

Figure对象可以拥有自己的文字、线条以及图像等简单类型的Artist对象。默认的坐标系以像素点为单位，但是可以通过设置Artist对象的transform属性修改其所使用的坐标系。例如Figure对象的坐标系是以图表的左下角为坐标原点(0,0)，右上角的坐标为(1,1)，关于坐标变换在后面的章节还会进行详细介绍。下面的程序创建一个Figure对象，并在其中添加两条直线：

```

from matplotlib.lines import Line2D

fig = plt.figure()

line1 = Line2D(
    [0, 1], [0, 1], transform=fig.transFigure, figure=fig,
color="r")

line2 = Line2D(
    [0, 1], [1, 0], transform=fig.transFigure, figure=fig,
color="g")

fig.lines.extend([line1, line2])

```

为了让所创建的Line2D对象使用Figure对象的坐标系，我们将Figure对象的transFigure属性赋给Line2D对象的transform属性。为了让Line2D对象知道它是在Figure对象中，还设置其figure属性为fig。最后还需要将这两个Line2D对象添加到Figure对象的lines属性列表中。

表4-2列出了Figure对象中包含其他Artist对象的属性：

表4-2 包含其他Artist对象的Figure对象属性

属性	说明
axes	Axes对象列表
patch	作为背景的Rectangle对象
images	FigureImage对象列表
legends	Legend对象列表
lines	Line2D对象列表

patches	Patch对象列表
texts	Text对象列表，用于显示文字

4.2.3 Axes容器

Axes容器(子图)是整个matplotlib的核心，它包含了组成图表的众多Artist对象，并且有许多方法函数帮助我们创建和修改这些对象。和Figure容器一样，它有一个patch属性作为背景，当它是笛卡尔坐标时，patch属性是一个Rectangle对象；而当它是极坐标时，patch属性则是Circle对象。例如下面的语句将Axes对象的背景色设置为绿色：

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.patch.set_facecolor("green")
```

当调用Axes对象的绘图方法plot()时，它将创建一组Line2D对象，并将它们添加进Axes对象的lines属性中，最后返回包含所有创建的Line2D对象的列表。plot()的所有关键字参数都将传递给这些Line2D对象以设置它们的属性：

```
x, y = np.random.rand(2, 100)
line = ax.plot(x, y, "--", color="blue", linewidth=2)[0]
line is ax.lines[0]
True
```

注意`plot()`返回的是一个`Line2D`对象列表，因为可以传递多组X-Y轴的数据给`plot()`，同时绘制多条曲线。

与`plot()`类似，绘制柱状图的函数`bar()`和绘制直方统计图的函数`hist()`将创建一个`Patch`对象的列表，每个元素实际上都是从`Patch`类派生的`Rectangle`对象，所创建的`Patch`对象都被添加进了`Axes`对象的`patches`属性中：

```
fig, ax = plt.subplots()
    n, bins, rects = ax.hist(np.random.randn(1000), 50,
facecolor="blue")

rects[0] is ax.patches[0]

True
```

一般我们不会直接对`lines`或`patches`属性进行操作，而是调用`add_line()`或`add_patch()`等方法，这些方法帮助我们完成许多属性的设置工作。下面首先创建`Axes`对象`ax`和`Rectangle`对象`rect`：

```
fig, ax = plt.subplots()

rect = plt.Rectangle((1,1), width=5, height=12)
```

然后通过`add_patch()`将`rect`添加进`ax`中：

```
ax.add_patch(rect) # 将rect添加进ax

rect.get_axes() is ax

True
```

接下来，为了完整显示`rect`，调用`ax`的`autoscale_view()`方法让它自动调节X-Y轴的显示范围：

```
print ax.get_xlim() # ax的x轴范围为0到1，无法显示完整的rect
print ax.dataLim._get_bounds() # 数据的范围和rect的大小一致
ax.autoscale_view() # 自动调整坐标轴的范围
print ax.get_xlim() # 于是x轴可以完整显示rect
(0.0, 1.0)
(1.0, 1.0, 5.0, 12.0)
(1.0, 6.0)
```

表4-3列出了Axes对象中可以包含其他Artist对象的属性：

表4-3 包含其他Artist对象的Axes对象属性

属性	说明
artists	Artist对象列表
patch	作为Axes背景的Patch对象，可以是Rectangle或Circle
collections	Collection对象列表
images	AxesImage对象列表
legends	Legend对象列表
lines	Line2D对象列表
patches	Patch对象列表
texts	Text对象列表
xaxis	XAxis对象

yaxis	YAxis对象
-------	---------

表4-4列出了Axes对象的各种创建其他Artist对象的方法：

表4-4 Axes对象提供的创建其他Artist对象的方法

Axes的方法	所创建的对象	添加进的列表
annotate	Annotate	texts
bars	Rectangle	patches
errorbar	Line2D、Rectangle	lines, patches
fill	Polygon	patches
hist	Rectangle	patches
imshow	AxesImage	images
legend	Legend	legends
plot	Line2D	lines
scatter	PolygonCollection	collections
text	Text	texts

例如下面的程序调用scatter() 绘制散列图，它返回的是一个PathCollection对象，该对象被添加进ax.collections列表：

```
fig, ax = plt.subplots()
t = ax.scatter(np.random.rand(20), np.random.rand(20))
```

```
print t, t in ax.collections
      <matplotlib.collections.PathCollection object at
0x082339B0> True
```

4.2.4 Axis容器

Axis容器包括坐标轴上的刻度线、刻度文本、坐标网格以及坐标轴标题等内容。刻度包括主刻度和副刻度，分别通过`get_major_ticks()`和`get_minor_ticks()`方法获得。每个刻度线都是一个XTick或YTick对象，它包括实际的刻度线和刻度文本。为了方便访问刻度线和文本，Axis对象提供了`get_ticklabels()`和`get_ticklines()`方法来直接获得刻度线和刻度文本。

下面先创建一个子图并获得其X轴对象axis:

```
fig, ax = plt.subplots()
axis = ax.xaxis
```

下面获得axis对象的刻度位置的列表:

```
axis.get_ticklocs()
array([ 0. ,  0.2,  0.4,  0.6,  0.8,  1. ])
```

下面获得axis对象的刻度标签以及标签中的文字:

```
print axis.get_ticklabels() # 获得刻度标签的列表
print [x.get_text() for x in axis.get_ticklabels()] # 获得刻
度的文本字符串
```

```
<a list of 6 Text major ticklabel objects>  
[u'0.0', u'0.2', u'0.4', u'0.6', u'0.8', u'1.0']
```

下面获得轴上表示主刻度线的列表，可以看到X轴上共有12条刻度线，它们是子图的上下两个X轴上的所有刻度线：

```
axis.get_ticklines()  
<a list of 12 Line2D ticklines objects>
```

而由于图中没有副刻度线，因此副刻度线列表的长度为0：

```
axis.get_ticklines(minor=True) # 获得副刻度线列表  
<a list of 0 Line2D ticklines objects>
```

获得刻度线或刻度标签之后，可以设置其各种属性，下面设置刻度线为绿色粗线，文本为红色并且旋转45°。最终结果如图4-8所示：

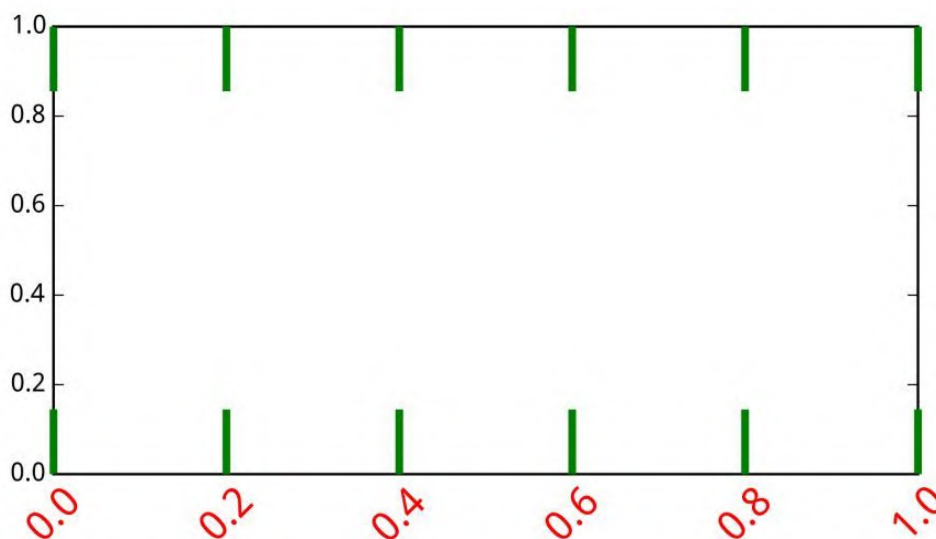


图4-8 配置X轴的刻度线和刻度文本的样式


```
for label in axis.get_ticklabels():
    label.set_color("red")
    label.set_rotation(45)
    label.set_fontsize(16)

for line in axis.get_ticklines():
    line.set_color("green")
    line.set_markersize(25)
    line.set_markedgewidth(3)

fig
```

这个例子只是为了演示Artist对象的各种属性，实际上使用pyplot模块中的xticks()能够更快地完成X轴上的刻度文本的配置。不过，xticks()只能设置刻度文本的属性，不能设置刻度线的属性。感兴趣的读者可以在IPython中输入plt.xticks??来查看源代码。

```
plt.xticks(fontsize=16, color="red", rotation=45)
```

在前面的例子中，副刻度线列表为空，这是因为用于计算副刻度位置的对象默认为NullLocator，它不产生任何刻度线。而计算主刻度位置的对象为AutoLocator，它会根据当前的缩放等配置自动计算刻度的位置：

```
print axis.get_minor_locator() # 计算副刻度位置的对象
print axis.get_major_locator() # 计算主刻度位置的对象
<matplotlib.ticker.NullLocator object at 0x08364F50>
<matplotlib.ticker.AutoLocator object at 0x084285D0>
```

matplotlib提供了多种配置刻度线位置的Locator类和控制刻度文本显示的Formatter类。下面的程序设置X轴的主刻度为 $\pi/4$ ，副刻度为 $\pi/20$ ，并且主刻度上的文本用数学符号显示 π 。程序的输出如图4-9所示。

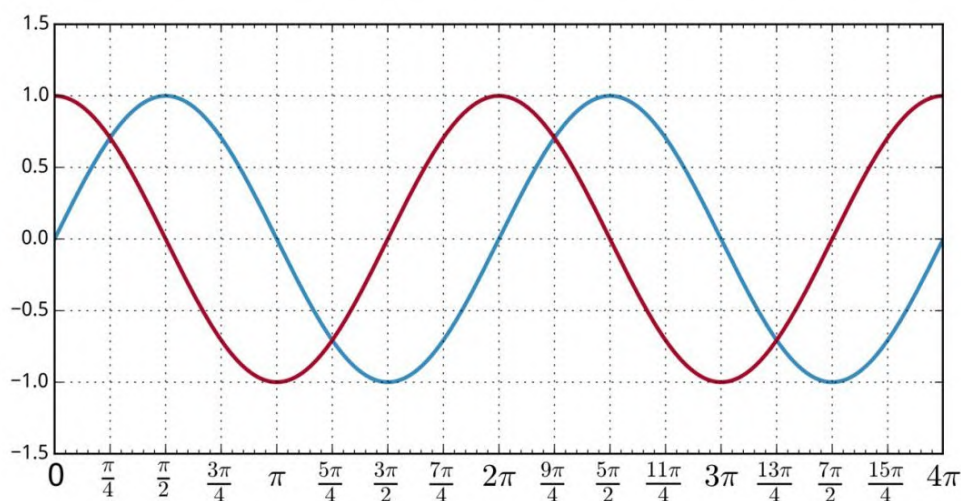


图4-9 配置X轴的刻度线的位置和文本，并开启副刻度线

```
from fractions import Fraction

from matplotlib.ticker import MultipleLocator,
FuncFormatter ❶

x = np.arange(0, 4*np.pi, 0.01)
fig, ax = plt.subplots(figsize=(8,4))
plt.plot(x, np.sin(x), x, np.cos(x))

def pi_formatter(x, pos): ❷
    frac = Fraction(int(np.round(x / (np.pi/4))), 4)
    d, n = frac.denominator, frac.numerator
    if frac == 0:
        return "0"
```

```

        elif frac == 1:
            return "$\pi$"

        elif d == 1:
            return r"${%d} \pi$" % n

        elif n == 1:
            return r"$\frac{\pi}{{%d}}$" % d

        return r"$\frac{%d \pi}{{%d}}$" % (n, d)

# 设置两个坐标轴的范围
plt.ylim(-1.5,1.5)
plt.xlim(0, np.max(x))

# 设置图的底边距
plt.subplots_adjust(bottom = 0.15)

plt.grid() #开启网格

# 主刻度为pi/4
ax.xaxis.set_major_locator( MultipleLocator(np.pi/4) ) ❸

# 主刻度文本用pi_formatter函数计算
ax.xaxis.set_major_formatter( FuncFormatter( pi_formatter )
) ❹

# 副刻度为pi/20
ax.xaxis.set_minor_locator( MultipleLocator(np.pi/20) ) ❺

```

```
# 设置刻度文本的大小

for tick in ax.xaxis.get_major_ticks():
    tick.label1.set_fontsize(16)
```

①刻度定位和文本格式化相关的类都在`matplotlib.ticker`模块中定义，程序从中载入了如下两个类：

- `MultipleLocator`：③⑤以指定值的整数倍为刻度放置主副刻度线。
- `FuncFormatter`：④使用指定的函数计算刻度文本，它会将刻度值和刻度的序号作为参数传递给计算刻度文本的函数。②程序中通过`pi_formatter()`计算出与刻度值对应的刻度文本。

4.2.5 Artist对象的关系

为了方便读者理解图表中各种Artist对象之间的关系，本书提供了一个输出Artist对象关系图的小程序。



`scipy2.common.GraphvizMatplotlib`：将`matplotlib`的对象关系输出成如图4-10所示的关系图。

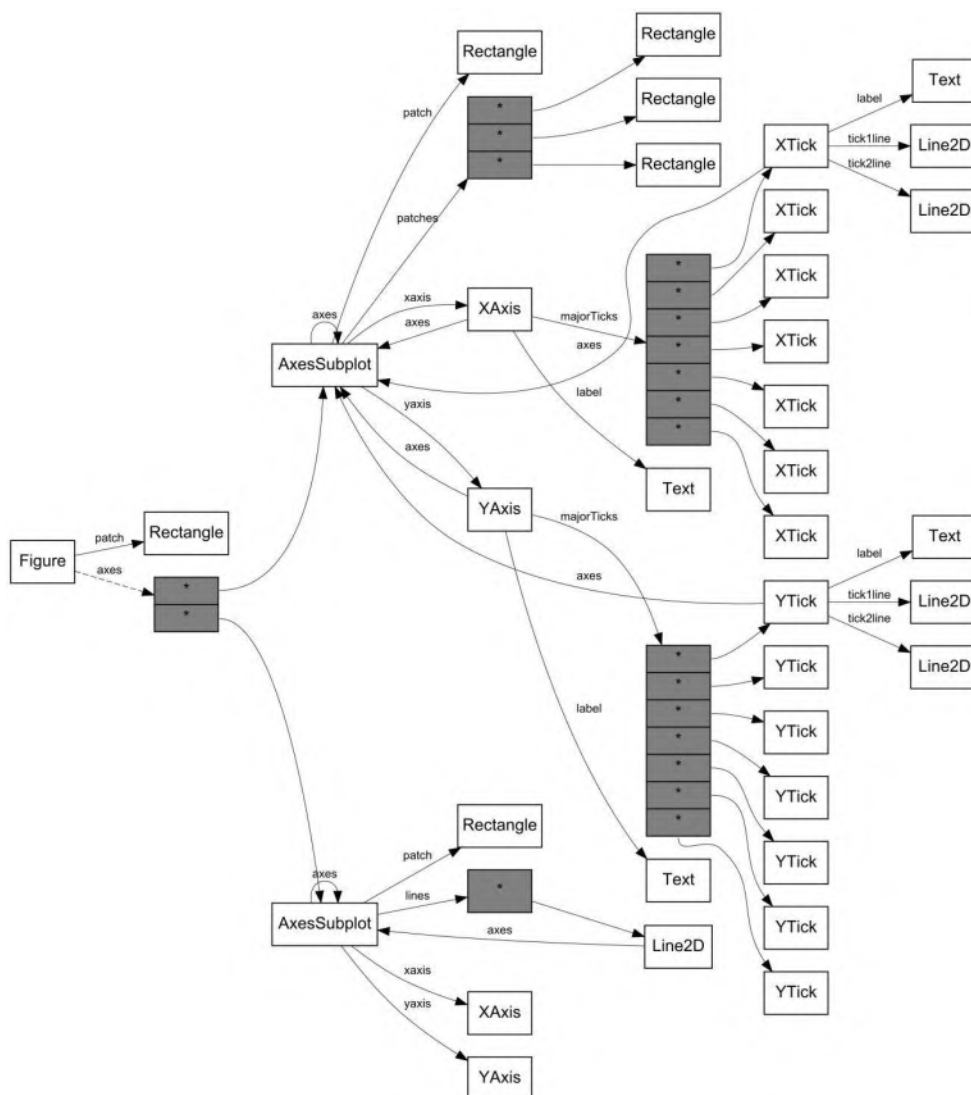


图4-10 使用GraphvizMatplotlib生成图表对象中各个Artist对象之间的关系图

为了生成关系图，读者可以从Graphviz的官方网站下载Graphviz软件包，或者使用Graphviz的在线编辑器。下面看一个例子：

```
fig = plt.figure()
plt.subplot(211)
plt.bar([1, 2, 3], [1, 2, 3])
plt.subplot(212)
plt.plot([1, 2, 3])
```

下面调用 `GraphvizMatplotlib.graphviz()`，将 `fig` 内部各个 `Artist` 对象的关系输出为 `dot` 代码，并使用 `%dot` 魔法命令将其转换为 SVG 图像，显示在 Notebook 中。结果为图4-10所示的关系图：

```
from scipy2.common import GraphvizMatplotlib
%dot GraphvizMatplotlib.graphviz(fig)
```

图4-10中以灰色填充矩形表示列表，其他的矩形表示各种 `Artist` 对象。`Artist` 对象之间的关系使用带箭头的细线表示，细线旁边的文本为属性名。

例如从 `Figure` 矩形到 `Rectangle` 矩形的箭头表示 `Figure` 对象的 `patch` 属性是一个 `Rectangle` 对象。而 `Figure` 对象的 `axes` 属性是一个有两个元素的列表，每个元素都是一个 `AxesSubplot` 对象。请读者仔细观察图4-10，并在 IPython 中输入相应的语句，确认各个 `Artist` 对象之间的关系。

4.3 坐标变换和注释



与本节内容对应的 Notebook 为：04-matplotlib/matplotlib-300-transform.ipynb。

一幅图表中涉及多种坐标系以及坐标变换，理解各种坐标系的含义并掌握其用法才能随心所欲地使用 `matplotlib` 绘制出理想效果的图

表。本节以图表中的文字、箭头和标注为例介绍各种坐标系及其变换。

```
def func1(x): ❶
    return 0.6*x + 0.3

def func2(x): ❶
    return 0.4*x*x + 0.1*x + 0.2

def find_curve_intersects(x, y1, y2):
    d = y1 - y2
    idx = np.where(d[:-1]*d[1:]<=0)[0]
    x1, x2 = x[idx], x[idx+1]
    d1, d2 = d[idx], d[idx+1]
    return -d1*(x2-x1)/(d2-d1) + x1

x = np.linspace(-3,3,100) ❷
f1 = func1(x)
f2 = func2(x)
fig, ax = plt.subplots(figsize=(8,4))
ax.plot(x, f1)
ax.plot(x, f2)

x1, x2 = find_curve_intersects(x, f1, f2) ❸
ax.plot(x1, func1(x1), "o")
ax.plot(x2, func1(x2), "o")
```

```
ax.fill_between(x, f1, f2, where=f1>f2, facecolor="green",  
alpha=0.5) ④
```

```
from matplotlib import transforms  
trans = transforms.blended_transform_factory(ax.transData,  
ax.transAxes)
```

```
ax.fill_between([x1, x2], 0, 1, transform=trans, alpha=0.1)  
⑤
```

```
a = ax.text(0.05, 0.95, u"直线和二次曲线的交点", ⑥  
transform=ax.transAxes,  
verticalalignment = "top",  
fontsize = 18,  
bbox={"facecolor":"red", "alpha":0.4, "pad":10}  
)
```

```
arrow = {"arrowstyle":"fancy,tail_width=0.6",  
"facecolor":"gray",  
"connectionstyle":"arc3,rad=-0.3"}
```

```
ax.annotate(u"交点", ⑦  
xy=(x1, func1(x1)), xycoords="data",  
xytext=(0.05, 0.5), textcoords="axes fraction",  
arrowprops = arrow)
```

```
ax.annotate(u"交点", ⑦  
xy=(x2, func1(x2)), xycoords="data",
```



```

xytext=(0.05, 0.5), textcoords="axes fraction",
arrowprops = arrow)

xm = (x1+x2)/2
ym = (func1(xm) - func2(xm))/2+func2(xm)
o = ax.annotate(u"直线大于曲线区域", ⑦
               xy=(xm, ym), xycoords="data",
               xytext=(30, -30), textcoords="offset points",
               bbox={"boxstyle":"round", "facecolor":(1.0, 0.7, 0.7),
"edgecolor":"none"},
               fontsize=16,
               arrowprops={"arrowstyle":"->"}
)

```

程序的输出如图4-11所示。在图4-11中演示了下面列出的标注效果：

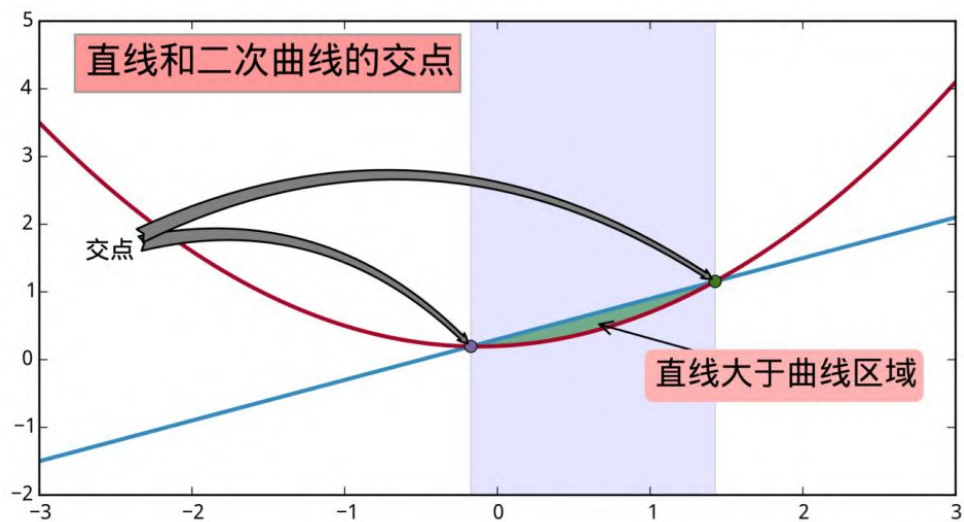


图4-11 为图表添加各种注释元素

- 用两个小圆点表示直线和曲线的两个交点。
- 对两个交点之间、位于直线和曲线之间的面积进行了填充。
- 使用一个高为整个子图高度、左右边位于两个交点的矩形表示两个交点之间的区间。
- 在图4-11的左上角放置了说明文字。
- 对两个交点和填充面积使用了带箭头的注释说明。

首先，❶定义了两个函数func1和func2，它们分别是计算一条直线和一条二次曲线的函数。❷然后计算这两个函数在区间(-3, 3)上的值，并且调用plot()绘制成曲线图。

❸为了标出两个交点，我们用find_curve_intersects()计算两条曲线f1和f2的交点对应的X轴坐标x1和x2。交点处的小圆点仍然使用plot()进行绘制，这时所传递的X-Y轴的数据为单一的数值，并且以'o'为样式进行绘图。

如何计算两条曲线的交点

当两条曲线的Y轴坐标值y1和y2使用相同的X轴坐标数组x计算时，很容易计算它们的交点。首先计算两条曲线在Y轴的差值 $d = y1 - y2$ ，然后找到符号相反的两个连续的差值的下标idx和idx + 1。计算直线 $(x[idx], d[idx]) - (x[idx+1], d[idx+1])$ 和X轴的交点就可得到两条曲线交点的X轴坐标xc。如果要计算交点的Y轴坐标，只需要调用np.interp(xc, x, y1)对曲线进行线性插值即可。

④接下来调用`fill_between()`绘制X轴上在两个交点之间、Y轴上在两条曲线之间的面积部分，并通过`facecolor`和`alpha`参数指定填充的颜色和透明度。`fill_between()`的调用参数如下：

```
fill_between(x, y1, y2=0, where=None)
```

其中，`x`参数是长度为N的数组，`y1`和`y2`参数是长度为N的数组或单个数值。当`y1`或`y2`为单个数值时，它们相当于一个长度为N、元素数值都相同的数组。`fill_between()`将填充Y轴在`y1`和`y2`之间的部分。如果`where`参数为`None`，就对数组`x`中的所有元素进行填充；如果`where`是一个布尔数组，则只填充其中`True`所对应的部分。程序中的数组`x`的取值范围为 $(-3, 3)$ ，由于设置了条件`where = f1 > f2`，因此只绘制直线在二次曲线之上的部分。

⑤绘制X轴上在两个交点之间的矩形区域；⑥用`text()`在图表中添加说明文字；⑦最后用`annotate()`为图表添加三个带箭头的注释。

为了真正理解程序的细节，首先需要了解`matplotlib`中坐标变换的工作原理。

4.3.1 4种坐标系

在`matplotlib`所绘制的一幅图表中，有4种坐标系：

- 数据坐标系：它是描述数据空间中位置的坐标系，例如对于图4-11，它的数据坐标系的范围为X轴在 $(-3, 3)$ 之间，Y轴在 $(-2, 5)$ 之间。

- 子图坐标系：描述子图中位置的坐标系，子图的左下角坐标为(0, 0)，右上角坐标为(1, 1)。
- 图表坐标系：一幅图表可以包含多个子图，并且子图周围都有一定的余白，因此还需要用图表坐标系描述图表显示区域中的某个点，图表的左下角坐标为(0, 0)，右上角坐标为(1, 1)。
- 窗口坐标系：它是绘图窗口中以像素为单位的坐标系。左下角坐标为(0, 0)，右上角坐标为(width, height)。其中的width和height分别是以像素为单位的绘图窗口的内宽和内高，不包括标题栏、工具条以及状态栏等部分。

Axes对象的transData属性是数据坐标变换对象，transAxes属性是子图坐标变换对象。Figure对象的transFigure属性是图表坐标变换对象。

通过上述坐标变换对象的transform()方法，可以将此坐标系下的坐标转换为窗口坐标系中的坐标。下面的程序计算数据坐标系中的坐标点(-3, -2)和(3, 5)在绘图窗口中的坐标：

```
print type(ax.transData)
ax.transData.transform([(-3,-2), (3,5)])
<class 'matplotlib.transforms.CompositeGenericTransform'>
array([[ 80.,   32.],
       [ 576.,  288.]])
```

下面的程序计算子图坐标系中的坐标点(0, 0)和(1, 1)在绘图窗口中的位置，得到的结果和上面的相同。即子图的左下角坐标(0, 0)

和数据坐标系中的坐标(-3, -2)在屏幕上是一个点。观察图4-11可以知道这显然是正确的。

```
ax.transAxes.transform([(0,0), (1,1)])  
array([[ 80.,  32.],  
       [ 576., 288.]])
```

最后计算图表坐标系中坐标点(0, 0)和(1, 1)在绘图窗口中的位置, 可以看出绘图区域的宽为640个像素, 高为320个像素:

```
fig.transFigure.transform([(0,0), (1,1)])  
array([[ 0.,  0.],  
       [640., 320.]])
```

通过坐标变换对象的`inverted()`方法, 可以获得它的逆变换对象。例如下面的程序计算绘图窗口中的坐标点(320, 160)在数据坐标系中的坐标, 结果为(-0.09677419, 1.5):

```
inv = ax.transData.inverted()  
print type(inv)  
inv.transform((320, 160))  
<class 'matplotlib.transforms.CompositeGenericTransform'>  
array([-0.09677419,  1.5      ])
```

请读者仔细观察程序所输出的图表, 子图的上下余白相同, 而左侧余白略大于右侧余白, 因此绘图区域的中心点(320, 160)并不是数据区域的中心点(0, 1.5)。

当调用`set_xlim()`修改子图所显示的X轴范围之后，它的数据坐标变换对象也同时发生了变化：

```
print ax.set_xlim(-3, 2) # 设置x轴的范围为-3到2
print ax.transData.transform((3, 5)) # 数据坐标变换对象已经发生了变化
(-3, 2)
[ 675.2  288. ]
```

下面回头看看图4-11中绘制矩形区间的程序：

```
from matplotlib import transforms
trans = transforms.blended_transform_factory(ax.transData,
ax.transAxes)
ax.fill_between([x1, x2], 0, 1, transform=trans, alpha=0.1)
```

矩形区间使用`fill_between()`绘制。由于所绘制矩形的左右两边要始终经过两个交点，因此矩形的X轴坐标必须使用数据坐标系中的坐标：`x1`和`x2`。而由于矩形的高度始终充满整个子图的高度，因此矩形的Y轴坐标必须是子图坐标系中的坐标：`0`和`1`。



使用`axvspan()`和`axhspan()`可以快速绘制垂直方向和水平方向上的区间。

程序中，使用`blended_transform_factory()`创建这种混合坐标系。它的两个参数都是坐标变换对象，它从第一个参数获得X轴的坐标

变换，从第二个参数获得Y轴的坐标变换。因此它所返回的坐标变换对象trans的X轴使用数据坐标系，而Y轴使用子图坐标系。程序中，将混合坐标变换对象trans传递给fill_between()的transform参数，这样所绘制的填充区域就能始终保持左右边通过两个交点，而上下边位于子图边框之上。

4.3.2 坐标变换的流水线

从一个坐标系变换到另一个坐标系，中间需要经过几个步骤。而且数据坐标系不一定是笛卡尔坐标系，它可能是极坐标系或对数坐标系。因此坐标系的变换并不是简单的二维仿射变换 (2D Affine Transformation)。让我们从最简单的图表坐标变换对象transFigure开始，介绍matplotlib的坐标变换是如何进行的。

通过本书提供的GraphvizMPLTransform可以将坐标变换对象显示为关系图，图4-12显示了fig.transFigure的内部结构。

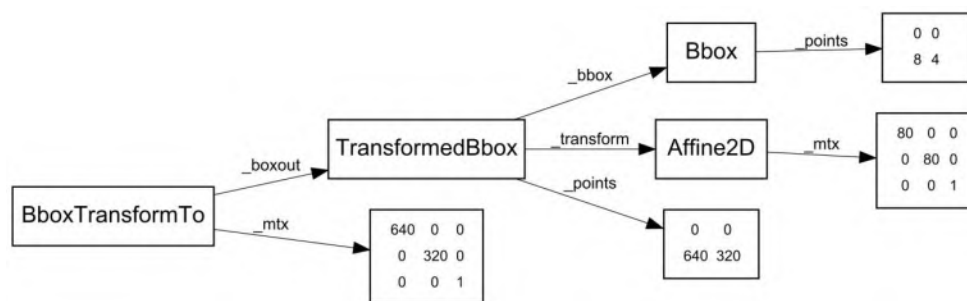


图4-12 图表坐标变换对象的内部结构

```

from scipy2.common import GraphvizMPLTransform

%dot GraphvizMPLTransform.graphviz(fig.transFigure)

```

这个坐标变换对象的内容有些复杂，它是一个BboxTransformTo对象，其中包含一个TransformedBbox对象，而TransformedBbox对象又包含一个Bbox对象和一个Affine2D对象：

- Bbox：定义一个矩形区域—— $[[x_0, y_0], [x_1, y_1]]$ 。在本例中，矩形的两个顶点坐标分别为(0, 0)和(8, 4)，它是窗口的英寸大小，通过figsize参数传递给figure()。
- Affine2D：二维仿射变换对象，它是一个矩阵，通过它和齐次向量相乘得到变换之后的坐标。由于矩阵中只有对角线上的值不为零，因此该仿射变换只进行缩放变换。它将坐标(x, y)变换为(80*x, 80*y)。

仿射变换

二维空间的仿射变换矩阵的大小为 3×3 ，为了进行仿射变换需要使用齐次坐标，即用三维向量(x, y, 1)表示二维平面上的点(x, y)。仿射变换就是仿射矩阵和向量的乘积。由于变换矩阵最下一行的数值始终是(0, 0, 1)，因此有时也将它写成 2×3 的矩阵形式。

- TransformedBbox：将矩形区域通过仿射变换之后得到一个新的矩形区域。例子中，所得到的矩形区域的两个顶点为(0, 0)和(640, 320)。为了避免重复运算，它的_points属性缓存了这两个顶点的坐标。它正好是以像素点为单位的窗口的大小，因此仿射变换矩阵中的数值80实际上是Figure对象的dpi属性。

- BboxTransformTo: 它是一个从单位矩形区域转换到指定的矩形区域的变换。在本例中，它是一个将矩形区域(0, 0)-(1, 1)变换到矩形区域(0, 0)-(640, 320)的坐标变换对象，因此它能将坐标从图表坐标系转换为窗口坐标系中的坐标。其`_mtx`属性缓存了该变换矩阵。

`fig.transFigure` 中的仿射变换对象可以通过 `fig.dpi_scale_trans` 获得:

```
fig.dpi_scale_trans == fig.transFigure._boxout._transform
True
```

接下来我们查看子图坐标变换对象的内容(内容结构参见图4-13):

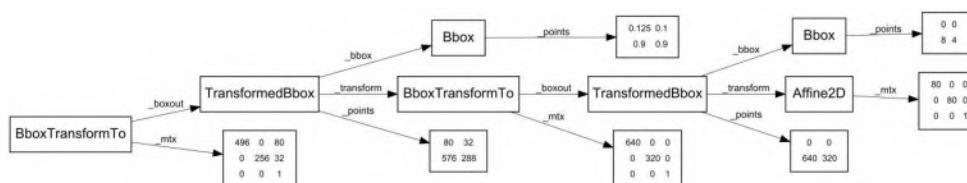


图4-13 子图坐标变换对象的内部结构

```
%dot GraphvizMPLTransform.graphviz(ax.transAxes)
```

`ax.transAxes` 是一个 `BboxTransformTo` 对象，因此它也将 (0, 0)-(1, 1) 区域变换为另一个区域。而此区域是一个 `TransformedBbox` 对象，它是将矩形区域 (0.125, 0.1)-(0.9, 0.9) 通过 `fig.transFigure` 变换之后的区域。因此在 `transAxes` 对象内部使用了 `transFigure` 变换:

```
ax.transAxes._boxout._transform == fig.transFigure
True
```

而此变换中的矩形区域(0.125, 0.1)-(0.9, 0.9)是子图在图表坐标系中的位置:

```
ax.get_position()
Bbox('array([[ 0.125,  0.1  ],\n          [ 0.9  ,  0.9  ]])')
```

子图在窗口坐标系中的矩形区域为:

```
ax.transAxes._boxout.bounds
(80.0, 31.999999999999993, 496.0, 256.0)
```

因此ax.transAxes实际上是一个将矩形区域(0,0)-(1,1)变换到矩形区域(80.0, 32)-(496.0, 256.0)的坐标变换对象。

最后我们观察数据坐标系的变换对象ax.transData(内部结构参见图4-14)。它由ax.transScale、ax.transLimits和ax.transAxes共同构成,因此先看看ax.transLimits和ax.transScale的内容。transLimits是一个BboxTransformFrom对象,它是一个将指定的矩形区域变换为(0,0)-(1,1)矩形区域的变换对象。

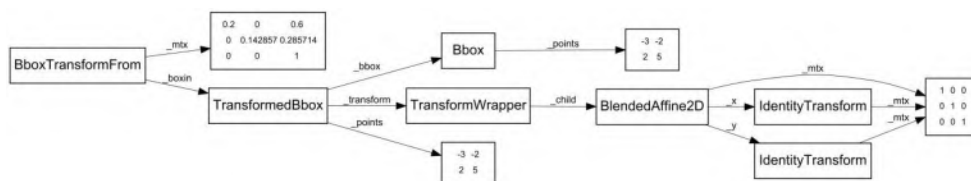


图4-14 数据坐标变换对象的内部结构

```
%dot GraphvizMPLTransform.graphviz(ax.transLimits)
```

而transLimits的源矩形区域为一个TransformedBbox对象，它是一个将矩形区域(-3, -2)-(2, 5)通过坐标变换之后的矩形区域。而此处的变换由TransformWrapper对象定义，在图4-14中它是一个恒等变换。因此transLimits的最终效果就是将矩形区域(-3, -2)-(2, 5)变换为矩形区域(0, 0)-(1, 1)：

```
print ax.transLimits.transform((-3, -2))
print ax.transLimits.transform((2, 5))

[ 0.  0.]
[ 1.  1.]
```

而矩形区域(-3, -2)-(2, 5)由X轴和Y轴的显示范围决定：

```
print ax.get_xlim() # 获得x轴的显示范围
print ax.get_ylim() # 获得y轴的显示范围

(-3.0, 2.0)
(-2.0, 5.0)
```

由于transLimits将数据坐标系的显示范围变换为单位矩形，而transAxes将单位矩形变换为以像素为单位的窗口矩形范围，因此这两个变换的综合效果就是将数据坐标变换为窗口坐标。可以用“+”号将两个变换连接起来创建一个新的变换对象，例如ax.transLimits + ax.transAxes表示先进行ax.transLimits变换，然后进行ax.transAxes变换，变换对象就像流水线上生产产品一样，一步一步地对坐标点进行变换。下面的程序比较它和ax.transData的变换结果：

```
t = ax.transLimits + ax.transAxes
print t.transform((0,0))
print ax.transData.transform((0,0))

[ 377.6          105.14285714]
[ 377.6          105.14285714]
```

为了支持不同比例的坐标轴，transData中还包括一个transScale变换，即 $\text{transData} = \text{transScale} + \text{transLimits} + \text{transAxes}$ 。本例中transScale是一个恒等变换，因此 $\text{ax.transLimits} + \text{ax.transAxes}$ 和 ax.transData 的变换效果一样：

```
ax.transScale

TransformWrapper(BlendedAffine2D(IdentityTransform(), IdentityTr
ansform()))
```

当使用`semilogx()`、`semilogy()`以及`loglog()`等绘图函数绘制对数坐标轴的图表时，或者使用Axes的`set_xscale()`和`set_yscale()`等方法将坐标轴设置为对数坐标时，transScale就不再是恒等变换了，其内部结构如图4-15所示。

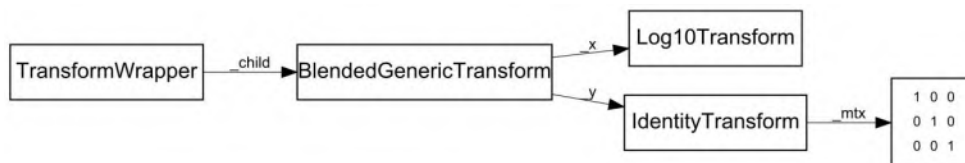


图4-15 X轴为对数坐标时transScale对象的内部结构



由于本例中X轴的取值范围包含负数，因此如果将X轴改为对数坐标，并且重新绘图，会产生很多错误信息。

```
ax.set_xscale("log") # 将x轴改为对数坐标  
  
%dot GraphvizMPLTransform.graphviz(ax.transScale)  
  
ax.set_xscale("linear") # 将x轴改为线性坐标
```

4.3.3 制作阴影效果

下面用上节介绍的坐标变换绘制带阴影效果的曲线。完整程序如下，效果如图4-16所示：

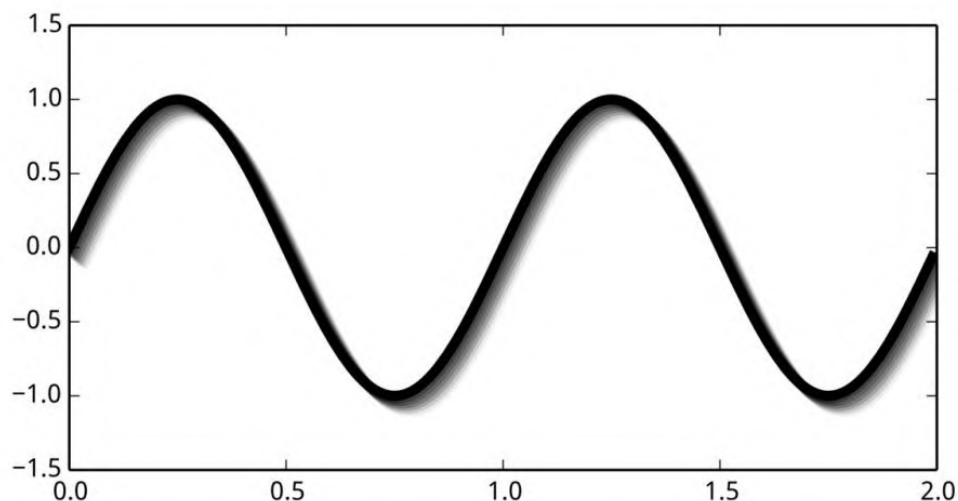


图4-16 使用坐标变换绘制的带阴影的曲线

```
fig, ax = plt.subplots()  
  
x = np.arange(0., 2., 0.01)
```

```

y = np.sin(2*np.pi*x)

N = 7 # 阴影的条数

for i in xrange(N, 0, -1):
    offset = transforms.ScaledTranslation(i, -i,
transforms.IdentityTransform()) ❶

    shadow_trans = plt.gca().transData + offset ❷

    ax.plot(x,y,linewidth=4,color="black",
            transform=shadow_trans, ❸
            alpha=(N-i)/2.0/N)

ax.plot(x,y,linewidth=4,color='black')
ax.set_ylim((-1.5, 1.5))

```

首先使用循环绘制N条透明度和偏移量逐渐变化的曲线，然后绘制实际的曲线，以实现阴影效果。

❶offset是一个ScaledTranslation对象，它的前两个参数决定了X轴和Y轴的偏移量，而第三个参数是一个坐标变换对象，经过它变换之后，再进行偏移变换。由于程序中的第三个参数是一个恒等变换，因此offset实际上是一个单纯的偏移变换：对X轴坐标增加i，对Y轴坐标减少i。

下面查看i为1时的offset：

```

offset.transform((0,0)) # 将(0,0)变换为(1,-1)
array([ 1., -1.])

```

❷阴影曲线的坐标变换由shadow_trans完成，它由数据坐标变换对象transData和offset组成。

```
print ax.transData.transform((0,0)) # 对(0,0)进行数据坐标变换
print shadow_trans.transform((0,0)) # 对(0,0)进行数据坐标变换
和偏移变换

[ 60.  120.]
[ 61.  119.]
```

❸最后通过参数transform将shadow_trans传递给plot()绘图。由于shadow_trans是在完成数据坐标到窗口坐标的变换之后，再进行偏移变换，因此无论当前的缩放比例如何，阴影效果将始终保持一致。

4.3.4 添加注释

在pyplot模块中提供了两个绘制文字的函数：text()和figtext()。它们分别调用当前Axes对象和当前Figure对象的text()方法进行绘图。text()默认在数据坐标系中添加文字，而figtext()则默认在图表坐标系中添加文字。可以通过transform参数改变文字所在的坐标系，下面的程序演示了在数据坐标系、子图坐标系以及图表坐标系中添加文字：

```
x = np.linspace(-1,1,10)
y = x**2

fig, ax = plt.subplots(figsize=(8,4))
ax.plot(x,y)
```

```

for i, (_x, _y) in enumerate(zip(x, y)):
    ax.text(_x, _y, str(i), color="red", fontsize=i+10) ❶

    ax.text(0.5, 0.8, u"子图坐标系中的文字", color="blue",
ha="center",
transform=ax.transAxes) ❷

plt.figtext(0.1, 0.92, u"图表坐标系中的文字", color="green") ❸

```

❶由于没有设置transform参数，text()默认在数据坐标系中创建文字，这里通过fontsize参数修改文字的大小。❷通过transform参数将文字的坐标变换改为ax.transAxes，因此文字在子图坐标系中。ha参数为'center'表示坐标点(0.5, 0.8)在水平方向上是文字的中心，ha是horizontalalignment的缩写，其含义是水平对齐。❸调用figtext()在图表坐标系中添加文字。

程序的输出如图4-17所示。请读者使用缩放和平移工具改变子图的显示范围，你会发现数据坐标系中的文字将跟随曲线变动，而其他两个坐标系中的文字位置不变。单击绘图窗口工具栏中的倒数第二个图标按钮，打开“Subplot Configuration Tool”对话框，调节top、right、bottom和left等参数，你会发现子图坐标系中的文字也会跟着改变位置，水平方向上它和子图的中心始终保持一致。而图表坐标系中文字的位置，只有在改变窗口大小时才会发生变化。

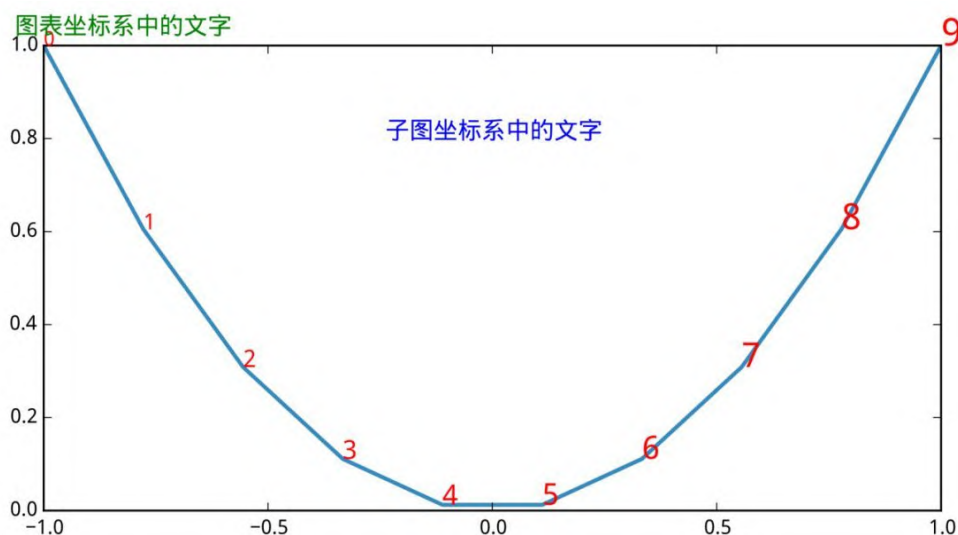


图4-17 三个坐标系中的文字

绘制文字的函数还有许多关键字参数用于设置文字、外框的样式，请读者参考matplotlib的用户手册，这里就不再详细介绍了。

通过pyplot模块的`annotate()`绘制带箭头的注释文字，其调用参数如下：

```
annotate(s, xy, xytext=None, xycoords='data', textcoords='data', arrowprops=None, ...)
```

其中s参数是注释文本，xy是箭头所指处的坐标，xytext是注释文本所在的坐标。xycoords和textcoords分别指定箭头坐标和注释文本坐标的坐标变换方式。

带箭头的注释需要指定两个坐标：箭头所指处的坐标和注释文字所在的坐标。而这两个坐标可以使用不同的坐标变换。参数xycoords和textcoords都是字符串，它们可以有表4-5所示的几种选项：

表4-5 属性值与相应的坐标变换方式

属性值	坐标变换方式
figure points	以点为单位，相对于图表左下角的坐标
figure pixels	以像素为单位，相对于图表左下角的坐标
figure fraction	图表坐标系中的坐标
axes points	以点为单位，相对于子图左下角的坐标
axes pixels	以像素为单位，相对于子图左下角的坐标
axes fraction	子图坐标系中的坐标
data	数据坐标系中的坐标
offset points	以点为单位，相对于点xy的坐标
polar	数据坐标系中的极坐标

其中'figure fraction'、'axes fraction'和'data'分别表示使用图表坐标系、子图坐标系和数据坐标系中的坐标变换对象。由于图表和子图坐标系都是正规化之后的坐标，使用起来不太方便，因此对于图表和子图还分别提供了以点为单位和以像素为单位的坐标变换方式。点和像素的单位类似，但是它不会随着图表的dpi属性值而发生变化，它始终以每英寸72个点进行计算。

上述几种坐标变换都以固定的点为原点进行变换，有时我们希望以距离箭头的偏移量指定文字的坐标，这时可以使用'offset points'选项。

在图4-11中，所有注释的箭头坐标都采用'data'，因此无论如何放大或平移绘图区域，箭头始终指向数据坐标系中的固定点。而注释文本“交点”的坐标变换方式采用'axes fraction'，因此“交点”始终保持在子图中的固定位置。而“直线大于曲线区域”注释文本的坐标采用'offset points'变换，因此文字和箭头的相对位置始终保持不变。

最后，arrowprops参数是一个描述箭头样式的字典。关于注释样式的详细配置请参考matplotlib的相关文档。

4.4 块、路径和集合



与本节内容对应的 Notebook 为：04-matplotlib/matplotlib-400-patch-collections.ipynb。

本节介绍构成绘图元素的几个重要的类，熟练掌握这些类的用法可以绘制出标准的绘图函数无法实现的效果，并且能极大地提高绘图速度。

4.4.1 Path与Patch

Patch对象(块)是一种拥有填充和边线的Artist对象，例如多边形、椭圆等都是Patch对象，它的边线由Path对象描述。而Path对象的vertices和codes属性是两个数组，分别用于描述坐标点和每个坐标点对应的绘图命令代码。表4-6列出了各种命令代码：

表4-6 命令代码及说明

代码	定义	说明
0	STOP	停止绘图
1	MOVETO	将当前位置移动到对应的坐标点
2	LINE TO	从当前位置绘制直线到对应的坐标点
3	CURVE3	使用2个坐标点绘制曲线
4	CURVE4	使用3个坐标点绘制曲线
79	CLOSEPOLY	关闭多边形

下面创建一个左下角位于(0, 1)、宽为2、高为1的Rectangle矩形对象，并查看与之对应的Path对象的vertices和codes属性：

```
rect_patch = plt.Rectangle((0, 1), 2, 1)
rect_path = rect_patch.get_path()
rect_path.vertices      rect_path.codes
-----
[[ 0.,  0.],           [ 1,  2,  2,  2, 79]
 [ 1.,  0.],
```

```
[ 1.,  1.],  
[ 0.,  1.],  
[ 0.,  0.]]
```

对照前面的命令代码表，很容易理解矩形是如何绘制出来的。但是细心的读者会发现，vertices中的坐标并不是我们创建矩形时指定的4个顶点坐标。这是因为所有的矩形对象都共用同一个Path对象，然后通过前面介绍过的transform对象将单位矩形的Path对象变换到指定的坐标之上。下面通过get_patch_transform()获得Patch的坐标变换对象，并用它将单位矩形的顶点坐标变换为我们所创建矩形的顶点坐标：

```
tran = rect_patch.get_patch_transform()  
tran.transform(rect_path.vertices)  
array([[ 0.,  1.],  
       [ 2.,  1.],  
       [ 2.,  2.],  
       [ 0.,  2.],  
       [ 0.,  1.]])
```

对于表示复杂曲线的Patch对象，可以通过诸如InkScape的矢量图设计软件绘制并将曲线保存成SVG文档，然后从SVG文档中提取曲线信息，创建相应的Patch对象。本书提供了从简单的SVG文件中获取路径的read_svg_path()函数，下面是用它绘制的Python图标，参见图4-18：

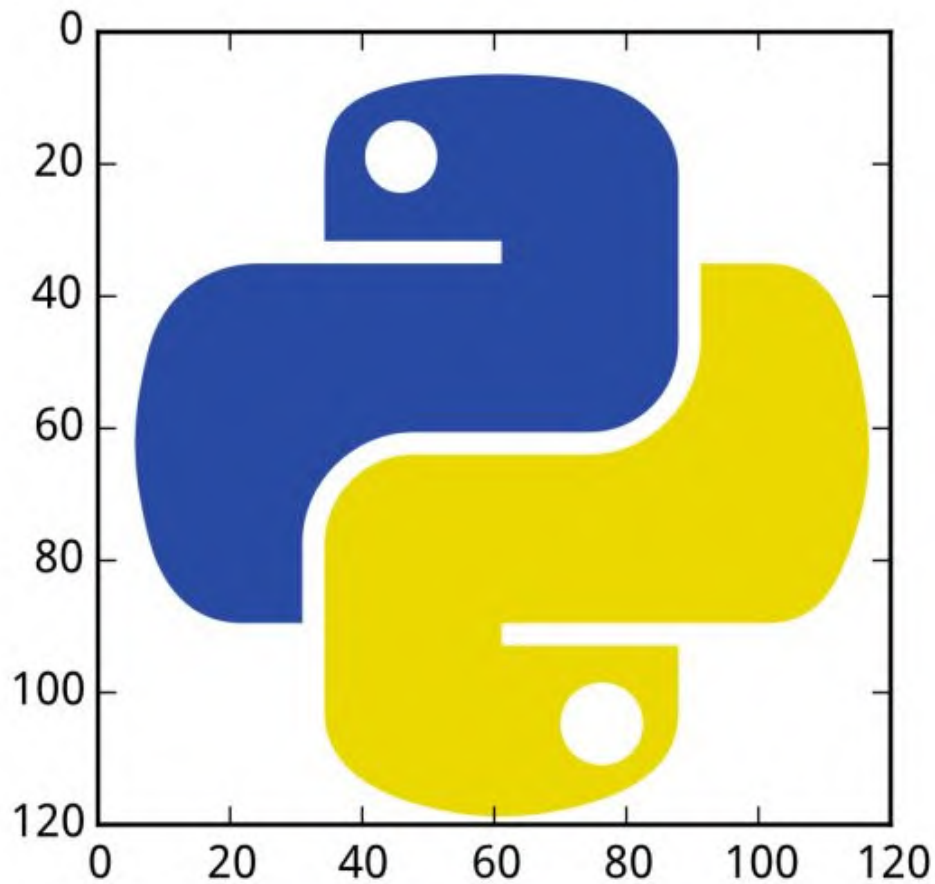


图4-18 使用本书提供的`read_svg_path()`读入SVG文件中的路径并显示为Patch对象



`scipy2.matplotlib.svg_path`: 从SVG文件中获取简单的路径信息。可以使用该模块将矢量绘图软件创建的图形转换为Patch对象。

```
from scipy2.matplotlib.svg_path import read_svg_path

ax = plt.gca()
```

```

patches = read_svg_path("python-logo.svg")

for patch in patches:
    ax.add_patch(patch)

ax.set_aspect("equal")
ax.invert_yaxis()
ax.autoscale()

```

4.4.2 集合

当需要绘制大量图形时，可以使用从Collection类派生的各种集合对象。在绘制图形时，Collection对象会将其中多个保存绘图信息的列表传递给C++编写的绘图函数，从而提高绘图速度。表4-7列出了这些列表属性，当列表长度不统一时，短列表中的元素将被循环使用。

表4-7 Collection类的列表属性及说明

属性	说明
paths	绘图路径
transforms	坐标变换对象
edgecolors	边线颜色
facecolors	填充颜色
linewidths	边线宽度

offsets	坐标偏移量
---------	-------

Collection对象中的路径从路径坐标到屏幕坐标要进行三次坐标变换：

- transform：主坐标变换。
- transforms：与Collection中每条路径对应的路径变换。
- transOffset和offsets：坐标偏移用的变换对象和偏移量，对offsets中的每个坐标采用transOffset进行坐标变换得到实际的坐标偏移量。

根据offset_position参数的值，分为如下两种情况：screen(默认值)和data。

- screen：坐标变换的顺序为——路径变换、主变换、坐标偏移。
- data：坐标变换的顺序为——坐标偏移、路径变换、主变换。

下面通过几个实例帮助读者理解上述各种属性和变换。

1. 曲线集合(LineCollection)

在文件“butterfly.txt”中每一行保存一条封闭曲线上各点的坐标：x0 y0 x1 y1 x2 y2 ...。❶在循环读入这些坐标时，将第一个点的坐标添加到列表尾部，❷然后将其转换为形状为(N, 2)的数组，其中N为曲线上的点数加1。

lines是一个保存多个数组的列表，每个数组表示一条曲线。可以通过LineCollection绘制lines保存的曲线集合。❸colors参数设置所有曲线的颜色为黑色，❹也可以通过cmap参数设置所使用的颜色映射表，曲线的颜色由array参数数组中的值和颜色映射表决定。这里将曲线的点数的对数作为颜色映射表的输入值。

```
from matplotlib import collections as mc

lines = []

with open("butterfly.txt", "r") as f:
    for line in f:
        points = line.strip().split()
        points.extend(points[:2]) ❶
        points = np.array(points).reshape(-1, 2) ❷
        lines.append(points)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4))

lc1 = mc.LineCollection(lines, colors="k", linewidths=1) ❸
lc2 = mc.LineCollection(lines, cmap="Paired", linewidths=1,
❹
                        array=np.log2(np.array([len(line) for line in
lines])))

ax1.add_collection(lc1)
ax2.add_collection(lc2)

for ax in ax1, ax2:
    ax.set_aspect("equal")
```

```
ax.autoscale()  
ax.axis("off")
```



图4-19 使用LineCollection显示大量曲线

lc1和lc2中都有145条路径，lc1的edgecolors属性长度为1，因此所有的曲线都采用相同的颜色。而lc2的edgecolors属性长度为145，其中的每个颜色都是通过norm()和cmap()计算得到的。

```
print "number of lc1 paths:", len(lc1.get_paths())  
print "number of lc1 colors:", len(lc1.get_edgecolors())  
print "number of lc2 colors:", len(lc2.get_edgecolors())  
print np.all(lc2.get_edgecolors() ==  
lc2.cmap(lc2.norm(lc2.get_array())))  
  
number of lc1 paths: 145  
number of lc1 colors: 1  
number of lc2 colors: 145  
True
```

下面显示路径变换、主变换、坐标偏移，可以看到唯一起作用的是主变换，它就是数据坐标变换对象，它将曲线上各个点的坐标从数

据坐标系转换到屏幕坐标系。

```
print lc1.get_transforms() # 路径变换
print lc1.get_transform() is ax1.transData # 主变换为数据坐标
变换对象
print lc1.get_offset_transform(), lc1.get_offsets()
[]
True
IdentityTransform() [[ 0.  0.]]
```

使用LineCollection可以绘制颜色或宽度渐变的曲线，例如下面对一个二维平面上的矢量场积分，并将所得的路径保存到streams列表中。它的长度为25，其中每个数组表示一条积分路径，形状为(50, 2)。

```
from scipy.integrate import odeint

def field(s, t):
    x, y = s
    return 0.3 * x - y, 0.3 * y + x
    return [u, v]

X, Y = np.mgrid[-2:2:5j, -2:2:5j]
init_pos = np.c_[X.ravel(), Y.ravel()]
t = np.linspace(0, 5, 50)

streams = []
for pos in init_pos:
```

```

    r = odeint(field, pos, t)

    streams.append(r)

print len(streams), streams[0].shape

25 (50, 2)

```

为了采用渐变颜色显示每条积分路径，先将streams转换成一个三维数组lines，形状为(25*(50-1), 2, 2)，也就是由1225条线段组成的集合。我们使用两种数值作为颜色映射表的输入：time_value和speed_value。其中time_value为到达对应坐标点所需的时间，speed_value为对应坐标点处的速度大小，效果如图4-20所示。

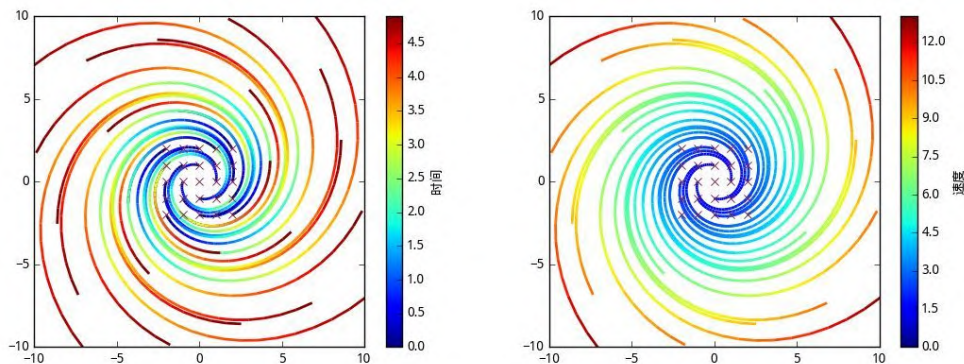


图4-20 使用LineCollection绘制颜色渐变的曲线

```

lines = np.concatenate([
    np.concatenate((r[:-1, None, :], r[1:, None, :]),
axis=1)

    for r in streams], axis=0)

time_value = np.concatenate([t[:-1]] * len(streams))
x, y = lines.mean(axis=1).T
u, v = field([x, y], 0)

```

```

speed_value = np.sqrt(u ** 2 + v ** 2)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 3.5))
fig.subplots_adjust(0, 0, 1, 1)
ax1.plot(init_pos[:, 0], init_pos[:, 1], "x")
ax2.plot(init_pos[:, 0], init_pos[:, 1], "x")

lc1 = mc.LineCollection(lines, linewidths=2,
array=time_value)
lc2 = mc.LineCollection(lines, linewidths=2,
array=speed_value)

ax1.add_collection(lc1)
ax2.add_collection(lc2)

plt.colorbar(ax=ax1, mappable=lc1, label=u"时间")
plt.colorbar(ax=ax2, mappable=lc2, label=u"速度")

for ax in ax1, ax2:
    ax.plot(init_pos[:, 0], init_pos[:, 1], "x")
    ax.autoscale()
    ax.set_aspect("equal")
    ax.set_xlim(-10, 10)
    ax.set_ylim(-10, 10)

```

2. 多边形集合(PolyCollection)

多边形集合PolyCollection的用法和LineCollection类似，不过它会自动封闭并填充颜色。下面的star_polygon()创建以(x, y)为中心、r为半径、旋转theta的N角星，s参数为内圆的半径与外圆半径的比值，效果如图4-21所示。在随机创建1000个N角星后，调用PolyCollection绘制。

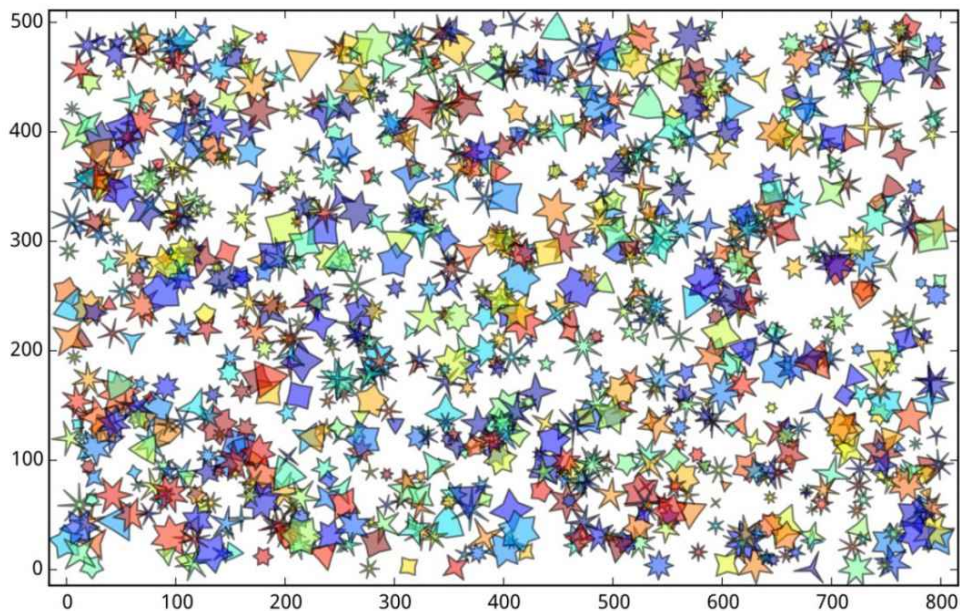


图4-21 用PolyCollection绘制大量多边形

```
from numpy.random import randint, rand, uniform

def star_polygon(x, y, r, theta, n, s):
    angles = np.arange(0, 2*np.pi, 2*np.pi/2/n) + theta
    xs = r * np.cos(angles)
    ys = r * np.sin(angles)
    xs[1::2] *= s
    ys[1::2] *= s
    xs += x
```

```

        ys += y
    return np.vstack([xs, ys]).T

stars = []
for i in range(1000):
    star = star_polygon(randint(800), randint(500),
                        uniform(5, 20), uniform(0,
2*np.pi),
                        randint(3, 9), uniform(0.1, 0.7))
    stars.append(star)

fig, ax = plt.subplots(figsize=(10, 5))
    polygons = mc.PolyCollection(stars, alpha=0.5,
array=np.random.rand(len(stars)))
    ax.add_collection(polygons)
    ax.autoscale()
    ax.margins(0)
    ax.set_aspect("equal")

```

每个多边形的填充颜色由颜色映射表决定，因此facecolors的长度与多边形的个数相同，而所有的多边形共用边线颜色，所以edgecolors的长度为1。PolyCollection的坐标变换方式与LineCollection相同，就不再重复了。

```

        print "length of facecolors:",
len(polygons.get_facecolors())
        print "length of edgecolors:",

```

```
len(polygons.get_edgecolors())  
    length of facecolors: 1000  
    length of edgecolors: 1
```

3. 路径集合(PathCollection)

`scatter()` 用于绘制散列图，它返回的是一个 `PathCollection` 对象：

```
N = 30  
  
np.random.seed(42)  
  
x = np.random.rand(N)  
y = np.random.rand(N)  
  
size = np.random.randint(20, 60, N)  
value = np.random.rand(N)  
  
  
fig, ax = plt.subplots()  
pc = ax.scatter(x, y, s=size, c=value)
```

`pc` 是一个 `PathCollection` 对象，它的 `facecolors` 长度为散列点数，颜色由颜色映射表决定。`edgecolors` 的长度为1，所以每个圆形的边线颜色相同。

所有散列点的形状都是相似的，因此 `paths` 属性的长度为1。每个散列点的大小由路径变换 `transforms` 决定，它是一个形状为 (30, 3, 3) 的三维数组。每个散列点对应其中的一个 3×3 的变换矩阵。下面是

下标为0的散列点的路径变换对象，它将路径在X轴和Y轴方向上放大5.916倍：

```
print pc.get_transforms().shape
print pc.get_transforms()[0] #下标为0的点对应的缩放矩阵
(30, 3, 3)
[[ 5.91607978  0.          0.          ]
 [ 0.          5.91607978  0.          ]
 [ 0.          0.          1.          ]]
```

散列点的中心位置为offsets经过offset_transform变换之后的坐标。由下面的结果可以看出offset_transform变换就是数据坐标变换对象，它将数据空间中的坐标变换为以像素为单位的屏幕坐标，因此offsets中保存的就是数据坐标系中的坐标偏移量。

```
print pc.get_offsets()[0] #下标为0的点对应的中心坐标
#计算下标为0的点对应的屏幕坐标
print pc.get_offset_transform().transform(pc.get_offsets())
[0]
print pc.get_offset_transform() is ax.transData
[ 0.37454012  0.60754485]
[ 212.66351729  134.74900826]
True
```

transforms决定散列点的大小，offset_transform和offsets决定散列点的位置，因此主变换transform对象无须做任何变换，它是一个恒等变换。

```
print pc.get_transform()

IdentityTransform()
```

由于offset_position的值为screen，因此坐标变换的顺序为：路径变换、主变换、坐标偏移。路径变换对单位圆进行缩放，而坐标偏移则把圆形移动到指定的位置。

```
pc.get_offset_position()

u'screen'
```

4. 椭圆集合(EllipseCollection)

EllipseCollection用于绘制大量的椭圆，每个椭圆都可以拥有独立的长轴和短轴长度以及旋转角度。它的参数如下：

```
EllipseCollection(self, widths, heights, angles,
units='points', **kwargs)
```

其中widths、heights和angles是三个长度相同的数组，分别为每个椭圆的两个轴的长度以及旋转角度。而units参数则指定widths和heights的单位，有如下选项：

```
'points' | 'inches' | 'dots' | 'width' | 'height' | 'x' | 'y' | 'xy'
```

其中'points'、'inches'和'dots'为屏幕坐标系中的长度，'dots'的单位为像素点，而'points'和'inches'则根据图表对象的DPI属性按照不同的比例变换为'dots'单位。'width'和'height'分别用子图的宽度或高度作为长度单位，'x'和'y'则采用数据坐标系中

的X轴或Y轴长度，'xy'表示采用数据坐标系中的X轴和Y轴的长度单位。如果子图的X轴和Y轴的长度单位不同，椭圆呈现的旋转角度与angles指定的值也会有所不同。

下面的程序演示了unit为'x'和'xy'的区别，效果如图4-22所示，左图中椭圆的长度单位为'x'，宽度为X轴上的两个单位距离，高度为X轴上的一个单位距离。由于高度和宽度采用同样的单位，因此椭圆显示的角度与angles指定的值相同。当unit为'xy'时，椭圆的高度采用Y轴的单位距离，由于Y轴的单位距离的像素点数小于X轴的单位距离，因此右图中的椭圆比左图的更扁一些，而且由于两个方向上的长度单位不同，椭圆呈现的方向也与angles指定的值不同。如果使用`axes[1].set_aspect("equal")`将X和Y轴的单位长度设置为相同，则椭圆的角度和angles指定的角度相同，图中的12个椭圆将均匀地分布在一个正圆的圆周上。

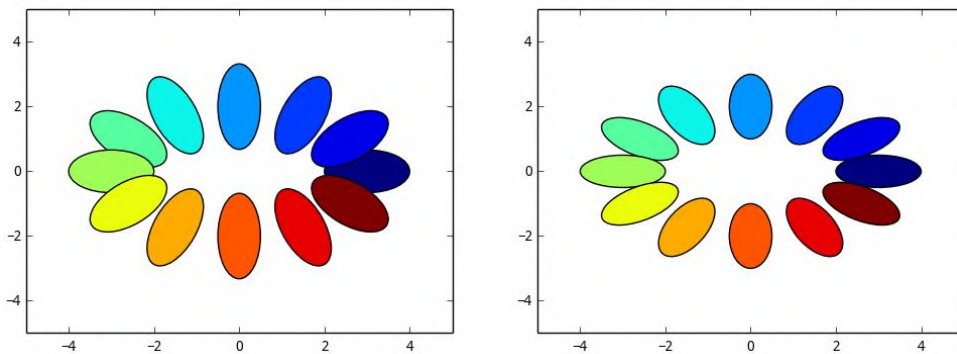


图4-22 EllipseCollection的unit参数：unit='x' (左图)、unit='xy' (右图)

```
angles = np.linspace(0, 2*np.pi, 12, endpoint=False)
offsets = np.c_[3*np.cos(angles), 2*np.sin(angles)]
angles_deg = np.rad2deg(angles)
widths = np.full_like(angles, 2)
heights = np.full_like(angles, 1)
```

```

fig, axes = plt.subplots(1, 2, figsize=(12, 4))

    ec0 = mc.EllipseCollection(widths, heights, angles_deg,
units="x", array=angles,
                                offsets=offsets,
transOffset=axes[0].transData)
    axes[0].add_collection(ec0)
    axes[0].axis((-5, 5, -5, 5))

    ec1 = mc.EllipseCollection(widths, heights, angles_deg,
units="xy", array=angles,
                                offsets=offsets,
transOffset=axes[1].transData)
    axes[1].add_collection(ec1)
    axes[1].axis((-5, 5, -5, 5))
    #axes[1].set_aspect("equal")

```

5. 数据空间中的圆形集合对象

仿照图4-22(右)，可以使用EllipseCollection绘制以数据空间中的长度为半径的圆的集合，只需要将widths和heights设置为圆形的直径即可。虽然在collections模块中还有一个CircleCollection类，但是它所设置的圆的大小是屏幕空间中的大小，无法控制其在数据空间中的大小。

下面我们自定义一个能绘制数据空间中圆形集合的 DataCircleCollection 类:

```
from matplotlib.collections import CircleCollection,
Collection

from matplotlib.transforms import Affine2D

class DataCircleCollection(CircleCollection):

    def set_sizes(self, sizes):
        self._sizes = sizes

    def draw(self, render):
        ax = self.axes
        ms = np.zeros((len(self._sizes), 3, 3))
        ms[:, 0, 0] = self._sizes
        ms[:, 1, 1] = self._sizes
        ms[:, 2, 2] = 1
        self._transforms = ms ❶

        m = ax.transData.get_affine().get_matrix().copy()
        m[:2, 2:] = 0
        self.set_transform(Affine2D(m)) ❷

        return Collection.draw(self, render)
```

- ①设置每个圆形对应的路径变换，它们将圆形缩放指定的倍数。
- ②使用数据坐标转换对象的缩放系数创建一个新的二维仿射变换对象，并将其设置为主变换。路径变换与主变换叠加起来，就将半径为1的单位圆缩放为数据空间中半径为sizes的圆形。

每个圆形对应的路径变换将单位圆变换为数据坐标系中指定大小的圆形，然后通过主坐标变换将数据坐标系中的圆形变换为屏幕坐标系中的大小，最后再由坐标偏移变换将圆形放到指定的位置。

下面用DataCircleCollection绘制一幅漂亮的圆形拼图，效果如图4-23所示。“venus-face.csv”中的每一行有6个数值表示一个圆形，每个数值的含义为：圆形X轴坐标、圆形Y轴坐标、半径、红色、绿色和蓝色。



图4-23 使用DataCircleCollection绘制大量的圆形

```
data = np.loadtxt("venus-face.csv", delimiter=",")
offsets = data[:, :2]
sizes = data[:, 2] * 1.05
colors = data[:, 3:] / 256.0

fig, axe = plt.subplots(figsize=(8, 8))
axe.set_rasterized(True)
cc = DataCircleCollection(sizes, facecolors=colors,
edgecolors="w", linewidths=0.1,
```

```
offsets=offsets,  
transOffset=axe.transData)  
  
axe.add_collection(cc)  
axe.axis((0, 512, 512, 0))  
axe.axis("off")
```

4.5 绘图函数简介



与本节内容对应的 Notebook 为：04-matplotlib/matplotlib-500-plot-functions.ipynb。

本节介绍如何使用matplotlib绘制一些常用的图表。matplotlib的每个绘图函数都有许多关键字参数用来设置图表的各种属性，由于篇幅有限，本书不能对其一一进行介绍。一般来说，如果读者需要对图表进行某种特殊的设置，可以在绘图函数的说明文档或者matplotlib的演示页面中找到相关的说明。

4.5.1 对数坐标图

前面介绍过如何使用plot()绘制曲线图，所绘制图表的X-Y轴坐标都是算术坐标。下面我们看看如何在对数坐标系中绘图。

绘制对数坐标图的函数有三个：`semilogx()`、`semilogy()`、`loglog()`。它们分别绘制X轴为对数坐标、Y轴为对数坐标以及两个轴都为对数坐标的图表。

下面的程序使用4种不同的坐标系绘制低通滤波器的频率响应曲线，结果如图4-24所示。其中，左上图为`plot()`绘制的算术坐标系，右上图为`semilogx()`绘制的X轴对数坐标系，左下图为`semilogy()`绘制的Y轴对数坐标系，右下图为`loglog()`绘制的双对数坐标系。使用双对数坐标系表示的频率响应曲线通常被称为波特图。

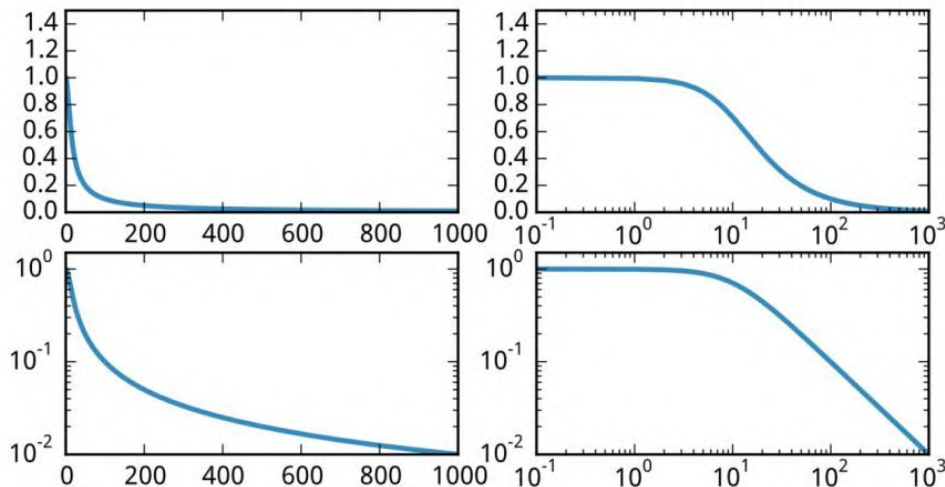


图4-24 低通滤波器的频率响应：算术坐标(左上)、X轴对数坐标(右上)、Y轴对数坐标(左下)、双对数坐标(右上)

```
w = np.linspace(0.1, 1000, 1000)
p = np.abs(1/(1+0.1j*w)) # 计算低通滤波器的频率响应

fig, axes = plt.subplots(2, 2)

functions = ("plot", "semilogx", "semilogy", "loglog")
```

```

for ax, fname in zip(axes.ravel(), functions):
    func = getattr(ax, fname)
    func(w, p, linewidth=2)
    ax.set_ylim(0, 1.5)

```

4.5.2 极坐标图

极坐标系是和笛卡尔坐标系完全不同的坐标系，极坐标系中的点由一个夹角和一段相对中心点的距离表示。下面的程序绘制极坐标图，效果如图4-25所示。

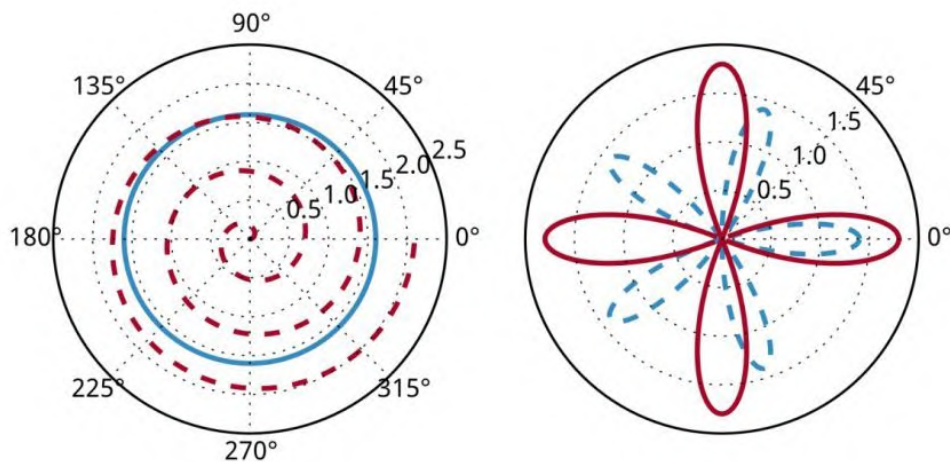


图4-25 极坐标中的圆、螺旋线和玫瑰线

```

theta = np.arange(0, 2*np.pi, 0.02)

plt.subplot(121, polar=True) ❶
plt.plot(theta, 1.6*np.ones_like(theta), linewidth=2) ❷
plt.plot(3*theta, theta/3, "--", linewidth=2)

```

```
plt.subplot(122, polar=True)
plt.plot(theta, 1.4*np.cos(5*theta), "--", linewidth=2)
plt.plot(theta, 1.8*np.cos(4*theta), linewidth=2)
plt.rgrids(np.arange(0.5, 2, 0.5), angle=45) ❸
plt.thetagrids([0, 45]) ❹
```

❶调用`subplot()`创建子图时通过设置`polar`参数为`True`，创建一个极坐标子图。❷然后调用`plot()`在极坐标子图中绘图。也可以使用`polar()`直接创建极坐标子图并在其中绘制曲线。

❸`rgrids()`设置同心圆栅格的半径大小和文字标注的角度。因此右图中的虚线圆圈有三个，半径分别为0.5、1.0和1.5，这些文字沿着45°线排列。❹`thetagrids()`设置放射线栅格的角度，因此右图中只有两条放射线栅格线，角度分别为0°和45°。

4.5.3 柱状图

柱状图用其每根柱子的长度表示值的大小，它们通常用来比较两组或多组值。下面的程序从文件中读入中国人口的年龄的分布数据(人口分布数据由维基百科提供，仅供参考，不保证正确性)，并使用柱状图比较男性和女性的年龄分布，效果如图4-26所示。

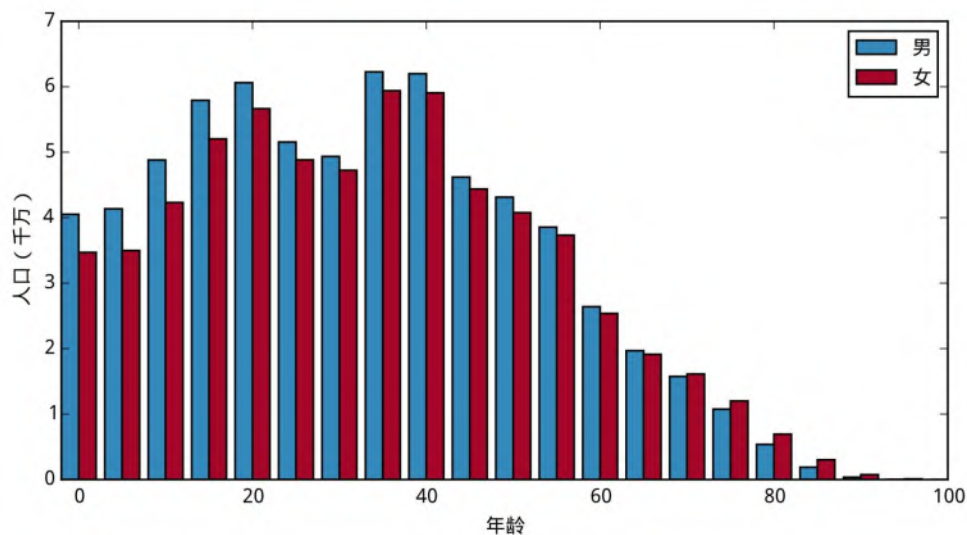


图4-26 中国男女人口的年龄分布图

```
data = np.loadtxt("china_population.txt")
width = (data[1,0] - data[0,0])*0.4 ❶
plt.figure(figsize=(8, 4))
c1, c2 = plt.rcParams['axes.color_cycle'][:2]
plt.bar(data[:,0]-width, data[:,1]/1e7, width, color=c1,
label=u"男") ❷
plt.bar(data[:,0], data[:,2]/1e7, width, color=c2,
label=u"女") ❸
plt.xlim(-width, 100)
plt.xlabel(u"年龄")
plt.ylabel(u"人口(千万)")
plt.legend()
```

读入的数据中，第一列为年龄，它将作为柱状图的横坐标。❶首先计算柱状图中每根柱子的宽度，因为要在每个年龄段上绘制两根柱子，因此柱子的宽度应该小于年龄段的二分之一。这里以年龄段的0.4倍作为柱子的宽度。

❷调用`bar()`绘制男性人口分布的柱状图。它的第一个参数为每根柱子的左边缘的横坐标，为了让男性和女性的柱子以年龄刻度为中心，这里让每根柱子左侧的横坐标为“年龄减去柱子的宽度”。`bar()`的第二个参数为每根柱子的高度，第三个参数指定所有柱子的宽度。当第三个参数为序列时，可以为每根柱子指定宽度。

❸绘制女性人口分布的柱状图，这里以年龄为柱子的左边缘横坐标，因此女性和男性的人口分布图以年龄刻度为中心。由于`bar()`不自动修改颜色，因此程序中通过`color`参数设置两个柱状图的颜色。

4.5.4 散列图

使用`plot()`绘图时，如果指定样式参数为只绘制数据点，那么所绘制的就是一幅散列图。例如：

```
plt.plot(np.random.random(100), np.random.random(100), "o")
```

但是这种方法所绘制的点无法单独指定颜色和大小。`scatter()`所绘制的散列图可以指定每个点的颜色和大小。下面的程序演示了`scatter()`的用法，效果如图4-27所示。

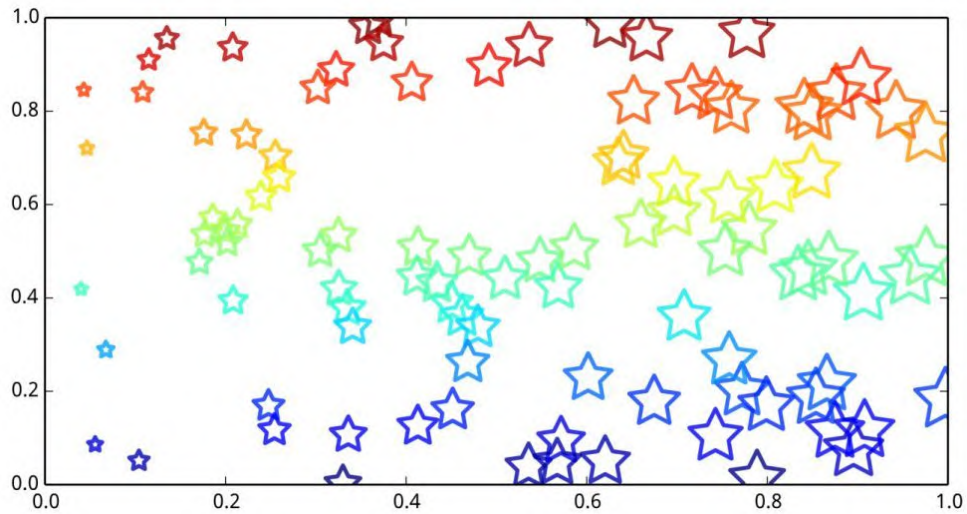


图4-27 可指定点的颜色和大小散列

```
plt.figure(figsize=(8, 4))
x = np.random.random(100)
y = np.random.random(100)
plt.scatter(x, y, s=x*1000, c=y, marker=(5, 1),
            alpha=0.8, lw=2, facecolors="none")
plt.xlim(0, 1)
plt.ylim(0, 1)
```

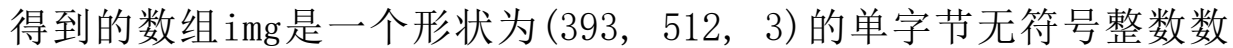
`scatter()` 的前两个参数是两个数组，分别指定每个点的X轴和Y轴的坐标。`s`参数指定点的大小，其值和点的面积成正比，可以是单个数值或数组。

`c`参数指定每个点的颜色，也可以是数值或数组。这里使用一维数组为每个点指定了一个数值。通过颜色映射表，每个数值都会与一个颜色相对应。默认的颜色映射表中蓝色与最小值对应，红色与最大值对应。当`c`参数是形状为(N, 3)或(N, 4)的二维数组时，则直接表示每个点的RGB颜色。

`marker`参数设置点的形状，可以是一个表示形状的字符串，或是表示多边形的两个元素的元组，第一个元素表示多边形的边数，第二个元素表示多边形的样式，取值范围为0、1、2、3。0表示多边形，1表示星形，2表示放射形，3表示忽略边数显示为圆形。

最后，通过`alpha`参数设置点的透明度，`lw`参数设置线宽，它是`linewidth`的缩写。`facecolors`参数为“none”表示散列点没有填充色。

4.5.5 图像

`imread()` 和 `imshow()` 提供了简单的图像载入和显示功能。`imread()` 可以从图像文件读入数据，得到一个表示图像的NumPy数组。它的第一个参数是文件名或文件对象，`format`参数指定图像类型，如果省略则由文件的扩展名决定图像类型。对于灰度图像，它返回一个形状为(M, N)的数组；对于彩色图像，它返回形状为(M, N, C)的数组。其中M为图像的高度，N为图像的宽度，C为3或4，表示图像的通道数。下面的程序从`lena.jpg`中读入图像数据，效果如图4-28所示。所得到的数组是一个形状为(393, 512, 3)的单字节无符号整数数组。这是因为通常所使用的图像采用单字节分别保存每个像素的红、绿、蓝三个通道的分量：

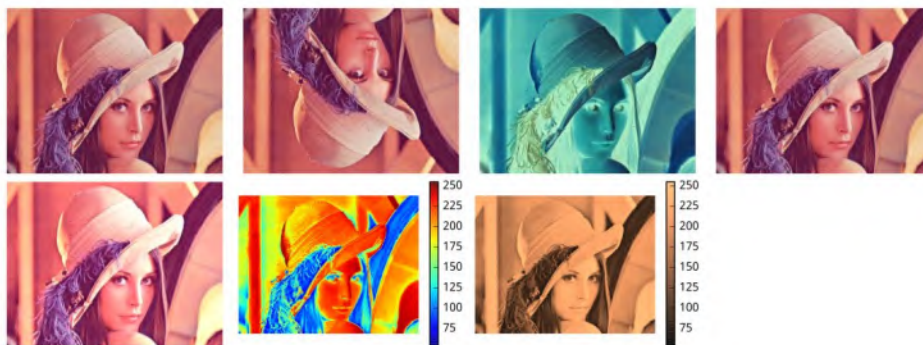


图4-28 用`imread()`和`imshow()`显示图像

```
img = plt.imread("lena.jpg")  
print img.shape, img.dtype  
(393, 512, 3) uint8
```

下面使用`imshow()`显示所表示的图像：

❶ `imshow()`可以用来显示`imread()`所返回的数组。如果数组是表示多通道图像的三维数组，则每个像素的颜色由各个通道的值决定。

❷ `imshow()`所绘制图表的Y轴的正方向是从上往下的。如果设置`imshow()`的`origin`参数为“lower”，则所显示图表的原点在左下角，但是整个图像就上下颠倒了。

❸ 如果三维数组的元素类型为浮点数，则元素值的取值范围为0.0到1.0，与颜色值0到255对应。超过这个范围可能会出现颜色异常的像素。下面的例子将数组转换为浮点数组并用`imshow()`进行显示，由于数值范围超过了0.0~1.0，因此颜色显示异常。

❹ 而取值在0.0~1.0的浮点数组和原始图像完全相同。

⑤使用`clip()`将超出范围的值限制在取值范围之内，可以使整个图像变亮。

⑥如果`imshow()`的参数是二维数组，则使用颜色映射表决定每个像素的颜色。这里显示图像中的红色通道，它是一个二维数组。其显示效果比较吓人，因为默认的图像映射将最小值映射为蓝色、将最大值映射为红色。可以使用`colorbar()`将颜色映射表在图表中显示出来。

⑦通过`imshow()`的`cmap`参数可以修改显示图像时所采用的颜色映射表，使用名为`copper`的颜色映射表显示图像的红色通道。

```
img = plt.imread("lena.jpg")
fig, axes = plt.subplots(2, 4, figsize=(11, 4))
fig.subplots_adjust(0, 0, 1, 1, 0.05, 0.05)

axes = axes.ravel()

axes[0].imshow(img) ①
axes[1].imshow(img, origin="lower") ②
axes[2].imshow(img * 1.0) ③
axes[3].imshow(img / 255.0) ④
axes[4].imshow(np.clip(img / 200.0, 0, 1)) ⑤

axe_img = axes[5].imshow(img[:, :, 0]) ⑥
plt.colorbar(axe_img, ax=axes[5])

axe_img = axes[6].imshow(img[:, :, 0], cmap="copper") ⑦
```

```
plt.colorbar(axe_img, ax=axes[6])

for ax in axes:
    ax.set_axis_off()
```

颜色映射表是一个ColorMap对象，matplotlib中已经预先定义了很多颜色映射表，可以通过下面的语句找到这些颜色映射表的名字：

```
import matplotlib.cm as cm
cm._cmapnames[:5]

['Spectral', 'copper', 'RdYlGn', 'Set2', 'summer']
```

使用imshow()可以显示任意的二维数据，例如下面的程序使用图像直观地显示了二元函数 $f(x,y) = xe^{x^2-y^2}$ ，效果如图4-29所示。

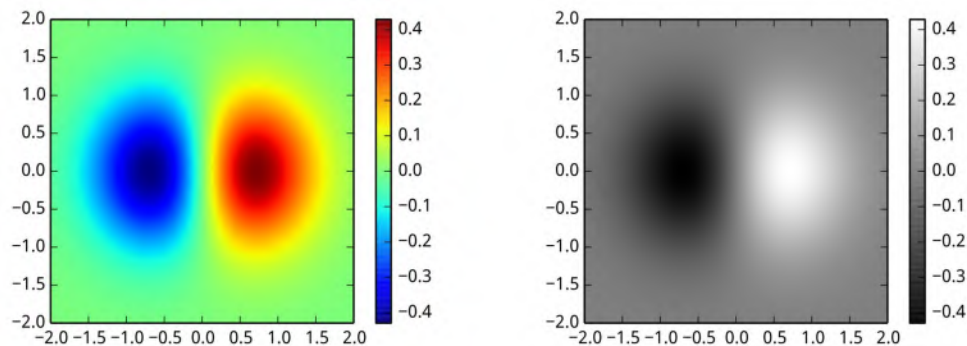


图4-29 使用imshow()可视化二元函数

```
y, x = np.ogrid[-2:2:200j, -2:2:200j]
z = x * np.exp(- x**2 - y**2) ❶

extent = [np.min(x), np.max(x), np.min(y), np.max(y)] ❷

plt.figure(figsize=(10,3))
```

```
plt.subplot(121)

plt.imshow(z, extent=extent, origin="lower") ❸

plt.colorbar()

plt.subplot(122)

plt.imshow(z, extent=extent, cmap=cm.gray, origin="lower")

plt.colorbar()
```

❶首先通过数组的广播功能计算出表示函数值的二维数组 z ，注意它的第0轴表示Y轴、第1轴表示X轴。❷然后将X、Y轴的取值范围保存到`extent`列表中。❸将`extent`列表传递给`imshow()`的`extent`参数，这样图表的X、Y轴的刻度标签将使用`extent`列表指定的范围。

4.5.6 等值线图

还可以使用等值线图表示二元函数。所谓等值线，是指由函数值相等的各点连成的平滑曲线。等值线可以直观地表示二元函数值的变化趋势，例如等值线密集的地方表示函数值在此处的变化较大。`matplotlib`中可以使用`contour()`和`contourf()`描绘等值线，它们的区别是`contourf()`所得到的是带填充效果的等值线。下面的程序演示了这两个函数的用法，效果如图4-30所示：

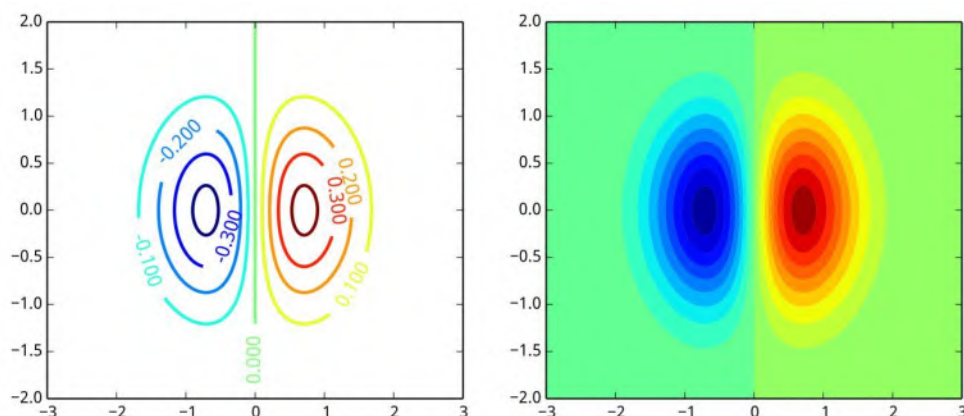


图4-30 用contour(左)和contourf(右)描绘等值线图

```

y, x = np.ogrid[-2:2:200j, -3:3:300j] ❶
z = x * np.exp( - x**2 - y**2)

extent = [np.min(x), np.max(x), np.min(y), np.max(y)]

plt.figure(figsize=(10,4))
plt.subplot(121)
cs = plt.contour(z, 10, extent=extent) ❷
plt.clabel(cs) ❸
plt.subplot(122)
plt.contourf(x.reshape(-1), y.reshape(-1), z, 20) ❹

```

❶为了更清楚地区分X轴和Y轴，这里让它们的取值范围和等分次数均不相同。这样所得到的数组 z 的形状为(200, 300)，它的第0轴对应Y轴，第1轴对应X轴。

❷调用`contour()`绘制数组 z 的等值线图，第二个参数为10表示将整个函数的取值范围等分为10个区间，即其所显示的等值线图中将有9条等值线。和`imshow()`一样，可以使用`extent`参数指定等值线图的X轴

和Y轴的数据范围。❸ `contour()` 所返回的是一个 `QuadContourSet` 对象，将它传递给 `clabel()`，为其中的等值线标上对应的值。

❹ 调用 `contourf()` 绘制带填充效果的等值线图。这里演示了另一种设置X、Y轴取值范围的方法。它的前两个参数分别是计算数组 `z` 时所使用的X轴和Y轴上的取样点，这两个数组必须是一维数组或是形状与数组 `z` 相同的数组。



如果需要对散列点数据绘制等值线图，可以先使用 `scipy.interpolate` 模块中提供的插值函数将散列点数据插值为网格数据。

还可以使用等值线绘制隐函数曲线。所谓隐函数，是指在一个方程中，若令 `x` 在某一区间内取任意值时总有相应的 `y` 满足此方程，则可以说方程在该区间上确定了 `x` 的隐函数 `y`，如隐函数 $x^2 + y^2 - 1 = 0$ 表示一个单位圆。

显然无法像绘制一般函数那样，先创建一个等差数组表示变量 `x` 的取值点，然后计算出数组中每个 `x` 所对应的 `y` 值。可以使用等值线解决这个问题，显然隐函数的曲线就是值等于0的那条等值线。下面的程序绘制函数：

$$f(x, y) = (x^2 + y^2)^4 - (x^2 - y^2)^2$$

在 $f(x, y) = 0$ 和 $f(x, y) - 0.1 = 0$ 时的曲线，效果如图4-31(左)所示。

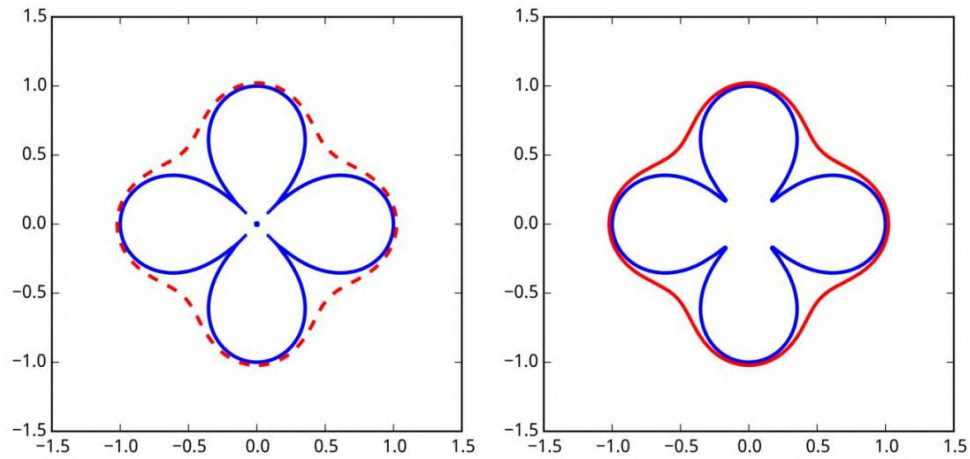


图4-31 使用等值线绘制隐函数曲线(左)，获取等值线数据并绘图(右)

```

y, x = np.ogrid[-1.5:1.5:200j, -1.5:1.5:200j]
f = (x**2 + y**2)**4 - (x**2 - y**2)**2

plt.figure(figsize=(9, 4))
plt.subplot(121)
extent = [np.min(x), np.max(x), np.min(y), np.max(y)]
cs = plt.contour(f, extent=extent, levels=[0, 0.1], ❶
                colors=["b", "r"], linestyles=["solid", "dashed"],
linewidths=[2, 2])

plt.subplot(122)
for c in cs.collections: ❷
    data = c.get_paths()[0].vertices
    plt.plot(data[:,0], data[:,1],
              color=c.get_color()[0],
linewidth=c.get_linewidth()[0])

```

❶在调用`contour()`绘制等值线时，可以通过`levels`参数指定等值线所对应的函数值，这里设置`levels`参数为`[0, 0.1]`，因此最终将绘制两条等值线。通过`colors`、`linestyles`、`linewidths`等参数可以分别指定每条等值线的颜色、线型以及线宽。

仔细观察图4-31(左)会发现，表示隐函数 $f(x, y)=0$ 的蓝色实线并不是完全连续的，在图的中间部分它由许多孤立的小段构成。因为等值线在原点附近无限靠近，所以无论对函数 f 的取值空间如何进行细分，总是会有无法分开的地方，最终造成了图中的那些孤立的细小区域，而表示隐函数 $f(x, y)-0.1=0$ 的红色虚线则是闭合且连续的。

❷从等值线集合`cs`中找到表示等值线的路径，并使用`plot()`将其绘制出来，效果如图4-31(右)所示。

`contour()`返回一个`QuadContourSet`对象，其`collections`属性是一个等值线列表，每条等值线用一个`LineCollection`对象表示：

```
print cs
cs.collections
<matplotlib.contour.QuadContourSet instance at 0x057EFC10>
<a list of 2 mcoll.LineCollection objects>
```

每个`LineCollection`对象都有它自己的颜色、线型、线宽等属性，注意这些属性所获得结果的外面还有一层包装，要获得其第0个元素才是真正的配置：

```
print cs.collections[0].get_color()[0]
print cs.collections[0].get_linewidth()[0]
```

```
[ 0.  0.  1.  1.]  
2
```

在前面的章节介绍过LineCollection对象是一组曲线的集合，因此它可以表示蓝色实线那样由多条线构成的等值线。它的get_paths()方法获得构成等值线的所有路径，本例中蓝色实线所表示的等值线由42条路径构成：

```
len(cs.collections[0].get_paths())  
42
```

路径是一个Path对象，通过它的vertices属性可以获得路径上所有点的坐标：

```
path = cs.collections[0].get_paths()[0]  
path.vertices  
array([[ -0.08291457, -0.98938936],  
       [ -0.09039269, -0.98743719],  
       [ -0.09798995, -0.98513674],  
       ...,  
       [ -0.05276382, -0.99548781],  
       [ -0.0678392 , -0.99273907],  
       [ -0.08291457, -0.98938936]])
```

4.5.7 四边形网格

`pcolormesh(X, Y, C)` 绘制由X、Y和C三个数组定义的四边形网格。这三个数组是二维数组，X和Y的形状相同，C的形状可以和X、Y相同，也可以比它们少一行一列。每个四边形的4个顶点的X轴坐标由X中上下左右相邻的4个元素决定，Y轴坐标由Y中对应的4个元素决定。四边形的颜色由C中对应的元素以及颜色映射表决定。

在下面的例子中，X和Y的形状都是(2, 3)，其中有两组上下左右相邻的4个元素，定义两个四边形的4个顶点：

第一个四边形的顶点 第二个四边形的顶点

=====	=====
(0, 0), (1, 0.2)	(1, 0.2), (2, 0)
(0, 1), (1, 0.8)	(1, 0.8), (2, 1)

每个四边形的填充颜色与Z中的一个元素对应：

```
X = np.array([[0, 1, 2],
               [0, 1, 2]])
Y = np.array([[0, 0.2, 0],
               [1, 0.8, 1]])
Z = np.array([[0.5, 0.8]])
```

下面将X和Y平坦化之后用`plot()`绘制出这些顶点的坐标，然后调用`pcolormesh()`绘制这两个四边形。与左边的四边形对应的颜色映射值为0.5，与右边的四边形对应的颜色映射值为0.8，因此一个显示为蓝色，另一个显示为红色。

```
plt.plot(X.ravel(), Y.ravel(), "ko")
plt.pcolormesh(X, Y, Z)
```

```
plt.margins(0.1)
```

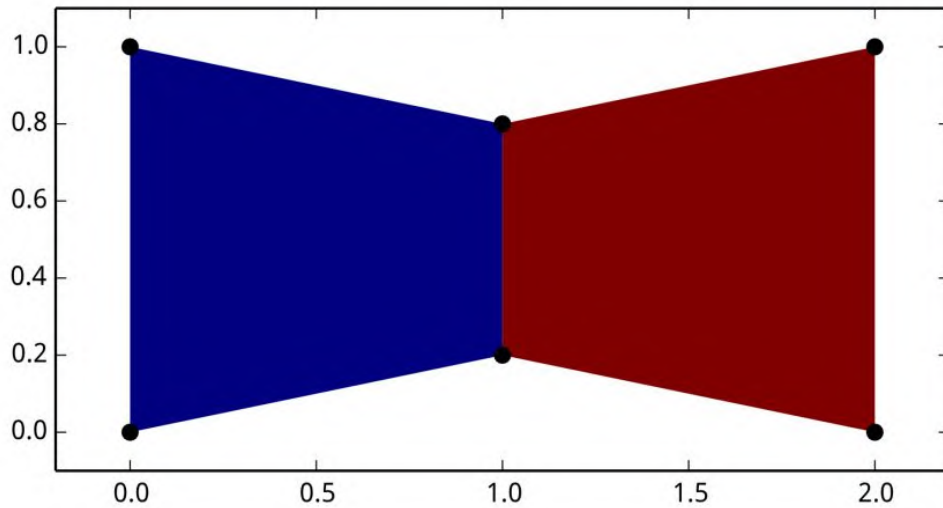


图4-32 演示pcolormesh()绘制的四边形及其填充颜色

在下面的例子中，使用pcolormesh()绘制复数平面上的坐标变换。在图4-33中，左侧的图表显示s平面上的矩形区域，右侧的图表显示通过公式 $z = \frac{2+s}{2-s}$ 坐标变换之后的网格，左侧中的矩形被变换成右侧同颜色的四边形。由于axes[2]和axes[3]中的网格由近4万个四边形组成，为了在输出SVG图像时提高绘图速度，这里将rasterized参数设置为True，这些四边形将作为一幅点阵图像输出到SVG图像中。

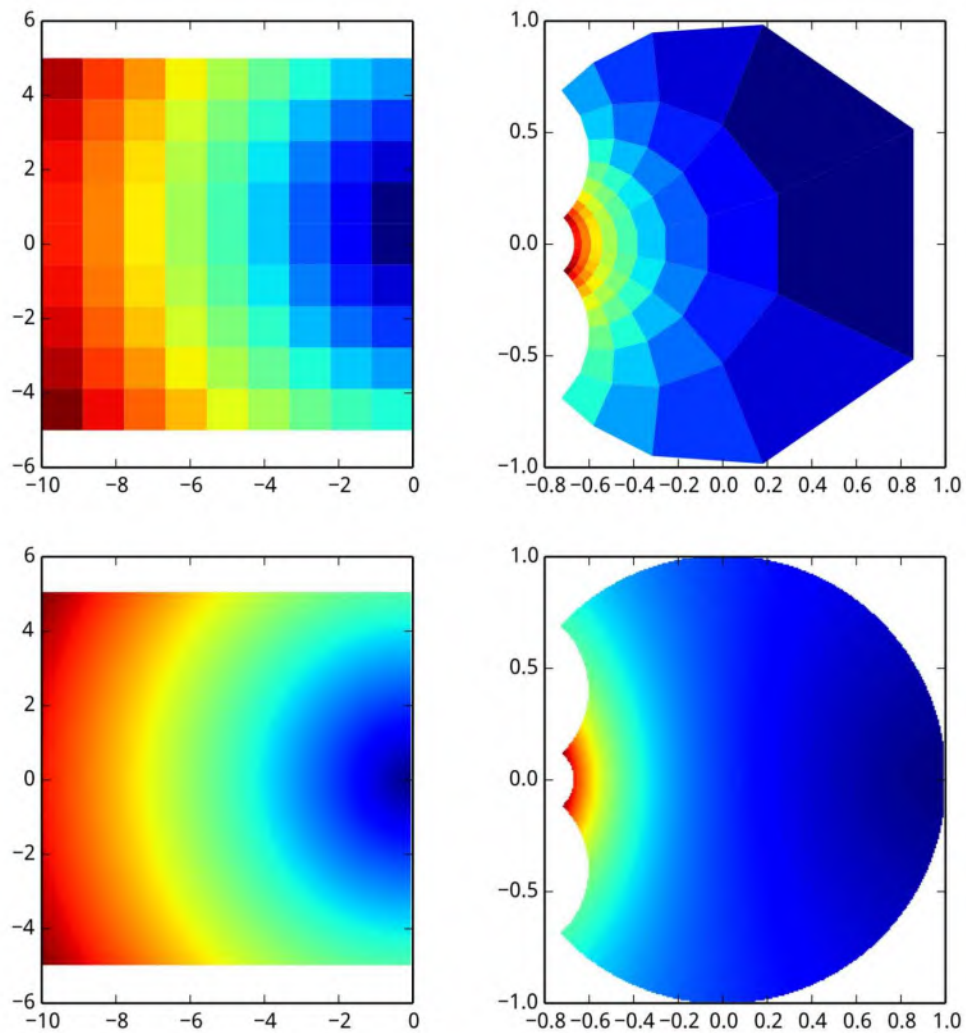


图4-33 使用pcolormesh()绘制复数平面上的坐标变换

```
def make_mesh(n):
    x, y = np.mgrid[-10:0:n*1j, -5:5:n*1j]

    s = x + 1j*y
    z = (2 + s) / (2 - s)

    return s, z

fig, axes = plt.subplots(2, 2, figsize=(8, 8))
```

```

axes = axes.ravel()
for ax in axes:
    ax.set_aspect("equal")

s1, z1 = make_mesh(10)
s2, z2 = make_mesh(200)
axes[0].pcolormesh(s1.real, s1.imag, np.abs(s1))
axes[1].pcolormesh(z1.real, z1.imag, np.abs(s1))
axes[2].pcolormesh(s2.real, s2.imag, np.abs(s2),
rasterized=True)
axes[3].pcolormesh(z2.real, z2.imag, np.abs(s2),
rasterized=True)

```

还可以在极坐标中使用pcolormesh()绘制网格，下面的例子使用mgrid[]创建极坐标中的等间隔网格，然后在projection为polar的子图中绘制这个网格：

```

def func(theta, r):
    y = theta * np.sin(r)
    return np.sqrt(y*y)

T, R = np.mgrid[0:2*np.pi:360j, 0:10:100j]
Z = func(T, R)

ax=plt.subplot(111, projection="polar", aspect=1.)
ax.pcolormesh(T, R, Z, rasterized=True)

```

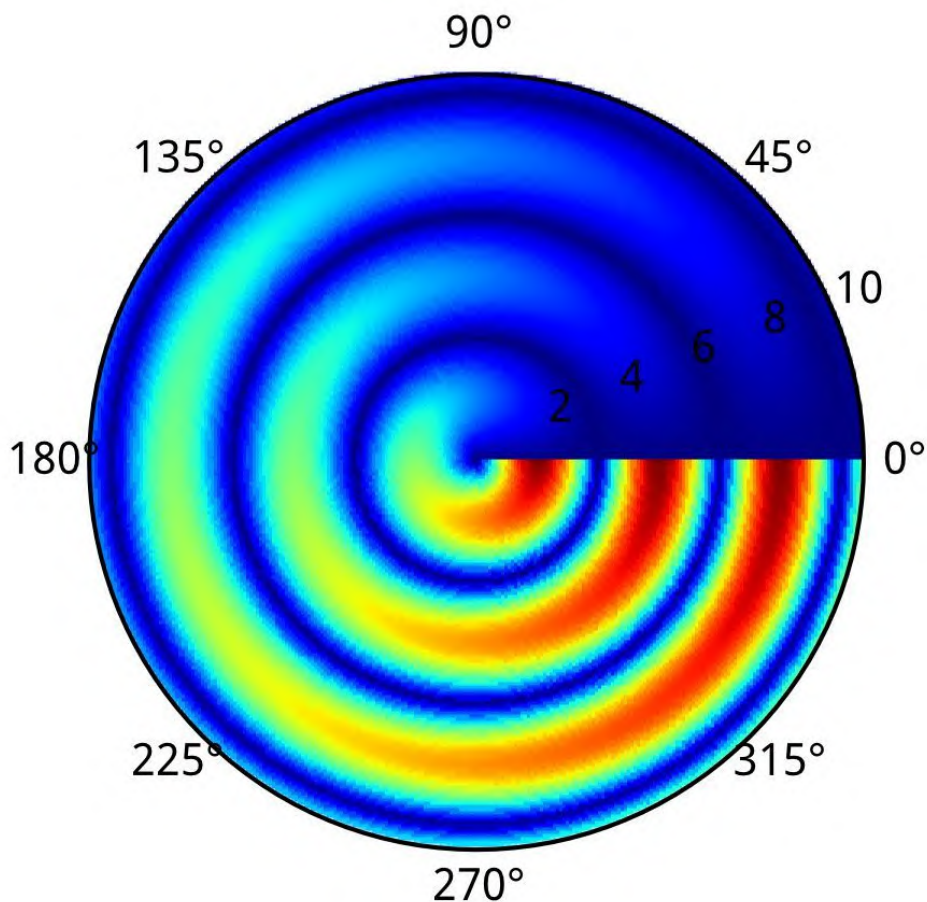


图4-34 使用`pcolormesh()`绘制极坐标中的网格

4.5.8 三角网格

在工业工程设计与分析中，经常将分析对象使用三角网格离散化，然后用有限元法进行模拟。在`matplotlib`中提供了下面的三角网格绘制函数：

- `triplot()`：绘制三角网格的边线。
- `tripcolor()`：与`pcolormesh()`类似，绘制填充颜色的三角网格。

- `tricontour()` 和 `tricontourf()`：绘制三角网格的等高线。

`diffusion.txt` 是使用 FiPy 对二维稳态热传导问题进行有限元模拟的结果。该文件分为三个部分：

- 以 `#points` 开头的部分是一个形状为 $(N_points, 2)$ 的数组，保存 N_points 个点的坐标。
- 以 `#triangles` 开头的部分是一个形状为 $(N_triangles, 3)$ 的数组，保存每个三角形三个顶点在 `points` 数组中的下标。
- 以 `#values` 开头的部分是一个形状为 $(N_triangles, 1)$ 的数组，保存每个三角形对应的温度。

下面的程序将这些数据读入 `data` 字典：

```
with open("diffusion.txt") as f:
    data = {"points": [], "triangles": [], "values": []}
    values = None
    for line in f:
        line = line.strip()
        if not line:
            continue
        if line.startswith("#"):
            values = data[line[1:]]
            continue
        values.append([float(s) for s in line.split()])

data = {key: np.array(data[key]) for key in data}
```

然后就可以调用`tripcolor()`，用三角形网格显示目标区域的温度，结果如图4-35所示。

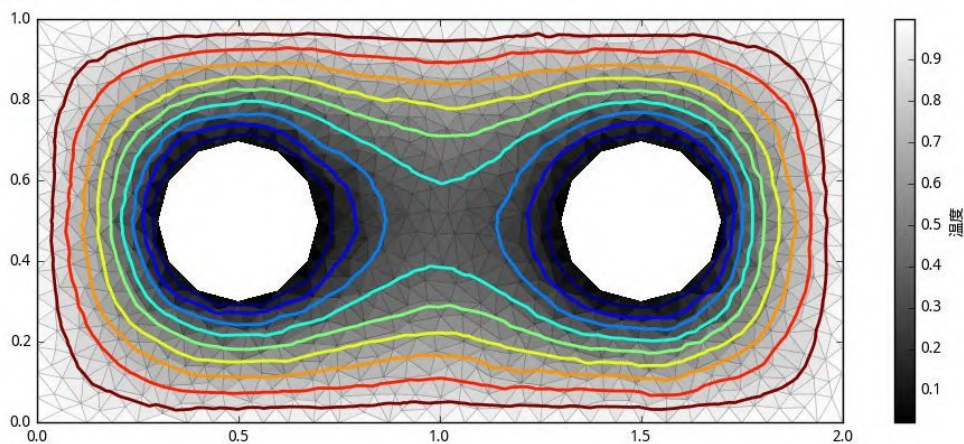


图4-35 使用`tripcolor()`和`tricontour()`绘制三角网格和等值线

❶`tripcolor()`的参数从左到右分别为各点的X轴坐标、Y轴坐标、三角形顶点下标、标量数组。标量数组中的每个值可以与每个顶点对应，也可以与每个三角形对应。在本例中由于`values`的长度与`triangles`的第0轴长度相同，因此每个值与三角形相对应。若标量数组的长度与顶点数相同，则每个三角形对应的值由其三个顶点的平均值决定。

❷调用`tripplot()`绘制所有三角形的边线。❸调用`tricontour()`绘制等高线。由于要求标量数组与三角形顶点对应，而本例中标量数组与三角形对应，因此先计算每个三角形的重心坐标 X_c 和 Y_c ，这样`values`中的每个值就可以与每个三角形的重心对应。在调用`tricontour()`时没有传递三角形顶点下标信息，这时会调用`matplotlib`自带的三角化算法计算出每个三角形对应的顶点。

```
X, Y = data["points"].T
triangles = data["triangles"].astype(int)
```

```

values = data["values"].squeeze()

fig, ax = plt.subplots(figsize=(12, 4.5))
ax.set_aspect("equal")

mapper = ax.tripcolor(X, Y, triangles, values, cmap="gray")
❶

plt.colorbar(mapper, label=u"温度")

plt.triplot(X, Y, triangles, lw=0.5, alpha=0.3, color="k")
❷

Xc = X[triangles].mean(axis=1)
Yc = Y[triangles].mean(axis=1)
plt.tricontour(Xc, Yc, values, 10) ❸

```

4.5.9 箭头图

使用`quiver()`可以用大量的箭头表示矢量场。下面的程序显示 $f(x, y) = xe^{x^2-y^2}$ 的梯度场，结果如图4-36所示。`vec_field(f, x, y)`近似计算函数 f 在 x 和 y 处的偏导数。

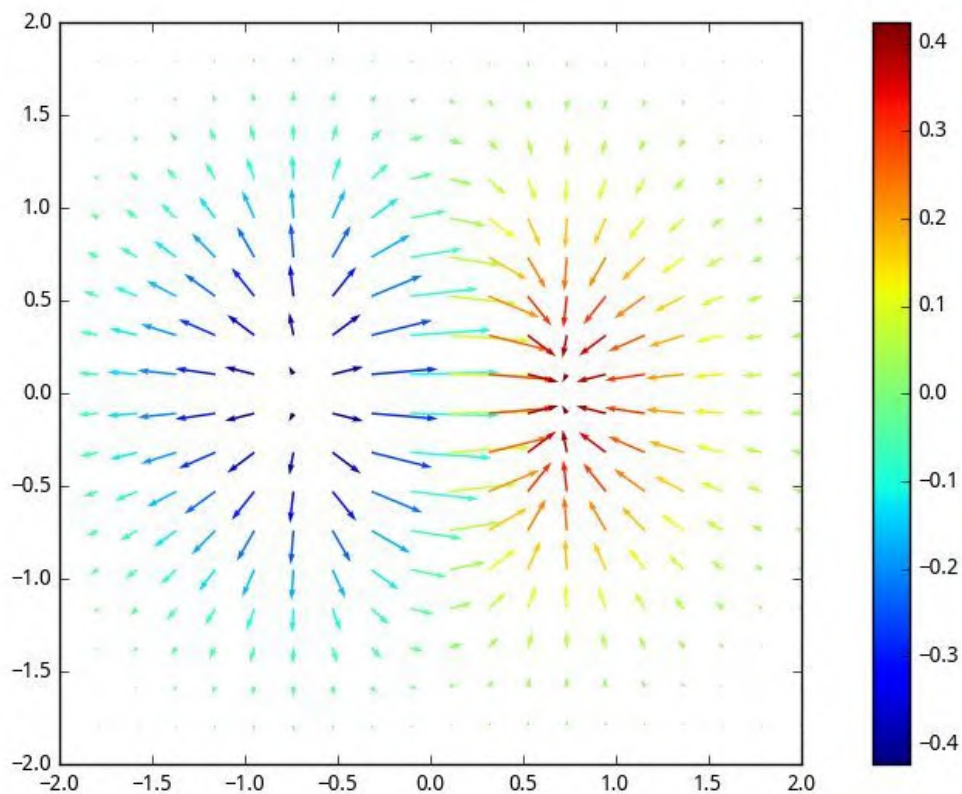


图4-36 用quiver() 绘制矢量场

quiver() 的前5个参数中，X、Y是箭头起点的X轴和Y轴坐标，U、V是箭头方向和大小的矢量，C是箭头对应的值。

```
def f(x, y):  
    return x * np.exp(- x**2 - y**2)  
  
def vec_field(f, x, y, dx=1e-6, dy=1e-6):  
    x2 = x + dx  
    y2 = y + dy  
    v = f(x, y)  
    vx = (f(x2, y) - v) / dx  
    vy = (f(x, y2) - v) / dy
```

```

        return vx, vy

X, Y = np.mgrid[-2:2:20j, -2:2:20j]
C = f(X, Y)
U, V = vec_field(f, X, Y)
plt.quiver(X, Y, U, V, C)
plt.colorbar();
plt.gca().set_aspect("equal")

```

此外，`quiver()` 还提供许多参数来配置箭头的大小和方向：

- 箭头的长度由`scale`和`scale_units`决定。其中`scale`为数值，表示箭头的缩放尺度，而`scale_units`为箭头的长度单位，可选
单 位
有`'width'`、`'height'`、`'dots'`、`'inches'`、`'x'`、`'y'`、`'xy'`等。其中`'width'`、`'height'`为子图的宽和高，`'dots'`和`'inches'`以点和英寸为单位，`'x'`、`'y'`、`'xy'`则以数据坐标系的X轴、Y轴或单位矩形的对角线为单位。箭头的长度按照“UV矢量的长度 * 箭头的长度单位 / 缩放尺度”计算。例如，如果`scale`为2，`scale_units`为`'x'`，而UV矢量的长度为3，则对应的箭头的长度为1.5个X轴的单位长度。
- `width`、`headwidth`、`headlength`和`headaxislength`等参数决定箭头的杆部分粗细、箭头部分的大小以及长度，而`units`参数决定这些参数的单位，可选值与`scale_units`相同。这些参数的含义如图4-37所示。

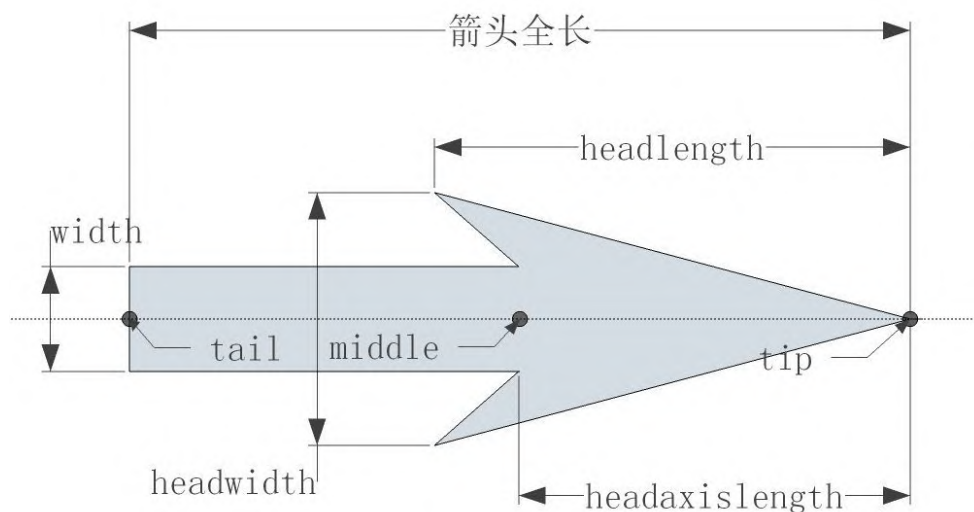


图4-37 quiver箭头的各个参数的含义

- `pivot` 参数决定箭头旋转的中心，可以为'tail'、'middle'、'tip'等值，在图4-37中使用灰色圆点表示这些旋转点。
- `angles`参数决定箭头的方向。正方形可能由于X轴和Y轴的缩放尺度不同而显示为长方形，因此方向有两种计算方式：'uv'和'xy'。其中'uv'只采用U和V的值计算方向，因此若U和V的值相同，则方向为45度；而'xy'在使用U和V计算角度时考虑X轴和Y轴的缩放尺度。

下面通过两个例子帮助读者理解这些参数的用法，如图4-38所示。首先绘制了一条参数曲线，然后沿着该曲线绘制了40个等分曲线的箭头，箭头的方向表示箭头处曲线的切线方向，颜色表示箭头所在处参数的大小。计算部分留给读者自行分析，下面仔细分析这些参数是如何决定箭头的大小和方向的。

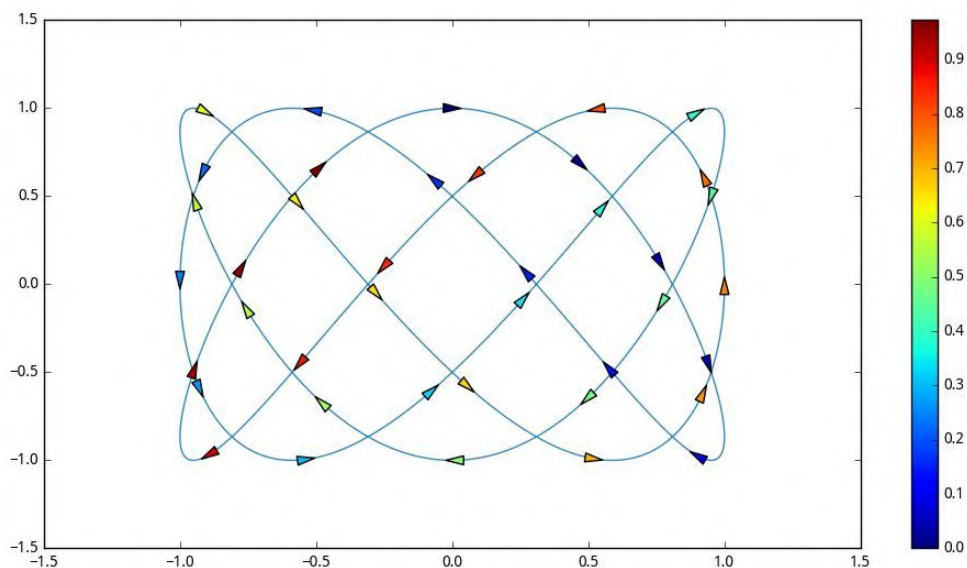


图4-38 使用箭头表示参数曲线的切线方向

箭头的长度和其他尺寸的单位由`scale_units`和`units`决定，在本例中均为`'dots'`，即以像素点为单位。`dx`和`dy`为描述箭头的矢量，长度为1，将`scale`参数设置为`1.0/arrow_size`，这样所有箭头的长度均为`arrow_size`个像素点。箭杆的宽度由`width`参数指定，本例中的宽度为1个像素。而`headwidth`、`headlength`和`headaxislength`等参数决定箭头部分的宽度、长度以及箭头与箭杆接触部分的长度，这些参数为对应长度与箭杆宽度的比例系数。在本例中，由于箭杆宽度为1个像素，因此箭头宽度为`arrow_size * 0.5`个像素，而箭头部分的长度和箭头的长度相同，因此图中的箭头没有箭杆部分。

由于子图的X轴和Y轴的缩放比例不同，因此设置`angles`参数为`"xy"`，这样箭头的方向才能与曲线的切线方向相同。

```
n = 40
arrow_size = 16
t = np.linspace(0, 1, 1000)
```

```

x = np.sin(3*2*np.pi*t)
y = np.cos(5*2*np.pi*t)
line, = plt.plot(x, y, lw=1)

lengths = np.cumsum(np.hypot(np.diff(x), np.diff(y)))
length = lengths[-1]
arrow_locations = np.linspace(0, length, n, endpoint=False)
index = np.searchsorted(lengths, arrow_locations)
dx = x[index + 1] - x[index]
dy = y[index + 1] - y[index]
ds = np.hypot(dx, dy)
dx /= ds
dy /= ds
plt.quiver(x[index], y[index], dx, dy, t[index],
           units="dots", scale_units="dots",
           angles="xy", scale=1.0/arrow_size,
pivot="middle",
           edgecolors="black", linewidths=1,
           width=1, headwidth=arrow_size*0.5,
           headlength=arrow_size, headaxislength=arrow_size,
           zorder=100)

plt.colorbar()
plt.xlim([-1.5, 1.5])
plt.ylim([-1.5, 1.5])

```

还可以用`quiver()`绘制起点和终点的箭头集合。下面的例子绘制神经网络结构示意图，效果如图4-39所示。为了让箭头能够连接两个

神经节点，将scale_units设置为“xy”，将angles设置为“xy”，并且将scale设置为1。这样箭头的长度就为箭头对应的矢量在数据空间中的长度。

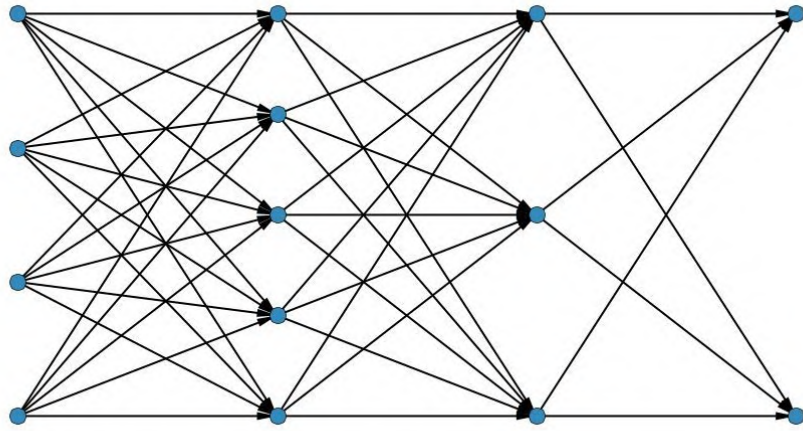


图4-39 使用quiver()绘制神经网络结构示意图

```
levels = [4, 5, 3, 2]
x = np.linspace(0, 1, len(levels))

for i in range(len(levels) - 1):
    j = i + 1
    n1, n2 = levels[i], levels[j]
    y1, y2 = np.mgrid[0:1:n1*1j, 0:1:n2*1j]
    x1 = np.full_like(y1, x[i])
    x2 = np.full_like(y2, x[j])
    plt.quiver(x1, y1, x2-x1, y2-y1,
               angles="xy", units="dots", scale_units="xy",
               scale=1, width=2, headlength=10,
               headaxislength=10, headwidth=4)
```

```
yp = np.concatenate([np.linspace(0, 1, n) for n in levels])
xp = np.repeat(x, levels)
plt.plot(xp, yp, "o", ms=12)
plt.gca().axis("off")
plt.margins(0.1, 0.1)
```

4.5.10 三维绘图

`mpl_toolkits.mplot3d`模块在`matplotlib`的基础上提供了三维作图的功能。由于它使用`matplotlib`的二维绘图功能实现三维图形的绘制工作，因此绘图速度有限，不适合用于大规模数据的三维绘图。如果读者需要更复杂的三维数据可视化功能，请阅读TVTK与Mayavi章节。

下面是绘制三维曲面的程序，程序的输出如图4-40所示。

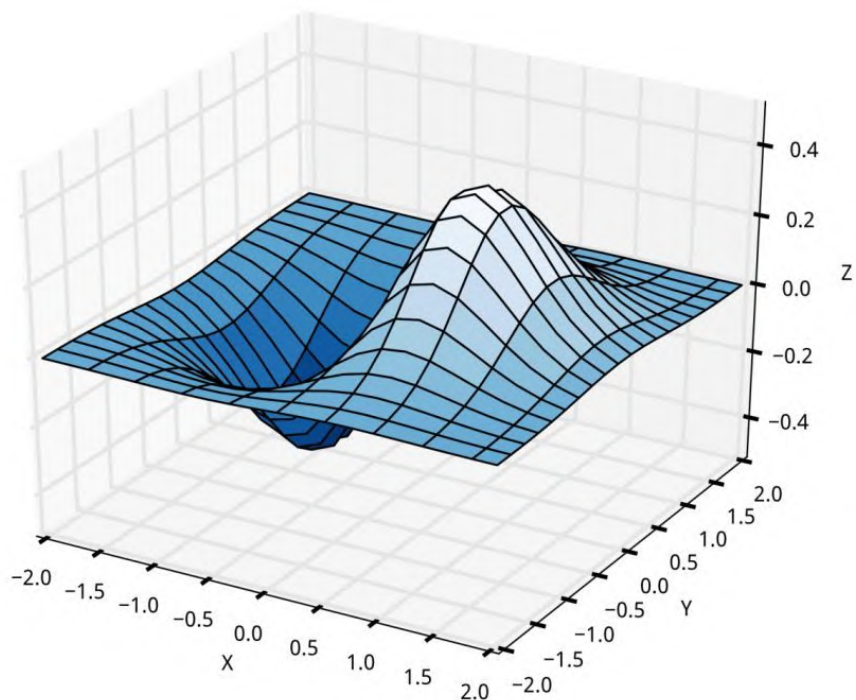


图4-40 使用matplotlib3D绘制的三维曲面图

```
import mpl_toolkits.mplot3d ❶

x, y = np.mgrid[-2:2:20j, -2:2:20j] ❷
z = x * np.exp( - x**2 - y**2)

fig = plt.figure(figsize=(8, 6))
ax = plt.subplot(111, projection='3d') ❸
    ax.plot_surface(x, y, z, rstride=2, cstride=1, cmap =
plt.cm.Blues_r) ❹
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
```


❶首先载入`mplot3d`模块，`matplotlib`中三维绘图相关的功能均在此模块中定义。❷使用`mgrid`创建X-Y平面的网格并计算网格上每点的高度`z`。由于绘制三维曲面的函数要求其X、Y和Z轴的数据都用相同形状的二维数组表示，因此这里不能使用`ogrid`创建。和之前的`imshow()`不同，数组的第0轴可以表示X和Y轴中的任意一个，在本例中第0轴表示X轴，第1轴表示Y轴。

❸在当前图表中创建一个子图，通过`projection`参数指定子图的投影模式为“3d”，这样`subplot()`将返回一个用于三维绘图的`Axes3D`子图对象。

投影模式

投影模式决定了点从数据坐标转换为屏幕坐标的方式。可以通过下面的语句获得当前有效的投影模式的名称：

```
>>> from matplotlib import projections
>>> projections.get_projection_names()
['3d', 'aitoff', 'hammer', 'lambert', 'mollweide',
'polar', 'rectilinear']
```

只有在载入`mplot3d`模块之后此列表中才会出现‘3d’投影模式。‘aitoff’、‘hammer’、‘lambert’、‘mollweide’等均为地图投影，‘polar’为极坐标投影，‘rectilinear’则是默认的直线投影模式。

④调用Axes3D对象的`plot_surface()`绘制三维曲面图。其中参数`x`、`y`、`z`都是形状为(20, 20)的二维数组。数组`x`和`y`构成了X-Y平面上的网格，而数组`z`则是网格上各点在曲面上的取值。通过`cmap`参数指定值和颜色之间的映射，即曲面上各点的高度值与其颜色的对应关系。`rstride`和`cstride`参数分别是数组的第0轴和第1轴的下标间隔。对于很大的数组，使用较大的间隔可以提高曲面的绘制速度。

除了绘制三维曲面之外，Axes3D对象还提供了许多其他的三维绘图方法。请读者在官方网站查看各种三维绘图的演示程序。

4.6 matplotlib技巧集



与 本 节 内 容 对 应 的 Notebook 为 : 04-matplotlib/matplotlib-600-tips.ipynb。

作为本章的最后一节，让我们介绍一些比较特别的用法。

4.6.1 使用agg后台在图像上绘图

matplotlib绘制的图表十分细腻，这是因为它的后台绘图库是用C++开发的高质量反锯齿二维绘图库：Anti-Grain Geometry (AGG)。

如果想绘制一些二维图形，但不需要matplotlib的图表功能，我们可以直接在内存中绘制图像，然后将其转换成NumPy数组。

下面的代码载入RendererAgg(画布)，并创建一个长宽都是250个像素的RendererAgg对象，其第三个参数为DPI，该参数不影响画布的大小。其buffer_rgba()方法获得画布中保存绘图结果的缓存，通过frombuffer()将该缓存转换为NumPy数组，并按照画布的大小调用reshape()。最后得到的数组arr的形状为(250, 250, 4)，其中第2轴的4表示画布有4个通道：红、绿、蓝、透明。

```
import numpy as np
from matplotlib.backends.backend_agg import RendererAgg

w, h = 250, 250
renderer = RendererAgg(w, h, 90)
buf = renderer.buffer_rgba()
arr = np.frombuffer(buf, np.uint8).reshape(h, w, -1)
print arr.shape
(250, 250, 4)
```

RendererAgg对象提供了一些draw_*()方法用于在画布上绘图，例如下面的代码首先创建一个Path对象，然后调用renderer.draw_path()在画布上绘制该Path对象，如图4-41所示。其第一个参数为一个GraphicsContextBase对象，用来设置绘图时的一些属性，例如线宽、线条颜色等。第三个参数是一个坐标变换对象，在本例中使用恒等变换，第4个参数为路径的填充颜色。

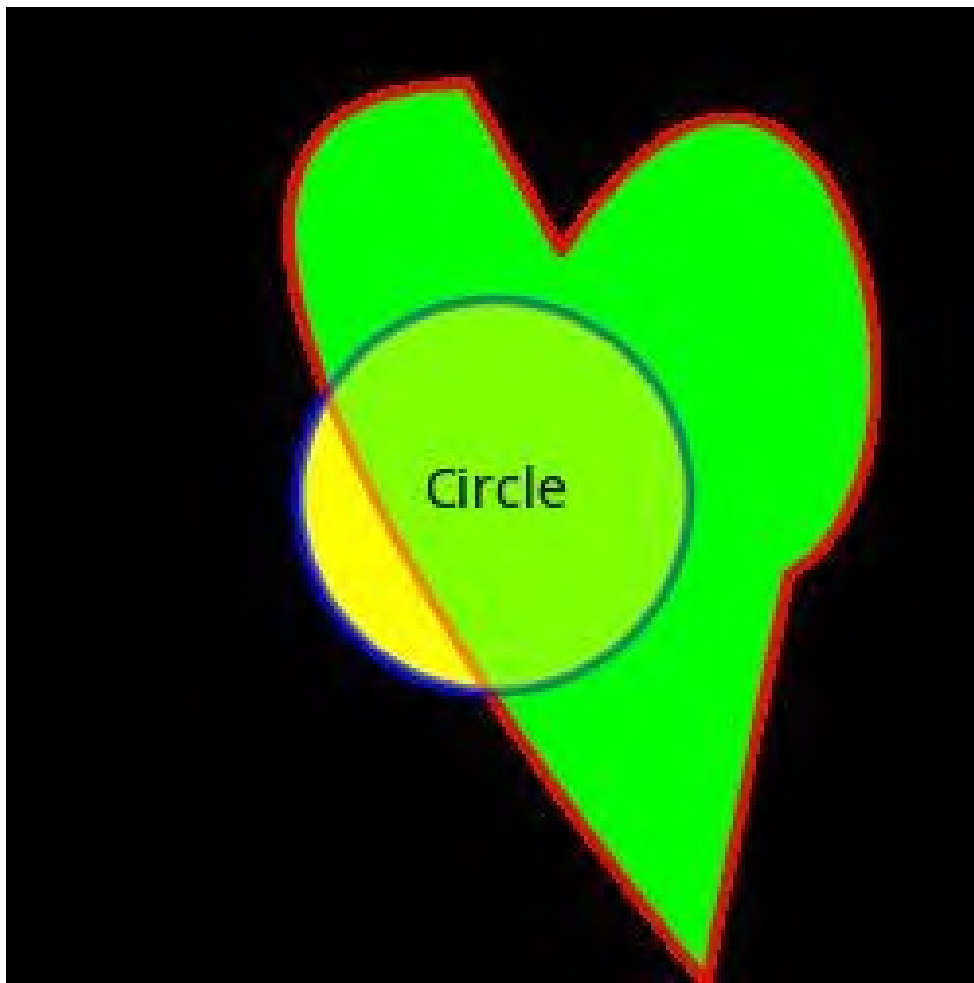


图4-41 直接使用RendererAgg绘图

```
from matplotlib.path import Path
from matplotlib import transforms
```

```
path_data = [
    (Path.MOVETO, (179, 1)),
    (Path.CURVE4, (117, 75)),
    (Path.CURVE4, (12, 230)),
    (Path.CURVE4, (118, 230)),
    (Path.LINETO, (142, 187)),
    (Path.CURVE4, (210, 290)),
```

```

        (Path.CURVE4, (250, 132)),
        (Path.CURVE4, (200, 105)),
        (Path.CLOSEPOLY, (179, 1)),
    ]

    code, points = zip(*path_data)
    path = Path(points, code)

    gc = renderer.new_gc()
    gc.set_linewidth(2)
    gc.set_foreground((1, 0, 0))
    gc.set_antialiased(True)

    renderer.draw_path(gc, path,
transforms.IdentityTransform(), (0, 1, 0))

```

也可以使用matplotlib中提供的Artist对象绘图。下面首先创建一个Circle对象和一个Text对象，然后调用它们的draw()方法在画布上绘图。由于Text对象在绘图时需要获取画布的dpi属性，因此在调用draw()之前先将其figure属性设置为renderer。

```

from matplotlib.patches import Circle
from matplotlib.text import Text

c = Circle((w/2, h/2), 50, edgecolor="blue",
facecolor="yellow", linewidth=2, alpha=0.5)
c.draw(renderer)

```

```
text = Text(w/2, h/2, "Circle", va="center", ha="center")
text.figure = renderer
text.draw(renderer)
```

为了在IPython Notebook中显示arr所表示的图像，可以调用pypolt.imshow()。其第一个参数可以是文件名或者拥有文件接口的对象，这里使用BytesIO对象将PNG图像的内容保存在png_buf中。然后使用IPython的display_png()显示该内存中的PNG图像。

```
from io import BytesIO
from IPython.display import display_png
png_buf = BytesIO()
plt.imshow(arr, format="png")
display_png(png_buf.getvalue(), raw=True)
```

本书提供了一个可以在图像上绘图的ImageDrawer类。下面使用ImageDrawer在图像上绘制标记、文字、直线、圆形、矩形以及椭圆，并使用本书提供的%array_image魔法命令将结果图像显示在Notebook中，结果如图4-42所示。

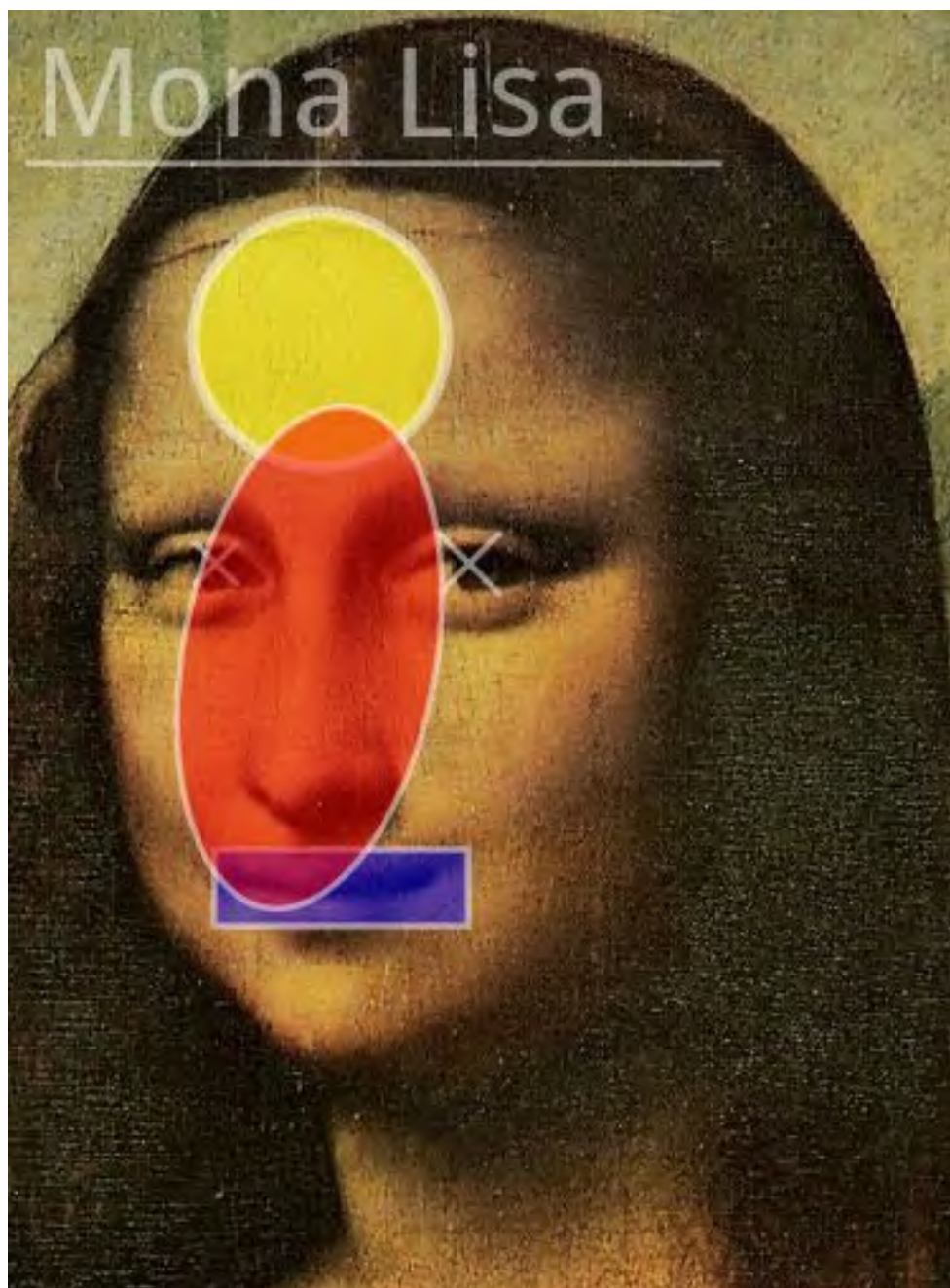


图4-42 使用本书提供的ImageDrawer在图像上绘图

ImageDrawer的reverse参数决定Y轴的方向，默认值为True，表示Y轴方向向下，和图像的像素坐标系方向相同，False表示Y轴方向向上，和数学上的笛卡尔坐标系的定义相同。



`scipy2.matplotlib.ImageDrawer`: 使用`RendererAgg`直接在图像上绘图, 以方便用户在图像上标注信息。

```
from scipy2.matplotlib import ImageDrawer
img = plt.imread("vinci_target.png")
drawer = ImageDrawer(img)
drawer.set_parameters(lw=2, color="white", alpha=0.5)
drawer.line(8, 60, 280, 60)
drawer.circle(123, 130, 50, facecolor="yellow", lw=4)
drawer.markers("x", [82, 182], [218, 218], [50, 100])
drawer.rectangle(81, 330, 100, 30, facecolor="blue")
drawer.text(10, 50, u"Mona Lisa", fontsize=40)
drawer.ellipse(119, 255, 200, 100, 100, facecolor="red")
%array_image drawer.to_array()
```

4.6.2 响应鼠标与键盘事件



为了在Notebook中执行本节代码, 需要启动GUI事件处理线程, 例如执行`%gui qt`和`%matplotlib qt`, 将启动Qt的GUI线程, 并将`Qt4Agg`设置为`matplotlib`的绘图后台。如果希望切换到嵌入模式, 可以再执行`%matplotlib inline`。

界面中的事件绑定都是通过`Figure.canvas.mpl_connect()`进行的，它的第一个参数为事件名，第二个参数为事件响应函数，当指定的事件发生时，将调用指定的函数。

```
%gui qt
%matplotlib qt
```

1. 键盘事件

下面的程序响应键盘按键，并输出按键的值：



`scipy2.matplotlib.key_event_show_key`：显示触发键盘按键事件的按键名称。

```
import sys

fig, ax = plt.subplots()

def on_key_press(event):
    print event.key
    sys.stdout.flush()

fig.canvas.mpl_connect('key_press_event', on_key_press)

control
```

```
ctrl+a
alt
alt+l
```

可以通过mpl_connect的帮助文档查看所有的事件名称。表4-8列出了其支持的所有事件名：

表4-8 支持的事件名及含义

事件名	含义
button_press_event	按下鼠标按键
button_release_event	释放鼠标按键
draw_event	界面重新绘制
key_press_event	按下键盘上的按键
key_release_event	释放键盘按键
motion_notify_event	鼠标移动
pick_event	鼠标点选绘图对象
scroll_event	鼠标滚轴事件
figure_enter_event	鼠标移进图表
figure_leave_event	鼠标移出图表
axes_enter_event	鼠标移进子图
axes_leave_event	鼠标移出子图

close_event	关闭图表
-------------	------

当前所有注册的响应函数可以通过 `Figure.canvas.callbacks.callbacks` 查看。下面的程序输出所有的事件响应函数，响应函数的执行按照显示的顺序进行，可以看出除了我们绑定的响应函数之外，`matplotlib` 还绑定了处理快捷键的键盘响应函数。

```
for key, funcs in
fig.canvas.callbacks.callbacks.iteritems():
    print key
    for cid, wrap in sorted(funcs.items()):
        func = wrap.func
        print " {0}:{1}.{2}".format(cid,
func.__module__, func)
    key_press_event
        3:matplotlib.backend_bases.<function key_press at
0x093A4C30>
        6:__main__.<function on_key_press at 0x09B05FB0>
    motion_notify_event
        4:matplotlib.backend_bases.<function mouse_move at
0x093A4FB0>
    scroll_event
        2:matplotlib.backend_bases.<function pick at
0x093A40F0>
    button_press_event
```

```
1:matplotlib.backend_bases.<function pick at
0x093A40F0>
```

下面的程序通过键盘按键修改曲线的颜色。由于某些按键与默认的快捷键重复，因此这里通过`mpl_disconnect()`取消默认快捷键的响应函数的绑定，其参数为调用`mpl_connect()`时所返回的整数。默认快捷键的响应函数对应的整数可以通过`Figure.canvas.manager.key_press_handler_id`获得。

❶在响应函数中通过`line.set_color()`修改曲线的颜色，❷并调用`Figure.canvas.draw_idle()`重新绘制整个图表。



`scipy2.matplotlib.key_event_change_color`: 通过按键修改曲线的颜色。

```
fig, ax = plt.subplots()
x = np.linspace(0, 10, 1000)
line, = ax.plot(x, np.sin(x))

def on_key_press(event):
    if event.key in 'rgbcmyk':
        line.set_color(event.key) ❶
    fig.canvas.draw_idle() ❷
```

```
fig.canvas.mpl_disconnect(fig.canvas.manager.key_press_handler_  
id)  
  
fig.canvas.mpl_connect('key_press_event', on_key_press)
```

2. 鼠标事件

当鼠标在子图范围内产生动作时，将触发鼠标事件。鼠标事件分为三种：

- 'button_press_event'：鼠标按键按下时触发。
- 'button_release_event'：鼠标按键释放时触发。
- 'motion_notify_event'：鼠标移动时触发。

鼠标事件的相关信息可以通过event对象的属性获得：

- name：事件名。
- button：鼠标按键，1、2、3表示左中右按键，None表示无按键。
- x, y：鼠标在图表中的像素坐标。
- xdata, ydata：鼠标在数据坐标系中的坐标。

下面的程序显示了鼠标事件的各种信息：



scipy2.matplotlib.mouse_event_show_info：显示子图中的鼠标事件的各种信息。

```

import sys

fig, ax = plt.subplots()

text = ax.text(0.5, 0.5, "event", ha="center", va="center",
fontdict={"size":20})

def on_mouse(event):
    global e
    e = event

    info = "{}\nButton:{}\nFig x,y:{}, {}\nData x,y:
{:3.2f}, {:3.2f}".format(
        event.name, event.button, event.x, event.y,
event.xdata, event.ydata)

    text.set_text(info)

    fig.canvas.draw()

fig.canvas.mpl_connect('button_press_event', on_mouse)
fig.canvas.mpl_connect('button_release_event', on_mouse)
fig.canvas.mpl_connect('motion_notify_event', on_mouse)

```

程序的执行结果如图4-43所示：

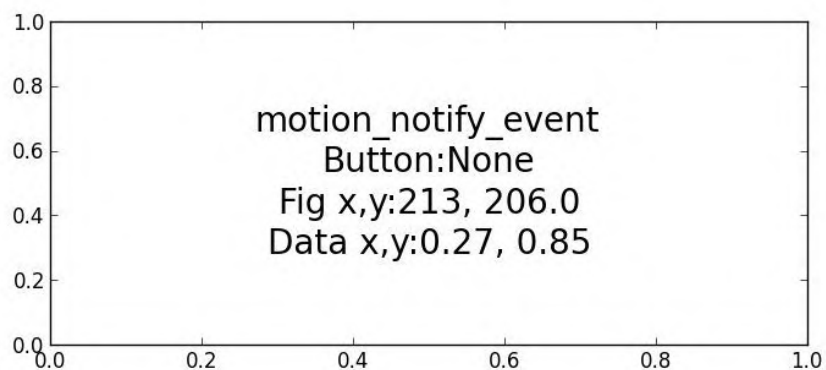


图4-43 显示鼠标事件信息

下面的例子通过响应上述三个鼠标事件，实现图表中形状的移动。我们将所有的事件响应封装到PatchMover类中，它有三个内部使用的属性：

- selected_patch：保存当前被选中的Patch对象。
- start_mouse_pos：保存Patch对象被选中时鼠标在子图中的坐标。
- start_patch_pos：保存Patch对象被选中时在子图中的坐标。

❶在on_press()中，对子图中的所有Patch对象进行循环判断，这里采用zorder属性排序之后的逆序循环，保证在最上层的Patch对象优先被选中。❷通过Patch.contains_point()判断当前的鼠标坐标是否在Patch对象之内，注意这里需要使用图表坐标系中的像素坐标。一旦判断鼠标在当前的Patch对象之中，则保存当前Patch对象，以及相关的坐标信息。注意我们保存数据坐标系的坐标，因为Patch对象的移动是在数据坐标系中进行的。

❸在on_motion()中，通过当前的鼠标坐标计算被选中Patch对象的当前位置，所有的计算都在数据坐标系中进行。❹调用Figure.canvas.draw_idle()重新绘制整个图表。

❺在on_release()中，取消被选中的Patch对象。



scipy2.matplotlib.mouse_event_move_polygon: 演示通过鼠标移动Patch对象。

```
from numpy.random import rand, randint
from matplotlib.patches import RegularPolygon

class PatchMover(object):
    def __init__(self, ax):
        self.ax = ax

        self.selected_patch = None
        self.start_mouse_pos = None
        self.start_patch_pos = None

        fig = ax.figure

        fig.canvas.mpl_connect('button_press_event',
self.on_press)

        fig.canvas.mpl_connect('button_release_event',
self.on_release)

        fig.canvas.mpl_connect('motion_notify_event',
self.on_motion)

    def on_press(self, event): ❶
        patches = self.ax.patches[:]
```



```

        patches.sort(key=lambda patch:patch.get_zorder())
        for patch in reversed(patches):
            if patch.contains_point((event.x, event.y)): ❷
                self.selected_patch = patch
                self.start_mouse_pos =
np.array([event.xdata, event.ydata])
                self.start_patch_pos = patch.xy
                break

    def on_motion(self, event): ❸
        if self.selected_patch is not None:
            pos = np.array([event.xdata, event.ydata])
            self.selected_patch.xy = self.start_patch_pos + pos -
self.start_mouse_pos
            self.ax.figure.canvas.draw_idle() ❹

    def on_release(self, event): ❺
        self.selected_patch = None

fig, ax = plt.subplots()
ax.set_aspect("equal")
for i in range(10):
    poly = RegularPolygon(rand(2), randint(3, 10), rand() *
0.1 + 0.1, facecolor=rand(3),
                                zorder=randint(10, 100))
    ax.add_patch(poly)

```

```
ax.relim()

ax.autoscale()

pm = PatchMover(ax)

plt.show()
```

3. 点选事件

上面通过`Patch.contains_point()`判断鼠标的点击事件是否发生在`Patch`对象内部。但是对于曲线这样的对象，没有类似的判断方法。为了响应鼠标点选图形的事件，可以设置图形对象的`picker`属性为`True`，然后在`'pick_event'`事件响应函数中加以处理。

在下面的例子中，创建了一个`Rectangle`对象和一个`Line2D`对象，并分别设置其`picker`属性，表示这两个图形对象支持点选事件。由于`Rectangle`占据一块面积，因此只需要设置为`True`即可；而对于表示曲线的`Line2D`对象，为了方便点选，在调用`plot()`创建曲线时通过`picker`参数设置了一个容错值，鼠标坐标到曲线的距离小于8.0就认为该曲线被点选。

在点选事件处理函数`on_pick()`中，我们修改曲线的线宽和矩形的填充颜色。



`scipy2.matplotlib.pick_event_demo`: 演示绘图对象的点选事件。

```
fig, ax = plt.subplots()

    rect = plt.Rectangle((np.pi, -0.5), 1, 1,
fc=np.random.random(3), picker=True)

ax.add_patch(rect)

x = np.linspace(0, np.pi*2, 100)
y = np.sin(x)

line, = plt.plot(x, y, picker=8.0)

def on_pick(event):
    artist = event.artist

    if isinstance(artist, plt.Line2D):
        lw = artist.get_linewidth()
        artist.set_linewidth(lw % 5 + 1)
    else:
        artist.set_fc(np.random.random(3))

    fig.canvas.draw_idle()

fig.canvas.mpl_connect('pick_event', on_pick)
```

4. 实时高亮显示曲线

为了方便用户分辨多条曲线，可以通过响应鼠标事件，当鼠标靠近某条曲线时，高亮显示该曲线。下面的例子中，我们采用面向对象的设计模式，将鼠标事件处理函数包装在CurveHighLighter内部。它

的alpha参数为非高亮显示时曲线的透明度，alpha为1表示完全不透明，0表示完全透明；linewidth参数为高亮显示曲线时曲线的宽度。

❶ 绑定鼠标移动事件motion_notify_event到on_move()方法之上。当鼠标在图表中移动时将调用on_move()方法。

❷ 所有高亮显示的逻辑判断都在highlight()中进行，其参数target为高亮显示的Line2D对象，如果为None表示取消高亮显示。

- 当无高亮显示时，将所有曲线的alpha属性和linewidth设置为1。
- 当有高亮显示时，将高亮显示的曲线的alpha属性设置为1，将linewidth设置为3；将非高亮显示的曲线的alpha属性设置为0.3，将linewidth设置为1。
- 若有任意一条曲线的属性被修改，则需要调用Figure.canvas.draw_idle()重新绘制整个图表，它会等到空闲时重绘。

❸ 子图中的所有曲线(Line2D对象)都在其属性lines列表中，对此列表中的Line2D对象进行循环。Line2D.contains()可以用于判断事件是否发生在此对象内部。当鼠标坐标离曲线的像素距离小于其pickradius属性时，将会判断事件发生在Line2D对象之上。contains()返回一个有两个元素的元组，其第0个元素为判断结果，第1个元素为保存详细信息的字典。



scipy2.matplotlib.mouse_event_highlight_curve: 鼠标移到曲线之上时高亮显示该曲线。

```
import matplotlib.pyplot as plt
import numpy as np

class CurveHighLighter(object):

    def __init__(self, ax, alpha=0.3, linewidth=3):
        self.ax = ax
        self.alpha = alpha
        self.linewidth = 3

        ax.figure.canvas.mpl_connect('motion_notify_event',
self.on_move) ❶

    def highlight(self, target): ❷
        need_redraw = False
        if target is None:
            for line in self.ax.lines:
                line.set_alpha(1.0)
                if line.get_linewidth() != 1.0:
                    line.set_linewidth(1.0)
                    need_redraw = True
        else:
            for line in self.ax.lines:
                lw = self.linewidth if line is target else
```

```

        if line.get_linewidth() != lw:
            line.set_linewidth(lw)
            need_redraw = True
alpha = 1.0 if lw == self.linewidth else self.alpha
        line.set_alpha(alpha)

    if need_redraw:
        self.ax.figure.canvas.draw_idle()

    def on_move(self, evt):
        ax = self.ax
        for line in ax.lines:
            if line.contains(evt)[0]: ❸
                self.highlight(line)
                break
        else:
            self.highlight(None)

fig, ax = plt.subplots()
x = np.linspace(0, 50, 300)

from scipy.special import jn

for i in range(1, 10):
    ax.plot(x, jn(i, x))

ch = CurveHighLighter(ax)

```

图4-44为实际运行效果，当鼠标悬停到某条曲线之上时，该曲线加粗显示，而其他曲线则变为半透明显示。

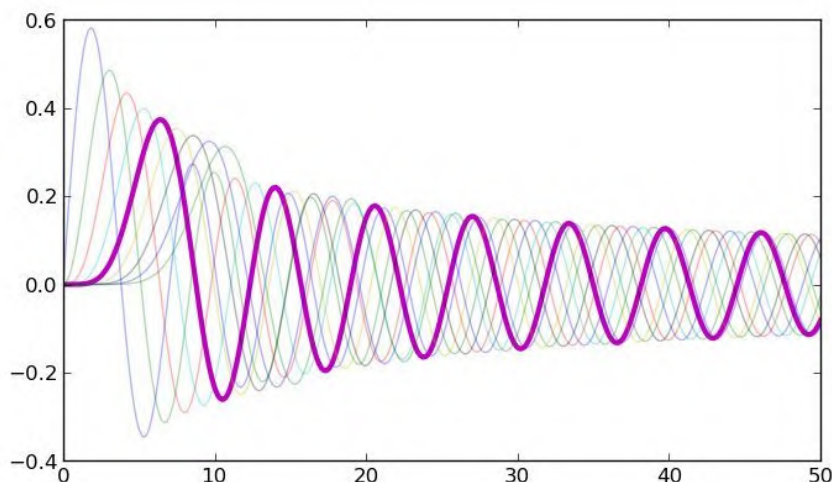


图4-44 高亮显示鼠标悬停曲线

4.6.3 动画

通过修改图形元素的各种属性并重新绘制图表，可以实现简单的动画效果。在下面的例子中：❶首先创建一个50毫秒的定时器timer，并调用其add_callback()添加定时事件，其第一个参数为定时事件发生时调用的函数，第二个参数为传递给此函数的对象。由于我们需要对图表中曲线的数据进行修改，因此需要将Line2D对象line传递给update_data()。❷调用Line2D.set_ydata()设置曲线的Y轴数据。❸调用Figure.canvas.draw()重绘整张图表。

```
import numpy as np
import matplotlib.pyplot as plt
```

```

fig, ax = plt.subplots()
x = np.linspace(0, 10, 1000)
line, = ax.plot(x, np.sin(x), lw=2)

def update_data(line):
    x[:] += 0.1
    line.set_ydata(np.sin(x)) ❷
    fig.canvas.draw()          ❸

timer = fig.canvas.new_timer(interval=50) ❶
timer.add_callback(update_data, line)
timer.start()

```

1. 使用缓存快速重绘图表

然而Figure.canvas.draw()的绘图速度较慢，为了提高重绘速度，可以快速重绘图表中的静态元素，只更新有动态效果的元素。在下面的例子中：

❶ 创建绘图元素时，设置其animated属性为True。❷ 在调用Figure.canva.draw()重绘整个图表时，会忽略所有animated为True的对象。❸ 这时所有的静态元素都已经绘制完毕，调用Figure.canvas.copy_from_bbox()保存子图对象对应区域中的图像信息到background中。子图对象在图表对象中的位置和大小可以通过其bbox属性获得。

在时钟事件处理函数中，❹首先调用 `Figure.canvas.restore_region()` 恢复所保存的图像信息，这相当于擦除所有动态元素，重新绘制了所有的静态元素。❺在更新曲线的Y轴数据之后，调用子图对象的 `draw_artist()` 在Canvas对象中绘制曲线，此时Canvas对象中已经是一幅完整的图表的图像了。❻调用 `Figure.canvas.blit()` 将Canvas中指定区域的内容绘制到屏幕上。

```
fig, ax = plt.subplots()
x = np.linspace(0, 10, 1000)
line, = ax.plot(x, np.sin(x), lw=2, animated=True) ❶

fig.canvas.draw() ❷
background = fig.canvas.copy_from_bbox(ax.bbox) ❸

def update_data(line):
    x[:] += 0.1
    line.set_ydata(np.sin(x))
    fig.canvas.restore_region(background) ❹
    ax.draw_artist(line) ❺
    fig.canvas.blit(ax.bbox) ❻

timer = fig.canvas.new_timer(interval=50)
timer.add_callback(update_data, line)
timer.start()
```

2. animation模块

通过前面两个简单的例子，我们了解了在matplotlib中制作动画的原理，不过在实际使用中，一般会使用animation模块制作动画效果。例如FuncAnimation对象将定期调用用户定义的函数来更新图表中的元素。

❶创建曲线对象时，设置animated参数为True。❷在动画回调函数update_line()中设置所有动画元素的数据，它有一个参数为当前的显示帧数，这里使用帧数修改波形的相位，并返回一个包含所有动画元素的序列。❸创建FuncAnimation对象定时调用update_line()，interval参数为每秒的帧数，blit为True表示使用缓存加速每帧图像的绘制。frames参数设置最大帧数，update_line()的帧数参数将在0到99之间循环变化。

```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.animation import FuncAnimation

fig, ax = plt.subplots()

x = np.linspace(0, 4*np.pi, 200)
y = np.sin(x)
line, = ax.plot(x, y, lw=2, animated=True) ❶

def update_line(i):
    y = np.sin(x + i*2*np.pi/100)
    line.set_ydata(y)
    return [line] ❷
```

```
ani = FuncAnimation(fig, update_line, blit=True,
interval=25, frames=100) ❸
```

若想将动画保存成动画文件，可以调用如下方法：



matplotlib会使用系统中安装的视频压缩软件（如ffmpeg.exe）生成视频文件。请读者确认视频压缩软件的可执行文件的路径是否在PATH环境变量中。

```
ani.save('sin_wave.mp4', fps=25)
```

4.6.4 添加GUI面板

在matplotlib的主页中可以找到将图表嵌入各种主流界面库的演示程序，这些程序都是首先创建一个GUI窗口，然后将图表作为控件嵌入窗口中。本节介绍一种更加简洁的方法：为matplotlib图表窗口添加GUI控件的控制面板。



scipy2.matplotlib.gui_panel：提供了TK与QT界面库的滑块控件面板类TkSliderPanel和QtSliderPanel。tk_panel_demo.py和qt_panel_demo.py为其演示程序。

下面是使用 TkSliderPanel 的一个例子。❶ 首先调用 `matplotlib.use()` 将后台界面库设置为“TkAgg”，该语句必须在 `matplotlib` 的其他函数之前被调用。❷ 我们希望绘制的曲线函数由 `exp_sin()` 定义，其第一个参数为自变量，其余参数为函数中的各个系数。❸ `update()` 函数接收新的系数，并调用 `exp_sin()` 计算新系数对应的函数值。❹ 然后调用 `line.set_data()` 更新曲线的数据。在更新子图的显示范围之后，❺ 调用 `fig.canvas.draw_idle()` 重新绘制整个图表。

❻ 创建一个 `TkSliderPanel` 对象，它的第一个参数为图表对象，第二个参数定义控制面板中的滑标名称以及取值范围，第三个参数为回调函数。当某个滑标控件的值发生变化时，将调用此函数。❼ 最后调用其 `set_parameters()` 设置各个滑标控件的初始值。程序的运行结果如图4-45(左)所示，而右图则是采用 `QtSliderPanel` 时的界面截图。❽ 为了在 Notebook 中显示 TK 界面库的窗口，需要执行 `%gui tk` 魔法命令，然后调用 `fig.show()` 显示图表窗口。如果是在单独的进程中运行该程序，则需要调用 `pl.show()` 以显示窗口。

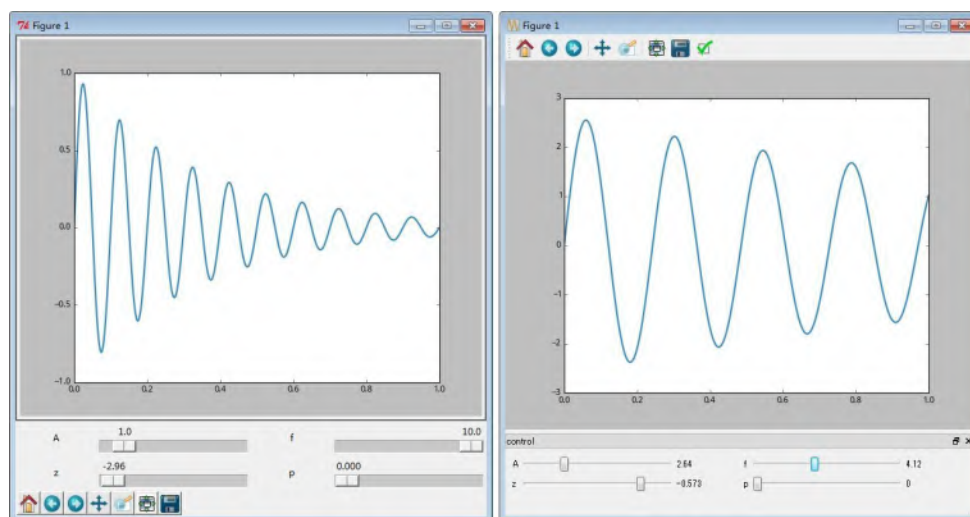


图4-45 为图表添加GUI控件：TK(左图)和QT(右图)

由于程序涉及GUI库的用法，限于篇幅这里不再详细叙述，请感兴趣的读者查看本书提供的相关源代码。

```
%gui tk

import numpy as np
import matplotlib

matplotlib.use("TkAgg") ❶

import pylab as pl

def exp_sin(x, A, f, z, p): ❷
    return A * np.sin(2 * np.pi * f * x + p) * np.exp(z *
x)

fig, ax = pl.subplots()

x = np.linspace(1e-6, 1, 500)
pars = {"A":1.0, "f":2, "z":-0.2, "p":0}
y = exp_sin(x, **pars)

line, = pl.plot(x, y)

def update(**kw): ❸
    y = exp_sin(x, **kw)
    line.set_data(x, y) ❹
    ax.relim()
```

```
ax.autoscale_view()

fig.canvas.draw_idle() ❸

from scipy2.matplotlib.gui_panel import TkSliderPanel

panel = TkSliderPanel(fig, ❹
                        [("A", 0, 10), ("f", 0, 10), ("z",
-3, 0), ("p", 0, 2*np.pi)],
                        update, cols=2, min_value_width=80)
panel.set_parameters(**pars) ❺
fig.show() ❻
```


第5章 Pandas-方便的数据分析库

NumPy虽然提供了方便的数组处理功能，但它缺少数据处理、分析所需的许多快速工具。Pandas基于NumPy开发，提供了众多更高级的数据处理功能。Pandas的帮助文档十分全面，因此本章主要介绍Pandas的一些基本概念和帮助文档中说明不够详细的部分。希望读者在阅读本章之后能更容易阅读官方文档。

```
import pandas as pd

pd.__version__

'0.16.2'
```

5.1 Pandas中的数据对象



与本节内容对应的Notebook为：[05-pandas/pandas-100-dataobjects.ipynb](#)。

Series和DataFrame是Pandas中最常用的两个对象。本节介绍这两种对象的基本概念以及常用属性，在后续章节将介绍对它们进行操作和运算的各种函数和方法。

5.1.1 Series对象

Series是Pandas中最基本的对象，它定义了NumPy的ndarray对象的接口__array__()，因此可以用NumPy的数组处理函数直接对Series对象进行

处理。Series对象除了支持使用位置作为下标存取元素之外，还可以使用索引标签作为下标存取元素，这个功能与字典类似。每个Series对象实际上都由两个数组组成：

- index：它是从ndarray数组继承的Index索引对象，保存标签信息。若创建Series对象时不指定index，将自动创建一个表示位置下标的索引。
- values：保存元素值的ndarray数组，NumPy的函数都对此数组进行处理。

下面创建一个Series对象，并查看上述两个属性：

```
s = pd.Series([1, 2, 3, 4, 5], index=["a", "b", "c", "d", "e"])
print u" 索引:", s.index
print u"值数组:", s.values

索引: Index([u'a', u'b', u'c', u'd', u'e'], dtype='object')
值数组: [1 2 3 4 5]
```

Series对象的下标运算同时支持位置和标签两种形式：

```
print u"位置下标    s[2]:", s[2]
print u"标签下标 s['d']:", s['d']

位置下标    s[2]: 3
标签下标 s['d']: 4
```

Series对象还支持位置切片和标签切片。位置切片遵循Python的切片规则，包括起始位置，但不包括结束位置；但标签切片则同时包括起始标签和结束标签。

s[1:3]	s['b':'d']
-----	-----
b 2	b 2
c 3	c 3
dtype: int64	d 4
	dtype: int64

和ndarray数组一样，还可以使用位置列表或位置数组存取元素，同样也可以使用标签列表和标签数组。

s[[1,3,2]]	s[['b','d','c']]
-----	-----
b 2	b 2
d 4	d 4
c 3	c 3
dtype: int64	dtype: int64

Series对象同时具有数组和字典的功能，因此它也支持字典的一些方法，例如Series.iteritems()：

```
list(s.iteritems())
[('a', 1), ('b', 2), ('c', 3), ('d', 4), ('e', 5)]
```

当两个Series对象进行操作符运算时，Pandas会按照标签对齐元素，也就是说运算操作符会对标签相同的两个元素进行计算。在下面的例子中，s中标签为'b'的元素和s2中标签为'b'的元素相加得到结果中的22。当某一方的标签不存在时，默认以NaN(Not a Number)填充。由于NaN是浮点数中的一个特殊值，因此输出的Series对象的元素类型被转换为float64。

s2 = pd.Series([20,30,40,50,60], index=["b","c","d","e","f"])					
s		s2		s+s2	
-----		-----		-----	
a	1	b	20	a	nan
b	2	c	30	b	22
c	3	d	40	c	33
d	4	e	50	d	44
e	5	f	60	e	55
dtype: int64		dtype: int64		f	nan
				dtype: float64	

5.1.2 DataFrame对象

DataFrame对象（数据表）是Pandas中最常用的数据对象。Pandas提供了将许多数据结构转换为DataFrame对象的方法，还提供了许多输入输出函数来将各种文件格式转换成DataFrame对象。在介绍这些函数之前，我们需要先理解DataFrame对象中的一些概念。

1. DataFrame的各个组成元素

下面的程序调用`read_csv()`从`Soils-simple.csv`读入数据，通过`index_col`参数指定第0和第1列为行索引，用`parse_dates`参数指定进行日期转换的列。在指定列时可以使用列的序号或列名。所得到的DataFrame对象如图5-1所示，图中标识出了DataFrame的各个组成部分的名称。

		列索引名		列索引				
		Measures	pH	Dens	Ca	Conduc	Date	Name
行索引名	Depth	Contour						
行索引	0-10	Depression	5.3525	0.9775	10.6850	1.4725	2015-05-26	Lois
		Slope	5.5075	1.0500	12.2475	2.0500	2015-04-30	Roy
		Top	5.3325	1.0025	13.3850	1.3725	2015-05-21	Roy
	10-30	Depression	4.8800	1.3575	7.5475	5.4800	2015-03-21	Lois
		Slope	5.2825	1.3475	9.5150	4.9100	2015-02-06	Diana
		Top	4.8500	1.3325	10.2375	3.5825	2015-04-11	Diana

第0级索引

第1级

行

列

数据

图5-1 DataFrame的结构

```
df_soil = pd.read_csv("data/Soils-simple.csv", index_col=[0, 1],
parse_dates=["Date"])

df_soil.columns.name = "Measures"
```

由图5-1可知DataFrame对象是一个二维表格。其中，每列中的元素类型必须一致，而不同的列可以拥有不同的元素类型。在本例中，有4列浮点数类型、1列日期类型和1列object类型。object类型的列可以保存任何Python对象，在Pandas中字符串列使用object类型。dtypes属性可以获得表示各个列类型的Series对象：

```
df_soil.dtypes
Measures
pH          float64
Dens        float64
Ca          float64
Conduc      float64
Date        datetime64[ns]
Name        object
dtype: object
```

与数组类似，通过shape属性可以得到DataFrame的行数和列数：

```
df_soil.shape
(6, 6)
```

DataFrame对象拥有行索引和列索引，可以通过索引标签对其中的数据进行存取。index属性保存行索引，而columns属性保存列索引。在本例中列索引是一个Index对象，索引对象的名称可以通过其name属性存取：

```
print df_soil.columns
print df_soil.columns.name
Index([u'pH', u'Dens', u'Ca', u'Conduc', u'Date', u'Name'],
      dtype='object', name=u'Measures')
Measures
```

行索引是一个表示多级索引的MultiIndex对象，每级的索引名可以通过names属性存取：

```
print df_soil.index
print df_soil.index.names
MultiIndex(levels=[[u'0-10', u'10-30'], [u'Depression',
u'Slope', u'Top']],
            labels=[[0, 0, 0, 1, 1, 1], [0, 1, 2, 0, 1, 2]],
            names=[u'Depth', u'Contour'])
[u'Depth', u'Contour']
```

与二维数组相同，DataFrame对象也有两个轴，它的第0轴为纵轴，第1轴为横轴。当某个方法或函数有axis、orient等参数时，该参数可以使用整数0和1或者“index”和“columns”来表示纵轴方向和横轴方向。

[]运算符可以通过列索引标签获取指定的列，当下标是单个标签时，所得到的是Series对象，例如df_soil[“pH”]，而当下标是列表时，则得到一个新的DataFrame对象，例如df_soil[[“Dens”, “Ca”]]：

df_soil["pH"]			df_soil[["Dens", "Ca"]]			
-----			-----			
Depth	Contour		Measures	Dens	Ca	
0-10	Depression	5.4	Depth	Contour		
	Slope	5.5	0-10	Depression	0.98	11
	Top	5.3		Slope	1.1	12
10-30	Depression	4.9	Top		1	13
	Slope	5.3	10-30	Depression	1.4	7.5
	Top	4.8		Slope	1.3	9.5
Name: pH, dtype: float64			Top		1.3	10

.loc[]可通过行索引标签获取指定的行，例如df.loc[“0-10”, “Top”]获得Depth为“0-10”、Contour为“Top”的行，而df.loc[“10-30”]获取Depth为“10-30”的所有行。当结果为一行时得到的是Series对象，结果为多行时得到的是DataFrame对象。注意由于原数据中列的类型不统一，因此得到的

Series对象的类型被转换为最通用的object类型。`.loc[]`的用法非常丰富，下一节还会详细介绍它的各种用法。

df_soil.loc["0-10", "Top"]		df_soil.loc["10-30"]						
-----		-----						
Measures		Measures	pH	Dens	Ca	Conduc	Date	Name
pH	5.3	Contour						
Dens	1	Depression	4.9	1.4	7.5	5.5	2015-03-21	Lois
Ca	13	Slope	5.3	1.3	9.5	4.9	2015-02-06	Diana
Conduc	1.4	Top	4.8	1.3	10	3.6	2015-04-11	Diana
Date	2015-05-21 00:00:00							
Name	Roy							
Name: (0-10, Top), dtype: object								

`values`属性将DataFrame对象转换成数组，由于本例中的列类型不统一，所得到的数组是一个元素类型为object的数组。

```
df_soil.values.dtype
dtype('O')
```

2. 将内存中的数据转换为DataFrame对象

调用`DataFrame()`可以将多种格式的数据转换成DataFrame对象，它的三个参数`data`、`index`和`columns`分别为数据、行索引和列索引。`data`参数可以是：

- 二维数组或者能转换为二维数组的嵌套列表。
- 字典：字典中的每对“键-值”将成为DataFrame对象的列。值可以是一维数组、列表或Series对象。

在下面的程序中，❶在将一个形状为(4, 2)的二维数组转换成DataFrame对象，通过`index`和`columns`参数指定行和列的索引。❷将字典转换为DataFrame对象，其列索引由字典的键决定，行索引由`index`参数指

定。❸将结构数组转换为DataFrame对象，其列索引由结构数组的字段名决定，行索引默认为从0开始的整数序列。

<pre>df1 = pd.DataFrame(np.random.randint(0, 10, (4, 2)), index=["A", "B", "C", "D"], columns=["a", "b"]) df2 = pd.DataFrame({"a": [1, 2, 3, 4], "b": [5, 6, 7, 8]}, index=["A", "B", "C", "D"]) arr = np.array([("item1", 1), ("item2", 2), ("item3", 3), ("item4", 4)], dtype=[("name", "10S"), ("count", int)]) df3 = pd.DataFrame(arr) ❸</pre>					
df1		df2		df3	
-----		-----		-----	
a b		a b		name count	
A	2 6	A	1 5	0	item1 1
B	3 1	B	2 6	1	item2 2
C	5 9	C	3 7	2	item3 3
D	8 0	D	4 8	3	item4 4

此外还可以调用以from_开头的类方法，将特定格式的数据转换成DataFrame对象。from_dict()将字典转换为DataFrame对象，其orient参数可以指定字典键对应的方向，默认值为“columns”，表示把字典的键转换为列索引，即字典中的每个值与一行对应。而orient参数为“index”时，字典中的每个值与一行对应。当字典为嵌套字典，即字典的值为字典时，另外一个轴的索引值由第二层字典中的键决定。下面分别将列表字典和嵌套字典转换为DataFrame对象。嵌套字典中缺失的数据使用NaN表示：

```
dict1 = {"a": [1, 2, 3], "b": [4, 5, 6]}
dict2 = {"a": {"A": 1, "B": 2}, "b": {"A": 3, "C": 4}}
df1 = pd.DataFrame.from_dict(dict1, orient="index")
df2 = pd.DataFrame.from_dict(dict1, orient="columns")
df3 = pd.DataFrame.from_dict(dict2, orient="index")
df4 = pd.DataFrame.from_dict(dict2, orient="columns")
```

df1				df2			df3				df4		
-----				-----			-----				-----		
0 1 2				a b			A B C				a b		
a	1	2	3	0	1	4	a	1	2	nan	A	1	3
b	4	5	6	1	2	5	b	3	nan	4	B	2	nan
				2	3	6					C	nan	4

`from_items()` 将 “（键，值）” 序列转换为 `DataFrame` 对象，其中 “值” 是表示一维数据的列表、数组或 `Series` 对象。当其 `orient` 参数为 “index” 时，需要通过 `columns` 指定列索引。

```
items = dict1.items()
df1 = pd.DataFrame.from_items(items, orient="index", columns=
["A", "B", "C"])
df2 = pd.DataFrame.from_items(items, orient="columns")
```

df1			df2		
-----			-----		
	A	B	C	a	b
a	1	2	3	0	1
b	4	5	6	2	3

3. 将 `DataFrame` 对象转换为其他格式的数据

`to_dict()` 方法将 `DataFrame` 对象转换为字典，它的 `orient` 参数决定字典元素的类型：

```
print df2.to_dict(orient="records") #字典列表
print df2.to_dict(orient="list") #列表字典
print df2.to_dict(orient="dict") #嵌套字典
```

```
[{'a': 1, 'b': 4}, {'a': 2, 'b': 5}, {'a': 3, 'b': 6}]
{'a': [1, 2, 3], 'b': [4, 5, 6]}
{'a': {0: 1, 1: 2, 2: 3}, 'b': {0: 4, 1: 5, 2: 6}}
```

`to_records()` 方法可以将 `DataFrame` 对象转换为结构数组，若其 `index` 参数为 `True`（默认值），则其返回的数组中包含行索引数据：


```
print df2.to_records().dtype
print df2.to_records(index=False).dtype
[('index', '<i8'), ('a', '<i8'), ('b', '<i8')]
[('a', '<i8'), ('b', '<i8')]
```

Pandas还提供了许多全局函数来从各种格式的文件读取数据，而各种以to开头的方法可以将其输出到文件中，关于文件的输入输出将在后面详细介绍。

5.1.3 Index对象

Index对象保存索引标签数据，它可以快速找到标签对应的整数下标，这种将标签映射到整数下标的功能与Python的字典类似。其values属性可以获得保存标签的数组，与Series一样，字符串使用object类型的数组保存：

```
index = df_soil.columns
index.values
array(['pH', 'Dens', 'Ca', 'Conduc', 'Date', 'Name'],
dtype=object)
```

Index对象可当作一维数组，通过与NumPy数组相同的下标操作可以得到一个新的Index对象，但是Index对象是只读的，因此一旦创建就无法修改其中的元素。

```
print index[[1, 3]]
print index[index > 'c']
print index[1::2]
Index([u'Dens', u'Conduc'], dtype='object', name=u'Measures')
```

```
Index([u'pH'], dtype='object', name=u'Measures')
      Index([u'Dens',    u'Conduc',    u'Name'],    dtype='object',
name=u'Measures')
```

Index对象也具有字典的映射功能，它将数组中的值映射到其位置：

- `Index.get_loc(value)`：获得单个值value的下标。
- `Index.get_indexer(values)`：获得一组值values的下标，当值不存在时，得到-1。

```
print index.get_loc('Ca')
print index.get_indexer(['Dens', 'Conduc', 'nothing'])
2
[ 1  3 -1]
```

可以直接调用`Index()`来创建Index对象，然后传递给`DataFrame()`的`index`或`columns`参数。由于Index是不可变对象，因此多个数据对象的索引可以是同一个Index对象。

```
index = pd.Index(["A", "B", "C", "D", "E"], name="level")
s1 = pd.Series([1, 2, 3, 4, 5], index=index)
df1 = pd.DataFrame({"a": [1, 2, 3, 4, 5], "b": [6, 7, 8, 9, 10]},
index=index)

print s1.index is df1.index
True
```

5.1.4 MultiIndex对象

MultiIndex表示多级索引，它从Index继承，其中的多级标签采用元组对象来表示。下面通过[]获取其中的单个元素，调用get_loc()和get_indexer()以获取单个标签和多个标签对应的下标。

```
mindex = df_soil.index
print mindex[1]
print mindex.get_loc(("0-10", "Slope"))
print mindex.get_indexer([("10-30", "Top"), ("0-10",
"Depression"), "nothing"])
('0-10', 'Slope')
1
[ 5  0 -1]
```

在MultiIndex内部并不直接保存元组对象，而是使用多个Index对象保存索引中每级的标签：

```
print mindex.levels[0]
print mindex.levels[1]
Index([u'0-10', u'10-30'], dtype='object', name=u'Depth')
Index([u'Depression', u'Slope', u'Top'], dtype='object',
name=u'Contour')
```

然后使用多个整数数组保存这些标签的下标：

```
print mindex.labels[0]
print mindex.labels[1]
FrozenNDArray([0, 0, 0, 1, 1, 1], dtype='int8')
FrozenNDArray([0, 1, 2, 0, 1, 2], dtype='int8')
```

下面的代码通过levels和labels属性得到多级索引中所有元组的列表，该列表也可以通过tolist()方法获得：

```

level0, level1 = minindex.levels
label0, label1 = minindex.labels
zip(level0[label0], level1[label1])
[('0-10', 'Depression'),
 ('0-10', 'Slope'),
 ('0-10', 'Top'),
 ('10-30', 'Depression'),
 ('10-30', 'Slope'),
 ('10-30', 'Top')]

```

当将一个元组列表传递给Index()时，将自动创建MultiIndex对象。若希望创建元素类型为元组的Index对象，可以设置tupleize_cols参数为False:

```

pd.Index([("A", "x"), ("A", "y"), ("B", "x"), ("B", "y")], name=
["class1", "class2"])

MultiIndex(levels=[[u'A', u'B'], [u'x', u'y']],
            labels=[[0, 0, 1, 1], [0, 1, 0, 1]],
            names=[u'class1', u'class2'])

```

此外可以使用以from_开头的MultiIndex类方法从特定的数据结构创建MultiIndex对象。例如from_arrays()方法从多个数组创建MultiIndex对象:

```

class1 = ["A", "A", "B", "B"]
class2 = ["x", "y", "x", "y"]

pd.MultiIndex.from_arrays([class1, class2], names=["class1",
"class2"])

MultiIndex(levels=[[u'A', u'B'], [u'x', u'y']],

```

```
labels=[[0, 0, 1, 1], [0, 1, 0, 1]],
names=[u'class1', u'class2'])
```

from_product() 则从多个集合的笛卡尔积创建MultiIndex对象。下面的程序将所创建的MultiIndex对象传递给index和columns参数，所创建的DataFrame对象的行和列使用同一个多级索引对象：

```
midx = pd.MultiIndex.from_product([["A", "B", "C"], ["x", "y"]],
                                   names=["class1", "class2"])
df1 = pd.DataFrame(np.random.randint(0, 10, (6, 6)), columns=midx, index=midx)
```

```
df1
-----
class1      A      B      C
class2      x  y  x  y  x  y
class1 class2
A      x      3  0  8  2  7  3
      y      9  7  6  9  2  4
B      x      8  8  0  4  8  3
      y      8  3  7  6  3  9
C      x      2  0  4  0  6  4
      y      0  7  5  6  0  5
```

5.1.5 常用的函数参数

在后续章节我们会详细介绍各种常用函数的用法。表5-1列出了一些常用的函数参数。

表5-1 常用的函数参数

参数名	常用值	说明
axis	0、1	运算对应的轴
level	整数或索引的级别名	指定运算对应的级别

DataFrame对象内部使用NumPy数组保存数据，因此也会出现和数组相同的共享数据存储区的问题。为了帮助读者理解Pandas的内存管理，让我们看看DataFrame对象的内部结构。

下面的程序通过本书提供的GraphvizDataFrame绘制df_soil的内部结构，结果如图5-2所示。图中的实线箭头表示一般属性，虚线箭头表示由property创建的属性，读取这些属性时实际上是获得它们对应的函数的返回值。在分析DataFrame对象的内部结构时，我们重点关注实线箭头所表示的属性。

```
from scipy2.common import GraphvizDataFrame
%dot GraphvizDataFrame.graphviz(df_soil)
```

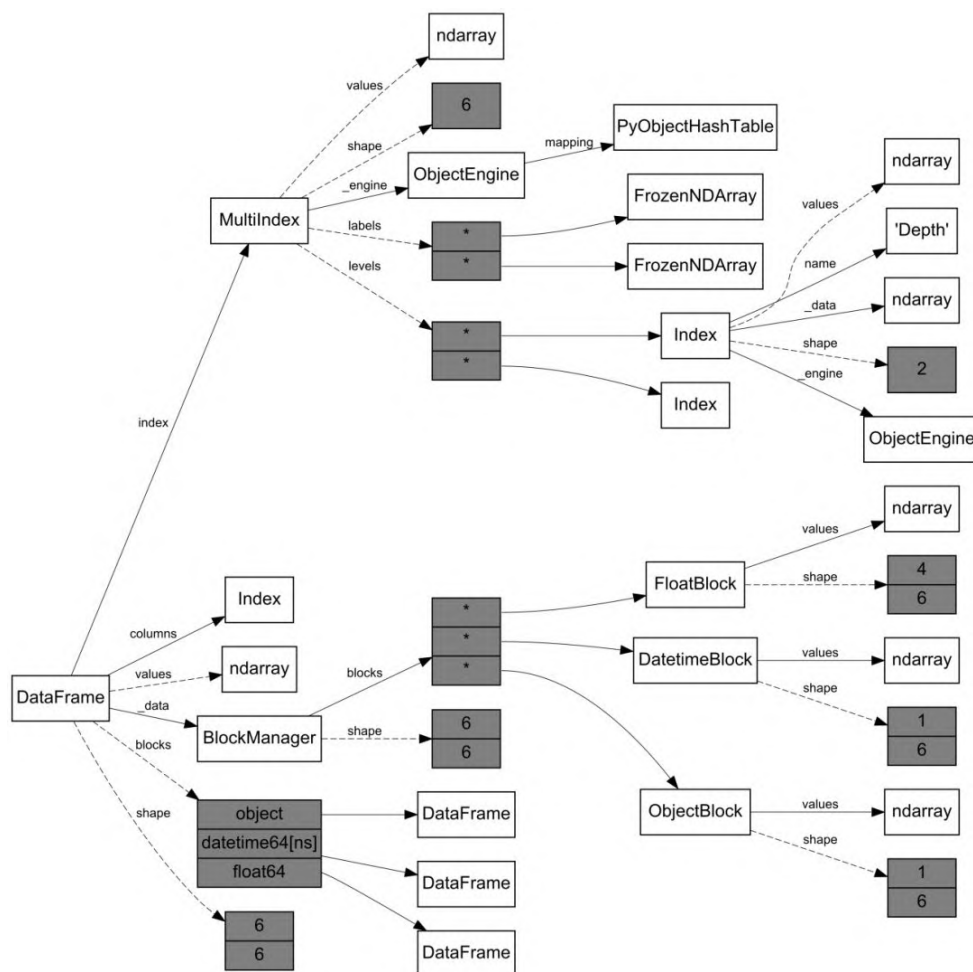


图5-2 DataFrame对象的内部结构

DataFrame对象的columns属性是Index对象，而index属性是表示多级索引的MultiIndex对象。Index对象的索引功能由其_engine属性——一个ObjectEngine对象提供，该对象使用一个哈希表PyObjectHashTable对象将标签映射到与其对应的整数下标。下面的代码获取列标签“Date”对应的整数下标：

```
df_soil.columns._engine.mapping.get_item("Date")  
4
```

DataFrame对象的数据都保存在_data属性中，它是一个BlockManager对象，其blocks属性是一个列表，其中有一个FloatBlock对象、一个DatetimeBlock对象和一个ObjectBlock对象。这些对象是管理实际数据的数据块，其values属性是保存数据的数组。

DataFrame对象尽量用一个数组保存相同类型的列，而将不同类型的列保存在不同的数组中。这些数组的形状为(4, 6)、(1, 6)和(1, 6)。它们的第0轴的长度对应4个浮点数列、1个时间列和1个字符串列。由此可知DataFrame中的整列数据是保存在连续的内存空间中，这有助于提高数据的存取速度。

当通过[]获取某一列时，所得到的Series对象与原DataFrame对象共享内存。下面查看保存Series对象数据的数组的base属性，也就是df_soil._data.blocks[0].values：

```
s = df_soil["Dens"]  
s.values.base is df_soil._data.blocks[0].values  
True
```


当通过[]获取多列时，将复制所有的数据，因此保存新DataFrame对象数据的数组的base属性为None：

```
print df_soil[["Dens"]]._data.blocks[0].values.base  
None
```

如果DataFrame对象只有一个数据块，则通过values属性得到的数组是数据块中数组的转置，因此它与DataFrame对象共享内存。例如在下面的程序中，df_float中所有列的元素类型相同，它只有一个数据块，因此df_float.values所得到的数组与df_float中保存数据的数组共享内存。

```
df_float = df_soil[['pH', 'Dens', 'Ca', 'Conduc']]  
df_float.values.base is df_float._data.blocks[0].values  
True
```

当DataFrame对象只有一个数据块时，获取其行数据所得到的Series对象也与其共享内存：

```
df_float.loc["0-10", "Top"].values.base is  
df_float._data.blocks[0].values  
True
```

而当BlockManager中使用多个数组保存数据时，则返回这些数据的拷贝，数组的元素类型为最通用的元素类型，以保存各种格式的数据。在下面的例子中，df.values的元素类型为object，因为它需要同时保存浮点数、时间和字符串。

```
df_soil.values.dtype  
dtype('O')
```

5.2 下标存取



与本节内容对应的Notebook为：05-pandas/pandas-200-getset.ipynb。

Series和DataFrame提供了丰富的下标存取方法，除了直接使用[]运算符之外，还可以使用.loc[]、.iloc[]、.at[]、.iat[]和.ix[]等存取器存取其中的元素。

下面的表5-2总结了DataFrame对象的各种存取方法：

表5-2 DataFrame对象的各种存取方法

方法	说明
[col_label]	以单个标签作为下标，获取与标签对应的列，返回Series对象
[col_labels]	以标签列表作为下标，获取对应的多个列，返回DataFrame对象
[row_slice]	整数切片或标签切片，得到指定范围之内的行
[row_bool_array]	选择布尔数组中True对应的行
.get(col_label, default)	与字典的get()方法的用法相同
.at[index_label, col_label]	选择行标签和列标签对应的值，返回单个元素
.iat[index, col]	选择行编号和列编号对应的值，返回单个元素

<code>.loc[index, col]</code>	通过单个标签值、标签列表、标签数组、布尔数组、标签切片等选择指定行与列上的数据
<code>.iloc[index, col]</code>	通过单个整数值、整数列表、整数数组、布尔数组、整数切片选择指定行与列上的数据
<code>.ix[index, col]</code>	同时拥有 <code>.loc[]</code> 和 <code>.iloc[]</code> 的功能，既可以使用标签下标也可以使用整数下标
<code>.lookup(row_labels, col_labels)</code>	选择行标签列表与列标签列表中每对标签对应的元素值
<code>.get_value(row_label, col_label)</code>	与 <code>.at[]</code> 的功能类似，不过速度更快
<code>.query()</code>	通过表达式选择满足条件的行
<code>.head()</code>	获取头部N行数据
<code>.tail()</code>	获取尾部N行数据

```
np.random.seed(42)

df = pd.DataFrame(np.random.randint(0, 10, (5, 3)),
                  index=["r1", "r2", "r3", "r4", "r5"],
                  columns=["c1", "c2", "c3"])
```

5.2.1 []操作符

通过[]操作符对DataFrame对象进行存取时，支持以下5种下标对象：

- 单个索引标签：获取标签对应的列，返回一个Series对象。
- 多个索引标签：获取以列表、数组(注意不能是元组)表示的多个标签对应的列，返回一个DataFrame对象。

- 整数切片：以整数下标获取切片对应的行。
- 标签切片：当使用标签作为切片时包含终值。
- 布尔数组：获取数组中True对应的行。
- 布尔DataFrame：将DataFrame对象中False对应的元素设置为NaN。

下面显示整数切片和标签切片的结果，注意标签切片包含终值“r4”：

df				df[2:4]				df["r2":"r4"]			
-----				-----				-----			
	c1	c2	c3		c1	c2	c3		c1	c2	c3
r1	6	3	7	r3	2	6	7	r2	4	6	9
r2	4	6	9	r4	4	3	7	r3	2	6	7
r3	2	6	7					r4	4	3	7
r4	4	3	7								
r5	7	2	5								

`df.c1 > 4`是一个布尔序列，因此`df[df.c1 > 4]`获得该序列中True对应的行。`df > 2`是一个布尔DataFrame对象，`df[df > 2]`将其中False对应的元素置换为NaN：

df[df.c1 > 4]				df[df > 2]			
-----				-----			
	c1	c2	c3		c1	c2	c3
r1	6	3	7	r1	6	3	7
r5	7	2	5	r2	4	6	9
				r3	nan	6	7
				r4	4	3	7
				r5	7	nan	5

5.2.2 .loc[]和.iloc[]存取器

`.loc[]`的下标对象是一个元组，其中的两个元素分别与DataFrame的两个轴相对应。若下标不是元组，则该下标对应第0轴，`:`对应第1轴。每个轴的下标对象都支持单个标签、标签列表、标签切片以及布尔数组。

`df.loc["r2"]` 获得 "r2" 对应的行，它返回一个 Series 对象。
`df.loc["r2", "c2"]` 获得 "r2" 行 "c2" 列的元素，它返回单个元素值。

	<code>df.loc["r2"]</code>	<code>df.loc["r2", "c2"]</code>
c1	4	6
c2	6	
c3	9	

Name: r2, dtype: int32

`df.loc[["r2", "r3"]]` 获得 "r2" 和 "r3" 对应的行。
`df.loc[["r2", "r3"], ["c1", "c2"]]` 则获得 "r2" 和 "r3" 行、"c1" 和 "c2" 列上的数据，所得到的数据都是新的 DataFrame 对象。

	<code>df.loc[["r2", "r3"]]</code>				<code>df.loc[["r2", "r3"], ["c1", "c2"]]</code>		
	c1	c2	c3		c1	c2	
r2	4	6	9	r2	4	6	
r3	2	6	7	r3	2	6	

在下面的程序中，第0轴的下标分别为标签切片和布尔数序列：

	<code>df.loc["r2": "r4", ["c2", "c3"]]</code>				<code>df.loc[df.c1>2, ["c1", "c2"]]</code>		
	c2	c3			c1	c2	
r2	6	9		r1	6	3	
r3	6	7		r2	4	6	
r4	3	7		r4	4	3	
r5	7	2					

`.iloc[]` 和 `loc[]` 类似，不过它使用整数下标：

```

df.iloc[2]          df.iloc[[2,4]]      df.iloc[[1,3]]      df.iloc[[1,3],[0,2]]
-----
c1    2              c1 c2 c3           c1 c2 c3           c1 c3
c2    6              r3  2  6  7         r2  4  6  9         r2  4  9
c3    7              r5  7  2  5         r4  4  3  7         r4  4  7
Name: r3, dtype: int32

df.iloc[2:4, [0,2]]  df.iloc[df.c1.values>2, [0,1]]
-----
      c1 c3              c1 c2
r3    2  7              r1    6  3
r4    4  7              r2    4  6
r4    4  3
r5    7  2

```

此外, `ix[]` 的存取器可以混用标签和位置下标, 例如:

```

df.ix[2:4, ["c1", "c3"]]  df.ix["r1":"r3", [0, 2]]
-----
      c1 c3              c1 c3
r3    2  7              r1    6  7
r4    4  7              r2    4  9
r3    2  7

```

如果DataFrame对象有整数索引, 则应该使用 `loc[]` 和 `iloc[]` 以避免混淆。

5.2.3 获取单个值

`.at[]` 和 `.iat[]` 分别使用标签和整数下标获取单个值, 此外 `get_value()` 与 `.at[]` 类似, 不过其执行速度要快一些:

```

df.at["r2", "c2"]  df.iat[1, 1]  df.get_value("r2", "c2")
-----
6              6              6

```

当 `loc[]` 的下标对象是两个标签列表时, 所获得的是这两个列表形成的网格上的元素, 这与NumPy的数组下标操作不一样。如果希望获取两个列

表中每对标签所对应的元素，可以使用lookup()，它返回一个包含指定元素的数组：

```
df.lookup(["r2", "r4", "r3"], ["c1", "c2", "c1"])  
array([4, 3, 2])
```

5.2.4 多级标签的存取

.loc[]和.at[]的下标可以指定多级索引中每级索引上的标签。这时多级索引轴对应的下标是一个下标元组，该元组中的每个元素与索引中的每级索引对应。若下标不是元组，则将其转换为长度为1的元组，若元组的长度比索引的层数少，则在其后面补slice(None)。

```
soil_df = pd.read_csv("data/Soils-simple.csv", index_col=[0, 1],  
parse_dates=["Date"])
```

在下面的例子中，“10-30”为第0轴的标签，根据前面的规则，将其转换为(“10-30”， slice(None))，即选择第0级中“10-30”对应的行：

```
soil_df.loc["10-30", ["pH", "Ca"]]
```

```
-----  
              pH  Ca  
Contour  
Depression  4.9  7.5  
Slope       5.3  9.5  
Top         4.8  10
```

如果需要选择第1级中“Top”对应的行，则需要把slice(None)作为第0级的下标。由于Python中只有直接在[]中才能使用以:分隔的切片语法，因此这里使用np.s_对象创建第0轴对应的下标：(slice(None), “Top”)。

```
soil_df.loc[np.s_[0, "Top"], ["pH", "Ca"]]
-----
              pH  Ca
Depth Contour
0-10 Top      5.3  13
10-30 Top      4.8  10
```

5.2.5 query() 方法

当需要根据一定的条件对行进行过滤时，通常可以先创建一个布尔数组，使用该数组获取True对应的行，例如下面的程序获得pH值大于5、Ca含量小于11%的行。由于Python中无法自定义not、and和or等关键字的行为，因此需要改用~、&、|等位运算符。然而这些运算符的优先级比较运算符要高，因此需要用括号将比较运算括起来：

```
soil_df[(soil_df.pH > 5) & (soil_df.Ca < 11)]
```

使用query()可以简化上述程序：

```
print soil_df.query("pH > 5 and Ca < 11")
-----
              pH  Dens  Ca  Conduc  Date  Name
Depth Contour
0-10 Depression  5.4  0.98  11    1.5 2015-05-26  Lois
10-30 Slope      5.3  1.3  9.5    4.9 2015-02-06  Diana
```

query()的参数是一个运算表达式字符串。其中可以使用not、and和or等关键字进行向量布尔运算，表达式中的变量名表示与其对应的列。如果希望在表达式中使用其他全局或局域变量的值，可以在变量名之前添加@，例如：

```
pH_low = 5
Ca_hi = 11
print soil_df.query("pH > @pH_low and Ca < @Ca_hi")
```


5.3 文件的输入输出



与本节内容对应的 Notebook 为：05-pandas/pandas-300-io.ipynb。

本节介绍表5-3中的输入输出函数：

表5-3 输入输出函数

函数名	说明
<code>read_csv()</code>	从CSV格式的文本文件读取数据
<code>read_excel()</code>	从Excel文件读入数据
<code>HDFStore()</code>	使用HDF5文件读写数据
<code>read_sql()</code>	从SQL数据库的查询结果载入数据
<code>read_pickle()</code>	读入Pickle序列化之后的数据

5.3.1 CSV文件

`read_csv()` 从文本文件读入数据，它的可选参数非常多，下面只简要介绍一些常用参数：

- `sep` 参数指定数据的分隔符号，可以使用正则表达式，默认值为逗号。有时CSV文件为了便于阅读，在逗号之后添加了一些空格以对

齐每列的数据。如果希望忽略这些空格，可以将`skipinitialspace`参数设置为`True`。

- 如果数据使用空格或制表符分隔，可以不设置`sep`参数，而将`delim_whitespace`参数设置为`True`。
- 默认情况下第一行文本被作为列索引标签，如果数据文件中没有保存列名的行，可以设置`header`参数为0。
- 如果数据文件之前包含一些说明行，可以使用`skiprows`参数指定数据开始的行号。
- `na_values`、`true_values`和`false_values`等参数分别指定`NaN`、`True`和`False`对应的字符串列表。
- 如果希望从字符串读入数据，可以使用`io.BytesIO(string)`将字符串包装成输入流。
- 如果希望将字符串转换为时间，可以使用`parse_dates`指定转换为时间的列。
- 如果数据中包含中文，可以使用`encoding`参数指定文件的编码，例如“`utf-8`”、“`gbk`”等。指定编码之后得到的字符串列为Unicode字符串。
- 可以使用`usecols`参数指定需要读入的列。
- 当文件很大时，可以用`chunksize`参数指定一次读入的行数。当使用`chunksize`时，`read_csv()`返回一个迭代器。
- 当文件名包含中文时，需要使用Unicode字符串指定文件名。

下面使用上面介绍的各个参数读入上海市的空气质量数据文件。该文件的文字编码为UTF-8，并且带BOM。所谓BOM，是指在文件开头的3个特殊字节表示该文件为UTF-8文件。对于带BOM的UTF-8文件，可以指定编码参数`encoding`为“`utf-8-sig`”。

该文件中有两种字符表示缺失数据：一个是减号，另一个是全角的横杠。由于`read_csv()`在将字节字符串转换为Unicode之前判断`NaN`，因此需

要使用与文件相同的编码表示这些缺失数据的字符串。



<http://air.epmap.org/>

空气质量数据来源：青悦空气质量历史数据库。

```
df_list = []

for df in pd.read_csv(
    u"data/aqi/上海市_201406.csv",
    encoding="utf-8-sig", #文件编码
    chunksize=100,        #一次读入的行数
    usecols=[u"时间", u"监测点", "AQI", "PM2.5", "PM10"], #只读入这些列
    na_values=["-", "-"], #这些字符串表示缺失数据
    parse_dates=[0]):      #第一列为时间列
    df_list.append(df) #在这里处理数据
```

df_list[0].count()		df_list[0].dtypes	
-----		-----	
时间	100	时间	datetime64[ns]
监测点	90	监测点	object
AQI	100	AQI	int64
PM2.5	100	PM2.5	int64
PM10	98	PM10	float64
dtype: int64		dtype: object	

注意“时间”列为datetime64[ns]类型，而由于存在缺失数据，因此“PM10”列被转换为浮点数类型，其他的数值列为整数类型，而“监测点”列中保存的是Unicode字符串。

```
print type(df.loc[0, u"监测点"])
<type 'unicode'>
```

5.3.2 HDF5文件

HDF5是存储科学计算数据的一种文件格式，支持大于2GB的文件，可以把它看作针对科学计算的数据库文件。关于HDF5文件格式的更多信息，请参考下面的链接：



http://www.nsmc.cma.gov.cn/FENGYUNCast/docs/HDF5.0_chinese.pdf

中文的HDF5使用简介。

HDF5文件像一个保存数据的文件系统，其中只有两种类型的对象：资料数据(dataset)和目录(group)：

- 资料数据：像文件系统中的文件一样用于保存各种数据，例如NumPy数组。
- 目录：类似于文件系统中的文件夹，可以包含其他的目录或资料数据。

使用Pandas可以很方便地将多个Series和DataFrame保存进HDF5文件。HDF5文件采用二进制格式保存数据，可以对数据进行压缩存储，比文本文件更节省空间，存取也更迅速。

下面创建一个HDFStore对象，通过complib参数指定使用blosc压缩数据，通过complevel参数指定压缩级别。

```
store = pd.HDFStore("a.hdf5", complib="blosc", complevel=9)
```

HDFStore对象支持字典接口，例如使用[]存取元素、get()和keys()等方法。

```

df1 = pd.DataFrame(np.random.rand(100000, 4),
columns=list("ABCD"))

df2 = pd.DataFrame(np.random.randint(0, 10000, (10000, 3)),
columns=["One", "Two", "Three"])

s1 = pd.Series(np.random.rand(1000))

store["dataframes/df1"] = df1
store["dataframes/df2"] = df2
store["series/s1"] = s1

print store.keys()

print df1.equals(store["dataframes/df1"])

['/dataframes/df1', '/dataframes/df2', '/series/s1']

True

```

HDFStore采用pytables扩展库存取HDF5文件，其get_node()方法可以获得pytables中定义的Node对象。使用该对象可以遍历文件中的所有节点，关于Node对象的用法，请读者参考pytables的文档：



http://pytables.github.io/usersguide/libref/hierarchy_classes.html

pytables官方文档。

下面用get_node()获得根节点，然后调用_f_walknodes()遍历其包含的所有节点。由结果可知，HDFStore中的Series和DataFrame对象与HDF5的目录对应，目录中通过多个资料数据保存具体的数据。

```

root = store.get_node("/")
for node in root._f_walknodes():
    print node

/dataframes (Group) u''
/series (Group) u''
/dataframes/df1 (Group) u''
/dataframes/df2 (Group) u''
/series/s1 (Group) u''
/series/s1/index (CArray(1000,), shuffle, blosc(9)) ''
/series/s1/values (CArray(1000,), shuffle, blosc(9)) ''
/dataframes/df1/axis0 (CArray(4,), shuffle, blosc(9)) ''
/dataframes/df1/axis1 (CArray(100000,), shuffle, blosc(9)) ''
/dataframes/df1/block0_items (CArray(4,), shuffle, blosc(9)) ''
    /dataframes/df1/block0_values (CArray(100000, 4), shuffle,
blosc(9)) ''
/dataframes/df2/axis0 (CArray(3,), shuffle, blosc(9)) ''
/dataframes/df2/axis1 (CArray(10000,), shuffle, blosc(9)) ''
/dataframes/df2/block0_items (CArray(3,), shuffle, blosc(9)) ''
    /dataframes/df2/block0_values (CArray(10000, 3), shuffle,
blosc(9)) ''

```

通过前面介绍的方法将DataFrame对象保存进HDFStore之后，无法再为其追加数据。在数据采集和导入大量CSV文件时，我们通常希望能不断地往同一DataFrame中添加新的数据。可以使用append()方法实现该功能。

❶append参数为False表示将覆盖已存在的数据，如果指定键值不存在，可省略该参数。❷将df3追加到指定键，因此读取该键将得到一个长度为100100的DataFrame对象。

```
store.append('dataframes/df_dynamic1', df1, append=False) ❶
df3 = pd.DataFrame(np.random.rand(100, 4), columns=list("ABCD"))
store.append('dataframes/df_dynamic1', df3) ❷
store['dataframes/df_dynamic1'].shape
(100100, 4)
```

使用append()将创建pytables中支持索引的表格(Table)节点，默认使用DataFrame的index作为索引。通过select()可以对表格进行查询以获取满足查询条件的行。在下面的程序中，通过where参数指定查询条件，index表示DataFrame的标签数据。该条件获取标签在97到102之间的所有行。由于我们将两个默认标签的DataFrame添加进该表格，因此98和99各对应两行数据。使用该方式读取部分数据时，可以减少内存使用量和磁盘读取量，提高数据的访问速度。

```
print store.select('dataframes/df_dynamic1', where='index > 97
& index < 102')
      A      B      C      D
98  0.95 0.072 0.78  0.18
99  0.19 0.043 0.24 0.075
100 0.21  0.78 0.86  0.47
101 0.71  0.87 0.63  0.74
98  0.058 0.18 0.91 0.083
99  0.47  0.81 0.71  0.59
```

如果希望对DataFrame的指定列进行索引，可以在用append()创建新的表格时，通过data_columns指定索引列，或将其设置为True以对所有列创建索引。

```
store.append('dataframes/df_dynamic1', df1, append=False,
data_columns=["A", "B"])
```

```
print store.select('dataframes/df_dynamic1', where='A > 0.99 &
B < 0.01')
```

	A	B	C	D
3656	0.99	0.0018	0.67	0.47
5091	1	0.004	0.43	0.15
17671	1	0.0042	0.99	0.31
41052	1	0.00081	0.9	0.32
45307	1	0.0093	0.72	0.065
67976	0.99	0.0096	0.93	0.79
69078	1	0.0055	0.97	0.88
87871	1	0.008	0.59	0.35
94421	0.99	0.0049	0.36	0.9

下面循环读入data\aqi路径之下的所有CSV文件，并将数据写入HDF5文件中。在将多个文本文件的数据逐次写入HDF5文件时，需要注意以下几点事项：

- HDF5文件不支持Unicode字符串，因此需要对Unicode字符串进行编码，转换为字节字符串。在本例中直接从文件读取UTF-8编码的字符串，因此在读入CSV文件时无须指定encoding参数。❶但是由于文件可能包含UTF-8的BOM，因此需要先读入文件的头三个字节并与BOM比较，这样才能保证读入的数据中与第一列对应的标签不包含BOM。
- 由于可能存在缺失数据，因此读入的数值列的类型可能为整数和浮点数。由于HDF5文件中的每列数据只能对应一种类型，❷因此需要使用dtype参数指定这些数值列的类型为浮点数。
- ❸需要为HDF5文件中的字符串列指定最大长度，否则该最大长度将由第一个被添加进HDF5文件的数据对象决定。



由于所有从CSV文件读入DataFrame对象的行索引都为默认值，因此HDF5文件中数据的行索引并不是唯一的。

```
def read_aqi_files(fn_pattern):
    from glob import glob
    from os import path

    UTF8_BOM = b"\xEF\xBB\xBF"

    cols = "时间,城市,监测点,质量等级,AQI,PM2.5,PM10,CO,NO2,O3,S02".split(",")
    float_dtypes = {col:float for col in "AQI,PM2.5,PM10,CO,NO2,O3,S02".split(",")}
    names_map = {"时间":"Time",
                  "监测点":"Position",
                  "质量等级":"Level",
                  "城市":"City",
                  "PM2.5":"PM2_5"}

    for fn in glob(fn_pattern):
        with open(fn, "rb") as f:
            sig = f.read(3) ❶
            if sig != UTF8_BOM:
                f.seek(0, 0)
            df = pd.read_csv(f,
                             parse_dates=[0],
                             na_values=["-", "-"],
                             usecols=cols,
                             dtype=float_dtypes) ❷
            df.rename_axis(names_map, axis=1, inplace=True)
            df.dropna(inplace=True)
            yield df

store = pd.HDFStore("data/aqi/aqi.hdf5", complib="blosc", complevel=9)
string_size = {"City": 12, "Position": 30, "Level":12}

for idx, df in enumerate(read_aqi_files(u"data/aqi/*.csv")):
    store.append('aqi', df, append=idx!=0, min_itemsize=string_size, data_columns=True)❸

store.close()
```

下面打开aqi.hdf5文件并读入所有数据：

```
store = pd.HDFStore("data/aqi/aqi.hdf5")
df_aqi = store.select("aqi")
print len(df_aqi)
337250
```

下面只读取PM2.5值大于500的行：

```
df_polluted = store.select("aqi", where="PM2_5 > 500")
print len(df_polluted)
87
```

5.3.3 读写数据库

用`to_sql()`可以将数据写入SQL数据库，它的第一个参数为数据库的表名，第二个参数为表示与数据库连接的Engine对象，Engine在sqlalchemy库中定义。下面首先从sqlalchemy中载入`create_engine()`，并调用它使用SQLite打开数据库文件“data/aqi/aqi.db”。当该文件不存在时，将创建新的数据库文件：

```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///data/aqi/aqi.db')
```

为了避免重复写入，下面先通过engine对象执行SQL语句，删除aqi表：

```
try:
    engine.execute("DROP TABLE aqi")
except:
    pass
```

然后调用`to_sql()`将数据写入数据库，`if_exists`参数为“append”表示当表存在时，将新数据添加到表中。由于本例中`DataFrame`对象的行索引无实际意义，因此设置`index`参数为`False`，表示不保存行索引。由于数据库要求使用Unicode字符串，因此在写入数据库之前对字符串列进行解码，将其数据转换为Unicode字符串。如果在从CSV文件读入数据时，通过`encoding`参数指定了文本编码，则不必执行此步骤。

```
str_cols = ["Position", "City", "Level"]

for df in read_aqi_files("data/aqi/*.csv"):
    for col in str_cols:
        df[col] = df[col].str.decode("utf8")
    df.to_sql("aqi", engine, if_exists="append", index=False)
```

下面调用`read_sql()`从数据库读入整个名为`aqi`的表：

```
df_aqi = pd.read_sql("aqi", engine)
```

也可以通过SQL查询语句读入部分数据，下面只读入PM2.5值大于500的行：

```
df_polluted = pd.read_sql("select * from aqi where PM2_5 > 500", engine)

print len(df_polluted)

87
```

5.3.4 使用Pickle序列化

还可以使用`to_pickle()`和`read_pickle()`对`DataFrame`对象进行序列化和反序列化：

```
df_aqi.to_pickle("data/aqi/aqi.pickle")
df_aqi2 = pd.read_pickle("data/aqi/aqi.pickle")
df_aqi.equals(df_aqi2)
True
```

Pickle是Python特有的对象序列化格式，因此很难使用其他软件、程序设计语言读取Pickle化之后的数据，但是作为临时保存运算的中间结果还是很方便的。

5.4 数值运算函数



与本节内容对应的Notebook为：[05-pandas/pandas-400-calculation.ipynb](#)。

`Series`和`DataFrame`对象都支持NumPy的数组接口，因此可以直接使用NumPy提供的ufunc函数对它们进行运算。此外它们还提供各种运算方法，例如`max()`、`min()`、`mean()`、`std()`等。这些函数都有如下三个常用参数：

- `axis`：指定运算对应的轴。
- `level`：指定运算对应的索引级别。
- `skipna`：运算是否自动跳过NaN。


```

10-30 Depression    6.8
      Slope         11
      Top           10
dtype: float64

```

Pandas还提供了`rolling_*`()函数来对序列中相邻的N个元素进行移动窗口运算。例如可以使用`rolling_median()`实现中值滤波，使用`rolling_mean()`计算移动平均。图5-3显示了使用这两个函数对带脉冲噪声的正弦波进行处理的结果。它们的第二个参数为窗口包含的元素个数，而`center`参数为`True`表示移动窗口以当前元素为中心。

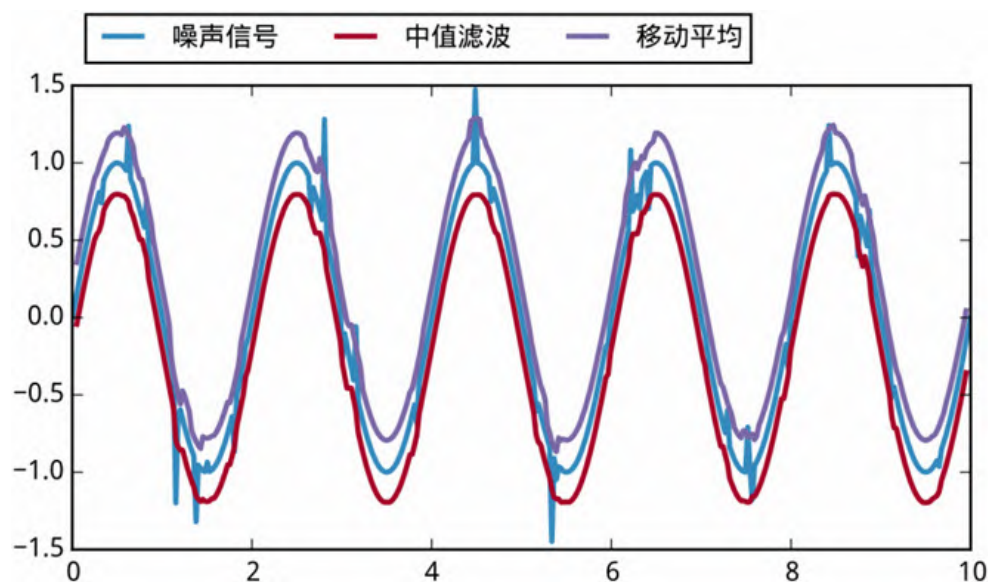


图5-3 中值滤波和移动平均

由于`rolling_median()`采用了更高效的算法，因此当窗口很大时它的运算速度比SciPy章节中介绍过的`signal.order_filter()`更快。

```

t = np.linspace(0, 10, 400)
x = np.sin(0.5*2*np.pi*t)
x[np.random.randint(0, len(t), 40)] += np.random.normal(0, 0.3,
40)

```

```
s = pd.Series(x, index=t)
s_mean = pd.rolling_mean(s, 5, center=True)
s_median = pd.rolling_median(s, 5, center=True)
```

`expanding_*`() 函数对序列进行扩展窗口运算，例如 `expanding_max()` 返回到每个元素为止的历史最大值。图5-4显示了 `expanding_max()`、`expanding_mean()` 和 `expanding_min()` 的运算结果。

? 请读者思考如何使用NumPy提供的ufunc函数计算图5-4中的三条曲线。

```
np.random.seed(42)
x = np.cumsum(np.random.randn(400))
x_max = pd.expanding_max(x)
x_min = pd.expanding_min(x)
x_mean = pd.expanding_mean(x)
```

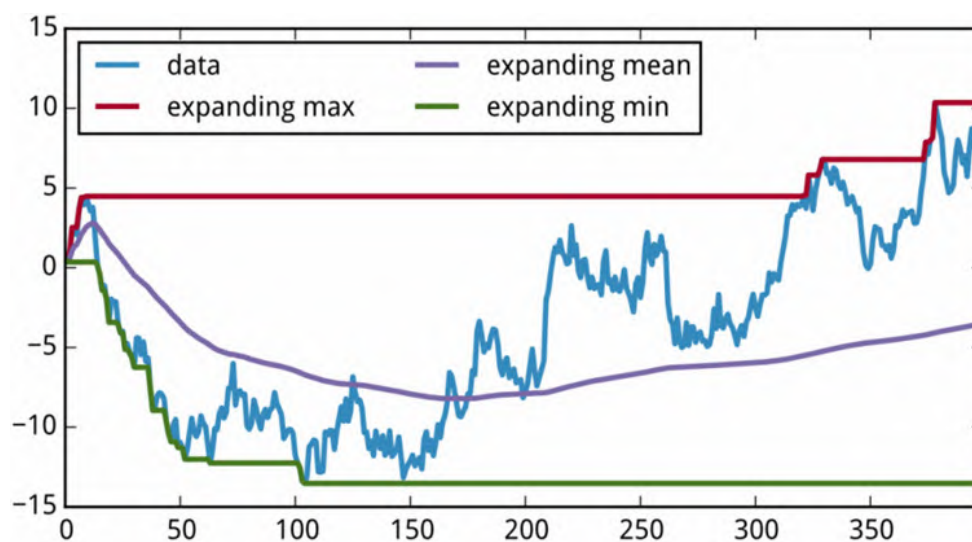


图5-4 用 `expanding_*` 计算历史最大值、平均值、最小值

字符串处理



与本节内容对应的Notebook为：[05-pandas/pandas-500-string.ipynb](#)。

Series对象提供了大量的字符串处理方法，由于数量众多，因此Pandas使用了一个类似名称空间的对象`str`来包装这些字符串相关的方法。例如下面的程序调用`str.upper()`将序列中的所有字母都转换为大写：

```
s_abc = pd.Series(["a", "b", "c"])
print s_abc.str.upper()

0    A
1    B
2    C

dtype: object
```

Python中包含两种字符串：字节字符串和Unicode字符串。通过`str.decode()`可以将字节字符串按照指定的编码解码为Unicode字符串。例如在UTF-8编码中，一个汉字占用三个字符，因此下面的`s_utf8`中的字符串长度分别为6、9、12。当调用`str.decode()`将其转换为Unicode字符串的序列之后，其各个元素的长度为实际的文字个数。`str.encode()`可以把Unicode字符串按照指定的编码转换为字节字符串，在常用的汉字编码GB2312中，一个汉字占用两个字节，因此`s_gb2312`的元素长度分别为4、6、8。


```
s_utf8 = pd.Series([b"北京", b"北京市", b"北京地区"])
s_unicode = s_utf8.str.decode("utf-8")
s_gb2312 = s_unicode.str.encode("gb2312")

s_utf8.str.len()  s_unicode.str.len()  s_gb2312.str.len()
-----
0      6          0      2          0      4
1      9          1      3          1      6
2     12          2      4          2      8
dtype: int64      dtype: int64      dtype: int64
```

无论Series对象包含哪种字符串对象，其dtype属性都是object，因此无法根据它判断字符串类型。在处理文本数据时，需要格外注意字符串的类型。

可以对str使用整数或切片下标，相当于对Series对象中的每个元素进行下标运算，例如：

```
print s_unicode.str[:2]

0      北京
1      北京
2      北京
dtype: object
```

字符串序列与字符串一样，支持加法和乘法运算，例如：

```
print s_unicode + u"-" + s_abc * 2

0      北京-aa
1      北京市-bb
2      北京地区-cc
dtype: object
```

也可以使用str.cat()连接两个字符串序列的对应元素：

```
print s_unicode.str.cat(s_abc, sep="-")

0      北京-a
```

```
1      北京市-b
2      北京地区-c
dtype: object
```

调用`astype()`方法可以对Series对象中的所有元素进行类型转换，例如下面将整数序列转换为字符串序列：

```
print s_unicode.str.len().astype(unicode)
0      2
1      3
2      4
dtype: object
```

`str`中的有些方法可以对元素类型为列表的Series对象进行处理，例如下面调用`str.split()`将`s`中的每个字符串使用字符“|”分隔，所得到的结果`s_list`的元素类型为列表。然后调用它的`str.join()`方法以逗号连接每个列表中的元素：

```
s = pd.Series(["a|bc|de", "x|xyz|yz"])
s_list = s.str.split("|")
s_comma = s_list.str.join(",")
```

	s		s_list		s_comma
0	a bc de	0	[a, bc, de]	0	a,bc,de
1	x xyz yz	1	[x, xyz, yz]	1	x,xyz,yz
	dtype: object		dtype: object		dtype: object

对字符串序列进行处理时，经常会得到元素类型为列表的序列。Pandas没有提供处理这种序列的方法，不过可以通过`str[]`获取其中的元素：

```
s_list.str[1]
0      bc
```

```
1    xyz
dtype: object
```

或者先将其转换为嵌套列表，然后再转换为DataFrame对象：

```
print pd.DataFrame(s_list.tolist(), columns=["A", "B", "C"])
   A    B    C
0  a   bc   de
1  x  xyz   yz
```

Pandas 还提供了一些正则表达式相关的方法。例如使用其中的 `str.extract()` 可以从字符串序列中抽取出需要的部分，得到 DataFrame 对象。下面的例子中，`df_extract1` 对应的正则表达式包含三个未命名的组，因此其结果包含三个自动命名的列。而 `df_extract2` 对应的正则表达式包含两个命名组，因此其列名为组名。

```
df_extract1 = s.str.extract(r"(\w+)\|(\w+)\|(\w+)")
df_extract2 = s.str.extract(r"(?P<A>\w+)\|(?P<B>\w+)\|")
df_extract1    df_extract2
-----
   0    1    2      A    B
0  a   bc   de  0  a   bc
1  x  xyz   yz  1  x  xyz
```

在处理数据时，经常会遇到这种以特定分隔符分隔关键字的数据，例如下面的数据可以用于表示有向图，其第一列为边的起点、第二列为以“|”分隔的多个终点。下面使用 `read_csv()` 读入该数据，得到一个两列的 DataFrame 对象：

```
import io
text = "" "A, B|C|D
```

```

B, E|F
C, A
D, B|C
"""

df = pd.read_csv(io.BytesIO(text), skipinitialspace=True,
header=None)

print df

```

	0	1
0	A	B C D
1	B	E F
2	C	A

可以使用下面的程序将上述数据转换为每行对应一条边的数据。

❶nodes是一个元素类型为列表的Series对象。❷调用NumPy数组的repeat()方法将第一列数据重复相应的次数。由于repeat()只能接受32位整数，而str.len()返回的是64位整数，因此还需要进行类型转换。❸将嵌套列表平坦化，转换为一维数组。

```

nodes = df[1].str.split("|") ❶
from_node = df[0].values.repeat(nodes.str.len().astype(np.int32)) ❷
to_node = np.concatenate(nodes) ❸

print pd.DataFrame({"from_node":from_node, "to_node":to_node})

```

	from_node	to_node
0	A	B
1	A	C
2	A	D
3	B	E
4	B	F
5	C	A
6	D	B
7	D	C

还可以把原始数据的第二列看作第一列数据的标签，为了后续的数据分析，通常使用`str.get_dummies()`将这种数据转换为布尔DataFrame对象，每一列与一个标签对应，元素值为1表示对应的行包含对应的标签：

```
print df[1].str.get_dummies(sep="|")
```

	A	B	C	D	E	F
0	0	1	1	1	0	0
1	0	0	0	0	1	1
2	1	0	0	0	0	0
3	0	1	1	0	0	0

当字符串操作很难用向量化的字符串方法表示时，可以使用`map()`函数，将针对每个元素运算的函数运用到整个序列之上：

```
df[1].map(lambda s:max(s.split("|")))
```

```
0    D
```

```
1    F
```

```
2    A
```

```
3    C
```

```
Name: 1, dtype: object
```

当用字符串序列表示分类信息时，其中会有大量相同的字符串，将其转换为分类(Category)序列可以节省内存、提高运算效率。例如在下面的`df_soil`对象中，`Contour`、`Depth`和`Gp`列都是表示分类的数据，因此有许多重复的字符串。

```
df_soil = pd.read_csv("Soils.csv", usecols=[2, 3, 4, 6])
```

```
print df_soil.dtypes
```

```
Contour    object
```

```
Depth      object
```

```
Gp         object
```

```
pH         float64
```

```
dtype: object
```

下面循环调用`astype("category")`将这三列转换为分类列：

```
for col in ["Contour", "Depth", "Gp"]:  
    df_soil[col] = df_soil[col].astype("category")  
print df_soil.dtypes  
Contour      category  
Depth        category  
Gp            category  
pH            float64  
dtype: object
```

与名称空间对象`str`类似，元素类型为`category`的`Series`对象提供了名称空间对象`cat`，其中保存了与分类序列相关的各种属性和方法。例如`cat.categories`是保存所有分类的`Index`对象：

```
Gp = df_soil.Gp  
print Gp.cat.categories  
Index([u'D0', u'D1', u'D3', u'D6', u'S0', u'S1', u'S3', u'S6',  
u'T0', u'T1',  
       u'T3', u'T6'],  
      dtype='object')
```

而`cat.codes`则是保存下标的整数序列，元素类型为`int8`，因此一个元素用一个字节表示。

Gp.head(5)		Gp.cat.codes.head(5)	
-----		-----	
0	T0	0	8
1	T0	1	8
2	T0	2	8
3	T0	3	8
4	T1	4	9
Name: Gp, dtype: category		dtype: int8	
Categories (12, object): [D0, D1, D3, ..., T1, T3, T6]			

分类数据有无序和有序两种，无序分类中的不同分类无法比较大小，例如性别；有序分类则可以比较大小，例如年龄段。上面创建的三个分类列为无序分类，可以通过`cat.as_ordered()`和`cat.as_unordered()`在这两种分类之间相互转换。下面的程序通过`cat.as_ordred()`将深度分类列转换为有序分类，注意最后一行分类名之间使用“<”连接，表示是有序分类。

```
depth = df_soil.Depth
depth.cat.as_ordered().head()
-----
0      0-10
1      0-10
2      0-10
3      0-10
4     10-30
dtype: category
Categories (4, object): [0-10 < 10-30 < 30-60 < 60-90]
```

如果需要自定义分类中的顺序，可以使用`cat.reorder_categories()`指定分类的顺序：

```
contour = df_soil.Contour
categories = ["Top", "Slope", "Depression"]
contour.cat.reorder_categories(categories, ordered=True).head()
-----
0      Top
1      Top
2      Top
3      Top
4      Top
```

```
dtype: category
```

```
Categories (3, object): [Top < Slope < Depression]
```

5.5 时间序列



与本节内容对应的Notebook为：05-pandas/pandas-600-datetime.ipynb。

Pandas提供了表示时间点、时间段和时间间隔等三种与时间有关的类型，以及元素为这些类型的索引对象，并提供了许多时间序列相关的函数。本节简要介绍一些与时间相关的对象和函数。在本章最后一节还会介绍一些相关的实例。

5.5.1 时间点、时间段、时间间隔

Timestamp对象从Python标准库中的datetime类继承，表示时间轴上的一个时刻。它提供了方便的时区转换功能。下面调用Timestamp.now()获取当前时间now，它是不包含时区信息的本地时间。调用其tz_localize()可以得到指定时区的Timestamp对象。而带时区信息的Timestamp对象可以通过其tz_convert()转换时区。下面的now_shanghai的时间以"+08:00"结尾，表示它是东八区的时间，将其转换为东京时间得到now_tokyo，它是东九区的时间：

```
now = pd.Timestamp.now()
now_shanghai = now.tz_localize("Asia/Shanghai")
```



```
now_tokyo = now_shanghai.tz_convert("Asia/Tokyo")
print u"本地时间:", now
print u"上海时区:", now_shanghai
print u"东京时区:", now_tokyo
本地时间: 2015-07-25 11:50:46.264000
上海时区: 2015-07-25 11:50:46.264000+08:00
东京时区: 2015-07-25 12:50:46.264000+09:00
```

不同时区的时间可以比较，而本地时间和时区时间无法比较：

```
now_shanghai == now_tokyo
True
```

通过pytz模块的common_timezones()可以获得常用的表示时区的字符串：

```
import pytz
pytz.common_timezones
['Africa/Abidjan',
 'Africa/Accra',
 'Africa/Addis_Ababa',
 'Africa/Algiers',
 ...]
```

Period对象表示一个标准的时间段，例如某年、某月、某日、某小时等。时间段的长短由freq属性决定。下面的程序调用Period.now()，分别获得包含当前时间的日周期时间段和小时周期时间段。

```
now_day = pd.Period.now(freq="D")
now_hour = pd.Period.now(freq="H")

now_day                                now_hour
```

```
-----
```

Period('2015-07-25', 'D')	Period('2015-07-25 11:00', 'H')
---------------------------	---------------------------------

freq属性是一个描述时间段的字符串，其可选值可以通过下面的代码获得：

```
from pandas.tseries import frequencies
frequencies._period_code_map.keys()
frequencies._period_alias_dictionary()
```

对于周期为年度和星期的时间段，可以通过freq指定开始的时间。例如“W”表示以星期天开始的星期时间段，而“W-MON”则表示以星期一开始的星期时间段：

```
now_week_sun = pd.Period.now(freq="W")
now_week_mon = pd.Period.now(freq="W-MON")
```

	now_week_sun
now_week_mon	

```
-----
```

Period('2015-07-20/2015-07-26', 'W-SUN')	Period('2015-07-21/2015-07-27', 'W-MON')
--	--

时间段的起点和终点可以通过start_time和end_time属性获得，它们都是表示时间点的Timestamp对象：

now_day.start_time	now_day.end_time
--------------------	------------------

```
-----
```

Timestamp('2015-07-25 00:00:00')	Timestamp('2015-07-25 23:59:59.999999999')
----------------------------------	--

调用Timestamp对象的to_period()方法可以把时间点转换为包含该时间点的时间段。注意时间段不包含时区信息：

```
now_shanghai.to_period("H")
Period('2015-07-25 11:00', 'H')
```

Timestamp和Period对象可以通过其属性获得年、月、日等信息。下面分别获得年、月、日、星期几、一年中的第几天、小时等信息：

now.year	now.month	now.day	now.dayofweek	now.dayofyear	now.hour
-----	-----	-----	-----	-----	-----
2015	7	25	5	206	11

将两个时间点相减可以得到表示时间间隔的Timedelta对象，下面计算当前时刻离2015年国庆节还有多少时间：

```
national_day = pd.Timestamp("2015-10-1")
td = national_day - pd.Timestamp.now()
td
Timedelta('67 days 12:09:04.039000')
```

时间点和时间间隔之间可以进行加减运算：

```
national_day + pd.Timedelta("20 days 10:20:30")
Timestamp('2015-10-21 10:20:30')
```

Timedelta对象的days、seconds、microseconds和nanoseconds等属性分别获得它包含的天数、秒数、微秒数和纳秒数。注意这些值与对应的单位相乘并求和才是该对象表示的总时间间隔：

```
td.days    td.seconds    td.microseconds
-----
67L        43744L        39000L
```

也可以通过关键字参数直接指定时间间隔的天数、小时数、分钟数和秒数：

```
print pd.Timedelta(days=10, hours=1, minutes=2, seconds=10.5)
print pd.Timedelta(seconds=1000000)

10 days 01:02:10.500000
1 days 03:46:40
```

5.5.2 时间序列

上节介绍的Timestamp、Period和Timedelta对象都是表示单个值的对象，这些值可以放在索引或数据列中。下面的程序调用random_timestamps()创建一个包含5个随机时间点的DatetimeIndex对象ts_index，然后通过ts_index创建PeriodIndex类型的索引对象pd_index和TimedeltaIndex类型的索引对象td_index。DatetimeIndex、PeriodIndex和TimedeltaIndex都从Index继承，可以作为Series或DataFrame的索引。

random_timestamps()中的date_range()函数创建以start为起点、以end为终点、周期为freq的DatetimeIndex对象。

```
def random_timestamps(start, end, freq, count):
    index = pd.date_range(start, end, freq=freq)
    locations = np.random.choice(np.arange(len(index)), size=count, replace=False)
    locations.sort()
    return index[locations]

np.random.seed(42)
ts_index = random_timestamps("2015-01-01", "2015-10-01", freq="Min", count=5)
pd_index = ts_index.to_period("M")
td_index = pd.TimedeltaIndex(np.diff(ts_index))

print ts_index, "\n"
print pd_index, "\n"
print td_index, "\n"

DatetimeIndex(['2015-01-15 16:12:00', '2015-02-15 08:04:00',
                '2015-02-28 12:30:00', '2015-08-06 02:40:00',
                '2015-08-18 13:13:00'],
              dtype='datetime64[ns]', freq=None, tz=None)

PeriodIndex(['2015-01', '2015-02', '2015-02', '2015-08', '2015-08'], dtype='int64',
            freq='M')

TimedeltaIndex(['30 days 15:52:00', '13 days 04:26:00', '158 days 14:10:00',
                '12 days 10:33:00'],
              dtype='timedelta64[ns]', freq=None)
```

下面查看这三种索引对象的dtype属性。其中M8[ns]和m8[ns]是NumPy中表示时间点和时间间隔的dtype类型，内部采用64位整数存储时间信息，其中[ns]表示时间的最小单位为纳秒，能表示的时间范围大约是公元1678年到公元 2262年。PeriodIndex也使用64位整数，但是最小时间单位由其freq属性决定。

```
ts_index.dtype    pd_index.dtype    td_index.dtype
-----
dtype('<M8[ns]')  dtype('int64')    dtype('<m8[ns]')
```

这三种索引对象都提供了许多与时间相关的属性，例如：

```
ts_index.weekday    pd_index.month    td_index.seconds
-----
[3, 6, 5, 3, 1]    [1, 2, 2, 8, 8]    [57120, 15960, 51000, 37980]
```

`DatetimeIndex.shift(n, freq)`可以移动时间点，将当前的时间移动n个freq时间单位。对于天数、小时这样的精确单位，结果相当于与指定的时间间隔相加：

```
ts_index.shift(1, "H")
DatetimeIndex(['2015-01-15 17:12:00', '2015-02-15 09:04:00',
              '2015-02-28 13:30:00', '2015-08-06 03:40:00',
              '2015-08-18 14:13:00'],
              dtype='datetime64[ns]', freq=None, tz=None)
```

而对于月份这样不精确的时间单位，则移动一个单位相当于移到月头或月底：

```
ts_index.shift(1, "M")
DatetimeIndex(['2015-01-31 16:12:00', '2015-02-28 08:04:00',
              '2015-03-31 12:30:00', '2015-08-31 02:40:00',
              '2015-08-31 13:13:00'],
              dtype='datetime64[ns]', freq=None, tz=None)
```

`DatetimeIndex.normalize()`将时刻修改为当天的凌晨零点，可以理解为按日期取整：

```
ts_index.normalize()
DatetimeIndex(['2015-01-15', '2015-02-15', '2015-02-28', '2015-08-06',
              '2015-08-18'],
              dtype='datetime64[ns]', freq=None, tz=None)
```

如果希望对任意的时间周期取整，可以先通过`to_period()`将其转换为`PeriodIndex`对象，然后再调用`to_timestamp()`方法转换回`DatetimeIndex`

对象。to_timestamp()的how参数决定将时间段的起点还是终点转换为时间点，默认值为“start”。

```
ts_index.to_period("H").to_timestamp()
DatetimeIndex(['2015-01-15 16:00:00', '2015-02-15 08:00:00',
               '2015-02-28 12:00:00', '2015-08-06 02:00:00',
               '2015-08-18 13:00:00'],
              dtype='datetime64[ns]', freq=None, tz=None)
```

下面的Series对象ts_series的索引为DatetimeIndex对象，这种Series对象被称为时间序列：

```
ts_series = pd.Series(range(5), index=ts_index)
```

时间序列提供一些专门用于处理时间的方法，例如between_time()返回所有位于指定时间范围之内数据：

```
ts_series.between_time("9:00", "18:00")
2015-01-15 16:12:00    0
2015-02-28 12:30:00    2
2015-08-18 13:13:00    4
dtype: int64
```

而tshift()则将索引移动指定的时间：

```
ts_series.tshift(1, freq="D")
2015-01-16 16:12:00    0
2015-02-16 08:04:00    1
2015-03-01 12:30:00    2
2015-08-07 02:40:00    3
```

```
2015-08-19 13:13:00      4
dtype: int64
```

以PeriodIndex和TimedeltaIndex为索引的序列也可以使用tshift()对索引进行移动:

```
pd_series = pd.Series(range(5), index=pd_index)
td_series = pd.Series(range(4), index=td_index)

pd_series.tshift(1)    td_series.tshift(10, freq="H")
-----
2015-02      0          31 days 01:52:00      0
2015-03      1          13 days 14:26:00      1
2015-03      2          159 days 00:10:00      2
2015-09      3          12 days 20:33:00      3
2015-09      4          dtype: int64
Freq: M, dtype: int64
```

时间信息除了可以作为索引之外,还可以作为Series或DataFrame的列。下面分别将上述三种索引对象转换为Series对象,并查看其dtype属性:

```
ts_data = pd.Series(ts_index)
pd_data = pd.Series(pd_index)
td_data = pd.Series(td_index)

ts_data.dtype    pd_data.dtype    td_data.dtype
-----
dtype('<M8[ns]>') dtype('O')      dtype('<m8[ns]>')
```

可以看到Pandas的Series对象目前尚不支持使用64位整数表示时间段,因此使用对象数组保存所有的Period对象。而对于时间点和时间间隔数据则采用64位整数数组保存。

当序列的值为时间数据时,可以通过名字空间对象dt调用时间相关的属性和方法。例如:

ts_data.dt.hour	pd_data.dt.month	td_data.dt.days
-----	-----	-----
0 16	0 1	0 30
1 8	1 2	1 13
2 12	2 2	2 158
3 2	3 8	3 12
4 13	4 8	dtype: int64
dtype: int64	dtype: int64	

5.5.3 与NaN相关的函数



与本节内容对应的Notebook为：05-pandas/pandas-700-nan.ipynb。

Pandas使用NaN表示缺失的数据，由于整数列无法使用NaN，因此如果整数类型的列出现缺失数据，则会被自动转换为浮点数类型。下面将布尔类型的DataFrame对象传递给一个整数类型的DataFrame对象的where()方法。该方法将False对象的元素设置为NaN，注意其结果变成了浮点数类型，而没有NaN的列仍然为整数类型。

```

np.random.seed(41)
df_int = pd.DataFrame(np.random.randint(0, 10, (10, 3)), columns=list("ABC"))
df_int["A"] += 10
df_nan = df_int.where(df_int > 2)

```

```

df_int.dtypes  df_nan.dtypes
-----
A    int32    A    int32
B    int32    B    float64
C    int32    C    float64
dtype: object dtype: object

```

```

      df_int      df_nan
-----
      A B C      A B C
0  10 3 2  0  10 3 NaN
1  10 1 3  1  10 NaN 3
2  19 7 5  2  19 7 5
3  18 3 3  3  18 3 3

```

```

4  12 6 0  4  12 6 NaN
5  14 6 9  5  14 6 9
6  13 8 4  6  13 8 4
7  17 6 1  7  17 6 NaN
8  15 2 1  8  15 NaN NaN
9  15 3 2  9  15 3 NaN

```

`isnull()` 和 `notnull()` 用于判断元素值是否为NaN，它们返回全是布尔值的DataFrame对象。`df.notnull()` 和 `~df.isnull()` 的结果相同，但是由于 `notnull()` 少创建一个临时对象，其运算效率更高一些。

```

      df_nan.isnull()      df_nan.notnull()
-----
      A B C      A B C
0  False False True  0  True True False
1  False True False  1  True False True
2  False False False  2  True True True
3  False False False  3  True True True
4  False False True  4  True True False
5  False False False  5  True True True
6  False False False  6  True True True
7  False False True  7  True True False
8  False True True  8  True False False
9  False False True  9  True True False

```

`count()` 返回每行或每列的非NaN元素的个数：

df_nan.count()		df_nan.count(axis=1)	
-----		-----	
A	10	0	2
B	8	1	2
C	5	2	3
dtype:	int64	3	3
4	2		
		5	3
		6	3
		7	2
		8	1
		9	2
		dtype:	int64

对于包含NaN元素的数据，最简单的办法就是调用dropna()以删除包含NaN的行或列，当全部使用默认参数时，将删除包含NaN的所有行。可以通过thresh参数指定NaN个数的阈值，删除所有NaN个数大于等于该阈值的行。

df_nan.dropna()				df_nan.dropna(thresh=2)			
-----				-----			
	A	B	C		A	B	C
2	19	7	5	0	10	3	NaN
3	18	3	3	1	10	NaN	3
5	14	6	9	2	19	7	5
6	13	8	4	3	18	3	3
				4	12	6	NaN
				5	14	6	9
6	13	8	4				
				7	17	6	NaN
				9	15	3	NaN

当行数据按照某种物理顺序(例如时间)排列时，可以使用NaN前后的数据对其进行填充。ffill()使用之前的数据填充，而bfill()则使用之后的数据填充。interpolate()使用前后数据进行插值填充：

	df_nan.ffill()			df_nan.bfill()			df_nan.interpolate()		
	A	B	C	A	B	C	A	B	C
0	10	3	NaN	0	10	3	0	10	3.0 NaN
1	10	3	3	1	10	7	1	10	5.0 3
2	19	7	5	2	19	7	2	19	7.0 5
3	18	3	3	3	18	3	3	18	3.0 3
4	12	6	3	4	12	6	4	12	6.0 6
5	14	6	9	5	14	6	5	14	6.0 9
6	13	8	4	6	13	8	6	13	8.0 4
7	17	6	4	7	17	6 NaN	7	17	6.0 4
8	15	6	4	8	15	3 NaN	8	15	4.5 4
9	15	3	4	9	15	3 NaN	9	15	3.0 4

interpolate() 默认使用等距线性插值，可以通过其method参数指定插值算法。在下面的例子中，第0个元素和第2个元素的数值分别为3.0和7.0，因此当method参数默认省时下标为1的NaN被填充为前后两个元素的平均值5.0。而当method为"index"时，则使用索引值进行插值运算。由于第1个元素的索引与第2个元素的索引接近，因此其插值结果也接近第二个元素的值。

s = pd.Series([3, np.NaN, 7], index=[0, 8, 9])						
	s.interpolate()			s.interpolate(method="index")		
0	3		0	3.000000		
8	5		8	6.555556		
9	7		9	7.000000		
dtype: float64		dtype: float64				

此外还可以使用字典参数让fillna() 对不同的列使用不同的值填充NaN:

```
print df_nan.fillna({"B":-999, "C":0})
```

```
   A   B   C
0  10   3   0
1  10 -999  3
2  19   7   5
3  18   3   3
4  12   6   0
5  14   6   9
6  13   8   4
7  17   6   0
8  15 -999   0
9  15   3   0
```

各种聚合方法的skipna参数默认为True，因此计算时将忽略NaN元素，注意每行或每列是单独运算的。如果需要忽略包含NaN的整行，需要先调用dropna()。若将skipna参数设置为False，则包含NaN的行或列的运算结果为NaN。

df_nan.sum()	df_nan.sum(skipna=False)	df_nan.dropna().sum()
A 143	A 143	A 64
B 42	B NaN	B 24
C 24	C NaN	C 21
dtype: float64	dtype: float64	dtype: float64

df.combine_first(other)使用other填充df中的NaN元素。它将df中的NaN元素替换为other中对应标签的元素。在下面的例子中，df_nan中索引为1、2、8、9的行中的NaN被替换为df_other中相应的值：

```
df_other = pd.DataFrame(np.random.randint(0, 10, (4, 2)),
                        columns=["B", "C"],
                        index=[1, 2, 8, 9])
print df_nan.combine_first(df_other)
```

```
   A   B   C
0  10   3 NaN
1  10   4   3
2  19   7   5
3  18   3   3
4  12   6 NaN
```

```
5 14 6 9
6 13 8 4
7 17 6 NaN
8 15 4 5
9 15 3 5
```

5.5.4 改变DataFrame的形状



与本节内容对应的Notebook为：05-pandas/pandas-800-changeshape.ipynb。

本节介绍表5-4中的函数：

表5-4 本节要介绍的函数

函数名	功能	函数名	功能
concat	拼接多块数据	drop	删除行或列
set_index	设置索引	reset_index	将行索引转换为列
stack	将列索引转换为行索引	unstack	将行索引转换为列索引
reorder_levels	设置索引级别的顺序	swaplevel	交换索引中两个级别的顺序
sort_index	对索引排序	pivot	创建透视表
melt	透视表的逆变换	assign	返回添加新列之后的数据

DataFrame的shape属性和NumPy的二维数组相同，是一个有两个元素的元组。由于DataFrame的index和columns都支持MultiIndex索引对象，因此可以用DataFrame表示更高维的数据。

下面首先从CSV文件读入数据，并使用groupby()计算分组的平均值。关于groupby在后面的章节还会详细介绍。注意下面的soils_mean对象的行索引是多级索引：

```
soils = pd.read_csv("Soils.csv", index_col=0)[["Depth", "Contour", "Group", "pH", "N"]]
soils_mean = soils.groupby(["Depth", "Contour"]).mean()
```

soils.head()					soils_mean.head()				
	Depth	Contour	Group	pH	N				
1	0-10	Top	1	5.4	0.19	Depth	Contour		
2	0-10	Top	1	5.7	0.17	0-10	Depression	9	5.4 0.18
3	0-10	Top	1	5.1	0.26		Slope	5	5.5 0.22
4	0-10	Top	1	5.1	0.17		Top	1	5.3 0.2
5	10-30	Top	2	5.1	0.16	10-30	Depression	10	4.9 0.08
							Slope	6	5.3 0.1

1. 添加删除列或行

由于DataFrame可以看作一个Series对象的字典，因此通过DataFrame[colname] = values即可添加新列。有时新添加的列是从已经存在的列计算而来，这时可以使用eval()方法计算。例如下面的代码添加一个名为N_percent的新列，其值为N列乘上100：

```
soils["N_percent"] = soils.eval("N * 100")
```

assign()方法添加由关键字参数指定的列，它返回一个新的DataFrame对象，原数据的内容保持不变：


```
print soils.assign(pH2 = soils.pH + 1).head()
```

	Depth	Contour	Group	pH	N	N_percent	pH2
1	0-10	Top	1	5.4	0.19	19	6.4
2	0-10	Top	1	5.7	0.17	16	6.7
3	0-10	Top	1	5.1	0.26	26	6.1
4	0-10	Top	1	5.1	0.17	17	6.1
5	10-30	Top	2	5.1	0.16	16	6.1

`append()` 方法用于添加行，它没有 `inplace` 参数，只能返回一个全新对象。由于每次调用 `append()` 都会复制所有的数据，因此在循环中使用 `append()` 添加数据会极大地降低程序的运算速度。可以使用一个列表缓存所有的分块数据，然后调用 `concat()` 将所有这些数据沿着指定轴拼贴到一起。下面的程序比较二者的运算速度：

```
def random_dataframe(n):
    columns = ["A", "B", "C"]
    for i in range(n):
        nrow = np.random.randint(10, 20)
        yield pd.DataFrame(np.random.randint(0, 100, size=(nrow,
3)), columns=columns)
```

```
df_list = list(random_dataframe(1000))
```

```
%%time
df_res1 = pd.DataFrame([])
for df in df_list:
    df_res1 = df_res1.append(df)
Wall time: 1.37 s
```

```
%%time
df_res2 = pd.concat(df_list, axis=0)
```



```
Wall time: 118 ms
```

可以使用`keys`参数指定与每块数据对应的键，这样结果中的拼接轴将使用多级索引，方便快速获取原始的数据块。下面获取拼接之后的`DataFrame`对象中第0级标签为30的数据，并使用`equals()`方法判断它是否与原始数据中下标为30的数据块相同：

```
df_res3 = pd.concat(df_list, axis=0, keys=range(len(df_list)))
df_res3.loc[30].equals(df_list[30])

True
```

`drop()`删除指定标签对应的行或列，下面删除名为N和Group的两列：

```
print soils.drop(["N", "Group"], axis=1).head()
```

	Depth	Contour	pH	N_percent
1	0-10	Top	5.4	19
2	0-10	Top	5.7	16
3	0-10	Top	5.1	26
4	0-10	Top	5.1	17
5	10-30	Top	5.1	16

2. 行索引与列之间的相互转换

`reset_index()`可以将索引转换为列，通过`level`参数可以指定被转换为列的级别。如果只希望从索引中删除某个级别，可以设置`drop`参数为`True`。

```
print soils_mean.reset_index(level="Contour").head()
```

	Contour	Group	pH	N
Depth				
0-10	Depression	9	5.4	0.18
0-10	Slope	5	5.5	0.22
0-10	Top	1	5.3	0.2
10-30	Depression	10	4.9	0.08
10-30	Slope	6	5.3	0.1

`set_index()` 将列转换为行索引，如果 `append` 参数为 `False` (默认值)，则删除当前的行索引；若为 `True`，则为当前的索引添加新的级别。

```
print soils_mean.set_index("Group", append=True).head()
```

			pH	N
Depth	Contour	Group		
0-10	Depression	9	5.4	0.18
	Slope	5	5.5	0.22
	Top	1	5.3	0.2
10-30	Depression	10	4.9	0.08
	Slope	6	5.3	0.1

3. 行索引和列索引的相互转换

`stack()` 方法把指定级别的列索引转换为行索引，而 `unstack()` 则把行索引转换为列索引。下面的程序将行索引中的第一级转换为列索引的第一级，所得到的结果中行索引为单级索引，而列索引为多级索引：

```
print soils_mean.unstack(1)[["Group", "pH"]].head()
```

	Group			pH			
	Contour	Depression	Slope	Top	Depression	Slope	Top
Depth							
0-10		9	5	15.4	5.5	5.3	
10-30		10	6	24.9	5.3	4.8	
30-60		11	7	34.4	4.3	4.2	
60-90		12	8	44.2	3.9	3.9	

无论是 `stack()` 还是 `unstack()`，当所有的索引被转换到同一个轴上时，将得到一个 `Series` 对象：

```
print soils_mean.stack().head(10)
```

```
Depth Contour
0-10 Depression Group    9
      pH          5.4
      N          0.18
      Slope      Group    5
      pH          5.5
      N          0.22
      Top       Group    1
      pH          5.3
      N          0.2
10-30 Depression Group   10
dtype: float64
```

4. 交换索引的等级

`reorder_levels()` 和 `swaplevel()` 交换指定轴的索引级别。下面调用 `swaplevel()` 交换行索引的两个级别，然后调用 `sort_index()` 对新的索引进行排序：

```
print soils_mean.swaplevel(0, 1).sort_index()
```

```
          Group  pH    N
Contour  Depth
Depression 0-10    9  5.4  0.18
           10-30  10  4.9  0.08
           30-60  11  4.4  0.051
           60-90  12  4.2  0.04

Slope     0-10    5  5.5  0.22
           10-30    6  5.3  0.1
           30-60    7  4.3  0.061
           60-90    8  3.9  0.043

Top       0-10    1  5.3  0.2
           10-30    2  4.8  0.12
           30-60    3  4.2  0.08
           60-90    4  3.9  0.058
```

5. 透视表

`pivot()` 可以将DataFrame中的三列数据分别作为行索引、列索引和元素值，将这三列数据转换为二维表格：

```
df = soils_mean.reset_index()[["Depth", "Contour", "pH", "N"]]
df_pivot_pH = df.pivot("Depth", "Contour", "pH")
```

df				df_pivot_pH			
Depth	Contour	pH	N	Contour	Depression	Slope	Top
0	0-10	Depression	5.4 0.18	Depth			
1	0-10	Slope	5.5 0.22	0-10	5.4	5.5	5.3
2	0-10	Top	5.3 0.2	10-30	4.9	5.3	4.8
3	10-30	Depression	4.9 0.08	30-60	4.4	4.3	4.2
4	10-30	Slope	5.3 0.1	60-90	4.2	3.9	3.9
5	10-30	Top	4.8 0.12				
6	30-60	Depression	4.4 0.051				
7	30-60	Slope	4.3 0.061				
8	30-60	Top	4.2 0.08				
9	60-90	Depression	4.2 0.04				
10	60-90	Slope	3.9 0.043				
11	60-90	Top	3.9 0.058				

`pivot()` 的三个参数`index`、`columns`和`values`只支持指定一列数据。若不指定`values`参数，就将剩余的列都当作元素值列，得到多级列索引：

```
print df.pivot("Depth", "Contour")
```

Depth	pH			N			
	Contour	Depression	Slope	Top	Depression	Slope	Top
0-10		5.4	5.5	5.3	0.18	0.22	0.2
10-30		4.9	5.3	4.8	0.08	0.1	0.12
30-60		4.4	4.3	4.2	0.051	0.061	0.08
60-90		4.2	3.9	3.9	0.04	0.043	0.058

`melt()` 可以看作`pivot()` 的逆变换。由于它不能对行索引进行操作，因此先调用`reset_index()` 将行索引转换为列，然后用`id_vars` 参数指定该列为标识列：

```
df_before_melt = df_pivot_pH.reset_index()
df_after_melt = pd.melt(df_before_melt, id_vars="Depth", value_name="pH")
```

df_before_melt					df_after_melt		
Contour	Depth	Depression	Slope	Top	Depth	Contour	pH
0	0-10	5.4	5.5	5.3	0	0-10	Depression 5.4
1	10-30	4.9	5.3	4.8	1	10-30	Depression 4.9
2	30-60	4.4	4.3	4.2	2	30-60	Depression 4.4
3	60-90	4.2	3.9	3.9	3	60-90	Depression 4.2
					4	0-10	Slope 5.5
					5	10-30	Slope 5.3
					6	30-60	Slope 4.3
					7	60-90	Slope 3.9
					8	0-10	Top 5.3
					9	10-30	Top 4.8
					10	30-60	Top 4.2
					11	60-90	Top 3.9

5.6 分组运算



与本节内容对应的Notebook为：05-pandas/pandas-900-groupby.ipynb。

所谓分组运算是指使用特定的条件将数据分为多个分组，然后对每个分组进行运算，最后再将结果整合起来。Pandas中的分组运算由DataFrame或Series对象的groupby()方法实现。

下面以某种药剂的实验数据“dose.csv”为例介绍如何使用分组运算分析数据。在该数据集中使用了“ABCD”4种不同的药剂处理方式(Tmt)，针对不同性别(Gender)、不同年龄(Age)的患者进行药剂实验，记录下药剂的投药量(Dose)与两种药剂反应(Response)。

```
dose_df = pd.read_csv("dose.csv")
print dose_df.head(3)
```

	Dose	Response1	Response2	Tmt	Age	Gender
0	50	9.9	10	C	60s	F
1	15	0.002	0.004	D	60s	F
2	25	0.63	0.8	C	50s	M

5.6.1 groupby() 方法

如图5-5所示，分组操作中涉及两组数据：源数据和分组数据。将分组数据传递给源数据的groupby()方法以完成分组。groupby()的axis参数默认为0，表示对源数据的行进行分组。源数据中的每行与分组数据中的每个元素对应，分组数据中的每个唯一值对应一个分组。由于图中的分组数据中有两个唯一值，因此得到两个分组。



groupby()并不立即执行分组操作，而只是返回保存源数据和分组数据的GroupBy对象。在需要获取每个分组的实际数据时，GroupBy对象才会执行分组操作。

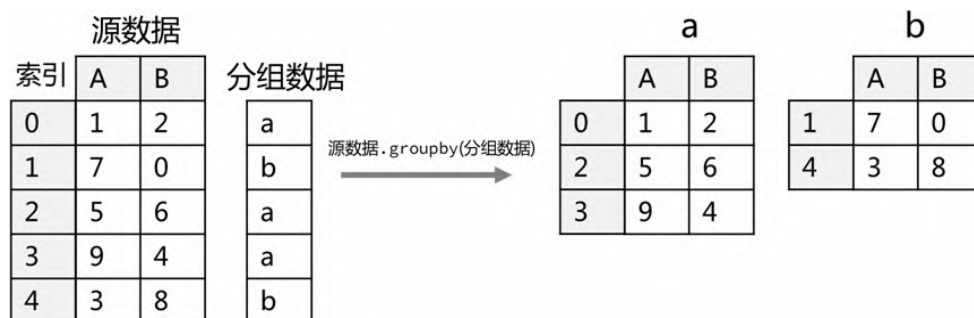


图5-5 groupby() 分组示意图

当分组用的数据在源数据中时，可以直接通过列名指定分组数据。当源数据是DataFrame类型时，groupby()方法返回一个DataFrameGroupBy对象。若源数据是Series类型，则返回SeriesGroupBy对象。在下面的例子中使用Tmt列对源数据分组：

```
tmt_group = dose_df.groupby("Tmt")
print type(tmt_group)

<class 'pandas.core.groupby.DataFrameGroupBy'>
```

还可以使用列表传递多组分组数据给groupby()，例如下面的程序使用处理方式与年龄对源数据分组：

```
tmt_age_group = dose_df.groupby(["Tmt", "Age"])
```

当分组数据不在源数据中时，可以直接传递分组数据。在下面的例子中对长度与源数据的行数相同、取值范围为[0, 5)的随机整数数组进行分组，这样就将源数据随机分成了5组：

```
random_values = np.random.randint(0, 5, dose_df.shape[0])
random_group = dose_df.groupby(random_values)
```

当分组数据可以通过源数据的行索引计算时，可以将计算函数传递给groupby()。下面的例子使用行索引值除以3的余数进行分组，因此将源数据的每行交替地分为3组。这是因为源数据的行索引为从0开始的整数序列。

```
alternating_group = dose_df.groupby(lambda n:n % 3)
```

上述三种分组数据可以任意自由组合，例如下面的例子同时使用源数据中的性别列、函数以及数组进行分组：

```
crazy_group = dose_df.groupby(["Gender", lambda n: n % 2,
random_values])
```

5.6.2 GroupBy对象

使用`len()`可以获取分组数：

```
print len(tmt_age_group), len(crazy_group)
10 20
```

GroupBy对象支持迭代接口，它与字典的`iteritems()`方法类似，每次迭代得到分组的键和数据。当使用多列数据分组时，与每个组对应的键是一个元组：

```
for key, df in tmt_age_group:
    print "key =", key, ", shape =", df.shape
key = ('A', '50s') , shape = (39, 6)
key = ('A', '60s') , shape = (26, 6)
key = ('B', '40s') , shape = (13, 6)
key = ('B', '50s') , shape = (13, 6)
key = ('B', '60s') , shape = (39, 6)
key = ('C', '40s') , shape = (13, 6)
key = ('C', '50s') , shape = (13, 6)
key = ('C', '60s') , shape = (39, 6)
key = ('D', '50s') , shape = (52, 6)
key = ('D', '60s') , shape = (13, 6)
```

由于Python的赋值语句支持迭代接口，因此可以使用下面的语句快速为每个分组数据指定变量名。这是因为我们知道只有4种药剂处理方式，并

且GroupBy对象默认会对分组键进行排序。可以将groupby()的sort参数设置为False以关闭排序功能，这样可以稍微提高大量分组时的运算速度。

```
(_, df_A), (_, df_B), (_, df_C), (_, df_D) = tmt_group
```



由于GroupBy对象有keys属性，因此无法通过dict(tmt_group)直接将其转换为字典，可以先将其转换为迭代器，再转换为字典dict(iter(tmt_group))。

get_group()方法可以获得与指定的分组键对应的数据，例如：

```
tmt_group.get_group("A").head(3)
```

	Dose	Response1	Response2	Tmt	Age	Gender
6	1	0	0	A	50s	F
10	15	5.2	5.2	A	60s	F
12	5	0	0.001	A	60s	F


```
tmt_age_group.get_group(("A", "50s")).head(3)
```

	Dose	Response1	Response2	Tmt	Age	Gender
6	1	0	0	A	50s	F
17	5	0	0.003	A	50s	M
34	40	11	10	A	50s	M

对GroupBy的下标操作将获得一个只包含源数据中指定列的新GroupBy对象，通过这种方式可以先使用源数据中的某些列进行分组，然后选择另一些列进行后续计算。

```
print tmt_group["Dose"]  
print tmt_group[["Response1", "Response2"]]  
<pandas.core.groupby.SeriesGroupBy object at 0x0C6076F0>  
<pandas.core.groupby.DataFrameGroupBy object at 0x0C6077F0>
```

GroupBy类中定义了__getattr__()方法，因此当获取GroupBy中未定义的属性时，将按照下面的顺序操作：

- 如果属性名是源数据对象的某列的名称，则相当于GroupBy[name]，即获取针对该列的GroupBy对象。
- 如果属性名是源数据对象的方法，则相当于通过apply()对每个分组调用该方法。注意Pandas中定义了转换为apply()的方法集合，只有在此集合之中的方法才会被自动转换。关于apply()方法将在下一小节详细介绍。

下面的程序得到对源数据中的Dose列进行分组的GroupBy对象：

```
print tmt_group.Dose
<pandas.core.groupby.SeriesGroupBy object at 0x05D96B70>
```

5.6.3 分组—运算—合并

通过GroupBy对象提供的agg()、transform()、filter()以及apply()等方法可以实现各种分组运算。每个方法的第一个参数都是一个回调函数，该函数对每个分组的数据进行运算并返回结果。这些方法根据回调函数的返回结果生成最终的分组运算结果。

1. agg()—聚合

agg()对每个分组中的数据进行聚合运算。所谓聚合运算是指将一组由N个数值组成的数据转换为单个数值的运算，例如求和、平均值、中间值甚至随机取值等都是聚合运算。其回调函数接收的数据是表示每个分组中每列数据的Series对象，若回调函数不能处理Series对象，则agg()会接着尝

试将整个分组的数据作为DataFrame对象传递给回调函数。回调函数对其参数进行聚合运算，将Series对象转换为单个数值，或将DataFrame对象转换为Series对象。agg() 返回一个DataFrame对象，其行索引为每个分组的键，而列索引为源数据的列索引。

在图5-6中，上方两个表格显示了两个分组中的数据，而下方左侧两个表格显示了对这两个分组执行聚合运算之后的结果。其中最左侧的表格是执行g.agg(np.max)的结果。由于np.max()能对Series对象进行运算，因此agg()将分组a和分组b中的每列数据分别传递给np.max()以计算每列的最大值，并将所有最大值聚合成一个DataFrame对象。例如分组a中的B列传递给np.max()的计算结果为6，该数值存放在结果的第a行、第B列中。

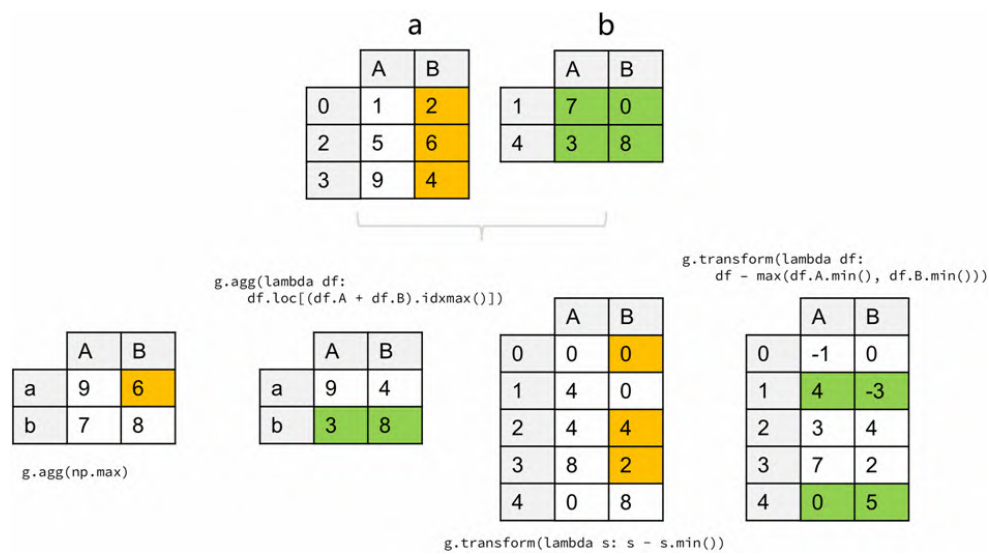


图5-6 agg() 和transform() 的运算示意图

左侧第二个表格对应的程序为：

```
g.agg(lambda df: df.loc[(df.A + df.B).idxmax()])
```

由于在回调函数中访问了属性A和B，这两个属性在表示每列数据的Series对象中不存在，因此传递Series对象给回调函数的尝试失败。于是agg() 接下来尝试将表示整个分组数据的DataFrame对象传递给回调函数。

该回调函数每次返回结果中的一行，例如图中分组b对应的运算结果为第b行。该回调函数返回每个分组中A+B最大的那一行。

下面是对tmt_group进行聚合运算的例子。❶计算每个分组中每列的平均值，注意结果中自动剔除了无法求平均值的字符串列。❷找到每个分组中Response1最大的那一行，由于回调函数对表示整个分组的DataFrame进行运算，因此结果中包含了源数据中的所有列。

```
agg_res1 = tmt_group.agg(np.mean) ❶
agg_res2 = tmt_group.agg(lambda df:df.loc[df.Response1.idxmax()]) ❷
```

agg_res1				agg_res2					
-----				-----					
	Dose	Response1	Response2		Dose	Response1	Response2	Age	Gender
Tmt				Tmt					
A	34	6.7	6.9	A	80	11	10	60s	F
B	34	5.6	5.5	B	1e+02	11	10	50s	M
C	34	4	4.1	C	60	10	11	50s	M
D	34	3.3	3.2	D	80	11	9.9	60s	F

2. transform() — 转换

transform()对每个分组中的数据进行转换运算。与agg()相同，首先尝试将表示每列的Series对象传递给回调函数，如果失败，将表示整个分组的DataFrame对象传递给回调函数。回调函数的返回结果与参数的形状相同，transform()将这些结果按照源数据的顺序合并在一起。

图5-6的下方右侧两个表格为transform()的运算结果，它们对应的回调函数分别对Series和DataFrame对象进行处理。注意这两个表格的行索引与源数据的行索引相同。

下面是对tmt_group进行转换运算的例子。❶回调函数能对Series对象进行运算，因此运算结果中不包含源数据中的字符串列。❷由于Series对象没有assign()方法，因此transform()在尝试Series失败之后，将表示整个

分组的DataFrame对象传递给回调函数。该回调函数只对Response1列进行转换。

<pre>transform_res1 = tmt_group.transform(lambda s:s - s.mean()) ❶</pre>									
<pre>transform_res2 = tmt_group.transform(lambda df:df.assign(Response1=df.Response1 - df.Response1.mean())) ❷</pre>									
transform_res1.head(5)				transform_res2.head(5)					

	Dose	Response1	Response2		Dose	Response1	Response2	Age	Gender
0	16	5.8	5.9	0	50	5.8	10	60s	F
1	-19	-3.3	-3.2	1	15	-3.3	0.004	60s	F
2	-8.5	-3.4	-3.3	2	25	-3.4	0.8	50s	M
3	-8.5	-2.7	-2.6	3	25	-2.7	1.6	60s	F
4	-19	-4	-4.1	4	15	-4	0.02	60s	F

3. filter() — 过滤

filter() 对每个分组进行条件判断。它将表示每个分组的DataFrame对象传递给回调函数，该函数返回True或False，以决定是否保留该分组。filter() 的返回结果是过滤掉一些行之后的DataFrame对象，其行索引与源数据的行索引的顺序一致。

下面的程序保留Response1列的最大值小于11的分组，注意结果中包含用于分组的列Tmt：

<pre>print tmt_group.filter(lambda df:df.Response1.max() < 11).head()</pre>									
	Dose	Response1	Response2	Tmt	Age	Gender			
0	50	9.9	10	C	60s	F			
1	15	0.002	0.004	D	60s	F			
2	25	0.63	0.8	C	50s	M			
3	25	1.4	1.6	C	60s	F			
4	15	0.01	0.02	C	60s	F			

4. apply() — 运用

`apply()` 将表示每个分组的DataFrame对象传递给回调函数并收集其返回值，并将这些返回值按照某种规则合并。`apply()` 的用法十分灵活，可以实现上述`agg()`、`transform()` 和`filter()` 方法的功能。它会根据回调函数的返回值的类型选择恰当的合并方式，然而这种自动选择有时会得到令人费解的结果。

图5-7显示了对包含a和b两个分组的GroupBy对象g执行`apply()` 后得到的4种结果：

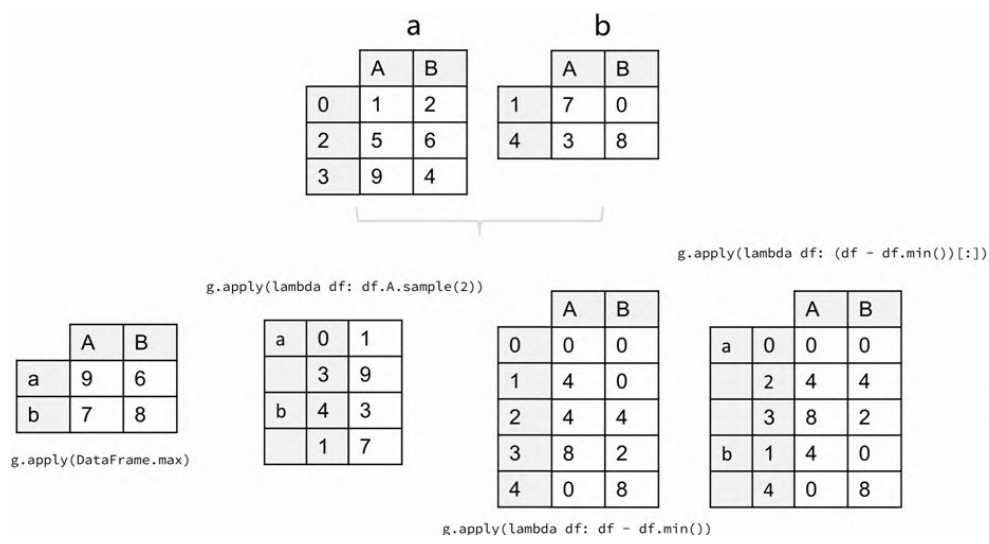


图5-7 `apply()` 的运算示意图

如图5-7(左一)所示，回调函数为`DataFrame.max`，它计算DataFrame对象中每列的最大值，返回一个以列名为索引的Series对象，因此对于所有的分组数据返回的索引都是相同的。这种情况下`apply()` 的结果与`agg()` 相同，是一个以每个分组的键为行索引、以所有返回对象的索引为列索引的DataFrame对象。

如图5-7(左二)所示，当回调函数返回的Series对象的索引不是全部一致时，`apply()` 将这些Series对象沿垂直方向连接在一起，得到一个多级索引的Series对象。多级索引由分组的键和每个Series对象的索引构成。

在图5-7(左三)中，回调函数返回的是DataFrame对象，并且结果的行索引与参数的行索引是同一索引对象，即满足如下条件：

```
df.index is (df - df.min()).index
```

则apply() 返回的DataFrame对象的行索引与源数据的索引一致，这与filter()的结果相同。

在图5-7(左四)中，回调函数的返回结果与(图5-7左三)相同，但由于使用[:]复制了整个数据，因此其返回对象与参数的索引不是同一对象。这种情况下，将按照分组的顺序沿垂直方向将所有返回结果连接在一起，得到一个多级索引的DataFrame对象。多级索引由分组的键和每个DataFrame对象的索引构成。



注意目前的版本采用is判断索引是否相同，很容易引起混淆，未来的版本可能会对这一点进行修改。

下面计算tmt_group的每个分组中每列的最大值和平均值。注意最大值的结果中包含字符串列，而平均值的结果中不包含字符串列：

tmt_group.apply(pd.DataFrame.max)							tmt_group.apply(pd.DataFrame.mean)			
-----							-----			
	Dose	Response1	Response2	Tmt	Age	Gender		Dose	Response1	Response2
Tmt							Tmt			
A	1e+02	11	11	A	60s	M	A	34	6.7	6.9
B	1e+02	11	10	B	60s	M	B	34	5.6	5.5
C	1e+02	10	11	C	60s	M	C	34	4	4.1
D	1e+02	11	9.9	D	60s	M	D	34	3.3	3.2

下面的程序从每个分组的Response1列随机取两个数值。❶由于sample() 保留与数值对应的标签，因此结果是一个多级标签的Series对象。❷对sample() 的结果调用reset_index() 方法，这样所有返回结果的标

签全部相同，因此得到的结果是一个DataFrame对象，其每一行与一个分组对应。

```
sample_res1 = tmt_group.apply(lambda df:df.Response1.sample(2)) ❶
sample_res2 = tmt_group.apply(
    lambda df:df.Response1.sample(2).reset_index(drop=True)) ❷
```

sample_res1			sample_res2		
-----			-----		
Tmt			Response1	0	1
A	248	10	Tmt		
	164	10	A	10	10
B	113	0.19	B	10	10
	26	9.4	C	0.004	9.9
C	191	10	D	0.33	11
	236	1.7			
D	188	0.061			
	8	0.001			
Name: Response1, dtype: float64					

当回调函数的返回值是DataFrame对象时，根据其行标签是否与参数对象的行标签为同一对象，会得到不同的结果：

```
group = tmt_group[["Response1", "Response1"]]
apply_res1 = group.apply(lambda df:df - df.mean())
apply_res2 = group.apply(lambda df:(df - df.mean())[:])
```

apply_res1.head()			apply_res2.head()		
-----			-----		
	Response1	Response1		Response1	Response1
0	5.8	5.8	Tmt		
1	-3.3	-3.3	A	6	-6.7
2	-3.4	-3.4		10	-1.5
3	-2.7	-2.7		12	-6.7
4	-4	-4		17	-6.7
				32	2.6

当回调函数返回None时，将忽略该返回值，因此可以实现filter()的功能。下面的程序从Response1的均值大于5的分组中随机取两行数据：


```
print tmt_group.apply(lambda df:None if df.Response1.mean() < 5 else df.sample(2))
```

	Dose	Response1	Response2	Tmt	Age	Gender
Tmt						
A	235	60	9.8	10	A	50s
	164	20	10	10	A	50s
B	9	40	11	10	B	60s
	16	30	9.8	10	B	60s

Pandas使用Cython对一些常用的聚合功能进行了优化处理，例如 `mean()`、`median()`、`var()` 等。此外，`GroupBy` 还自动将一些常用的 `DataFrame` 方法用 `apply()` 包装。因此，通过 `GroupBy` 对象调用这些方法就相当于将这些方法作为回调函数传递给 `apply()`。

下面的例子分别调用Cython编写的提速方法 `mean()` 和使用 `apply()` 包装之后的 `quantile()` 方法：

tmt_group.mean()				tmt_group.quantile(q=0.75)			
	Dose	Response1	Response2		Dose	Response1	Response2
Tmt				Tmt			
A	34	6.7	6.9	A	50	10	10
B	34	5.6	5.5	B	50	9.8	10
C	34	4	4.1	C	50	9.6	9.6
D	34	3.3	3.2	D	50	8.9	8.4

本节以 `DataFrameGroupBy` 对象为例介绍了分组运算的基本概念以及常用方法。Pandas提供的分组运算功能十分强大，建议读者在理解了本节的内容之后，再详细阅读官方的帮助文档，以了解更多的用法和技巧。

5.7 数据处理和可视化实例



与本节内容对应的 Notebook 为：05-pandas/pandas-A00-examples.ipynb。

作为本章的最后一节，让我们看两个用Pandas分析实际数据的例子。本节使用Pandas提供的绘图方法`plot()`将计算结果显示为图表，其内部使用`matplotlib`绘图。关于绘图方面的详细信息请读者参考`matplotlib`章节。

5.7.1 分析Pandas项目的提交历史

Pandas的源代码托管于GitHub，由世界各地的爱好者共同开发。下面让我们使用Pandas分析它的提交记录文件`data/pandas.log`。如果读者的计算机中安装了Git版本控制软件，可以通过如下命令创建该文件：

```
git clone https://github.com/pydata/pandas.git
cd pandas
git log > ../pandas.log
```

该文件中的一条提交记录由多行文本构成，例如：

```
commit 758ca05e2eb04532b5d78331ba87c291038e2c61
Author: Garrett-R <xxxx@xxxxx.com>
Date:   Sat Jun 27 15:11:12 2015 -0700

    DOC: Add warning for newbs not to edit auto-generated file,
#10456
```

由于该文件不是由特定分隔符分隔的文本文件，无法使用`read_csv()`读取，因此我们自己编写数据读取函数`read_git_log()`。它是一个生成器函数，每次迭代返回一个包含作者名、日期和提交说明的元组。

```
def read_git_log(log_fn):
    import io
```



```
"Message"])\n\nprint df_commit.shape\n\n(12109, 3)
```

为了分析时间数据，需要将提交时间的字符串转换为时间列，使用 `to_datetime()` 可以快速完成这个任务，它会自动尝试各种常用的日期格式。由下面的输出可知，它进行了时区转换，将所有的时间都转换成了世界标准时间：

```
df_commit["Date"] = pd.to_datetime(df_commit.DateString)\nprint df_commit[["DateString", "Date"]].head()
```

		DateString	Date
0	Tue Jul 7 23:43:31 2015 -0500	2015-07-08 04:43:31	
1	Tue Jul 7 12:18:50 2015 -0700	2015-07-07 19:18:50	
2	Tue Jul 7 13:37:38 2015 -0500	2015-07-07 18:37:38	
3	Sat Jun 27 15:11:12 2015 -0700	2015-06-27 22:11:12	
4	Tue Jul 7 10:53:55 2015 -0500	2015-07-07 15:53:55	

为了统计每个时区的提交次数，下面将表示时区的部分提取出来，保存到 `Timezone` 列中：

```
df_commit["Timezone"] = df_commit.DateString.str[-5:]
```

在提交说明的每行开头可能会有全部大写字母的单词，该单词通常用于描述提交的分类。在大致浏览提交记录之后，决定使用正则表达式 `"^([A-Z/]{2,12})"` 提取这种分类信息。其中 `^` 与 `re.MULTILINE` 配合使用，可以匹配每行的开头；`()` 括起来的部分就是要提取的内容；`[A-Z/]` 匹配大写字母或斜杠字符；`{2,12}` 表示前面的匹配重复2到12次。

```
import re\n\ndf_commit["Type"] = df_commit.Message.str.extract(r"^([A-Z/]{2,12})", flags=re.MULTILINE)
```

下面使用`value_counts()`统计时区和分类。可以看到美国所在时区的提交数最多，修复BUG的提交数最多。

```
type_counts = pd.value_counts(df_commit.Type)
```

-0500	2793	ENH	1720
-------	------	-----	------

+0200	1052	TST	1117
-------	------	-----	------

```
dtype: int64      dtype: int64
```

为了方便后续处理，下面将Date列设置为行索引，并且按照时间顺序排序。注意我们设置drop参数为False，保留Date列。

```
df.commit.sort_index(inplace=True)
```

下面统计两次连续提交之间的时间间隔，结果如图5-8所示。❶对Date列调用diff()方法计算前后两个时间点的差，由于最开始的数据无法计算，因此得到的值为NaN，这里调用dropna()来删除NaN。❷为了方便绘制直方统计图，这里将时间差转换为小时数。❸调用plot()方法绘图，通过kind参数指定图表的类型为“hist”，figsize参数指定图表的大小，其余的参数都传递给实际的绘图函数hist()，plot()方法返回表示子图的对象。

```
hours_delta = time_delta.dt.days * 24 + time_delta.dt.seconds /
```

```
ax = hours_delta.plot(kind="hist", figsize=(8, 3), 3
```

```
ax.set_xlabel("Hours")
```

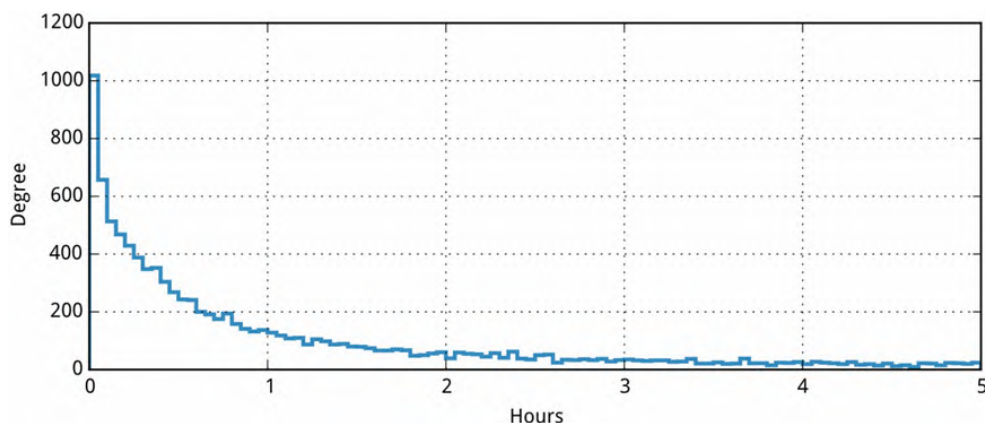


图5-8 两次提交的时间间隔统计

下面的程序绘制每个星期的提交数，结果如图5-9所示。先调用时间序列的`resample()`方法对其进行分组计算，其第一个参数为表示分组时间周期的字符串，“W”表示按星期分组。`how`参数指定对每个分组执行的运算。“count”表示返回每个分组中值不为NaN的元素个数。因此可以使用不包含NaN的任何列得到相同的结果。

```
ax = df_commit.Author.resample("W",
how="count").plot(kind="area", figsize=(8, 2.5))
ax.grid(True)
ax.set_ylabel(u"提交次数")
```

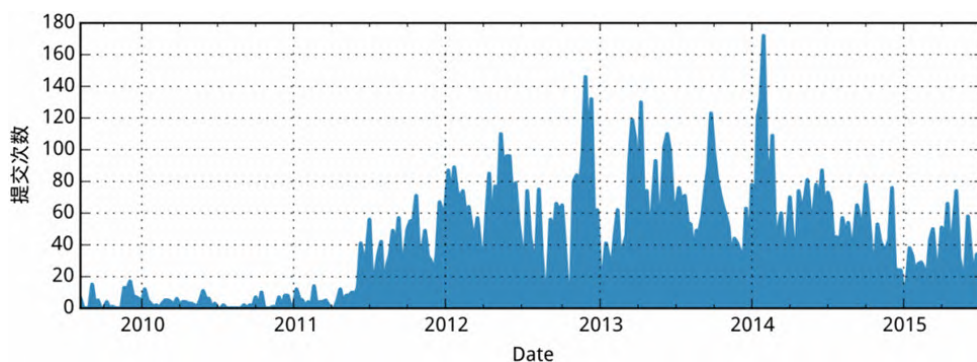


图5-9 每个星期的提交次数

`how`参数还支持回调函数，例如下面的程序绘制每个月的提交人数，结果如图5-10所示。这里使用`len(s.unique())`得到每个分组去重之后的长

度:

```
ax = df_commit.Author.resample("M", how=lambda  
s:len(s.unique()))).plot(  
kind="area", figsize=(8, 2.5))  
ax.set_ylabel(u"提交人数")
```

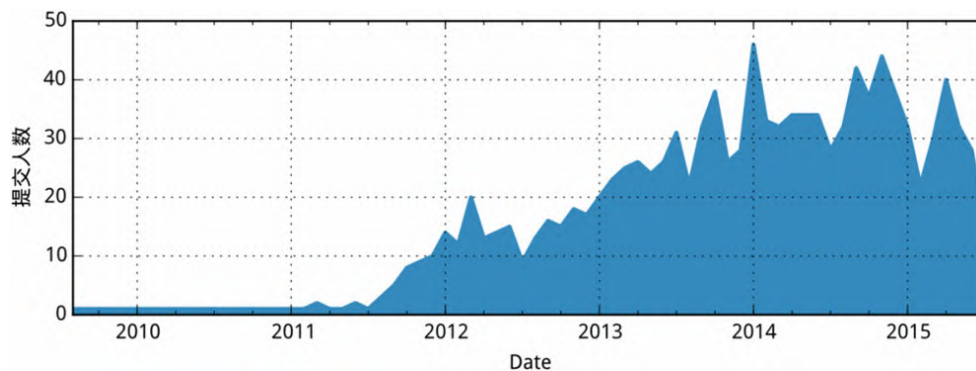


图5-10 每个月的提交人数



请读者思考如何使用groupby()实现与上述resample()相同的运算。

下面使用value_counts()方法统计每位作者的提交数。Pandas的创始人Wes McKinney目前还是排在第一位:

```
s_authors = df_commit.Author.value_counts()  
print s_authors.head()  
  
Wes McKinney      3115  
jreback            3009  
y-p                943  
Chang She          629
```

```
Phillip Cloud      596
dtype: int64
```

下面使用`crosstab()`统计每个月每位作者的提交数，所得到的结果`df_counts`的行索引为月份，列索引为作者。这里通过`DatetimeIndex`的`to_period()`将时间点转换为以月为单位的时间段。结果中包含72个月、485位作者的提交数：

```
df_counts = pd.crosstab(df_commit.index.to_period("M"),
df_commit.Author)
df_counts.index.name = "Month"
print df_counts.shape
(72, 485)
```

下面获取`s_authors`中排前5名的作者对应的列，并调用`plot()`绘图，结果如图5-11所示。默认情况下`plot()`将多列数据绘制在一个子图中。这里将`subplots`参数设置为`True`，让每列数据绘制在不同的子图中。通过将`sharex`和`sharey`参数设置为`True`，让所有子图的X-Y轴的数据范围保持一致，以便比较数据。由图可知创始人已于2013年下半年离开该项目，目前由jreback接手。

```
df_counts[s_authors.head(5).index].plot(kind="area",
subplots=True,
figsize=(8, 6),
color=pl.rcParams['axes.color_cycle'][0],
alpha=0.5,
sharex=True,
sharey=True)
```

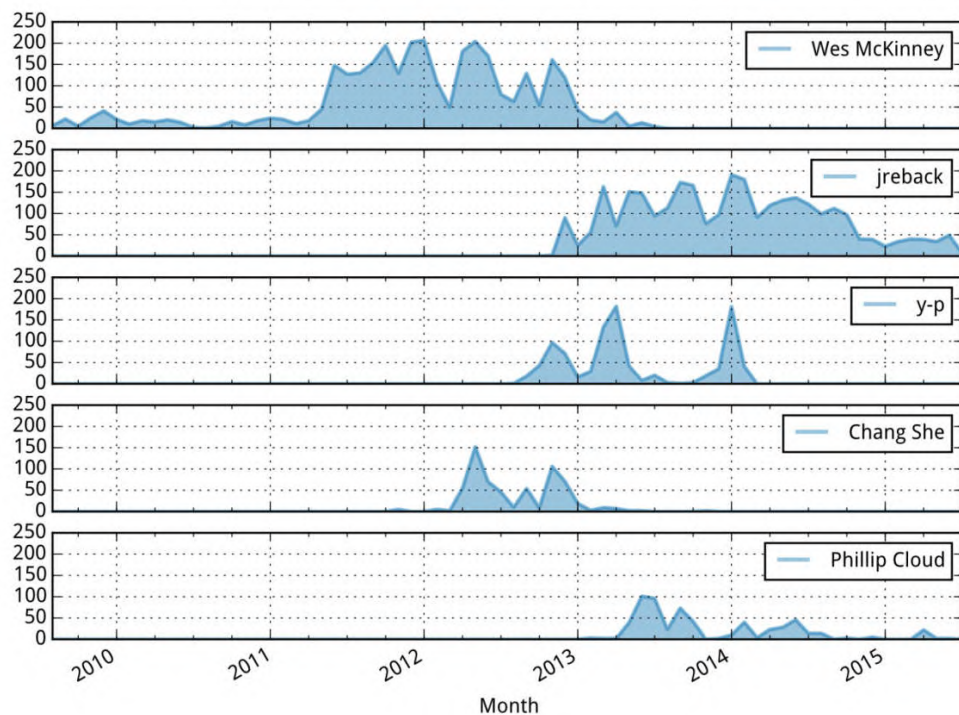



图5-11 前5名作者的每月提交次数

下面我们绘制与GitHub类似的活动记录图，效果如图5-12所示。图中的每个方块表示一天的提交数，每一列表示一个星期。❶首先将索引转换为以天为周期的时间段索引，然后统计次数，就得到了每天的提交数，其索引为以天为周期的时间段。❷将该索引转换多级索引，第0级是以星期为周期的索引，第1级是表示星期几的整数，0表示星期一。

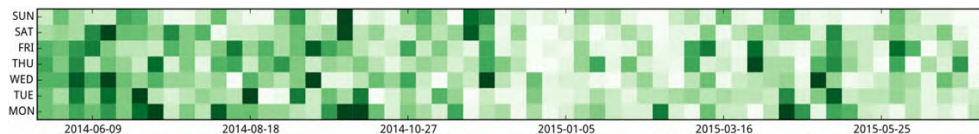


图5-12 Pandas项目的活动记录图

❸使用`unstack()`将第0级索引转换为列索引，得到一个DataFrame对象，获取其最后60周的数据之后，将其中的NaN填充为0。这样就得到了用于做图的`active_data`数据。

注意daily_commit的索引不是按照时间先后顺序排列的，但unstack()返回的结果中行索引和列索引都是按从小到大的顺序排列的：

```
daily_commit = df_commit.index.to_period("D").value_counts() ❶
daily_commit.index =
pd.MultiIndex.from_arrays([daily_commit.index.asfreq("W"), ❷
daily_commit.index.weekday])
daily_commit = daily_commit.sort_index()
active_data = daily_commit.unstack(0).iloc[:, -60:].fillna(0) ❸
```

下面是绘图部分，❹下调用pcolormesh()绘制填充方格，然后调用set_xticks()等函数修改X和Y轴上刻度的位置和文本：

```
fig, ax = pl.subplots(figsize=(15, 4))
ax.set_aspect("equal")
ax.pcolormesh(active_data.values, cmap="Greens",
              vmin=0, vmax=active_data.values.max() * 0.75) ❹

tick_locs = np.arange(3, 60, 10)
ax.set_xticks(tick_locs + 0.5)

ax.set_xticklabels(active_data.columns[tick_locs].to_timestamp(how="start").format())

ax.set_yticks(np.arange(7) + 0.5)

from pandas.tseries.frequencies import DAYS
ax.set_yticklabels(DAYS)
```

也可以使用本书提供的`plot_dataframe_as_colormesh()`来绘制，效果见图5-12：

```
from scipy2.pandas import plot_dataframe_as_colormesh

plot_dataframe_as_colormesh(active_data, xtick_start=3,
xtick_step=10,

                                xtick_format=lambda
period:str(period.start_time.date()),

                                ytick_format=dict(zip(range(7), DAYS)).get,
                                cmap="Greens", vmin=0, vmax=active_data.values.max() * 0.75)
```

5.7.2 分析空气质量数据

在前面介绍输入输出的小节里，我们将data/aqi文件夹下的所有空气质量数据的CSV文件写入了HDF5文件aqi.hdf5。下面我们利用Pandas分析这些数据。首先使用HDFStore打开保存数据的HDF5文件，并调用`select()`来读入所有数据，如表5-5所示：

表5-5 空气质量数据

index	Time	City	Position	AQI	Level	PM2_5	PM10	CO	NO2	O3	SO2
1	2014-04-11 15:00:00	上海	普陀	76	良	49	101	0.0	0	0	0
2	2014-04-11 15:00:00	上海	十五厂	72	良	52	94	0.479	53	124	9
3	2014-04-11 15:00:00	上海	虹口	80	良	59	98	0.612	52	115	11
4	2014-04-11 15:00:00	上海	徐汇上师大	74	良	54	87	0.706	43	113	14
5	2014-04-11 15:00:00	上海	杨浦四漂	84	良	62	99	0.456	43	82	9

```
store = pd.HDFStore("data/aqi/aqi.hdf5")
df_aqi = store.select("aqi")
df_aqi.head()
```

下面用`value_counts()`查看所有的城市名：

```
print df_aqi.City.value_counts()
天津      134471
北京      109999
上海       92745
天津市       13
北京市       12
上海市       10
dtype: int64
```

我们发现每座城市都有两种表示形式，下面使用`str.replace()`将结尾的“市”删除，并将其转换为分类类型：

```
df_aqi["City"] = df_aqi.City.str.replace(" 市 ",
").astype("category")
print df_aqi.City.value_counts()
天津      134484
北京      110011
上海       92755
dtype: int64
```

AQI列为空气质量的评分，而其他的数值列为各种成分的指标值。下面通过`corr()`计算这些值之间的相关性：

```
corr = df_aqi.corr()
print corr
```

	AQI	PM2_5	PM10	CO	NO2	O3	SO2
AQI	1	0.944	0.694	0.611	0.534	-0.136	0.42
PM2_5	0.944	1	0.569	0.633	0.556	-0.169	0.426
PM10	0.694	0.569	1	0.46	0.472	-0.136	0.414
CO	0.611	0.633	0.46	1	0.565	-0.233	0.538
NO2	0.534	0.556	0.472	0.565	1	-0.439	0.448
O3	-0.136	-0.169	-0.136	-0.233	-0.439	1	-0.198
SO2	0.42	0.426	0.414	0.538	0.448	-0.198	1

为了更直观地显示上面的相关矩阵，可以将其绘制成如图5-13所示的图表。由图可知空气质量指数与PM2.5的相关性最大，而与O3略呈负相关性。

```
fig, ax = plt.subplots()
    plot_dataframe_as_colormesh(corr, ax=ax, colorbar=True,
xtick_rot=90)
```

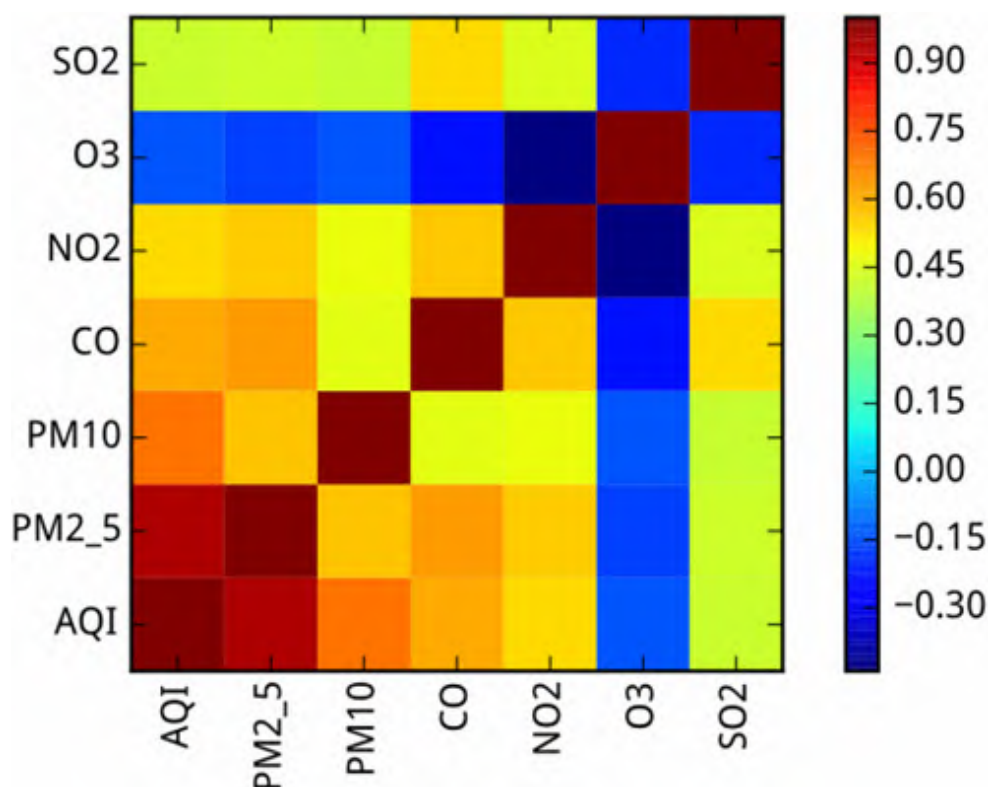


图5-13 空气质量参数之间的相关性

下面比较每座城市每天的PM2.5值的分布情况。❶首先调用groupby()，按照日期和城市分组，❷然后计算每个分组的PM2.5的平均值，结果是一个多级索引的Series对象。索引的第0级为日期，第1级为城市。接着调用unstack()将第1级转换为列，因此mean_pm2_5是一个行索引为日期、列索引为城市的DataFrame对象。❸调用plot(kind="hist")绘制直方统计图，它会对DataFrame对象中的每列数据进行运算，结果如图5-14(左)所示。

```
daily_city_groupby = df_aqi.groupby([df_aqi.Time.dt.date,
"City"]) ❶

mean_pm2_5 = daily_city_groupby.PM2_5.mean().unstack() ❷

mean_pm2_5.plot(kind="hist", histtype="step", bins=20,
normed=True, lw=2) ❸
```

还可以将kind参数设置为“kde”以绘制核密度估计分布，结果如图5-14(右)所示。由于使用了高斯核，因此图中均值小于0的概率密度不为零。

```
ax = mean_pm2_5.plot(kind="kde")

ax.set_xlim(0, 400)
```

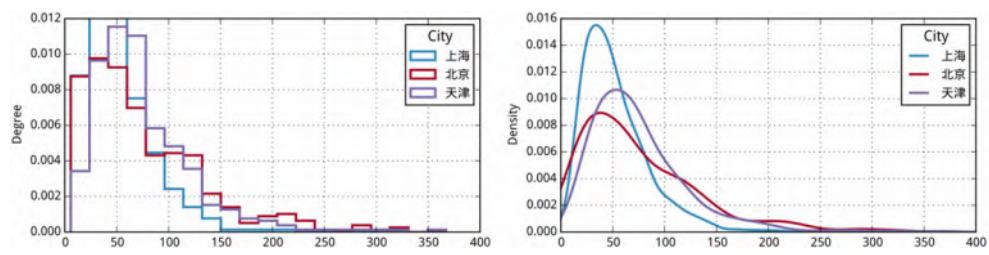


图5-14 每座城市的日平均PM2.5的分布图

下面计算三座城市每天的PM2.5之间的相关性，如表5-6所示。可以看出上海和北京基本无相关性，而天津和北京则有较强的相关性。

```
mean_pm2_5.corr()
```

表5-6 三座城市PM2.5之间的相关性

--	--	--	--

index	上海	北京	天津
上海	1.0	-0.146	0.0718
北京	-0.146	1.0	0.691
天津	0.0718	0.691	1.0

下面统计一个星期中每天的PM2.5的平均值，通过dt.dayofweek可以获得表示星期几的整数序列，其中0表示星期一。结果如图5-15所示，可以看出北京市星期四、星期五、星期六的空气质量要比其他时间的差一些。

```

week_mean = df_aqi.groupby([df_aqi.Time.dt.dayofweek,
"City"]).PM2_5.mean()

ax = week_mean.unstack().plot(kind="Bar")
ax.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
          ncol=3, mode="expand", borderaxespad=0.)

from pandas.tseries.frequencies import DAYS
ax.set_xticklabels(DAYS)

```

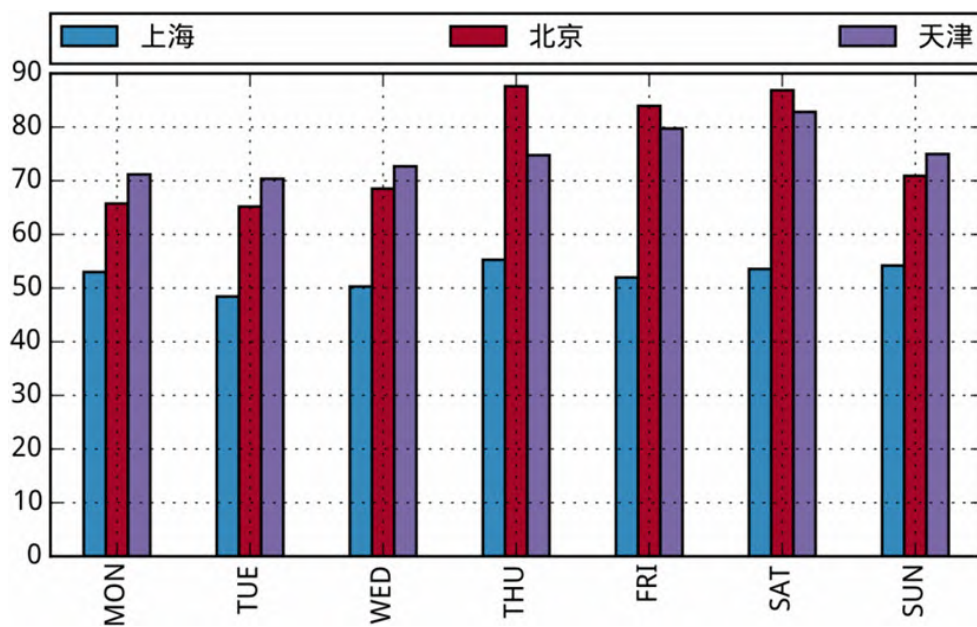


图5-15 一星期中PM2.5的平均值

下面统计一天中每个小时的PM2.5的平均值，结果如图5-16所示，可以看出北京市白天的空气质量比夜晚要好一些：

```
hour_mean = df_aqi.groupby([df_aqi.Time.dt.hour,
"City"]).PM2_5.mean()

ax = hour_mean.unstack().plot(kind="Bar", figsize=(10, 3))

ax.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
          ncol=3, borderaxespad=0.)
```

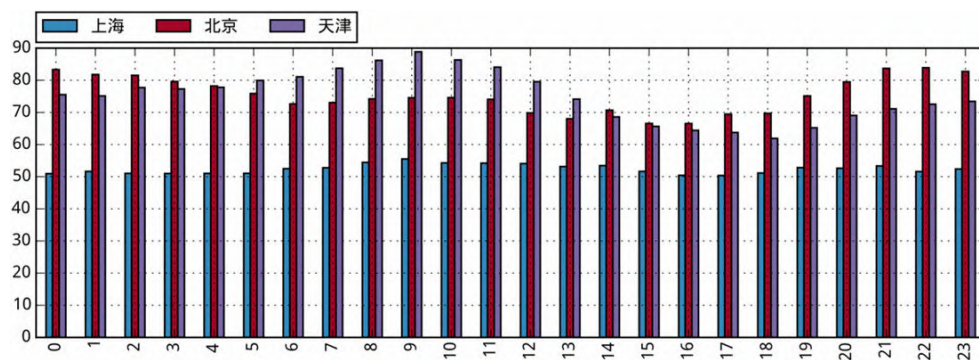


图5-16 一天中不同时段PM2.5的平均值

下面计算北京市每个观测点的每个月的PM2.5的平均值，得到一个多级索引的month_place_mean序列，其第0级为月份、第1级为观测点。然后调用其mean(level=1)方法，计算每个观测点的PM2.5的平均值，结果如图5-17所示。

```
df_bj = df_aqi.query("City=='北京'")

month_place_mean = df_bj.groupby([df_bj.Time.dt.to_period("M"),
"Position"]).PM2_5.mean()

place_mean = month_place_mean.mean(level=1).order()

place_mean.plot(kind="bar")
```

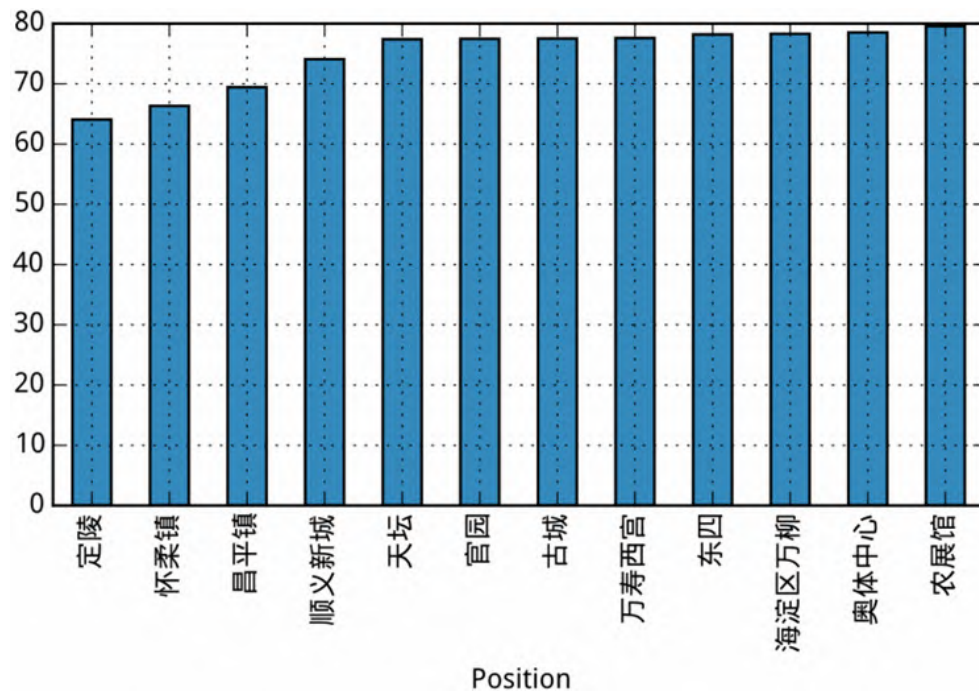



图5-17 北京市各个观测点的PM2.5的平均值

下面分别选出空气质量最好和最差的两个观测点，并使用柱状图显示它们每个月的PM2.5平均值：

```
places = place_mean.iloc[[0, 1, -2, -1]].index
month_place_mean.unstack().loc[:, places].plot(kind="bar",
figsize=(10, 3), width=0.8)
```

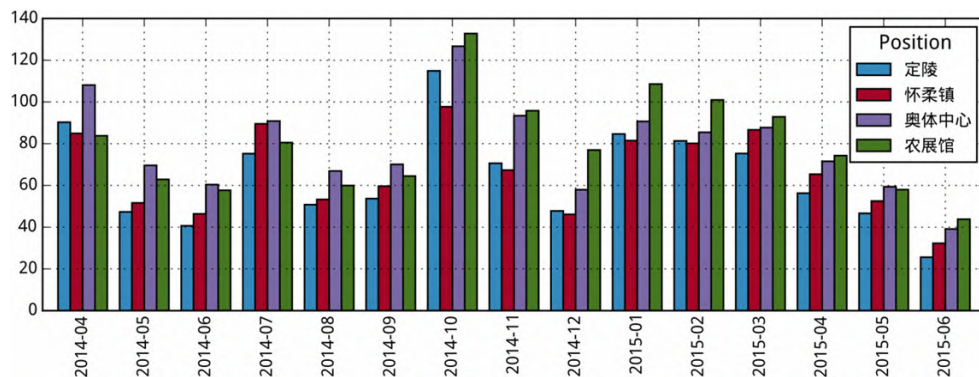


图5-18 北京市各观测点的月平均PM2.5值

第6章 SymPy-符号运算好帮手

SymPy是Python的数学符号计算库，用它可以进行数学表达式的符号推导和演算。随着SymPy 0.7.3的发布，它已经逐渐发展成熟。虽然与一些专业符号运算软件相比，SymPy的功能以及运算速度都还较弱，但由于它完全采用Python编写，因而能很好地与其他的Python科学计算库结合使用。例如本章最后一节将介绍如何使用SymPy得到一个简单的单摆系统的微分方程组，并将其自动转换成数值运算程序，通过SciPy中的`odeint()`求解该微分方程组，最后使用`matplotlib`完成动画模拟。

6.1 从例子开始



与本节内容对应的Notebook为：`06-sympy/sympy-100-intro.ipynb`。

在详细介绍SymPy的语法结构和各种运算功能之前，本节先通过几个实例说明用SymPy解决符号运算问题的一般步骤。

6.1.1 封面上的经典公式

下面是本书封面左上角的数学公式：

$$e^{i\pi} + 1 = 0$$

该公式被称为欧拉恒等式，其中 e 是自然常数， i 是虚数单位， π 是圆周率。它被誉为数学中最奇妙的公式，它将5个基本数学常数用加法、乘法

和幂运算联系起来。下面用SymPy验证这个公式。

首先从sympy库载入所有名字，其中E表示自然常数，I表示虚数单位，pi表示圆周率，因此可以用它们直接求欧拉公式的值：

```
from sympy import *  
  
E**(I*pi) + 1  
  
0
```

SymPy还可以帮助我们做数学公式的推导和证明。欧拉恒等式可以将 π 代入下面的欧拉公式来得到：

$$e^{ix} = \cos x + i \sin x$$

在SymPy中可以使用expand()将表达式展开，下面用它展开 e^{ix} 试试看：

```
x = symbols("x")  
  
expand( E**(I*x) )
```

e^{ix}

很遗憾没有成功，若将expand()的参数complex设置为True，则表达式将被分为实数和虚数两个部分：

```
expand(exp(I*x), complex=True)
```

$$ie^{-\Im x} \sin(\Re x) + e^{-\Im x} \cos(\Re x)$$

这次表达式展开了，但是得到的结果相当复杂。其中 $\Re x$ 是取实数值的函数， $\Im x$ 是取虚数值的函数。之所以会出现这两个函数，是因为expand()将x当作复数处理。为了指定x为实数，需要如下重新定义x：

```
x = Symbol("x", real=True)
expand(exp(I*x), complex=True)
```

$$\text{isin}(x) + \cos(x)$$

终于得到了欧拉公式，那么如何证明它呢？可以用泰勒多项式对其进行展开：

```
tmp = series(exp(I*x), x, 0, 10)
tmp
```

$$1 + ix - \frac{x^2}{2} - \frac{ix^3}{6} + \frac{x^4}{24} + \frac{ix^5}{120} - \frac{x^6}{720} - \frac{ix^7}{5040} + \frac{x^8}{40320} + \frac{ix^9}{362880} + \mathcal{O}(x^{10})$$

展开之后的虚数项和实数项交替出现。根据欧拉公式，虚数项之和应该等于 $\sin(x)$ 的泰勒级数，而实数项之和应该等于 $\cos(x)$ 的泰勒展开。下面获得tmp的实部：

```
re(tmp)
```

$$\frac{x^8}{40320} - \frac{x^6}{720} + \frac{x^4}{24} - \frac{x^2}{2} + \Re(\mathcal{O}(x^{10})) + 1$$

下面对 $\cos(x)$ 进行泰勒展开，可以看到其各项与上面的结果一致：

```
series(cos(x), x, 0, 10)
```

$$1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720} + \frac{x^8}{40320} + \mathcal{O}(x^{10})$$

下面获得tmp的虚部：

```
im(tmp)
```

$$\frac{x^9}{362880} - \frac{x^7}{5040} + \frac{x^5}{120} - \frac{x^3}{6} + x + \Im(\mathcal{O}(x^{10}))$$

下面对 $\sin(x)$ 进行泰勒展开，其各项也与上面的结果一致：

```
series(sin(x), x, 0, 10)
```

$$x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + \frac{x^9}{362880} + \mathcal{O}(x^{10})$$

由于 e^{ix} 的展开公式的实部和虚部分别等于 $\cos x$ 和 $\sin x$ ，因此验证了欧拉公式的正确性。

6.1.2 球体体积

在SciPy一章中介绍了如何使用数值定积分计算球体的体积，而SymPy中的`integrate()`则可以计算符号积分。例如下面的语句用`integrate()`做不定积分运算：

```
integrate(x*sin(x), x)
```

$$-x\cos(x) + \sin(x)$$

如果指定变量 x 的取值范围，`integrate()`则做定积分运算：

```
integrate(x*sin(x), (x, 0, 2*pi))
```

$$-2\pi$$

为了计算球体体积，首先看看如何计算圆形面积。假设圆形的半径为 r ，则圆上任意一点的 Y 坐标函数为：

$$y(x) = \sqrt{r^2 - x^2}$$

可以直接对函数 $y(x)$ 在 $-r$ 到 r 区间上求定积分得到半圆面积。下面的程序计算该定积分：首先需要定义运算中所需的符号，用`symbols()`可以一次

创建多个符号。定义半径r时需要设置positive参数为True，表示圆的半径为正数：

```
x, y = symbols('x, y')
r = symbols('r', positive=True)
circle_area = 2 * integrate(sqrt(r**2 - x**2), (x, -r, r))
circle_area
```

$$\pi r^2$$

接下来对此面积公式求定积分，就可以得到球体的体积，但是随着X轴坐标的变化，对应的切面半径会发生变化。假设X轴的坐标为x，球体的半径为r，则x处的切面的半径可以使用前面的公式y(x)计算出。因此需要对circle_area中的变量r进行替代：

```
circle_area = circle_area.subs(r, sqrt(r**2 - x**2))
circle_area
```

$$\pi(r^2 - x^2)$$

然后对circle_area中的变量x在区间-r到r上进行定积分，得到球体的体积公式：

```
integrate(circle_area, (x, -r, r))
```

$$\frac{4\pi}{3}r^3$$

subs()可以替换表达式中的符号，它有如下3种调用方式：

- expression.subs(x, y)：将算式中的x替换成y。
- expression.subs({x:y, u:v})：使用字典进行多次替换。
- expression.subs([(x, y), (u, v)])：使用列表进行多次替换。

请注意多次替换是顺序执行的，因此`expression.sub([(x, y), (y, x)])`并不能对符号`x`和`y`进行交换。

6.1.3 数值微分

所谓数值微分，是指根据函数在一些离散点的函数值，推算它在某点的导数或高阶导数的近似值的方法。例如当`h`足够接近于零时，可以使用下面的公式计算`f(x)`在`x`处的导数`f'(x)`：

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

上面的公式使用两个函数值计算导数值，被称为两点公式，使用的点数越多数值微分的精度也就越高。可以使用SymPy提供的`as_finite_diff()`自动计算N点公式。下面先使用`symbols()`定义三个符号对象，其中定义`f`时设置`cls`参数为`Function`表示它是表示数学函数的符号。

```
x = symbols('x', real=True)
h = symbols('h', positive=True)
f = symbols('f', cls=Function)
```

`f`是表示函数的符号，而`f(x)`则是自变量为`x`的函数，下面调用其`diff()`方法，对`x`求一阶导数：

```
f_diff = f(x).diff(x, 1)
f_diff
```

$$\frac{d}{dx}f(x)$$

然后调用`as_finite_diff()`，将一阶导数转换为使用`f(x)`、`f(x-h)`、`f(x-2*h)`、`f(x-3*h)`表达的4点公式：


```
expr_diff = as_finite_diff(f_diff, [x, x-h, x-2*h, x-3*h])
expr_diff
```

$$\frac{11}{6h}f(x) - \frac{1}{3h}f(-3h + x) + \frac{3}{2h}f(-2h + x) - \frac{3}{h}f(-h + x)$$

下面以 $f(x) = x \cdot e^{-x^2}$ 为例比较数值求导与符号求导的误差。首先使用`subs()`方法将`expr_diff`中的 $f(x)$ 替换为目标函数，并调用其`doit()`方法计算导函数：

```
sym_dexpr = f_diff.subs(f(x), x*exp(-x**2)).doit()
sym_dexpr
```

$$-2x^2e^{-x^2} + e^{-x^2}$$

然后调用`lambdify()`，将上面的`sym_dexpr`表达式转换为数值运算的函数。其第一个参数为自变量列表，第二个参数为运算表达式，这里将`modules`参数设置为“numpy”，因此`sym_dfunc()`可以对数组进行运算：

```
sym_dfunc = lambdify([x], sym_dexpr, modules="numpy")
sym_dfunc(np.array([-1, 0, 1]))
array([-0.36787944,  1.          , -0.36787944])
```

由于`expr_diff`是一个加法表达式，因此通过其`args`属性可以获得所有的加法项：

```
print expr_diff.args
(-3*f(-h + x)/h, -f(-3*h + x)/(3*h), 3*f(-2*h + x)/(2*h),
11*f(x)/(6*h))
```

上面的加法项没有按照自变量从小到大的顺序排列。下面使用通配符`w`和`c`组成的模板`c* f(w)`对每个加法项进行匹配，提取出每项的系数和函数

参数:

```
w = Wild("w")
c = Wild("c")
patterns = [arg.match(c * f(w)) for arg in expr_diff.args]
```

每个匹配结果是一个以通配符为键的字典，例如下面是第一项的匹配结果，它表示该项的系数为 $-3/h$ ，函数 f 的参数为 $-h + x$ 。

```
print patterns[0]
{w_: -h + x, c_: -3/h}
```

下面使用通配符 w 的匹配结果计算出排序用的键值，并从排序之后的列表中选择出每个匹配结果中与通配符 c 对应的表达式：

```
coefficients = [t[c] for t in sorted(patterns, key=lambda
t:t[w])]
print coefficients
[-1/(3*h), 3/(2*h), -3/h, 11/(6*h)]
```

下面将系数表达式列表中的 h 替换为 0.001 ，得到系数数组。注意SymPy的浮点数运算得到的是SymPy中的Float对象，还需要调用`float()`来将其转换为Python的float对象：

```
coeff_arr = np.array([float(coeff.subs(h, 1e-3)) for coeff in
coefficients])
print coeff_arr
[ -333.33333333  1500.          -3000.          1833.33333333]
```

接下来使用NumPy计算数值微分的值，并与`sym_dfunc()`的运算结果进行比较，输出最大绝对误差：

```

def moving_window(x, size):
    from numpy.lib.stride_tricks import as_strided
    x = np.ascontiguousarray(x)
    return as_strided(x, shape=(x.shape[0] - size + 1, size),
                      strides=(x.itemsize, x.itemsize))

x_arr = np.arange(-2, 2, 1e-3)
y_arr = x_arr * np.exp(-x_arr * x_arr)
num_res = (moving_window(y_arr, 4) * coeff_arr).sum(axis=1)
sym_res = sym_dfunc(x_arr[3:])
print np.max(abs(num_res - sym_res))

4.08944167418e-09

```

为了比较点数与误差之间的关系，在下面的 `finite_diff_coefficients()` 函数中计算间隔为 h 、点数为 $order$ 的系数，并绘制2、3、4点公式对应的误差曲线，结果如图6-1所示，注意Y轴为对数轴。

```

def finite_diff_coefficients(f_diff, order, h):
    v = f_diff.variables[0]
    points = [x - i * h for i in range(order)]
    expr_diff = as_finite_diff(f_diff, points)
    w = Wild("w")
    c = Wild("c")
    patterns = [arg.match(c*f(w)) for arg in expr_diff.args]
    coefficients = np.array([float(t[c])
                             for t in sorted(patterns,
key=lambda t:t[w])])
    return coefficients

```

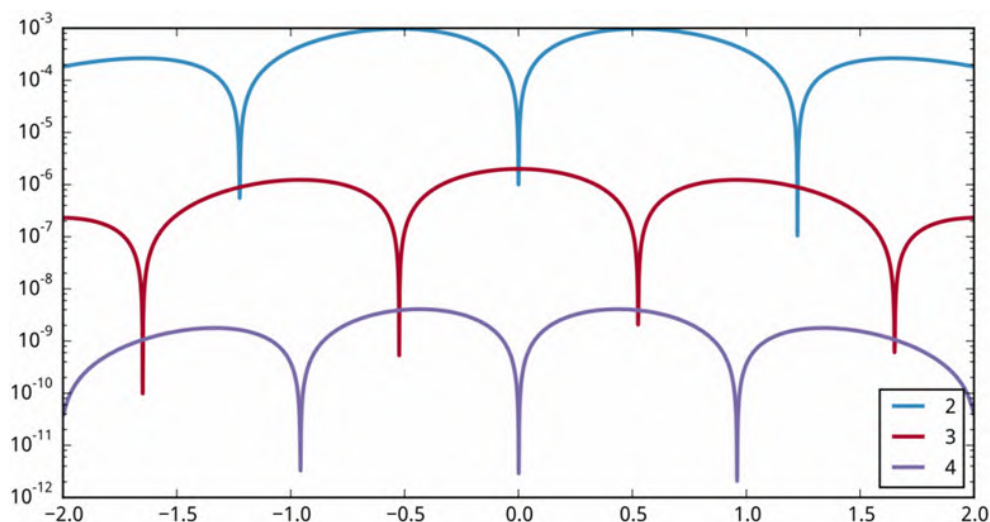


图6-1 比较不同点数的数值微分的误差

6.2 数学表达式



与本节内容对应的 Notebook 为：`06-sympy/sympy-200-expression.ipynb`。

本节详细介绍数学表达式的结构，虽然这部分内容比较枯燥，但只有了解表达式的结构，才能随心所欲地对其进行处理，将SymPy运用到更复杂的计算中。

6.2.1 符号

数学符号用Symbol对象表示，符号对象的name属性是符号名，符号名在显示由此符号构成的表达式时使用。Symbol对象的符号名和Python的变

量名没有内在联系，但是为了使用起来方便，通常让变量名与符号名相同。为了快速创建符号以及与之同名的变量，可以使用`var()`，例如：

```
print var("x0,y0,x1,y1")  
  
(x0, y0, x1, y1)
```

上面的语句创建了名为`x0`、`y0`、`x1`、`y1`的4个`Symbol`对象，同时当前名称空间中创建了4个名为`x0`、`y0`、`x1`、`y1`的变量来分别表示这些`Symbol`对象。因为符号对象在转换为字符串时直接使用`name`属性，因此在交互式环境中看到变量`x0`显示的值就是`x0`：

<code>type(x0)</code>	<code>x0.name</code>	<code>type(x0.name)</code>
-----	-----	-----
<code>sympy.core.symbol.Symbol</code>	<code>'x0'</code>	<code>str</code>

在交互环境中使用`var()`能快速创建变量和`Symbol`对象，但在程序中使用容易引起混淆，这时可以使用`symbols()`创建`Symbol`对象，再将它们显式地赋值给变量：

```
x1, y1 = symbols("x1, y1")  
  
type(x1)  
  
sympy.core.symbol.Symbol
```

当然，如果不嫌麻烦也可以直接使用`Symbol`类创建对象：

```
x2 = Symbol("x2")
```

变量名和符号名当然也可以是不一样的，下面使用变量`t`表示符号`x0`，然后创建两个名为`alpha`和`beta`的符号，并用变量`a`和`b`分别表示。当符号使用希腊字母名时，可以显示为希腊字母。

```
t = x0
a, b = symbols("alpha, beta")
sin(a) + sin(b) + t
```

$$x_0 + \sin(\alpha) + \sin(\beta)$$

数学公式中的符号一般都有特定的假设，例如 m 、 n 通常是整数，而 z 经常用来表示复数。在用`var()`、`symbols()`或`Symbol()`创建`Symbol`对象时，可以通过关键字参数指定所创建符号的假设条件，这些假设条件会影响到它们所参与的计算。例如，下面创建了两个整数符号 m 和 n 以及一个正数符号 x ：

```
m, n = symbols("m, n", integer=True)
x = Symbol("x", positive=True)
```

每个符号都有许多`is_*`属性，用以判断符号的各种假设条件，在IPython中使用自动完成功能可以快速查看这些假设的名称。注意下划线后为大写字母的属性用来判断对象的类型，而全小写字母的属性则用来判断符号的假设条件。

```
[attr for attr in dir(x) if attr.startswith("is_") and
attr.lower() == attr]
['is_algebraic',
 'is_algebraic_expr',
 'is_antihermitian',
 'is_bounded',
 ...]
```

在下面的判断中， x 是一个符号对象，它是一个正数，因为它可以比较大小，所以它不是虚数。 x 是一个复数，因为复数包括实数，而实数包括正数。

x.is_Symbol	x.is_positive	x.is_imaginary	x.is_complex
True	True	False	True

使用 `assumptions0` 属性可以快速查看所有的假设条件，其中 `commutative` 为 `True` 表示该符号满足交换律，其余的假设条件根据英文名很容易知道它们的含义，这里就不再详细叙述了。

x.assumptions0		
{'commutative': True,	'complex': True,	'hermitian': True,
'imaginary': False,	'negative': False,	'nonnegative': True,
'nonpositive': False,	'nonzero': True,	'positive': True,
'real': True,	'zero': False}	

在 SymPy 中所有的对象都从 `Basic` 类继承，实际上这些 `is_*` 属性和 `assumptions0` 属性都是在 `Basic` 类中定义的：

```
Symbol.mro()
[sympy.core.symbol.Symbol,
 sympy.core.expr.AtomicExpr,
 sympy.core.basic.Atom,
 sympy.core.expr.Expr,
 sympy.logic.boolalg.Boolean,
 sympy.core.basic.Basic,
 sympy.core.evalf.EvalfMixin,
 object]
```

6.2.2 数值

为了实现符号运算，在 SymPy 内部有一整套数值运算系统，因此 SymPy 的数值和 Python 的整数、浮点数是完全不同的对象。为了使用方便，SymPy

会尽量自动将Python的数值类型转换为SymPy的数值类型。SymPy提供了一个S对象以方便用户快速将Python的数值转换成SymPy的数值。在下面的例子中，当有SymPy的数值参与计算时，结果也为SymPy的数值对象：

```
1/2 + 1/3  S(1)/2 + 1/S(3)
-----
0          5/6
```

$\frac{5}{6}$ 是Rational对象，它由两个整数的商表示，数学上称之为有理数。也可以直接通过Rational创建：

```
type(S(5)/6)
sympy.core.numbers.Rational
```

```
Rational(5, 10) # 有理数会自动进行约分处理
```

$\frac{1}{2}$

实数用Float对象表示，它和标准的浮点数类似，但是它的精度(有效数字)可以通过参数指定。由于在浮点数或Float对象内部都使用二进制方式表示数值，因此它们都可能无法精确表示十进制中的精确小数。可以使用N()查看浮点数的实际数值，例如下面的语句查看浮点数0.1和10000.1的60位有效数字，可以看到数值的绝对精度随着数值的增大而减小：

```
print N(0.1, 60)
print N(10000.1, 60)

0.1000000000000000005551115123125782702118158340454101562500000
10000.10000000000000363797880709171295166015625000000000000000000
```


[illegible]

```
print N(pi, 50)

print N(sqrt(2), 50)

3.1415926535897932384626433832795028841971693993751

1.4142135623730950488016887242096980785696718753769
```

SymPy重新定义了所有的数学运算符和数学函数。例如Add类表示加法，Mul类表示乘法，而Pow类表示指数运算，sin类表示正弦函数。和Symbol对象一样，这些运算符和函数都从Basic类继承，请读者自行在IPython中查看它们的继承列表，例如Add.mro()。可以使用这些类创建复杂的表达式：

```
var("x, y, z")
Add(x, y, z)
```

$$x + y + z$$

```
Add(Mul(x, y, z), Pow(x, y), sin(z))
```

$$xyz + x^y + \sin(z)$$

由于在Basic类中重新定义了__add__()等操作符相关的魔法方法，因此可以使用和Python表达式相同的方式创建SymPy的表达式：

```
x*y*z + x**y + sin(z)
```

$$xyz + x^y + \sin(z)$$

在Basic类中定义了两个很重要的属性：func和args。func属性得到对象的类，而args得到其参数。使用这两个属性可以观察SymPy所创建的表达式。也许读者会对没有减法运算类感到奇怪，下面让我们看看减法运算所得到的表达式：

t = x - y			
t.func	t.args	t.args[0].func	t.args[0].args
-----	-----	-----	-----
sympy.core.add.Add	(-y, x)	sympy.core.mul.Mul	(-1, y)

通过上面的例子可以看出，表达式x-y在SymPy中实际上是用Add(Mul(-1, y), x)表示的。同样，SymPy中没有除法类，请读者使用和上面相同的方法观察x / y在SymPy中是如何表示的。

SymPy的表达式实际上是一个由Basic类的各种对象多层嵌套结构的树状结构。使用dotprint()可以将表达式转换成Graphviz的DOT语言描述的图形，使用本书提供的%dot命令可以将其显示在Notebook中。图6-2显示了表达式 $xy \frac{\sqrt{x^2-y^2}}{x+y}$ 的结构：

```

from sympy.printing.dot import dotprint

graph = dotprint(x * y * sqrt(x ** 2 - y ** 2) / (x + y))

%dot -f svg graph

```

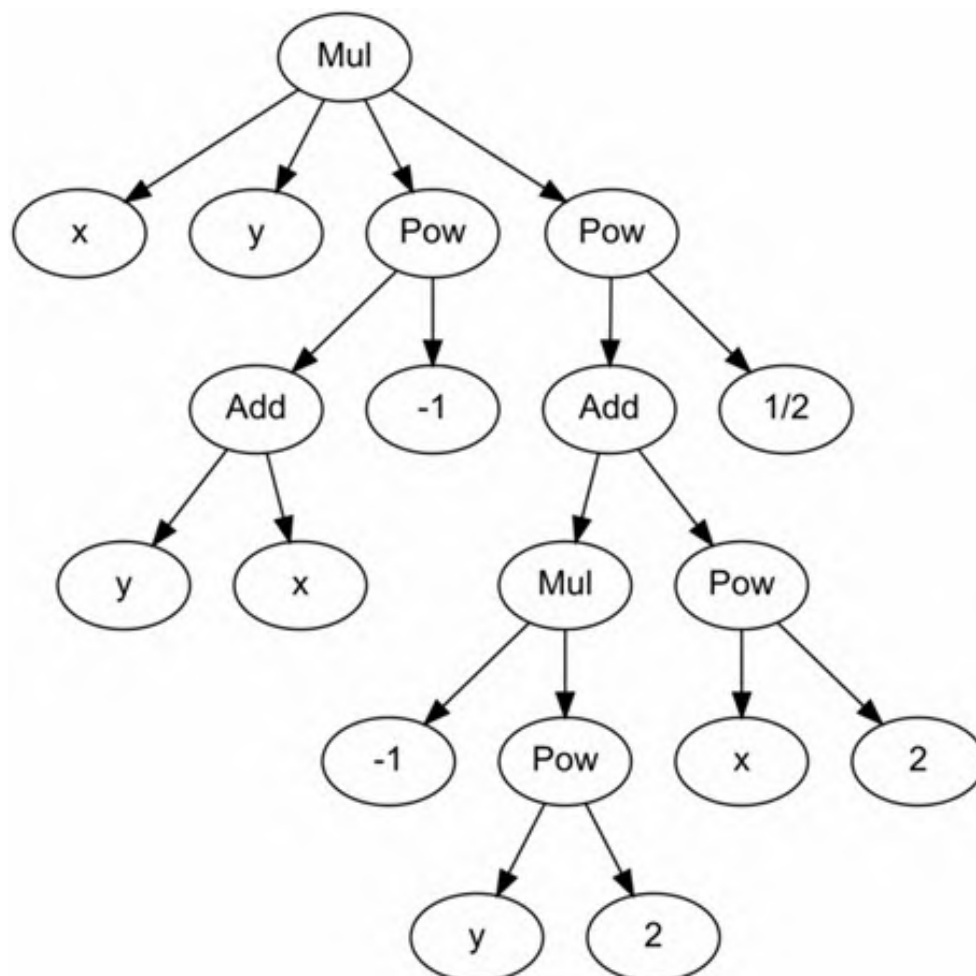


图6-2 表达式的树状结构

由于Basic对象的args属性类型是元组，因此表达式一旦创建就不能再改变。使用不可变的结构表示表达式有很多优点，例如可以用表达式作为字典的键。

除了使用SymPy中预先定义好的具有特殊运算含义的数学函数之外，还可以使用Function()创建自定义的数学函数：

```
f = Function("f")
```

注意Function虽然是一个类，但是上面的语句所得到的f并不是Function类的实例。和预定义的数学函数一样，f是一个类，它从Function类继承：

```
issubclass(f, Function)
True
```

当用f创建一个表达式时，就相当于创建它的一个实例：

```
t = f(x, y)
type(t)   t.func   t.args
-----
f         f        (x, y)
```

f的实例t可以参与表达式运算：

```
t + t * t
```

$f^2(x, y) + f(x, y)$

6.2.4 通配符

使用通配符可以创建匹配特定表达式的模板。例如在下面的例子中，a和b是Wild通配符对象，a*b**2是由这两个通配符创建的表达式模板。调用表达式对象的match()方法以使用指定的模板匹配表达式。它返回一个键为通配符、值为与该通配符匹配的子表达式的字典。



执行SymPy提供的`init_printing()`可以使用数学符号显示运算结果，但会将Python的内置对象也转换成LaTeX显示。为了编写方便，本书使用一般文本显示内置对象，而用本书提供的`%sympy_latex`魔法方法将内置对象转换为LaTeX。

```
x, y = symbols("x, y")
a = Wild("a")
b = Wild("b")
%sympy_latex (3 * x * (x + y)**2).match(a * b**2)
```

{a: 3x, b: x + y}

而`find()`方法则在表达式的树状结构中搜索所有与模板匹配的子树。下面在 $x^3 + 3x^2y + 3xy^2 + y^3$ 中搜索与模板匹配的所有子表达式：

```
expr = expand((x + y)**3)
%sympy_latex expr
%sympy_latex expr.find(a * b**2)
```

$x^3 + 3x^2y + 3xy^2 + y^3$

{2,3,x,x²,x³,y,y²,y³,3xy²,3x²y,x³ + 3x²y + 3xy² + y³}

整数2居然也与模板匹配，这有些出乎意外，我们使用下面的`find_match()`输出所有子表达式的与模板匹配的结果，如表6-1所示：

```
def find_match(expr, pattern):
    return [e.match(pattern) for e in expr.find(pattern)]

find_match(expr, a * b**2)
```

表6-1 与表达式的匹配结果一

表达式	匹配结果
$3xy^2$	$\{a: 3x, b: y\}$
2	$\{a: 1, b: \sqrt{2}\}$
3	$\{a: 1, b: \sqrt{3}\}$
y	$\{a: 1, b: \sqrt{y}\}$
(续表)	
表达式	匹配结果
x	$\{a: 1, b: \sqrt{x}\}$
$x^3 + 3x^2y + 3xy^2 + y^3$	$\{a: 1, b: \sqrt{x^3 + 3x^2y + 3xy^2 + y^3}\}$
x^3	$\{a: 1, b: x^{\frac{3}{2}}\}$
$3x^2y$	$\{a: 3y, b: x\}$
y^2	$\{a: 1, b: y\}$
y^3	$\{a: 1, b: y^{\frac{3}{2}}\}$
x^2	$\{a: 1, b: x\}$

由上面的结果可知，在模板与表达式匹配的过程中SymPy会对表达式进行变换，找到数学意义上正确匹配的结果。为了剔除掉这些由表达式变换而得到的结果，可以在定义通配符时使用exclude参数指定不能匹配的对象列表。当与通配符匹配的表达式中包含该参数中的任何对象时，匹配都会失败。下面的通配符a和b都不能包含整数1，而b不能包含指数运算，注意在SymPy中开平方使用指数表达式表示，见表6-2。

```
a = Wild("a", exclude=[1])
b = Wild("b", exclude=[1, Pow])
find_match(expr, a * b**2)
```

表6-2 与表达式的匹配结果二

表达式	匹配结果
$3xy^2$	$\{a: 3x, b: y\}$
$3x^2y$	$\{a: 3y, b: x\}$
y^3	$\{a: y, b: y\}$
x^3	$\{a: x, b: x\}$

可以使用`replace()`方法对表达式中的子表达式进行替换，例如下面将所有与`a*b**2`匹配的子表达式替换为`(a+b)**2`：

```
expr.replace(a * b**2, (a + b)**2)
```

$$4x^2 + 4y^2 + (x + 3y)^2 + (3x + y)^2$$

使用`WildFunction`可以定义与任意函数匹配的通配符。在下面的例子中，`f`与`exp`、`sin`和`abs`这三个函数匹配，匹配结果包含函数的参数。而指数运算则被当作运算符，`sqrt`实际上使用指数运算表示，它们都不与`f`匹配，见表6-3：

```
expr = sqrt(x) / sin(y**2) + abs(exp(x) * x)
find_match(expr, f)
```

表6-3 与表达式的匹配结果三

表达式	匹配结果
e^x	{WildFunction(f): e^x }
$\sin(y^2)$	{WildFunction(f): $\sin(y^2)$ }
$ xe^x $	{WildFunction(f): $ xe^x $ }

6.3 符号运算



与本节内容对应的 Notebook 为：06-sympy/sympy-300-calculations.ipynb

SymPy所提供的符号运算功能十分丰富，由于篇幅受限，本节只能简单介绍SymPy的一些常用的符号运算功能。

6.3.1 表达式变换和化简

`simplify()` 可以对数学表达式进行化简，例如：

```
simplify((x + 2) ** 2 - (x + 1) ** 2)
```

$2x+3$

`simplify()` 调用SymPy内部的多种表达式变换函数来对其化简。但是数学表达式的化简是一件非常复杂的工作，对于同一个表达式，根据其使用目的可以有多种化简方案。本节介绍SymPy提供的各种表达式变换函数，充分利用这些函数可以实现表达式的变换和化简。

`radsimp()` 对表达式的分母进行有理化，结果中的分母部分不含无理数。例如：

```
radsimp(1 / (sqrt(5) + 2 * sqrt(2)))
```

$\frac{1}{3}(-\sqrt{5} + 2\sqrt{2})$

也可以对带符号的表达式进行处理：

```
radsimp(1 / (y * sqrt(x) + x * sqrt(y)))
```

$\frac{-\sqrt{xy} + x\sqrt{y}}{xy(x-y)}$

`ratsimp()` 对表达式中的分母进行通分运算，即将表达式转换为分子除分母的形式：

```
ratsimp(x / (x + y) + y / (x - y))
```

$\frac{2y^2}{x^2 - y^2} + 1$

`fraction()` 返回包含表达式的分子与分母的元组，用它可以获得 `ratsimp()` 通分之后的分子或分母：

```
%sympy_latex fraction(ratsimp(1 / x + 1 / y))
```

$(x+y, xy)$

请注意 `fraction()` 不会自动对表达式进行通分运算，因此：

```
%sympy_latex fraction(1 / x + 1 / y)
```

$(\frac{1}{y} + \frac{1}{x}, 1)$

`cancel()` 对分式表达式的分子分母进行约分运算，去除它们的公因子：

```
cancel((x ** 2 - 1) / (1 + x))
```

$x-1$

`apart()` 对表达式进行部分分式分解，它将一个有理函数变为数个分子及分母次数较小的有理函数。下面用它将传递函数 $\frac{1}{s^3+s^2+s+1}$ 分解为两个次数较小的传递函数之和：

```
s = symbols("s")
trans_func = 1/(s**3 + s**2 + s + 1)
apart(trans_func)
```

$-\frac{s-1}{2s^2+2} + \frac{1}{2s+2}$

`trigsimp()` 化简表达式中的三角函数，通过 `method` 参数可以选择化简算法：

```
trigsimp(sin(x) ** 2 + 2 * sin(x) * cos(x) + cos(x) ** 2)
```

$\sin(2x)+1$

expand_trig() 展开三角函数表达式：

```
expand_trig(sin(2 * x + y))
```

$(2\cos^2(x) - 1)\sin(y) + 2\sin(x)\cos(x)\cos(y)$

expand() 根据用户设置的标志参数对表达式进行展开。默认情况下，表6-4中的标志参数为True：

表6-4 标志参数一

标志	表达式	结果	说明
mul	expand(x*(y+z))	xy + xz	展开乘法
log	expand(log(x*y**2))	log(xy ²)	展开对数函数的参数中的乘积和幂运算
multinomial	expand((x+y)**3)	x ³ + 3x ² y + 3xy ² + y ³	展开加减法表达式的整数次幂
power_base	expand((x*y)**z)	(xy) ^z	展开幂函数的底数乘积
power_exp	expand(x**(y+z))	x ^y x ^z	展开对幂函数的指数和

可以将默认为True的标志参数设置为False，不展开与之对应的表达式。在下面的例子中，将mul设置为False，因此不展开乘法：

```
x, y, z = symbols("x,y,z", positive=True)
expand(x * log(y * z), mul=False)
```

$x(\log(y) + \log(z))$

expand() 的表6-5中的标志参数默认为False：

表6-5 标志参数二

标志	表达式	结果	说明
complex	<code>expand(x*y)</code>	xy	展开复数
func	<code>expand(gamma(x + 1))</code>	$\Gamma(x + 1)$	对一些特殊函数进行展开
trig	<code>expand(sin(x + y))</code>	$\sin(x + y)$	展开三角函数

- complex: 展开复数的实部和虚部。

```
x, y = symbols("x,y", complex=True)
expand(x * y, complex=True)
```

$$\Re x \Re y + i \Re x \Im y + i \Re y \Im x - \Im x \Im y$$

- func: 对一些特殊函数进行展开。

```
expand(gamma(1 + x), func=True)
```

$$x\Gamma(x)$$

- trig: 展开三角函数。

```
expand(sin(x + y), trig=True)
```

$$\sin(x)\cos(y) + \sin(y)\cos(x)$$

`expand_log()`、`expand_mul()`、`expand_complex()`、`expand_trig()`、`expand_func()`等函数则通过将相应的标志参数设置为True来对`expand()`进行包装。

`factor()`可以对多项式表达式进行因式分解：

```
factor(15 * x ** 2 + 2 * y - 3 * x - 10 * x * y)
```

$$(3x - 2y)(5x - 1)$$

`collect()`收集表达式中指定符号的有理指数次幂的系数。下面用它获取表达式eq中x的各次幂的系数。首先需要对表达式进行展开，得到一系列

乘式之和，然后调用`collect()`对 x 的幂的系数进行收集：

```
eq = (1 + a * x) ** 3 + (1 + b * x) ** 2
eq2 = expand(eq)
collect(eq2, x)
```

$$a^3x^3 + x^2(3a^2 + b^2) + x(3a + 2b) + 2$$

在默认参数的情况下，`collect()`返回的是一个整理之后的表达式。如果希望得到 x 的各次幂的系数，可以设置参数`evaluate`为`False`，让它返回一个以 x 的幂为键、以系数为值的字典。注意常数项对应的键为SymPy中的符号对象`1`，因此需要使用`S(1)`作为键：

```
p = collect(eq2, x, evaluate=False)
p[S(1)]      p[x**2]
-----
2            3*a**2 + b**2
```

也可以使用`coeff()`方法获得系数，下面分别获得常数项和 x^2 的系数：

```
eq2.coeff(x, 0)  eq2.coeff(x, 2)
-----
2              3*a**2 + b**2
```

`collect()`也可以收集表达式的各次幂的系数，例如下面的程序收集表达式 $\sin 2x$ 的系数：

```
collect(a * sin(2 * x) + b * sin(2 * x), sin(2 * x))
```

$$(a+b)\sin(2x)$$

6.3.2 方程

在SymPy中表达式可以直接表示值为0的方程，也可以使用Eq()创建方程。solve()可以对方程进行符号求解，它的第一个参数是表示方程的表达式，其后的参数是表示方程中未知变量的符号。下面的例子使用solve()对一元二次方程进行求解：

```
a, b, c = symbols("a,b,c")
%sympy_latex solve(a * x ** 2 + b * x + c, x)
```

$$\left[\left(-\frac{\sqrt{3}i}{3}, -\frac{2i}{3}\sqrt{3} \right), \left(\frac{\sqrt{3}i}{3}, \frac{2i}{3}\sqrt{3} \right) \right]$$

由于方程的解可能有多组，因此solve()返回一个保存所有解的列表。可以传递包含多个表达式的元组或列表来让solve()对方程组求解，得到的解是两层嵌套的列表，其中的每个元组表示方程组的一组解：

```
%sympy_latex solve((x ** 2 + x * y + 1, y ** 2 + x * y + 2), x, y)
```

$$\left[\left(-\frac{\sqrt{3}i}{3}, -\frac{2i}{3}\sqrt{3} \right), \left(\frac{\sqrt{3}i}{3}, \frac{2i}{3}\sqrt{3} \right) \right]$$

roots()可以计算单变量多项式的根：

```
%sympy_latex roots(x**3 - 3*x**2 + x + 1)
```

$$\{1:1, 1+\sqrt{2}:1, -\sqrt{2}+1:1\}$$

6.3.3 微分

Derivative是表示导函数的类，它的第一个参数是需要进行求导的表达式，第二个参数是求导的自变量。请注意Derivative所得到的是一个导函数，它并不会进行求导运算：

```
t = Derivative(sin(x), x)
t
```

$$\frac{d}{dx}\sin(x)$$

调用其doit()方法可计算求导的结果：

```
t.doit()
```

$$\cos(x)$$

也可以使用diff()函数或表达式的diff()方法来计算导函数：

```
diff(sin(2*x), x)
```

$$2\cos(2x)$$

使用Derivative对象可以表示自定义的数学函数的导函数，例如：

```
Derivative(f(x), x)
```

$$\frac{d}{dx}f(x)$$

由于SymPy不知道如何对自定义的数学函数进行求导，因此diff()会返回和上面相同的结果。添加更多的自变量符号参数可以表示高阶导函数，例如：

```
Derivative(f(x), x, x, x) # 也可以写作Derivative(f(x), x, 3)
```

$$\frac{d^3}{dx^3}f(x)$$

也可以对不同的自变量符号计算偏导数：

```
Derivative(f(x, y), x, 2, y, 3)
```

$$\frac{\partial^5}{\partial x^2 \partial y^3}f(x, y)$$

diff()的参数和Derivative相同，例如下面的程序计算函数sin(xy)对x两次求导、对y三次求导的结果：

```
diff(sin(x * y), x, 2, y, 3)
```

$$x(x^2y^2\cos(xy) + 6xysin(xy) - 6\cos(xy))$$

6.3.4 微分方程

dsolve()可以对微分方程进行符号求解。它的第一个参数是带未知函数的表达式，第二个参数是需要进行求解的未知函数。例如下面的程序对微分方程 $f'(x) - f(x) = 0$ 进行求解。所得到的结果是一个自然指数函数，它有一个待定系数 C_1 。

```
x=symbols('x')
f=symbols('f', cls=Function)
dsolve(Derivative(f(x), x) - f(x), f(x))
```

$$f(x) = C_1 e^x$$

不同形式的微分方程需要使用不同的解法，使用classify_ode()可以查看与指定微分方程对应的解法列表。下面查看方程 $f'(x) + f(x) = (\cos(x) - \sin(x))f(x)^2$ 对应的解法：

```
eq = Eq(f(x).diff(x) + f(x), (cos(x) - sin(x)) * f(x)**2)
classify_ode(eq, f(x))
('1st_power_series', 'lie_group')
```

可以通过`dsolve()`的`hint`参数指定解法，默认值为'default'，表示采用`classify_ode()`返回值中的第一个解法：

```
dsolve(eq, f(x))
```

$$f(x) = C_1 - \frac{C_1 x^2}{2} - \frac{C_1 x^3}{6} + \frac{C_1 x^4}{4} + \frac{C_1 x^5}{120} (-C_1(C_1 - 3) - C_1(C_1 + 1) + 4C_1 + 12) + \mathcal{O}(x^6)$$

上面的解是采用泰勒级数展开之后的结果。如果使用'lie_group'解法，则可以得到更简洁的结果：

```
dsolve(eq, f(x), hint="lie_group")
```

$$f(x) = \frac{1}{C_1 e^x - \sin(x)}$$

也可以将`hint`设置为'all'，让`dsolve()`尝试`classify_ode()`返回的所有解法：

```
dsolve(eq, f(x), hint="all")
{'1st_power_series': f(x) == C1 - C1*x**2/2 - C1*x**3/6 +
C1*x**4/4 + C1*x**5*(-C1*(C1 - 3) - C1*(C1 + 1) + 4*C1 + 12)/120 +
O(x**6),
'best': f(x) == C1 - C1*x**2/2 - C1*x**3/6 + C1*x**4/4 +
C1*x**5*(-C1*(C1 - 3) - C1*(C1 + 1) + 4*C1 + 12)/120 + O(x**6),
'best_hint': '1st_power_series',
'default': '1st_power_series',
'lie_group': f(x) == 1/(C1*exp(x) - sin(x)),
'order': 1}
```


通过`sympy.ode.allhints`可以查看系统支持的所有解法：

```
sympy.ode.allhints
('separable',
 '1st_exact',
 '1st_linear',
 'Bernoulli',
 ...)
```

6.3.5 积分

`integrate()` 可以计算定积分和不定积分：

- `integrate(f, x)`：计算不定积分 $\int f dx$
- `integrate(f, (x, a, b))`：计算定积分 $\int_a^b f dx$

如果要对多个变量计算多重积分，只需要将被积分的变量依次列出即可：

- `integrate(f, x, y)`：计算双重不定积分 $\int \int f dx dy$
- `integrate(f, (x, a, b), (y, c, d))`：计算双重定积分 $\int_c^d \int_a^b f dx dy$

和`Derivative`对象表示微分表达式类似，`Integral`对象表示积分表达式，它的参数和`integrate()`的类似，例如：

```
e = Integral(x*sin(x), x)
e
```

$\int x \sin(x) dx$

调用积分对象的`doit()`方法以进行积分运算：

```
e.doit()
```

$-\text{xcos}(x) + \sin(x)$

有些积分表达式无法进行符号化简，这时可以调用求值方法`evalf()`或求值函数`N()`来对其进行数值运算：

```
e2 = Integral(sin(x)/x, (x, 0, 1))  
e2.doit()
```

$\text{Si}(1)$

`doit()` 返回的是一个用特殊函数表示的值。下面用`evalf()`求其数值：

```
print e2.evalf()  
print e2.evalf(50) # 可以指定精度  
0.946083070367183  
0.94608307036718301494135331382317965781233795473811
```

$\frac{\sin(x)}{x}$ 的积分被定义为一个特殊函数，它从0到无穷的定积分值为 $\pi/2$ ，即：

$$\int_0^{\infty} \frac{\sin(x)}{x} dx = \pi/2$$

但是SymPy的数值计算功能还不够强大，不能对应这种情况的定积分：

```
e3 = Integral(sin(x)/x, (x, 0, oo))  
e3.evalf()
```

-4.0

而调用`doit()`则能计算出精确的符号结果：

```
e3.doit()
```

$$\frac{\pi}{2}$$

6.4 输出符号表达式



与本节内容对应的 Notebook 为：[06-sympy/sympy-400-output.ipynb](#)。

SymPy 可以将表达式输出成各种格式，除了诸如 LaTeX、MathML 等显示的格式之外，还可以将表达式转换成 Python、C、Fortran 函数，从而将符号表达式转换成数值运算函数。

6.4.1 lambdify

`lambdify()` 可将表达式转换为数值运算函数。它的第一个参数是作为参数的符号序列，第二个参数是表达式或表达式序列。例如下面用 `solve()` 得到一元二次方程的两个符号解 `quadratic_roots`，然后调用 `lambdify()` 将其转换成数值运算函数：

```
a, b, c, x = symbols("a, b, c, x", real=True)
quadratic_roots = solve(a*x**2 + b*x + c, x)
lam_quadratic_roots_real = lambdify([a, b, c], quadratic_roots)
lam_quadratic_roots_real(2, -3, 1)
[1.0, 0.5]
```

所创建的函数默认使用math模块中定义的数值函数，因此无法计算根为复数的方程。可以通过modules参数指定表达式中函数所在的模块。下面改用cmath模块中的sqrt()函数，因此可以得到复数结果：

```
import cmath

    lam_quadratic_roots_complex = lambdify((a, b, c),
quadratic_roots, modules=cmath)

    lam_quadratic_roots_complex(2, 2, 1)

[(-0.5+0.5j), (-0.5-0.5j)]
```

还可以使用numpy模块中的函数计算多个一元二次方程的解。由于NumPy中的部分函数名与SymPy中的函数名不同，因此下面的程序中使用字符串'numpy'表示numpy模块，这样lambdify()会在其内部对函数名进行自动转换。下面的程序计算4个一元二次方程的解，为了得到复数解，需要使用dtype为complex的系数数组，运算结果是两个长度为4的数组：

```
    lam_quadratic_roots_numpy = lambdify((a, b, c), quadratic_roots,
modules="numpy")

    A = np.array([2, 2, 1, 2], np.complex)
    B = np.array([1, 4, 2, 1], np.complex)
    C = np.array([1, 1, 1, 2], np.complex)
    lam_quadratic_roots_numpy(A, B, C)

[array([-0.25000000+0.66143783j, -0.29289322+0.j           ,
        -1.00000000+0.j           , -0.25000000+0.96824584j]),
 array([-0.25000000-0.66143783j, -1.70710678+0.j           ,
        -1.00000000+0.j           , -0.25000000-0.96824584j])]
```

6.4.2 用autowrap()编译表达式

`lambdify()` 将表达式转换成Python的函数。然而对于需要大量运算的表达式，Python函数的运算能力有限。如果希望更快的运算速度，可以使用`autowrap()`将表达式转换成C语言或Fortran语言的函数，并编译为扩展模块以供Python调用。

下面先将保存两个解表达式的列表转换为表示矩阵的Matrix对象 `matrix_roots`，然后调用`autowrap()`将`matrix_roots`转换为两个函数，注意这两个函数只能返回实数解。通过`backend`参数可以指定编译扩展模块的方式：

- 'f2py'：默认值，使用f2py将Fortran语言程序包装成扩展模块。
- 'cython'：使用Cython将C语言程序包装成扩展模块。

为了让`autowrap()`正常工作，需要通过`tempdir`参数设置保存临时文件的路径，可以在该路径下找到输出的源程序以及编译之后的扩展模块。可以通过`flags`参数指定额外的编译命令，这里通过`-I`将NumPy头文件所在的路径添加进编译器的头文件搜索路径。

```
from sympy.utilities.autowrap import autowrap
matrix_roots = Matrix(quadratic_roots)
quadratic_roots_f2py = autowrap(matrix_roots, args=[a, b, c], tempdir=r".\tmp")
quadratic_roots_cython = autowrap(matrix_roots, args=[a, b, c], tempdir=r".\tmp",
                                   backend="cython", flags=["-I" + np.get_include()])

quadratic_roots_f2py(2, -3, 1)  quadratic_roots_cython(2, -3, 1)
-----
[[ 1. ],                      [[ 1. ],
 [ 0.5]]                      [ 0.5]]
```

还可以使用`ufuncify()`将表达式包装为NumPy的ufunc函数。由于不支持将Matrix对象作为输出，这里将其中的第一个解表达式转换为ufunc函数：

```
from sympy.utilities.autowrap import ufuncify

quadratic_roots_ufunc = ufuncify((a, b, c), quadratic_roots[0],
```

```
tempdir=r".\tmp")

quadratic_roots_ufunc([1, 2, 10.0], [6, 7, 12.0], [4, 5, 1.0])
array([-0.76393202, -1.          , -0.09009805])
```

codegen()将表达式转换成源程序。它的第一个参数是一个二元元组或是一个包含多个二元元组的列表，每个元组由函数名和表达式构成。language参数指定输出的源代码的语言，prefix参数指定输出的源代码的文件名。它返回两个元组，分别为源程序文件和头文件的文件名及其内容。如果符号表达式是表示矩阵的Matrix对象，在输出代码中使用数组保存其中的每个元素。

```
from sympy.utilities.codegen import codegen
(c_name, c_code), (h_name, c_header) = codegen(
    [("root0", quadratic_roots[0]),
     ("root1", quadratic_roots[1]),
     ("roots", matrix_roots)],
    language="C",
    prefix="quadratic_roots",
    header=False)

print h_name
print "-" * 40
print c_header
print
print c_name
print "-" * 40
print c_code
quadratic_roots.h
-----
```

```

#ifndef PROJECT__QUADRATIC_ROOTS__H
#define PROJECT__QUADRATIC_ROOTS__H

double root0(double a, double b, double c);
double root1(double a, double b, double c);
void roots(double a, double b, double c, double
*out_1451769269);

#endif

```

quadratic_roots.c

```
#include "quadratic_roots.h"
```

```
#include <math.h>
```

```
double root0(double a, double b, double c) {
```

```
    double root0_result;
```

```
    root0_result = (1.0L/2.0L)*(-b + sqrt(-4*a*c + pow(b, 2)))/a;
```

```
    return root0_result;
```

```
}
```

```
double root1(double a, double b, double c) {
```

```
    double root1_result;
```

```

        root1_result = -1.0L/2.0L*(b + sqrt(-4*a*c + pow(b, 2)))/a;
        return root1_result;

    }

    void roots(double a, double b, double c, double *out_1451769269)
    {

        out_1451769269[0] = (1.0L/2.0L)*(-b + sqrt(-4*a*c + pow(b,
2)))/a;

        out_1451769269[1] = -1.0L/2.0L*(b + sqrt(-4*a*c + pow(b,
2)))/a;

    }

```

此外，还可以用`ccode()`和`fcode()`将符号表达式输出为C和Fortran语言的表达式：

```

print ccode(matrix_roots, assign_to="y")
y[0] = (1.0L/2.0L)*(-b + sqrt(-4*a*c + pow(b, 2)))/a;
y[1] = -1.0L/2.0L*(b + sqrt(-4*a*c + pow(b, 2)))/a;

```

6.4.3 使用`cse()`分步输出表达式

通常符号运算的结果都是十分复杂的表达式，其中包含许多重复运算部分。使用`cse()`可以将表达式中重复的部分提取为分步运算。`cse()`的结果是一个由两个列表组成的元组，第一个列表是临时变量以及与之对应的表达式，第二个列表是计算结果。

下面对一元二次方程的两个根的符号表达式提取公共表达式，得到两个列表replacements和reduced_exprs。在replacements中引入两个临时符号x0和x1，对reduced_exprs中的临时符号逐步使用replacements的表达式进行替换，就可以得到与roots相同的表达式。由于根式部分使用临时符号表示只需要运算一次，因此节省了运算时间。

```
replacements, reduced_exprs = cse(quadratic_roots)
%sympy_latex replacements
```

$$\left[\left(x_0, \frac{1}{2a} \right), \left(x_1, \sqrt{-4ac + b^2} \right) \right]$$

```
%sympy_latex reduced_exprs
```

$$[x_0(-b + x_1), -x_0(b + x_1)]$$

临时符号默认以x开头，如果临时符号与表达式中的符号相冲突，可以使用symbols参数修改：

```
replacements, reduced_exprs = cse(quadratic_roots,
symbols=numbered_symbols("tmp"))
%sympy_latex replacements
```

$$\left[\left(\text{tmp}_0, \frac{1}{2a} \right), \left(\text{tmp}_1, \sqrt{-4ac + b^2} \right) \right]$$

虽然使用cse()能将复杂的表达式简化为一系列简单的运算步骤，但是目前SymPy的代码输出功能尚未使用cse()。因此本书提供了cse2func()来将表达式通过cse()转换之后再输出为Python函数。下面用该函数将roots中的两个表达式转换为quadratic_roots()函数：

```
from scipy2.sympy.cseprinter import cse2func
code = cse2func("cse_quadratic_roots(a, b, c)", quadratic_roots)
exec code
```

```

print code

def cse_quadratic_roots(a, b, c):
    from math import sqrt
    _tmp0 = 0.5/a
    _tmp1 = sqrt((b)**(2.0) - 4.0*a*c)
    return (_tmp0*(_tmp1 - b), -_tmp0*(_tmp1 + b))

```

下面调用生成的`quadratic_roots()`函数来计算某个一元二次方程的解：

```

cse_quadratic_roots(1, -4, 2)
(3.41421356237, 0.585786437627)

```

由于在函数中使用`math.sqrt()`进行开方运算，因此不支持结果为复数的情况。可以通过`module`参数指定使用`cmath`模块进行计算：

```

import cmath

exec cse2func("cse_quadratic_roots(a, b, c)", quadratic_roots,
module=cmath)

cse_quadratic_roots(1, -4, 10)
((2+2.449489742783178j), (2-2.449489742783178j))

```

此外，`cse2func()`还有`auto_import`、`calc_number`、`symbols`等参数，请感兴趣的读者分析该函数的源代码以理解这些参数的作用。

6.5 机械运动模拟



与本节内容对应的 Notebook 为：`06-sympy/sympy-500-mechanics.ipynb`。

SymPy 还提供了许多专业领域的符号运算功能，例如 `physics.mechanics` 模块可以用于计算刚体系统的运动方程。作为本章的最后一节，我们用该模块计算如图6-3所示系统的运动方程，并进行数值模拟。在图中，滑动方块可以沿参照系 I 的 X 轴自由运动，小球与滑块使用无质量连杆相连接，可以自由摆动。小球的初始摆动角度为 θ_0 。我们希望计算小球释放之后的运动轨迹。

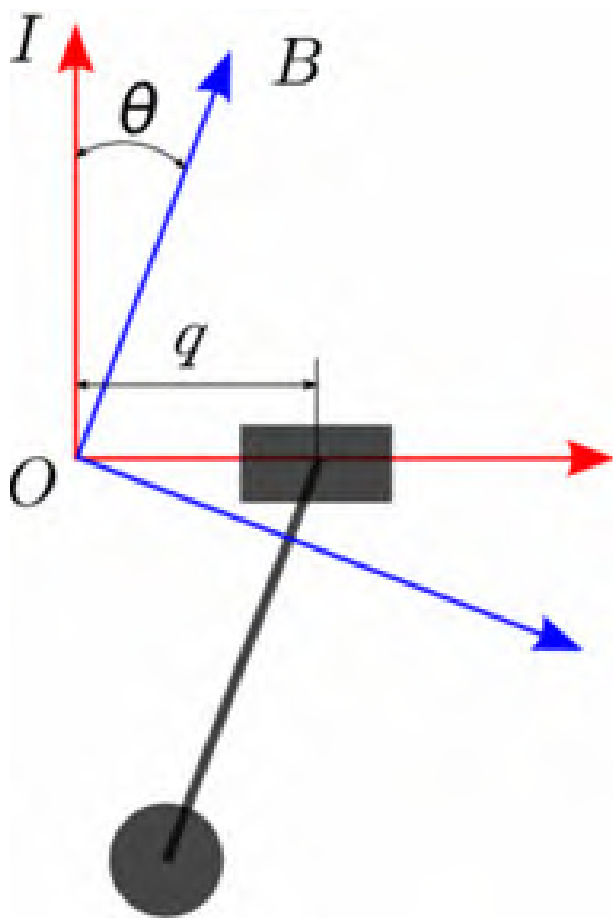


图6-3 滑块单摆系统的参照系示意图

6.5.1 推导系统的微分方程

首先从 `sympy.physics.mechanics` 载入所有符号，并使用其中的 `ReferenceFrame` 定义参照系 I ，用 `Point` 定义参照点 O 。最后调用 `O.set_vel()`，设置点 O 在参照系 I 中的运动速度为 0 。

```
from sympy.physics.mechanics import *

I = ReferenceFrame('I')          # 定义惯性参照系

O = Point('O')                   # 定义原点

O.set_vel(I, 0)                  # 设置点O在参照系I中的
速度为0

g = symbols("g")
```



<http://www.pydy.org/>

本节只介绍 `mechanics` 模块最基本的用法，若读者对使用 `SymPy` 求解多刚体系统感兴趣，可以参考 `PyDy` 扩展库。

下面定义方块在参照系 I 中的位置 q 和速度 u ，它们使用 `dynamicsymbols()` 定义。然后定义方块的质量为 m_1 ，质心为点 P_1 。接下来调用点 P_1 的 `set_pos()` 方法以设置它相对于点 O 的位移，`set_vel()` 方法设置它在参照系 I 中的速度，它们的方向沿着参照系 I 的 X 轴。最后在 P_1 处创建一个质量为 m_1 的质点来表示方块。

```
q = dynamicsymbols("q")
u = dynamicsymbols("u")
m1 = symbols("m1")
P1 = Point('P1')
P1.set_pos(O, q * I.x)          # 点P1的位置相对于点O，沿着参照系
```

I的x轴偏移 q

```
P1.set_vel(I, u * I.x) # 点P1在参照系I中的速度为x轴方向, 大小为u  
box = Particle('box', P1, m1) # 在点P1处放置质量为m1的方块box
```

使用`dynamicsymbols()`定义的符号是时间的函数:

```
%sympy_latex q, u
```

$(q(t), u(t))$

下面定义小球所在的参照系B, B为I绕Z轴旋转 θ 而得, 并设置B相对于I的角速度为 ω , 角速度围绕I的Z轴正方向旋转。角速度的正方向使用右手法则定义, 即右手大拇指指向围绕的轴, 四指的方向为正方向。

```
th = dynamicsymbols("theta")  
w = dynamicsymbols("omega")  
B = I.orientnew('B', 'Axis', [th, I.z]) # 将I围绕Z轴旋转theta  
得到参照系B  
B.set_ang_vel(I, w * I.z) # 设置B的角速度
```

细杆的长度为 l , 小球的质量为 m_2 。点 P_2 为小球的质心, 它的位置相对于点 P_1 , 沿着参照系B的Y轴负方向偏移 l , 并通过`v2pt_theory()`设置 P_2 在参照系I中的速度。若 P_2 和 P_1 在参照系B中相对静止, 当 P_1 在参照系I中的速度以及参照系B与参照系I之间的关系都确定时, 可以通过`P2.v2pt_theory(P1, I, B)`计算 P_2 在I中的速度。最后在 P_2 处放置质量为 m_2 的小球。

```
l, m2 = symbols("l,m2")  
P2 = P1.locatenew('P2', -l * B.y) # P2相对于P1沿着B的Y轴负方向偏移l
```

```
P2.v2pt_theory(P1, I, B)          # 使用二点理论设置P2在I中的速度
ball = Particle('ball', P2, m2)  # 在P2处放置质量为m2的小球
```

下面显示 P_2 在I中的速度:

```
P2.vel(I) #显示P2在I中的速度
```

$$u\hat{\mathbf{t}}_x + l\omega\hat{\mathbf{b}}_x$$

到此为止，各个惯性参照系、坐标点、质点之间的关系已经确定。下面创建KanesMethod对象，使用它可以推导出系统的微分方程组。 q_ind 参数为系统中所有与位移相关的独立状态列表， u_ind 参数为所有与速度相关的独立状态列表，而 kd_eqs 参数则是这些状态之间需要满足的微分方程。在本例中，方块的位移 q 的导数为速度 u ，细杆的旋转角度 θ 的导数为角速度 ω ：

```
eqs = [q.diff() - u, th.diff() - w] #q的导数为u, th的导数为w
kane = KanesMethod(I, q_ind=[q, th], u_ind=[u, w], kd_eqs=eqs)
```

然后调用`kane.equations()`方法推导微分方程。其中，`particles`为系统中所包含质点的列表，`forces`是系统所受外力的列表。每个作用力由作用点和矢量决定，这里定义两个质点上所受的重力。

```
particles = [box, ball] #系统包含的所有质点
forces = [(P1, -m1*g*I.y), (P2, -m2*g*I.y)] #系统所受的外力
fr, frstar = kane.kanes_equations(forces, particles)
```

`kane.equations()`返回系统的微分方程组。其中的每个方程为：`fr`和`frstar`中对应的表达式之和等于0。

```
%sympy_latex Eq(fr[0] + frstar[0], 0)
%sympy_latex Eq(fr[1] + frstar[1], 0)
```

$$lm_2\omega^2(t)\sin(\theta(t)) - lm_2\cos(\theta(t))\frac{d}{dt}\omega(t) - (m_1 + m_2)\frac{d}{dt}u(t) = 0$$

$$-glm_2\sin(\theta(t)) - l^2m_2\frac{d}{dt}\omega(t) - lm_2\cos(\theta(t))\frac{d}{dt}u(t) = 0$$

使用KanesMethod对象的mass_matrix_full和forcing_full属性可以写出求解系统状态的微分方程组，其中包含eqs中的两个方程：

```
from IPython import display

status = Matrix([[q],[th],[u],[w]])

display.Math(latex(kane.mass_matrix_full) + latex(status.diff()))
+
          "=" + latex(kane.forcing_full))
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & m_1 + m_2 & lm_2\cos(\theta(t)) \\ 0 & 0 & lm_2\cos(\theta(t)) & l^2m_2 \end{bmatrix} \begin{bmatrix} \frac{d}{dt}q(t) \\ \frac{d}{dt}\theta(t) \\ \frac{d}{dt}u(t) \\ \frac{d}{dt}\omega(t) \end{bmatrix} = \begin{bmatrix} u(t) \\ \omega(t) \\ lm_2\omega^2(t)\sin(\theta(t)) \\ -glm_2\sin(\theta(t)) \end{bmatrix}$$

6.5.2 将符号表达式转换为程序

当已知系统的状态 q 、 θ 、 u 、 ω 时，可以通过上面的公式计算出各个状态的导数，因此可以用`scipy.integrate.odeint()`对该微分方程组进行数值求解，从而计算出系统的运动轨迹。我们可以使用SymPy的表达式输出功能，自动生成计算各个状态的导数的函数。下面首先使用矩阵求逆与矩阵乘法计算线性方程组的解，结果`diff_status`是一个形状为(4, 1)的Matrix对象：

```
diff_status = kane.mass_matrix_full.inv() * kane.forcing_full
```

为了使用`autowrap()`将`diff_status`编译成扩展模块，需要将其中与`t`相关的表示状态的符号替换为一般符号。❶`status`中的每个元素都是与`t`相关的函数，可以通过`sym.func.__name__`获取`sym`对应的函数名，并使用该函数名创建与其同名的一般符号。❷调用`subs()`方法将所有与`t`相关的符号替换为一般符号，得到与`t`无关的矩阵`expr`。❸调用`autowrap()`将`expr`转换为计算其值的函数`_func_diff_status()`：

```
from sympy.utilities.autowrap import autowrap

status_symbols = [Symbol(sym.func.__name__) for sym in status] ❶
expr = diff_status.subs(zip(status, status_symbols)) ❷

_func_diff_status = autowrap(expr, args=[m1, m2, l, g] +
status_symbols, tempdir=r".\tmp_mechanics") ❸
```

由于`_func_diff_status()`的参数和返回值的形状不符合`odeint()`的要求，因此需要使用下面的`func_diff_status()`对其进行包装：

```
def func_diff_status(status, t, m1, m2, l, g):
    q, th, u, w = status
    return _func_diff_status(m1, m2, l, g, q, th, u, w).ravel()

init_status = np.array([0, np.deg2rad(45), 0, 0])
args = 1.0, 2.0, 1.0, 9.8
func_diff_status(init_status, 0, *args)
array([ 0.          ,  0.          ,  4.9          , -10.39446968])
```

下面调用`odeint()`对`func_diff_status()`进行积分，得到如图6-4所示的运动轨迹：


```

from scipy.integrate import odeint

t = np.linspace(0, 10, 500)

res = odeint(func_diff_status, init_status, t, args=args)

fig, (ax1, ax2) = plt.subplots(2, 1)
ax1.plot(t, res[:, 0], label=u"$q$")
ax1.legend()
ax2.plot(t, res[:, 1], label=u"$\theta$")
ax2.legend()

```

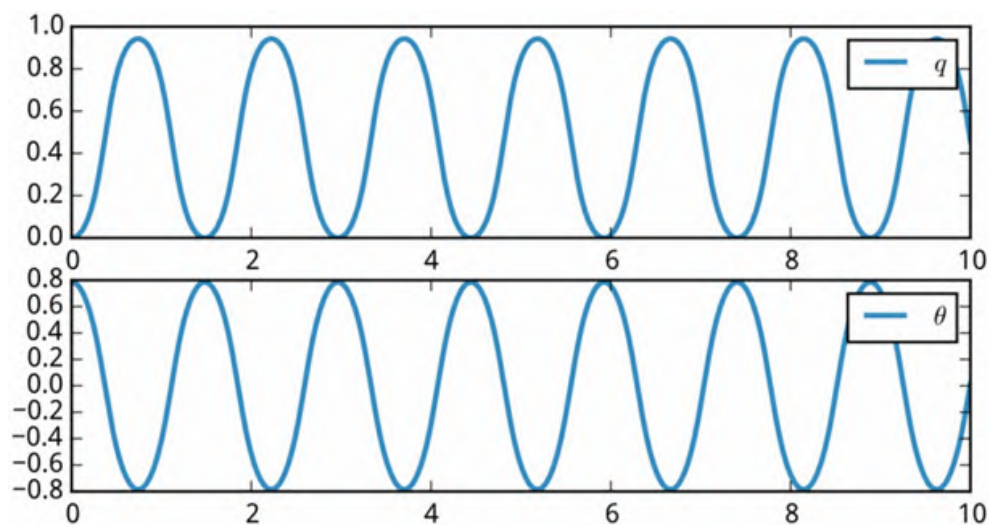


图6-4 使用odeint() 计算的运动轨迹

6.5.3 动画演示

可以通过matplotlib的动画制作功能，直观地显示系统的运动状况，效果如图6-5所示。关于这段程序的编写方法，请参照matplotlib一章中的相关说明。

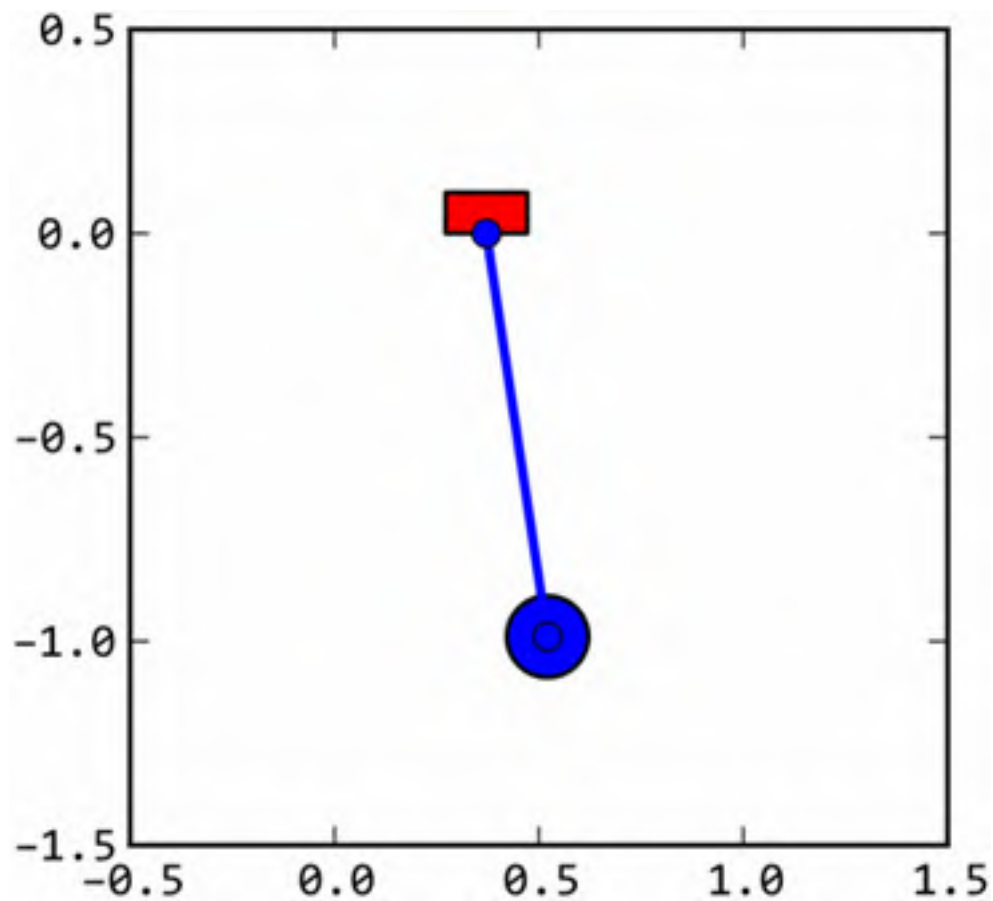


图6-5 动画演示效果

```
from matplotlib import animation
from matplotlib.patches import Rectangle, Circle
from matplotlib.lines import Line2D

def animate_system(t, states, blit=True):
    q, th, u, w = states.T
    fig = plt.figure()
    w, h = 0.2, 0.1
    ax = plt.axes(xlim=(-0.5, 1.5), ylim=(-1.5, 0.5),
aspect='equal')

    rect = Rectangle([q[0], 0, - w / 2.0, - h / 2],
```

```

        w, h, fill=True, color='red', ec='black', axes=ax,
animated=blit)

    ax.add_patch(rect)

    line = Line2D([], [], lw=2, marker='o', markersize=6,
animated=blit, axes=ax)

    ax.add_artist(line)

    circle = Circle((0, 0), 0.1, axes=ax, animated=blit)
    ax.add_patch(circle)

    def animate(i):
        x1, y1 = q[i], 0
        l = 1.0
        x2, y2 = l*sin(th[i]) + x1, -l*cos(th[i]) + y1
        rect.set_xy((x1-w*0.5, y1))
        line.set_data([x1, x2], [y1, y2])
        circle.center = x2, y2
        return rect, line, circle

    anim = animation.FuncAnimation(fig, animate, frames=len(t),
        interval=t[-1] / len(t) * 1000, blit=blit,
repeat=False)

    return anim

```

下面使用`%matplotlib`命令将matplotlib的后台绘图库改为qt，以便另外开启窗口来显示动画演示：

```
%gui qt  
  
%matplotlib qt  
  
anim = animate_system(t, res)
```


第7章 Traits&TraitsUI-轻松制作图形界面

Python作为一种高级的动态编程语言，它的变量没有类型，这种灵活性给快速开发带来很多便利，不过它也不是没有缺点。通过Traits库可以为对象的属性添加类型校验功能，从而提高程序的可读性，降低出错率。

Traits库是由Enthought公司开发的一套开源扩展库。虽然Traits库本身和科学计算没有关系，但它是该公司的其他各种科学计算库的基础，因此本书用一整章的篇幅对其进行详细介绍。

7.1 Traits类型入门



与本节内容对应的Notebook为：`07-traits/traits-100-intro.ipynb`。

Traits库最初是为了开发交互式绘图库Chaco而设计的。通常绘图库中有许多表示图形的对象，每个对象都有很多诸如线型、颜色和字体之类的属性。为了方便用户使用，每个属性可以允许多种形式的值。例如颜色属性可以是‘red’、0xff0000或(255, 0, 0)。也就是说，可以用字符串、整数或元组等类型的数据表示颜色。这样的需求初看起来用Python的无类型属性是一个很好的选择，因为我们可以把各种各样的值赋值给颜色属性。但是颜色属性虽然可以接受多样的

值，却不是能接受所有的值，比如'abc'和0.5等就不能很好地表示颜色。而且虽然为了方便用户使用，对外的接口可以接受多种类型的值，但是在程序内部必须有一个统一的表达方式来简化程序内部的实现。用Trait属性可以很好地解决这样的问题：

- 它可以接受能表示颜色的各种类型的值。
- 当给它赋值为不能表达颜色的值时，它能够立即捕捉到错误，并且提供一个有用的错误报告，告诉用户它能够接受什么样的值。
- 它提供内部的、标准的用于表达颜色的数据类型。

7.1.1 什么是Traits属性

下面我们通过一个简单的实例演示Trait属性的功能：

```
from traits.api import HasTraits, Color ❶  
  
class Circle(HasTraits): ❷  
    color = Color ❸
```

❶首先载入HasTraits和Color，推荐读者在使用Enthought公司开发的扩展库时采用和本例相同的载入方式。❷所有拥有Trait属性的类都需要从HasTraits继承。❸Color是Trait类型，在Circle类中用它定义了color属性。

熟悉Python的读者可能会觉得这个程序有些奇怪：按照标准的Python语法，直接在class下定义的属性color应该是Circle类的属

性，而程序的目的是为Circle类的实例添加color属性，是不是应该在初始化方法__init__()中运行self.color = Color呢？答案是否定的，记住Trait属性像类的属性一样定义，像实例的属性一样使用。我们不管HasTraits是如何实现这一点的，先看看如何使用Trait属性：

```
c = Circle()

Circle.color      #Circle类没有color属性

-----
-----

AttributeError                                Traceback (most
recent call last)

<ipython-input-4-8335a9908186> in <module>()
      1 c = Circle()
----> 2 Circle.color      #Circle类没有color属性

AttributeError: type object 'Circle' has no attribute
'color'
```

```
print c.color
print c.color.getRgb()
<PyQt4.QtGui.QColor object at 0x0542F270>
(255, 255, 255, 255)
```

从上面的运行结果可以看出Circle类没有color属性，而它的实例c则拥有color属性，其默认值为白色，PyQt4.QtGui.QColor是PyQt4界面库所使用的颜色类型。


```

c.color = "red"
print c.color.getRgb()
c.color = 0x00ff00
print c.color.getRgb()
c.color = (0, 255, 255)
print c.color.getRgb()

from traits.api import TraitError
try:
    c.color = 0.5
except TraitError as ex:
    print ex[0][:350], "..."
(255, 0, 0, 255)
(0, 255, 0, 255)
(0, 255, 255, 255)

The 'color' trait of a Circle instance must be a string of
the form (r,g,b) or (r,g,b,a) where r, g, b, and a are integers
from 0 to 255, a QColor instance, a Qt.GlobalColor, an integer
which in hex is of the form 0xRRGGBB, a string of the form
#RGB, #RRGGBB, #RRRGGBBB or #RRRRGGGGBBBB or 'aliceblue' or
'antiquewhite' or 'aqua' or 'aquamarine' or ...

```

由上面的运行结果可知，我们可以将' red'、0x00ff00和(0, 255, 255)等值赋给color属性，它们都被正确地转换为QColor类型的值。而当赋值为0.5时抛出TraitError异常，并且显示了一条很详细的出错信息来说明color属性能支持的所有值。最后看一个很酷的功能：

```
c.configure_traits()
```



当使用 wxPython 作为后台界面库时，由于 TraitsUI 4.4.0 中的一个错误，程序退出时会导致进程崩溃。请读者将本书提供的 `scipy2\patches\toolkit.py` 复制到 `site-packages\traitsui\wx` 目录下，覆盖原有的 `toolkit.py` 文件。

执行 `configure_traits()` 之后，会出现如图 7-1 所示的对话框界面以供我们修改 `color` 属性，任意选择一种颜色并单击 OK 按钮，`configure_traits()` 返回 `True`，而 `color` 属性已经变为我们通过界面所选择的颜色了：



如果在 Notebook 中运行 `c.configure_traits()`，它会立即返回 `False`，而不会等待对话框关闭。当程序单独运行时，`configure_traits()` 会等待界面关闭，并根据用户单击的按钮返回 `True` 或 `False`。

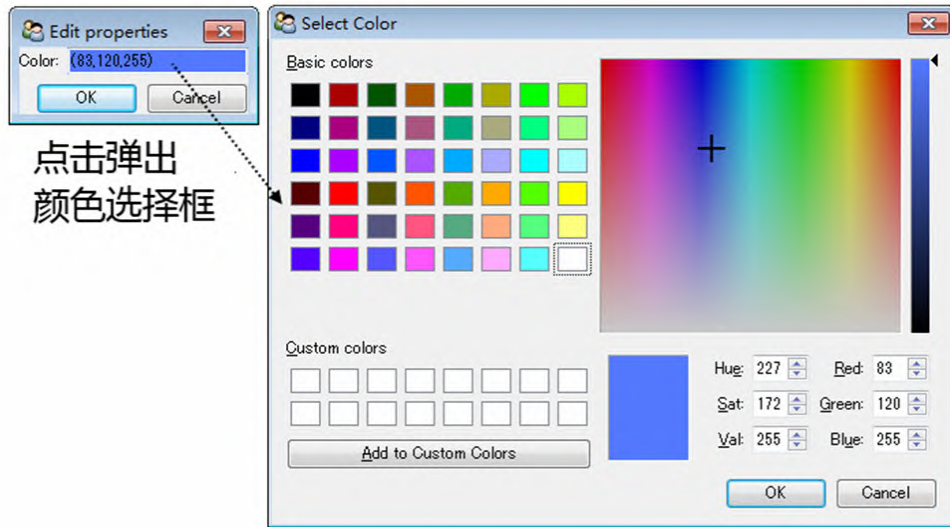


图7-1 自动生成的修改颜色属性的对话框

```
c.color.getRgb()  
(83, 120, 255, 255)
```

对于从HasTraits继承的对象，都可以调用configure_traits()方法以快速产生一个设置Trait属性的用户界面。在本例中，通过界面中的颜色输入框可以直接输入表示颜色的值，或者使用按钮打开颜色选择对话框。关于界面方面的功能将在本章的后半部分详细介绍。

7.1.2 Trait属性的功能

Traits库为对象的属性增加了类型定义的功能，此外还提供了如下额外功能：

- 初始化：每个Trait属性都有自己的默认值。
- 验证：Trait属性都有明确的类型定义，只有满足定义的值才能赋值给属性。

- 代理：Trait属性值可以代理给其他对象的属性。
- 监听：Trait属性值发生变化时，可以运行事先指定的函数。
- 可视化：拥有Trait属性的对象可以很方便地生成编辑Trait属性的界面。

下面的例子展示了Trait属性的上述功能：

```
from traits.api import Delegate, HasTraits, Instance, Int,
Str

class Parent ( HasTraits ):
    # 初始化：last_name被初始化为'Zhang'
    last_name = Str( 'Zhang' ) ❶

class Child ( HasTraits ):
    age = Int

    # 验证：father属性的值必须是Parent类的实例
    father = Instance( Parent ) ❷

    # 代理：将Child类的实例的last_name属性代理给其father属性的
last_name
    last_name = Delegate( 'father' ) ❸

    # 监听：当age属性的值被修改时，下面的函数将被运行
    def _age_changed ( self, old, new ): ❹
        print 'Age changed from %s to %s ' % ( old, new )
```

```
p = Parent()
c = Child()
```

程序中定义了Parent和Child这两个从HasTraits继承的类，并分别创建它们的对象p和c。❶用Str类型定义Parent对象的last_name属性是一个字符串，并且它的默认值为'Zhang'：

```
p.last_name
'Zhang'
```

❷用Instance类型定义Child对象的father属性是Parent类的实例，默认值为None。如果Parent类在Child类之后定义，可以用字符串表示类：father = Instance('Parent')。

❸通过Delegate类型为Child对象创建了一个代理属性last_name。它使c.last_name和c.father.last_name始终拥有相同的值。但是由于还没有设置对象c的father属性，因此无法正确获得对象c的last_name属性：

```
c.last_name
-----
-----
AttributeError                                Traceback (most
recent call last)
<ipython-input-10-fff30c984f1b> in <module>()
----> 1 c.last_name
```

```
AttributeError: 'NoneType' object has no attribute  
'last_name'
```

设置了对象c的father属性之后，就可以正确获取它的last_name属性了，并且对象c和p的last_name属性将始终保持一致：

```
c.father = p  
print c.last_name  
p.last_name = "ZHANG"  
print c.last_name  
  
Zhang  
  
ZHANG
```

④ 当对象c的age属性值发生变化时将调用监听函数_age_changed()：

```
c.age = 4  
  
Age changed from 0 to 4
```

下面调用configure_traits()以显示一个修改属性值的对话框，如图7-2(左)所示：

```
c.configure_traits()
```

从自动生成的界面可以看到，属性按照英文名排序，垂直排为一列。由于father属性是Parent类的对象，因此界面中以一个按钮表示它，单击此按钮将会弹出一个如图7-2(右)所示的对话框用于编辑father属性所对应的对象。



图7-2 为Child对象自动生成的属性修改对话框(左)，单击Father按钮弹出编辑Parent对象的对话框(右)

如果在编辑Father对象的对话框中修改last_name属性，Child对象的last_name属性也同时被修改，这是因为Child对象的last_name属性是一个代理属性，其值和father.last_name始终保持一致。

还可以调用print_traits()以输出所有的Trait属性名和属性值：

```
c.print_traits()
age:      4
father:    <__main__.Parent object at 0x05D9CC90>
last_name: 'ZHANG'
```

或调用get()以得到一个描述对象所有Trait属性的字典：

```
c.get()
{'age': 4, 'father': <__main__.Parent at 0x5d9cc90>,
'last_name': 'ZHANG'}
```

此外还可以调用set()以设置Trait属性的值，用set()可以同时配置多个Trait属性：

```
c.set(age = 6)
Age changed from 4 to 6
<__main__.Child at 0x5d9c600>
```

在创建HasTraits的派生类的对象时可以使用关键字参数设置各个Trait属性的值，例如：

```
c2 = Child(father=p, age=3)
Age changed from 0 to 3
```

当在派生类中定义了__init__()时，在其中必须调用其父类的__init__()方法，否则Trait属性的一些功能将无效。也许读者会对Trait属性的工作原理感兴趣，下面简单地介绍这些功能是如何实现的。

首先，Trait属性本身和普通Python对象的属性是一样的。但是每个Trait属性都有一个CTrait对象与之对应，这个CTrait对象为Trait属性提供了许多额外的功能。可以通过trait('属性名')获得与某个属性相对应的CTrait对象，或者用traits()获得包含所有CTrait对象的字典。下面的语句获得age属性对应的CTrait对象：

```
c.trait("age")
<traits.traits.CTrait at 0x9e23870>
```

Trait属性的默认值保存在与其对应的CTrait对象中：

```
p.trait("last_name").default
'Zhang'
```


给Trait属性赋值时的验证工作由CTrait对象的validate()完成。当验证失败时抛出异常，验证成功时则返回所要赋的值。因此validate()还可以用于修改所赋的值。下面直接调用father属性所对应的CTrait对象的validate()：

```
try:
    c.trait("father").validate(c, "father", 2)
except TraitError as ex:
    print ex
    The 'father' trait of a Child instance must be a Parent or
    None, but a value of 2 <type 'int'> was specified.
```

```
c.trait("father").validate(c, "father", p)
<__main__.Parent at 0x5d9cc90>
```

当 Trait 属性的值被改变时，HasTraits 对象的 trait_property_changed() 会被调用，trait_property_changed() 在 HasTraits 的父类 CHasTraits 中定义。在此方法中将会调用用户定义的属性监听函数。注意只调用监听函数，并不会修改属性的值，因此下面的语句将调用 _age_changed()，但不会修改 age 属性的值：

```
c.trait_property_changed("age", 8, 10)
c.age # age属性的值没有发生变化
Age changed from 8 to 10
6
```

CTrait对象是连接Trait属性和Trait类型的纽带，通过CTrait对象的trait_type属性可以获得定义Trait属性时所使用的Trait类型：

```
print c.trait("age").trait_type
print c.trait("father").trait_type
<traits.trait_types.Int object at 0x09DC0490>
<traits.trait_types.Instance object at 0x09DC0830>
```

7.1.3 Trait类型对象

在程序中使用Trait属性需要按照下面三个步骤进行：

(1) 从traits.api中载入所需的类型。

(2) 创建Trait类型对象。

(3) 创建一个从HasTraits类继承的新类，在其中使用所创建的Trait类型对象定义Trait属性。

通常步骤(2)和(3)是放在一起的，也就是说创建Trait类型对象的同时定义Trait属性，本书的大部分例子都是采用这种方式。例如：

```
from traits.api import Float, Int, HasTraits

class Person(HasTraits):
    age = Int(30)
    weight = Float
```

上面的程序为Person类定义了两个Trait属性：age和weight。其中age属性使用Int类型对象定义，30为默认值。而weight属性则直接使用Float类型定义，实际上它也会创建一个Float类型对象，其具体

的实现是在HasTraits类的内部进行。在上一节我们介绍过，每个Trait属性都对应一个CTrait对象，而通过CTrait对象的trait_type属性可以获得Trait类型对象。实际上，这些CTrait对象和Trait类型对象都是在类中保存的，因此对于同一个HasTraits派生类的多个实例，它们的某个Trait属性所对应的CTrait对象都是同一个对象。下面创建两个Person类的实例，并分别查看它们的Trait属性所对应的CTrait对象和Trait类型对象，可以看出多个实例之间共享CTrait对象的Trait类型对象：

```
p1 = Person()
p2 = Person()
print p1.trait("age") is p2.trait("age")
           print          p1.trait("weight").trait_type          is
p2.trait("weight").trait_type
True
True
```

也可以先单独创建一个Trait类型对象，然后用它定义多个Trait属性：

```
from traits.api import HasTraits, Range

coefficient = Range(-1.0, 1.0, 0.0)

class Quadratic(HasTraits):
    c2 = coefficient
    c1 = coefficient
    c0 = coefficient
```

```
class Quadratic2(HasTraits):  
    c2 = Range(-1.0, 1.0, 0.0)  
    c1 = Range(-1.0, 1.0, 0.0)  
    c0 = Range(-1.0, 1.0, 0.0)
```

上述程序中，Quadratic类有多个类型为Range的Trait属性，并且取值范围都是-1.0到1.0，初始值为0.0。为了尽量重用代码，我们先创建了一个Range类型对象，然后使用它定义了三个Trait属性。为了比较，在Quadratic2类中定义Trait属性时创建Range类型对象。

Quadratic对象的三个属性所对应的类型对象都是coefficient:

```
q = Quadratic()  
  
print coefficient is q.trait("c0").trait_type  
print coefficient is q.trait("c1").trait_type  
True  
True
```

而Quadratic2对象的属性所对应的类型对象则是不同的对象:

```
q2 = Quadratic2()  
q2.trait("c0").trait_type is q2.trait("c1").trait_type  
False
```

7.1.4 Trait的元数据

Trait类型可以拥有元数据属性，这些属性保存在与Trait属性对应的CTrait对象中。下面以一个实例解释什么是元数据属性。

```
from traits.api import HasTraits, Int, Str, Array, List

class MetadataTest(HasTraits):

    i = Int(99, myinfo="test my info") ❶

    s = Str("test", label=u"字符串") ❷

    # NumPy的数组
    a = Array ❸

    # 元素为Int的列表
    list = List(Int) ❹

test = MetadataTest()
```

下面查看所有的Trait属性：

```
test.traits()

{'a': <traits.traits.CTrait at 0x9e4fbe0>,
 'i': <traits.traits.CTrait at 0x9e4f9d0>,
 'list': <traits.traits.CTrait at 0x9e4fb30>,
 's': <traits.traits.CTrait at 0x9e4fa80>,
 'trait_added': <traits.traits.CTrait at 0x4fc2c38>,
 'trait_modified': <traits.traits.CTrait at 0x4fc2be0>}
```

通过traits()方法可以得到一个包含所有CTrait对象的字典。CTrait对象用于描述Trait属性，例如：test.trait('i')描述

`test.i`, `test.trait('s')`描述`test.s`。`test`对象有两个额外的CTrait对象：`trait_added`和`trait_modified`，它们在HasTraits类中定义。

每个Trait属性都有一个与之对应的CTrait对象用于描述它。而所谓元数据属性就是描述Trait属性的属性，它们保存在CTrait对象中。元数据属性可以分为三类：

- 内部属性：这些属性是CTrait对象自带的，只读不能写。
- 识别属性：这些属性可以自由设置，它们可以改变Trait属性的一些行为。
- 用户属性：用户自己添加的属性，需要自己编写程序来使用它们。

下面是一些内部元数据属性，可以读取它们的值，但不能修改：

- `array`：是否是数组，不是数组的Trait属性没有此属性。
- `default`：Trait属性的默认值。
- `default_kind`：一个描述默认值的类型的字符串，值可以是“value”、“list”、“dict”、“self”、“factory”、“method”等。
- `trait_type`：定义Trait属性时所使用的Trait类型对象。
- `inner_traits`：内部的CTrait对象，在List、Dict中使用，用来描述List和Dict的内部元素。
- `type`：Trait属性的分类，可以是“constant”、“delegate”、“event”、“property”、“trait”。

下面的元数据属性不是预定义的，但是可以被HasTraits对象使用：

- desc：描述Trait属性用的字符串，在生成界面时中使用它作为所创建的编辑器的帮助信息。
- editor：指定在界面中编辑Trait属性时所使用的编辑器类型。
- label：界面中的Trait属性编辑器的标签字符串。
- rich_compare：指定判断Trait属性值发生变化的方式。默认值为True，表示按值比较，False表示按照对象地址比较。
- trait_value：指定Trait属性是否接受TraitValue类的对象，默认值为False。当为True时，将Trait属性设置为TraitValue()，重置Trait属性为默认值。
- transient：指定当对象被保存(持久化)时是否保存此Trait属性值，当此属性不存在时使用默认值True。

下面查看前面示例中test对象的各个Trait属性的元数据属性：

❶在创建Int类型对象时，设置其默认值为99，并设置一个名为myinfo的用户元数据属性。这些信息都保存在与属性i对应的CTrait对象中：

```
print test.trait("i").default
print test.trait("i").myinfo
print test.trait("i").trait_type
99
test my info
<traits.trait_types.Int object at 0x05DA4F50>
```

②属性s的默认值为“test”，并且它有一个识别元数据属性label。在生成界面时，使用它作为编辑器的标签。为了在界面中使用中文，需要使用Unicode字符串。如果运行test.configure_traits()来显示图形界面，可以看到该属性对应的标签文本为“字符串”。

```
print test.trait("s").label
字符串
```

③array用于定义NumPy数组类型的Trait属性，因此属性a的元数据属性array为True：

```
test.trait("a").array
True
```

④属性list是一个元素类型为整数的列表。通过inner_traits元数据属性可以获得与列表元素对应的CTrait属性：

```
print test.trait("list")
print test.trait("list").trait_type
print test.trait("list").inner_traits # list属性的内部元素所对应的CTrait对象

print test.trait("list").inner_traits[0].trait_type # 内部元素所对应的Trait类型对象

<traits.traits.CTrait object at 0x09E4FB30>
<traits.trait_types.List object at 0x05DA46D0>
(<traits.traits.CTrait object at 0x09E4FC38>,)
<traits.trait_types.Int object at 0x05DA4E50>
```


7.2 Trait类型



与本节内容对应的Notebook为：07-traits/traits-200-types.ipynb。

本节介绍Traits库中提供的各种Traits类型以及如何监听Traits属性值的变化。

7.2.1 预定义的Trait类型

Traits库为Python的许多数据类型提供了预定义的Trait类型。对于Python的每个简单数据类型都有两种Trait类型与之对应，见表7-1：

- 强制Trait类型：当强制类型的Trait属性被赋值为类型不匹配的数据时，会抛出异常。
- 自动Trait类型：类型不匹配时会自动调用此类型对应的转换函数进行类型转换。

表7-1 预定义的Trait类型

强制类型	自动类型	内置默认值	自动转换函数
Bool	CBool	False	bool()

Complex	CComplex	0+0j	complex()
Float	CFloat	0.0	float()
Int	CInt	0	int()
Long	CLong	0L	int()
Str	CStr	''	str()
Unicode	CUnicode	u''	unicode()

下面的例子比较这两种类型：

```
from traits.api import HasTraits, CFloat, Float, TraitError

class Person(HasTraits):
    cweight = CFloat(50.0)
    weight = Float(50.0)
```

程序中用自动Trait类型CFloat定义了一个cweight属性，它可以接收能转换为数值的字符串“90”，而weight属性则使用强制Trait类型Float定义，将它赋值为“90”则会抛出异常：

```
p = Person()
p.cweight = "90"
print p.cweight
try:
    p.weight = "90"
except TraitError as ex:
    print ex
```

```
90.0

The 'weight' trait of a Person instance must be a float, but
a value of '90' <type 'str'> was specified.
```

除了简单类型以外，Traits库还定义了许多其他的常用数据类型。表7-2列出了一些常用的预定义Trait类型：

表7-2 一些常用的预定义Trait类型

类型名	参数	说明
Any	Any([value=None, **metadata])	任何对象
Array CArray	Array([dtype=None, shape=None, value=None, typecode=None, **metadata])	NumPy数组
Button	Button([label="", image=None, style="button", orientation="vertical", width_padding=7, height_padding=5, **metadata])	按钮类型，通常用于触发事件，参数用于描述界面中的按钮的样式
Callable	Callable([value=None, **metadata])	可调用对象
Code	Code([value="", minlen=0, maxlen=sys.maxint, regex="", **metadata])	某种编程语言的字符串

Color	Color([*args, **metadata])	界面库中所采用的颜色对象
CSet	CSet([trait=None, value=None, items=True, **metadata])	自动转换类型的集合对象
Constant	Constant(value*[, ***metadata])	常量对象，其值不能改变，必须指定初始值
Dict	Dict([key_trait=None, value_trait=None, value=None, items=True, **metadata])	字典对象，为了方便使用，在Traits库中还预定义了一些键的类型为字符串的字典类型，例如DictStrAny, DictStrBool等
Directory	Directory([value="", auto_set=False, entries=10, exists=False, **metadata])	表示某个目录的路径的字符串
Either	Either(val1*[, *val2, ..., valN, **metadata])	多个Trait类型的复合，例如Either(Str, Float)表示定义的属性可以是字符串或浮点数
Enum	Enum(values*[, ***metadata])	枚举数据，其值可以是候选值中的任意一个
Event	Event([trait=None, **metadata])	触发事件用的对象
Expression	Expression([value="0", **metadata])	Python的表达式对象

File	File([value="", filter=None, auto_set=False, entries=10, exists=False, **metadata])	表示文件路径的字符串
Font	Font([*args, **metadata])	界面库中表示字体的对象

读者可以查看各个Trait类型的文档以了解其具体用法。下面以枚举类型为例，介绍Trait类型的使用方法。使用Enum可以定义枚举类型，在Enum的定义中给出所有的候选值，这些值必须是Python的简单数据类型，例如字符串、整数、浮点数等，候选值的类型可以不一样。可以直接将候选值作为参数，或者将其放在列表中，第一个值为缺省值：

```
from traits.api import Enum, List

class Items(HasTraits):
    count = Enum(None, 0, 1, 2, 3, "many")
    # 或者:
    # count = Enum([None, 0, 1, 2, 3, "many"])
```

下面是运行结果：

```
item = Items()
item.count = 2
item.count = "many"
try:
```

```
        item.count = 5
    except TraitError as ex:
        print ex

    The 'count' trait of an Items instance must be None or 0 or
    1 or 2 or 3 or 'many', but a value of 5 <type 'int'> was
    specified.
```

如果希望候选值是动态的，可以用values参数指定候选值所对应的属性名：

```
class Items(HasTraits):
    count_list = List([None, 0, 1, 2, 3, "many"])
    count = Enum(values="count_list")
```

在上面的Items类中，先用List定义了一个列表类型的count_list属性，并为其指定了默认值。然后在用Enum定义枚举类型的属性时，用values参数指定枚举属性的候选值为count_list属性中的元素。

```
item = Items()

try:
    item.count = 5    #由于候选值列表中没有5，因此赋值失败
except TraitError as ex:
    print ex

item.count_list.append(5)
item.count = 5        #由于候选值列表中有5，因此赋值成功
item.count
```

```
The 'count' trait of an Items instance must be None or 0 or 1 or 2 or 3 or 'many', but a value of 5 <type 'int'> was specified.
```

```
5
```

在上面的例子中，因为5不在count_list属性中，第一次将count属性赋值为5时抛出异常。当将5添加进count_list属性中之后，就可以将count属性设置为5了。

7.2.2 Property属性

在标准的Python语法中可以使用property()为类创建Property属性。Property属性的用法和一般属性相同，但是在获取它的值或者给它赋值时会调用相应的方法。在Traits库中也提供了类似的功能，但是用法比标准Python更简洁。我们先看一个例子：

```
from traits.api import HasTraits, Float, Property,
cached_property

class Rectangle(HasTraits):
    width = Float(1.0)
    height = Float(2.0)

    #area是一个属性，当width、height的值变化时，它对应的_get_area
    函数将被调用

    area = Property(depends_on=['width', 'height']) ❶
```

```

# 通过cached_property修饰器缓存_get_area()的输出
@cached_property ❷
def _get_area(self): ❸
    "area的get函数，注意此函数名和对应的Property名的关系"
    print 'recalculating'
    return self.width * self.height

```

❶在Rectangle类中，使用Property()定义了一个area属性。与Python的标准property属性不同，它根据属性名直接决定属性所对应的方法。当读取area属性值时，得到的是❸_get_area()的返回值；而当设置area属性值时，所设置的值将传递给_set_area()。由于在本例中没有定义_set_area()，因此area属性是只读的。此外，通过depends_on参数可以指定Property属性的依赖关系。本例中，当Rectangle对象的width和height属性值发生变化时需要重新计算area属性。

❷_get_area()用@cached_property修饰，这样_get_area()的返回值将被缓存，除非area属性所依赖的width和height属性值发生变化，否则将一直使用缓存值，而不会每次调用_get_area()。下面看看实际的运行效果：

```

r = Rectangle()
print r.area # 第一次取得area，需要进行运算
r.width = 10
print r.area # 修改width之后，取得area，需要进行计算
print r.area # width和height都没有发生变化，因此直接返回缓存值，没有重新计算

```



```
recalculating
2.0
recalculating
20.0
20.0
```

通过depends_on和@cached_property，系统可以跟踪area属性的状态，判断是否需要调用_get_area()以重新计算area属性的值。注意在运行r.width=10之后，并没有立即调用_get_area()，而只是保存一个需要重新计算的标志，等到真正需要area的值时，_get_area()才会被调用。

下面用%qtconsole启动一个QtConsole，并在其中连续调用两次edit_traits()，弹出图7-3所示的两个编辑界面：

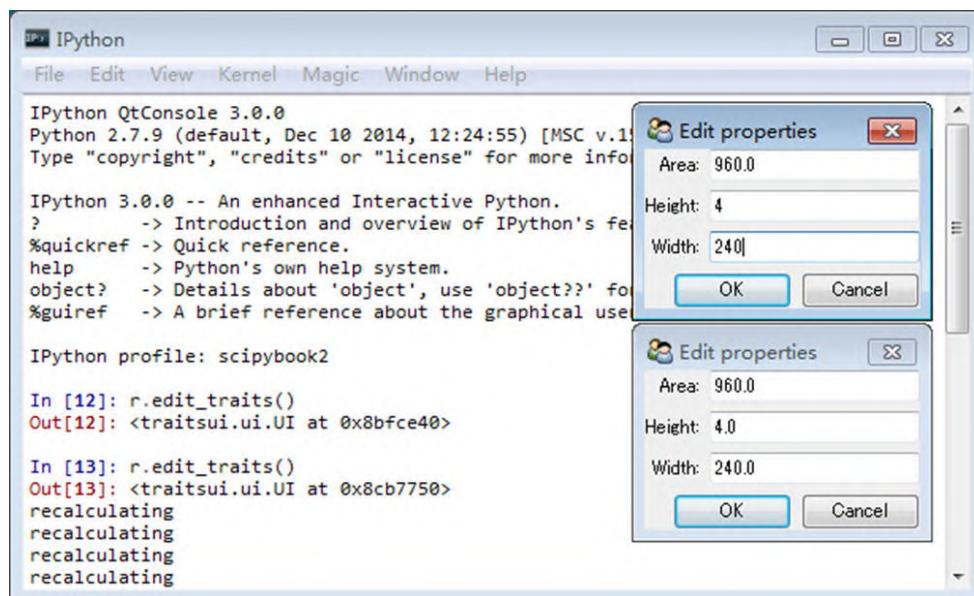


图7-3 修改两个对话框中的Height或Width属性会重新计算Area，并同时更新对话框中的显示

修改任意一个界面中的width或height属性，在输入数值的同时，两个界面中的Area、Height和Width等各个文本框同时更新，每次键盘按键都会调用`_get_area()`。此时在IPython窗口中直接修改width的值，也会调用`_get_area()`。

当打开界面之后，界面对象开始监听对象r的各个属性，因此在修改r.width之后，系统设置r.area的标志为需要重新计算，然后发现r.area的值有对象在监听，因此直接调用`_get_area()`更新其值，并且通知所有的监听对象，因此界面就一齐更新了。每个界面都会在Trait属性所对应的CTrait对象中添加监听对象：

```
t = r.trait("area") #获得与area属性对应的CTrait对象
t._notifiers(True) # _notifiers方法返回所有的通知对象，当area属性
改变时，这里对象将被通知

[<traits.trait_notifiers.FastUITraitChangeNotifyWrapper at
0x8b9e3f0>,
 <traits.trait_notifiers.FastUITraitChangeNotifyWrapper at
0x8bd4e10>]
```

由于我们弹出了两个界面，因此有两个需要通知的对象。如果再运行一次`r.edit_traits()`，这个列表将有3个元素。

7.2.3 Trait属性监听

HasTraits对象的所有Trait属性都自动支持监听功能。当某个Trait属性的值发生变化时，HasTraits对象会通知所有监听此属性的

函数。监听函数分为静态和动态两种，下面的程序演示了这两种监听方式：

```
from traits.api import HasTraits, Str, Int

class Child ( HasTraits ):
    name = Str
    age = Int
    doing = Str

    def __str__(self):
        return "%s<%x>" % (self.name, id(self))

    # 当age属性的值被修改时，下面的函数将被运行
    def _age_changed ( self, old, new ): ❶
        print "%s.age changed: form %s to %s" % (self, old,
new)

    def _anytrait_changed(self, name, old, new): ❷
        print "anytrait changed: %s.%s from %s to %s" %
(self, name, old, new)

    def log_trait_changed(obj, name, old, new): ❸
        print "log: %s.%s changed from %s to %s" % (obj, name,
old, new)

h = Child(name = "HaiYue", age=9)
```

```

k = Child(name = "KaiWen", age=2)
h.on_trait_change(log_trait_changed, name="doing") ④
anytrait changed: <8b823f0>.age from 0 to 9
<8b823f0>.age changed: form 0 to 9
anytrait changed: HaiYue<8b823f0>.name from  to HaiYue
anytrait changed: <8b823c0>.age from 0 to 2
<8b823c0>.age changed: form 0 to 2
anytrait changed: KaiWen<8b823c0>.name from  to KaiWen

```

①当Child对象的age属性值发生变化时，对应的静态监听函数 `_age_changed()` 将被调用。② `_anytrait_changed()` 是一个特殊的静态监听函数，任何Trait属性发生变化时都会调用此函数。

④ 通过调用 `h.on_trait_change()`，动态地将 ③ 普通函数 `log_trait_changed()` 和对象h的doing属性联系起来。当doing属性改变时，`log_trait_changed()` 将被调用。

下面分别改变h和k的属性：

```

h.age = 10
h.doing = "sleeping"
k.doing = "playing"
anytrait changed: HaiYue<8b823f0>.age from 9 to 10
HaiYue<8b823f0>.age changed: form 9 to 10
anytrait changed: HaiYue<8b823f0>.doing from  to sleeping
log: HaiYue<8b823f0>.doing changed from  to sleeping
anytrait changed: KaiWen<8b823c0>.doing from  to playing

```

静态监听函数的参数有如下几种形式：

```
_age_changed(self)
_age_changed(self, new)
_age_changed(self, old, new)
_age_changed(self, name, old, new)
```

而动态监听函数的参数有如下几种形式：

```
observer()
observer(new)
observer(name, new)
observer(obj, name, new)
observer(obj, name, old, new)
```

其中obj是拥有Trait属性的对象，name为值发生变化的属性名，old为改变之前的值，new为改变之后的值。

当多个Trait属性都需要使用同一个监听函数时，可以使用@on_trait_change装饰器：

```
@on_trait_change( names )
def any_method_name( self, ...):
    ...
```

当names所描述的Trait属性改变时，将调用any_method_name()，names是一个字符串或列表，它能够很灵活地描述一组Trait属性。下面列举了一些常用的属性匹配语法，当与之匹配的属性发生变化时将调用被修饰的监听函数：

- 用逗号隔开多个属性名：'foo,bar'，当self.foo或self.bar改变时。
- 用列表描述多个属性名：['foo','bar']，功能同上。
- 描述嵌套的属性名：'foo.bar'，当self.foo.bar或self.foo改变时。
- 描述嵌套的属性名：'foo:bar'，当self.foo.bar改变时。
- 列表属性：'foo[]'，self.foo是一个列表，当它本身或它的元素改变时。
- 指定属性名开头的字符串：'foo+'，当以foo开头的属性改变时。
- 指定元数据：'+foo'，当有名为foo的元数据的属性改变时。

完整的匹配方法请参考Traits库的用户手册。下面的程序演示了上述匹配语法：

```
from traits.api import HasTraits, Str, Int, Instance, List,
on_trait_change

class HasName(HasTraits):
    name = Str()

    def __str__(self):
        return "<%s %s>" % (self.__class__.__name__,
self.name)

class Inner(HasName):
    x = Int
```

```

y = Int

class Demo(HasName):
    x = Int
    y = Int
    z = Int(monitor=1) # 有元数据属性monitor的Int
    inner = Instance(Inner)
    alist = List(Int)
    test1 = Str()
    test2 = Str()

    def _inner_default(self):
        return Inner(name="inner1")

    @on_trait_change("x,y,inner.
[x,y],test+,+monitor,alist[]")
    def event(self, obj, name, old, new):
        print obj, name, old, new

```

下面是各种属性值改变时的运行结果:

```

d = Demo(name="demo")
d.x = 10 # 与x匹配
d.y = 20 # 与y匹配
d.inner.x = 1 # 与inner.[x,y]匹配
d.inner.y = 2 # 与inner.[x,y]匹配
d.inner = Inner(name="inner2") # 与inner.[x,y]匹配

```

```
d.test1 = "ok" #与 test+匹配
d.test2 = "hello" #与test+匹配
d.z = 30 # 与+monitor匹配
d.alist = [3] # 与alist[]匹配
d.alist.extend([4,5]) #与alist[]匹配
d.alist[2] = 10 # 与alist[]匹配
<Demo demo> x 0 10
<Demo demo> y 0 20
<Inner inner1> x 0 1
<Inner inner1> y 0 2
<Demo demo> inner <Inner inner1><Inner inner2>
<Demo demo> test1 ok
<Demo demo> test2 hello
<Demo demo> z 0 30
<Demo demo> alist [] [3]
<Demo demo> alist_items [] [4, 5]
<Demo demo> alist_items [5] [10]
```

7.2.4 Event和Button属性

Event和Button是两个专门用以处理事件的Trait类型，Button从Event继承，它除了Event的事件触发功能之外，还可以通过TraitsUI库自动生成界面中的按钮控件。Event属性和其他Trait属性相比有如下区别：

- 对Event属性赋值将触发与其绑定的属性监听事件，而通常的Trait属性只有在其值改变时才触发事件。
- Event属性不存储值，因此对其赋值只是起到触发事件的作用，而所赋的值将被忽略。试图获取Event属性值将抛出异常。由于Event属性所触发的事件不表示某个属性值的变化，因此它们所对应的静态监听函数名为_event_fired而不是_event_changed。下面是使用Event属性的例子：

```
from traits.api import HasTraits, Float, Event,
on_trait_change

class Point(HasTraits):           ❶
    x = Float(0.0)
    y = Float(0.0)
    updated = Event

    @on_trait_change( "x,y" )
    def pos_changed(self):         ❷
        self.updated = True

    def _updated_fired(self):      ❸
        self.redraw()

    def redraw(self):              ❹
        print "redraw at %s, %s" % (self.x, self.y)
```

❶在Point类中定义了x、y和updated三个Trait属性。❷使用@on_trait_change对pos_changed()方法进行修饰，当x或y属性被修改时pos_changed()会被调用。其中通过设置updated属性触发updated事件。❸在updated的事件处理方法_updated_fired()中调用redraw()以重新绘制。下面是修改各种属性时的事件触发情况：

```
p = Point()

p.x = 1

p.y = 1

p.x = 1 # 由于x的值已经为1，因此不触发事件

p.updated = True

p.updated = 0 # 给updated赋任何值都能触发

redraw at 1.0, 0.0

redraw at 1.0, 1.0

redraw at 1.0, 1.0

redraw at 1.0, 1.0
```

7.2.5 动态添加Trait属性

前面介绍的都是类的定义中声明Trait属性，以及在类的对象中使用Trait属性。由于Python是动态语言，因此Traits库也提供了直接为某个特定的对象添加Trait属性的方法。

在下面的例子中，直接生成一个HasTraits类的实例a，然后调用add_trait()方法以动态地为a添加一个名为x的Trait属性，其类型为Float，初始值为3.0：

```
a = HasTraits()
a.add_trait("x", Float(3.0))
a.x
3.0
```

接下来创建一个HasTraits类的实例b，用add_trait()为b添加一个属性a，指定其类型为HasTraits类的实例。然后把实例a赋值给实例b的属性a：

```
b = HasTraits()
b.add_trait("a", Instance(HasTraits))
b.a = a
```

然后为实例b添加一个类型为Delegate的属性y，在b.y和b.a.x之间建立代理连接。modify=True表示可以通过b.y修改b.a.x的值。可以看到当将b.y的值改为10时，a.x的值也同时改变了。

```
from traits.api import Delegate
b.add_trait("y", Delegate("a", "x", modify=True))
print b.y
b.y = 10
print a.x
3.0
10.0
```

实际上，通过赋值语句为HasTraits对象添加新属性时，这些属性都是Trait属性：

```
class A(HasTraits):  
    pass  
  
a = A()  
a.x = 3  
a.y = "string"  
a.traits()  
{'trait_added': <traits.traits.CTrait at 0x3927c90>,  
  'trait_modified': <traits.traits.CTrait at 0x3927c38>,  
  'x': <traits.traits.CTrait at 0x3927f50>,  
  'y': <traits.traits.CTrait at 0x3927f50>}
```

只不过它们的类型为Python，因此它们能够接收任何类型的对象，起不到校验的作用：

```
a.trait("x").trait_type  
<traits.trait_types.Python at 0x39399b0>
```

7.3 TraitsUI入门



与本节内容对应的Notebook为：07-traits/traits-300-
uiintro.ipynb。

Python有着丰富的界面开发库，除了缺省安装的Tkinter以外，wxPython、PyQt4等都是非常优秀的界面开发库。但是它们有如下共同

的问题：需要开发者掌握众多的API函数，许多细节需要开发者自己配置，例如控件的属性、位置以及事件响应等。

在开发科学计算程序时，通常希望快速实现一个够用的界面，让用户能够交互式地处理数据，而又不希望在界面制作上花费过多的精力。以Traits库为基础、以MVC模式为设计思想的TraitsUI库就是实现这一理想的最佳方案。

MVC的英文全称为Model-View-Controller，它的目的是实现一种动态的程序设计，简化程序的修改和扩展工作，并且使程序的各个部分能够充分被重复利用。

- Model(模型)：程序中存储数据以及对数据进行处理的部分。
- View(视图)：程序的界面部分，实现数据的显示。
- Controller(控制器)：起到视图和模型之间的组织作用，控制程序的流程，例如将界面操作转换为对模型的处理。

7.3.1 默认界面

TraitsUI库是一套建立在Traits库基础之上的用户界面库。它和Traits库紧密相连，如果读者已经设计好了一个从HasTraits继承的类，那么直接调用`configure_traits()`方法，系统将会使用TraitsUI库自动生成对话框界面，以供用户交互式地修改对象的Trait属性。下面是一个简单的例子：

```
from traits.api import HasTraits, Str, Int
```

```
class Employee(HasTraits):  
    name = Str  
    department = Str  
    salary = Int  
    bonus = Int  
  
Employee().configure_traits()
```

此程序创建一个Employee类的对象，然后调用configure_traits()来显示出如图7-4(左)所示的默认界面：

在此自动生成的界面中，所有的属性都采用文本框编辑，并且每个文本框的前面都有一个文字标签，上面的文字根据Trait属性名自动生成：第一个字母变为大写，所有的下划线变为空格。对话框的最下面提供了OK和Cancel按钮以确定或取消对Trait属性的修改。

由于salary属性定义为Int类型，当输入不能转换为整数时，输入框将以红色背景表示错误，并且OK按钮变成无效，如图7-4(右)所示。

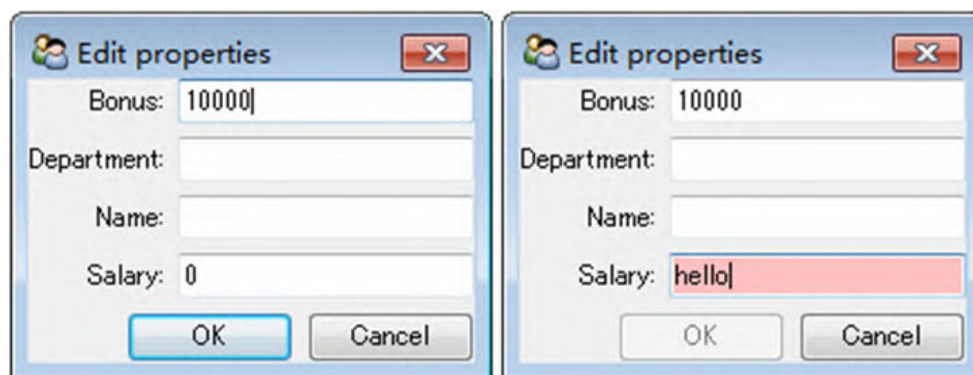


图7-4 自动生成的Employee类的对话框(左)，提醒非法的输入数据并且使OK按钮无效(右)

没有写一行界面相关的代码，就能得到一个够实用的界面，应该还是很令人满意的。为了手工控制界面的设计和布局，就需要添加自己的代码了。

7.3.2 用View定义界面

HasTraits的派生类用Trait属性保存数据，它相当于MVC模式中的模型。当没有指定界面显示方式时，Traits库会自动创建一个默认的界面。可以通过视图对象为模型设计更加实用的界面。

1. 外部视图和内部视图

下面是用视图对象定义界面的完整程序，图7-5显示了界面截图。



图7-5 使用视图对象描述界面

```
from traits.api import HasTraits, Str, Int
from traitsui.api import View, Item ❶

class Employee(HasTraits):
```

```

name = Str
department = Str
salary = Int
bonus = Int

view = View( ❷
    Item('department', label=u"部门", tooltip=u"在哪个部门
干活"), ❸
    Item('name', label=u"姓名"),
    Item('salary', label=u"工资"),
    Item('bonus', label=u"奖金"),
    title=u" 员 工 资 料 ", width=250, height=150,
resizable=True ❹
)

p = Employee()
p.configure_traits()

```

此程序在模型类Employee的基础之上添加了界面显示相关的代码。❶界面相关的内容都在traitsui.api中定义，这里从其中载入View和Item。View是描述界面的视图类，Item是描述界面中的控件和模型对象的Trait属性之间关系的类。

❷在Employee类下创建一个View对象，其中为Employee类的每个Trait属性创建了对应的Item对象。❸创建多个Item对象，并作为参数传递给View()。Item对象是视图的基本组成单位，每个Item对象描述界面中的一个编辑器，这些编辑器用于编辑模型对象中对应的Trait属

性的值。界面中的编辑器按照Item对象传递给View()的先后顺序显示，而不再按照Traits属性名排序。Item对象有许多参数，它们对Item对象的内容、表现以及行为进行描述。第一个参数指定与编辑器对应的Trait属性名，label和tooltip参数设置编辑器的标签和提示文本。Item对象还有很多其他属性，请读者参考TraitsUI的用户手册，或在IPython中输入Item??直接查看源代码。Item类从HasTraits继承，因此它的属性都是Trait属性。

除了Item之外，TraitsUI还定义了Item的几个派生类：Label、Heading和Spring。它们只用于辅助界面布局，因此不需要和模型对象的Trait属性关联。

④View类也从HasTraits继承，可以直接在创建View对象时，通过关键字参数设置其Trait属性。title属性为窗口标题栏中的文字，width和height属性为窗口的大小，resizable属性为True表示窗口的大小可变。

同一个模型对象可以通过多个视图对象用不同的界面显示，下面看一个例子：

```
from traits.api import HasTraits, Str, Int
from traitsui.api import View, Group, Item ❶

g1 = [Item('department', label=u"部门", tooltip=u"在哪个部门干活"), ❷
      Item('name', label=u"姓名")]
g2 = [Item('salary', label=u"工资"),
      Item('bonus', label=u"奖金")]
```

```
class Employee(HasTraits):  
    name = Str  
    department = Str  
    salary = Int  
    bonus = Int  
  
    traits_view = View( ❸  
        Group(*g1, label = u'个人信息', show_border = True),  
        Group(*g2, label = u'收入', show_border = True),  
        title = u"缺省内部视图")  
  
    another_view = View( ❹  
        Group(*g1, label = u'个人信息', show_border = True),  
        Group(*g2, label = u'收入', show_border = True),  
        title = u"另一个内部视图")  
  
    global_view = View( ❺  
        Group(*g1, label = u'个人信息', show_border = True),  
        Group(*g2, label = u'收入', show_border = True),  
        title = u"外部视图")  
  
p = Employee()  
  
# 使用内部视图traits_view  
p.edit_traits() ❻
```

❶从TraitsUI库载入Group类，用Group对象可以对界面中的编辑器分组。为了后续定义视图对象的程序更加简洁，❷程序中定义了两个全局列表g1和g2，它们的元素都是Item对象。

❸❹在Employee类内部用View()定义了两个视图对象：traits_view和another_view。而❺定义了一个全局的视图对象：global_view。在定义视图对象时，用Group对Item对象分组。

值得注意的是，Employee类中定义的两个视图对象既不是类的属性，也不是实例的属性。这些内部视图对象会放到Employee类的__view_traits__属性中。__view_traits__属性是一个ViewElements对象，它的content属性是保存所有内部视图的字典：

```
Employee.__view_traits__.content.keys()
['another_view', 'traits_view']
```

❻当调用edit_traits()以显示界面时，缺省使用模型类内部定义的缺省视图对象traits_view生成界面。也可以使用view参数指定显示界面时所使用的内部视图对象的名称：

```
# 使用内部视图another_view
p.edit_traits(view="another_view")
```

还可以直接将视图对象传递给view参数，这样可以使用在模型类之外定义的视图对象来生成界面：

```
# 使用外部视图view1
p.edit_traits(view=global_view)
```

图7-6显示了上面三种视图对象所生成的界面，每个视图的窗口标题都不同。由于这三个界面对应于同一个模型对象，因此无论在哪个窗口中修改模型对象的属性，另外两个窗口的内容也会同步更新。



图7-6 使用外部视图和内部视图定义界面

`edit_traits()`和`configure_traits()`一样用于生成界面，它们的区别在于`edit_traits()`显示界面之后不进入后台界面库的消息循环，因此如果直接运行只调用`edit_traits()`的程序，界面将在显示之后立即关闭，程序的运行也随之结束。而在`configure_traits()`中将进入消息循环，直到用户关闭所有窗口。因此通常主界面窗口或模式对话框使用`configure_traits()`显示，而无模式窗口或对话框则用`edit_traits()`显示。在IPython Notebook中，可以通过`%gui qt`或`%gui wx`命令启动界面消息循环，因此无论使用哪个方法显示界面，界面都不会阻塞Notebook内核的运行。



用TraitsUI库创建的界面可以选择后台界面库，目前支持的有qt4和wx两种。在启动程序时添加`-toolkit qt4`或`-toolkit wx`以选择使用何种界面库生成界面。本书中全部使用Qt作为后台界面库。

在本节的例子中，Employee类用于保存数据，因此它属于MVC模式中的模型；而View对象定义了Employee的界面显示部分，它属于视图。通过将视图对象传递给模型对象的configure_traits()方法，将模型对象和视图对象联系起来。在定义编辑器的Item对象中，不直接引用模型对象的属性，而是通过属性名与模型对象相关联。这样模型和视图之间的耦合性将会很弱，只需要属性名匹配，同一个视图对象可以运用到不同的模型对象之上。

有时候我们希望模型类知道如何显示自己，这时可以在模型类内部定义视图。在模型类中定义的视图可以被其派生类继承，因此派生类能使用父类的视图。在调用configure_traits()时如果不设置view参数，将使用模型对象内部的缺省视图对象生成界面。如果在模型类中定义了多个视图对象，则缺省使用名为traits_view的视图对象。

2. 多模型视图

在上节的例子中，一个模型可以对应多个视图。同样，使用一个视图可以将多个模型对象的数据显示在一个界面窗口中。下面是用一个视图对象同时显示多个模型对象的例子，程序的运行界面如图7-7所示：

```
from traits.api import HasTraits, Str, Int
from traitsui.api import View, Group, Item

class Employee(HasTraits):
    name = Str
```

```

    department = Str
    salary = Int
    bonus = Int

comp_view = View( ❶
    Group(
        Group(
            Item('p1.department', label=u"部门"),
            Item('p1.name', label=u"姓名"),
            Item('p1.salary', label=u"工资"),
            Item('p1.bonus', label=u"奖金"),
            show_border=True
        ),
        Group(
            Item('p2.department', label=u"部门"),
            Item('p2.name', label=u"姓名"),
            Item('p2.salary', label=u"工资"),
            Item('p2.bonus', label=u"奖金"),
            show_border=True
        ),
        orientation = 'horizontal'
    ),
    title = u"员工对比"
)

employee1 = Employee(department = u"开发", name = u"张三",
salary = 3000, bonus = 300) ❷

```

```

employee2 = Employee(department = u"销售", name = u"李四",
salary = 4000, bonus = 400)

HasTraits().configure_traits(view=comp_view, context=
{"p1":employee1, "p2":employee2}) ❸

```



图7-7 用一个视图同时显示多个模型对象

❶对于前面的模型类Employee，创建复合视图对象comp_view，它能同时显示两个Employee对象。注意Item对象的第一个参数不是简单的模型对象的属性名，它同时设置了Item对象的两个属性：object和name。例如参数“p1.department”将设置Item对象的object属性为“p1”，设置name属性为“department”。object属性告诉Item对象如何找到模型对象，而name属性则告诉Item对象如何找到模型对象中与之对应的属性。

❷接下来，创建组成模型对象的两个Employee对象：employee1和employee2。❸在显示界面时，使用context参数将包含两个模型对象的字典传递给configure_traits()。通过context参数传递的实际上是视图所对应的模型。这里的模型对象是一个字典，它的键和Item对象的object属性的值相同。由于已经通过context参数传递了模型对象，因此configure_traits()方法原本所属的对象将不会被用作界面的模

型对象。这里直接创建一个临时的HasTraits对象，然后调用`configure_traits()`方法。

如果读者认为这种写法有些取巧，也可以直接调用视图对象的`ui()`方法来显示界面，它的参数就是界面所要显示的模型对象。由于`ui()`和`edit_traits()`一样，不会启动界面库的消息循环，因此需要在运行`ui()`之后添加开始消息循环的代码。下面的消息循环代码支持所有的后台界面库：

```
from pyface.api import GUI

comp_view.ui({"p1":employee1, "p2":employee2})

GUI().start_event_loop() # 开始后台界面库的消息循环
```

3. Group对象

在前面例子的视图定义中，我们通过Group对象将一组相关的Item对象组织在一起。本节详细介绍如何使用Group对象组织界面。

在前面的两个例子中，将View对象中外层Group的`orientation`属性设置为`'horizontal'`，这样内部的两个Group对象将水平方向排列。下面的代码显示了View对象中Group的嵌套关系：

```
View(
    Group(
        Group(...),
        Group(...),
        orientation = 'horizontal'
```



```
)  
)
```

在创建Group时，可以通过设置orientation和layout等属性，改变Group内容的呈现方式。由于某些设置会经常用到，因此TraitsUI还提供了几个Group的派生类来覆盖这些属性的缺省值。例如下面是从Group类继承的HSplit类的代码：

```
class HSplit ( Group ):  
    # ...  
    layout      = 'split'  
    orientation = 'horizontal'
```

HSplit对象将它所包括的内容按照水平方向排列，并且在两块区域之间添加一个可调整位置的分隔条，使用HSplit和下面的代码等价：

```
Group( ... , layout = 'split', orientation = 'horizontal')
```

下面的程序演示了4种不同的界面分组方式，效果如图7-8所示。

```
from traits.api import HasTraits, Str, Int  
from traitsui.api import View, Item, Group, VGrid, VGroup,  
HSplit, VSplit  
  
class SimpleEmployee(HasTraits):  
    first_name = Str  
    last_name  = Str  
    department = Str
```

```
employee_number = Str
salary = Int
bonus = Int

items1 = [Item(name = 'employee_number', label=u'编号'),
           Item(name = 'department', label=u" 部 门 ",
tooltip=u"在哪个部门干活"),
           Item(name = 'last_name', label=u"姓"),
           Item(name = 'first_name', label=u"名")]

items2 = [Item(name = 'salary', label=u"工资"),
           Item(name = 'bonus', label=u"奖金")]

view1 = View(
    Group(*items1, label = u'个人信息', show_border = True),
    Group(*items2, label = u'收入', show_border = True),
    title = u"标签页方式",
    resizable = True
)

view2 = View(
    VGroup(
        VGrid(*items1, label = u'个人信息', show_border =
True, scrollable = True),
        VGroup(*items2, label = u'收入', show_border =
True),
```

```
    ),
    resizable = True, width = 400, height = 250, title =
u"垂直分组"
)

view3 = View(
    HSplit(
        VGroup(*items1, show_border = True, scrollable =
True),
        VGroup(*items2, show_border = True, scrollable =
True),
    ),
    resizable = True, width = 400, height = 150, title =
u"水平分组 (带调节栏) "
)

view4 = View(
    VSplit(
        VGroup(*items1, show_border = True, scrollable =
True),
        VGroup(*items2, show_border = True, scrollable =
True),
    ),
    resizable = True, width = 200, height = 300, title =
u"垂直分组 (带调节栏) "
)
```

```
sam = SimpleEmployee()

sam.configure_traits(view=view1)

sam.configure_traits(view=view2)

sam.configure_traits(view=view3)

sam.configure_traits(view=view4)
```

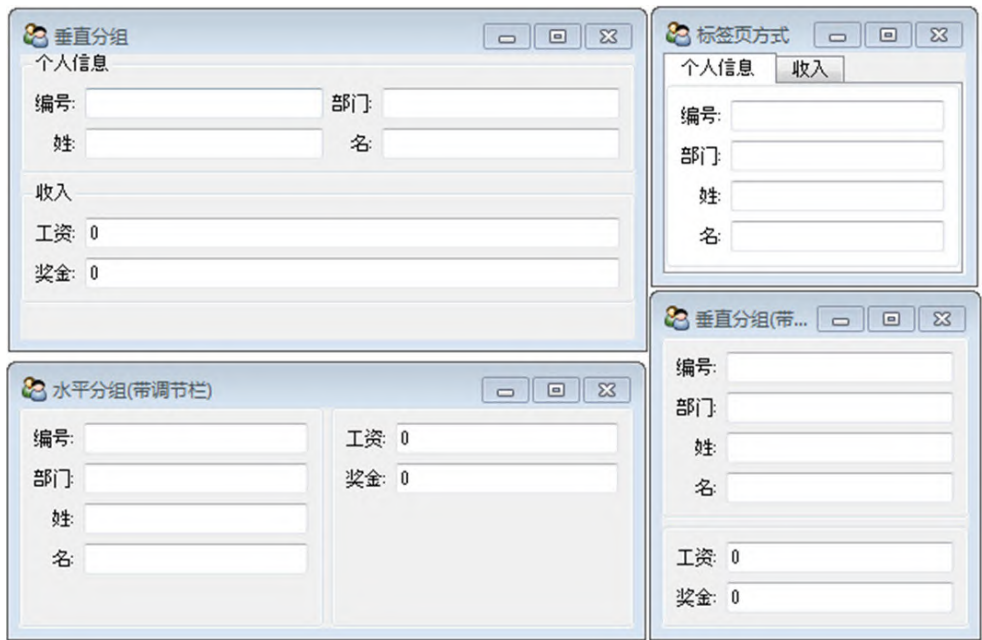


图7-8 在界面中用Group对象进行分组

表7-3是Group的各种派生类及对应属性的缺省设置：

表7-3 Group派生类及对应属性的设置

Group派生类	参数	说明
HGroup	orientation='horizontal'	水平排列
HFlow	orientation='horizontal'、	水平排列，当超过水平宽度时，将自动换行

	layout='flow'、 show_labels=False	show_labels属性为False表示隐藏其中的所有编辑器的标签文字
HSplit	orientation='horizontal'、 layout='split'	水平分隔，中间插入分隔条
Tabbed	orientation='horizontal'、 layout='tabbed'	分标签页显示
VGroup	orientation='vertical'	垂直排列
VFlow	orientation='vertical'、 layout='flow'、 show_labels=False	垂直排列，当超过垂直高度时，将自动换列
VFold	orientation='vertical'、 layout='fold'、 show_labels=False	垂直排列，可折叠
VGrid	orientation='vertical'、columns=2	按照多列的网格垂直排列，columns属性决定网格的列数
VSplit	orientation='vertical'、 layout='split'	垂直排列，中间插入分隔条

除了上述orientation、layout、show_labels、columns等属性之外，Group对象还提供了许多其他的属性来控制显示效果。和Item一

样，读者可以在Group类的源程序中找到每个属性的详细说明。

下面再通过一个例子说明visible_when和enabled_when属性的用法。这两个属性控制Group对象在界面中是否显示以及是否有效，效果如图7-9所示。



Item也提供了visible_when和enabled_when属性，用法和Group完全相同。

```
from traits.api import HasTraits, Int, Bool, Enum, Property
from traitsui.api import View, HGroup, VGroup, Item

class Shape(HasTraits):
    shape_type = Enum("rectangle", "circle")
    editable = Bool
    x, y, w, h, r = [Int]*5

    view = View(
        VGroup(
            HGroup(Item("shape_type"), Item("editable")),
            VGroup(Item("x"), Item("y"), Item("w"),
Item("h"),
                visible_when="shape_type=='rectangle'",
enabled_when="editable"),
            VGroup(Item("x"), Item("y"), Item("r"),
```

```

        visible_when="shape_type=='circle'",
enabled_when="editable"),

    ), resizable = True)

shape = Shape()
shape.configure_traits()

```

在上述程序中，Shape是一个表示矩形或圆形的类，具体形状由shape_type属性决定，而图形的参数则由x、y、w、h、r等属性决定。editable属性决定是否能通过用户界面修改图形参数。在视图定义中，使用VGroup对象定义了两个编辑器组，分别编辑矩形参数和圆形参数。通过设置VGroup的visible_when和enabled_when属性，将模型对象的shape_type和editable属性与编辑器的界面显示联系起来。

visible_when和enabled_when属性都是表示布尔表达式的字符串。当布尔表达式中涉及的模型对象的属性发生变化时，将会对字符串求值，并根据求值结果更新界面的显示。图7-9是程序的显示效果，其中左图对应的shape_type属性为“rectangle”，editable属性为True。当shape_type属性为“rectangle”时，将显示矩形参数的编辑器，隐藏圆形参数的编辑器。当editable属性为False时，所有编辑器都变成无效。

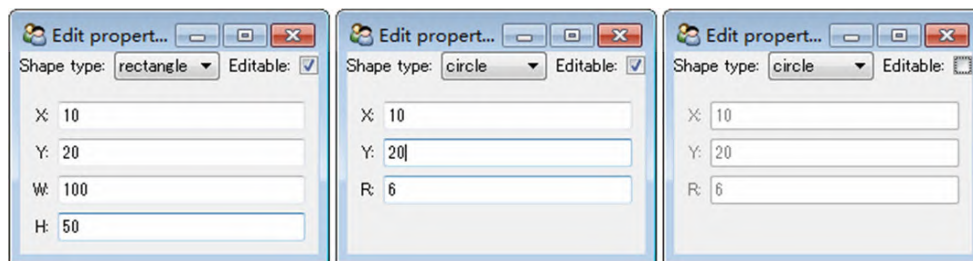


图7-9 演示visible_when和enabled_when属性的用法

4. 配置视图

前面介绍了如何使用Item和Group等对象组织窗口界面中的内容，本小节介绍如何配置窗口本身的属性。通过kind属性可以修改View对象的显示类型(见表7-4)：

表7-4 View对象的显示类型

类型	说明
modal	模式窗口，非即时更新
live	非模式窗口，即时更新
livemodal	模式窗口，即时更新
nonmodal	非模式窗口，非即时更新
wizard	向导类型
panel, subpanel	嵌入到其他窗口中的面板，即时更新，非模式

其中'modal'、'live'、'livemodal'、'nonmodal'这4种类型的View对象都采用窗口界面显示其内容。所谓模式窗口，是指在窗口关闭之前，程序中的其他窗口都不能被激活。而即时更新则是指当窗口中的编辑器内容改变时，会立即反映到编辑器所对应的模型对象的属性值。非即时更新的窗口则会复制模型对象，所有的改变在副本上进行，只有当用户单击OK或Apply按钮确定修改时，才会修改原始模型对象的属性。

'wizard' 由一系列特定的向导窗口组成，属于模式窗口，并且即时更新数据。

'panel' 和 'subpanel' 则是嵌入到窗口中的面板，'panel' 可以拥有自己的命令按钮，而 'subpanel' 则没有命令按钮。

在对话框中经常可以看到OK、Cancel、Apply之类的按钮，它们被称为命令按钮，完成所有对话框都需要的一些共同操作。在TraitsUI中，这些按钮可以通过View对象的buttons属性设置，其值为要显示按钮的列表。

TraitsUI中定义了UndoButton、ApplyButton、RevertButton、OKButton、CancelButton等6个标准的命令按钮，每个按钮对应一个名字，在指定buttons属性时，可以使用按钮的类名或对应的名字。与按钮类对应的名字就是类名除去Button，例如UndoButton对应"Undo"。

traitsui.menu中还预定义了一些命令按钮列表，以方便用户直接使用整套按钮：

```
OKCancelButton = [ OKButton, CancelButton ]

ModalButtons   = [  ApplyButton,  RevertButton,  OKButton,
CancelButton, HelpButton ]

LiveButtons    = [  UndoButton,   RevertButton,  OKButton,
CancelButton, HelpButton ]
```

```
from traitsui import menu

[btn.name for btn in menu.ModalButtons]

[u'Apply', u'Revert', u'OK', u'Cancel', u'Help']
```

7.4 用Handler控制界面和模型



与本节内容对应的Notebook为：[07-traits/traits-400-handler.ipynb](#)

虽然TraitsUI库的界面设计使用MVC模式，但是在前面的介绍中，没有出现过任何有关控制器的代码，这是因为TraitsUI库提供了缺省的控制器。我们可以通过继承Handler类来创建自己的控制器类，对视图和模型进行更自由的控制。

控制器最主要的功能是界面的事件处理，这些事件或对模型对象进行修改，或对界面进行修改。模型、视图以及控制器都是单独的实体。一个视图对象可以为多个模型对象生成界面，同样，一个控制器也可以用来处理不同视图界面中所产生的事件。因此控制器和视图以及模型之间不存在静态的联系。但是为了让控制器能够真正起作用，它必须知道自己所要处理的视图和模型。在TraitsUI中，控制器使用UIInfo对象获得它所对应的视图和模型。

当TraitsUI从视图创建界面时，将创建一个UIInfo对象，其中包括界面和模型对象的引用。当界面事件引起控制器的方法被调用时，这个UIInfo对象会作为参数传递给被调用的方法。

TraitsUI提供了三种指定控制器的方法：

- 将控制器作为视图的属性：使用视图对象的handler属性指定控制器对象，此视图所产生的界面都使用它进行事件处理。
- 显示界面时设置：调用edit_traits()、configure_traits()或ui()等方法显示界面时，将控制器对象传递给handler参数。它比视图的handler属性有更高的优先级。
- 将视图作为控制器的一部分进行定义。

7.4.1 用Handler处理事件

当显示某个视图时，将会按照下面的顺序执行控制器中的方法：

- (1) 创建一个UIInfo对象。
- (2) 运行控制器的init_info()方法。
- (3) 创建一个UI对象来表示实际的窗口。
- (4) 运行控制器的init()方法。
- (5) 运行控制器的position()方法。
- (6) 显示实际的窗口。

除了上面的init_info()、init()、position()之外，当用户操作界面时，会运行如下方法：

- apply()：用户单击窗口中的Apply按钮，模型对象的数据更新之后。
- close()：用户关闭窗口，在窗口关闭之前。

- `closed()`: 窗口关闭之后。
- `revert()`: 用户单击了Revert或Cancel按钮。
- `setattr()`: 用户通过界面修改了模型对象的某个Trait属性。
- `show_help()`: 用户单击了窗口中的Help按钮。

下面通过一个实例演示上述各个方法的使用:

```
from traits.api import HasTraits, Str, Int
from traitsui.api import View, Item, Group, Handler
from traitsui.menu import ModalButtons

g1 = [Item('department', label=u"部门"),
      Item('name', label=u"姓名")]
g2 = [Item('salary', label=u"工资"),
      Item('bonus', label=u"奖金")]

class Employee(HasTraits):
    name = Str
    department = Str
    salary = Int
    bonus = Int

    def _department_changed(self): ❶
        print self, "department changed to ",
self.department
```

```

def __str__(self): ❷
    return "<Employee at 0x%x>" % id(self)

view1 = View(
    Group(*g1, label = u'个人信息', show_border = True),
    Group(*g2, label = u'收入', show_border = True),
    title = u"外部视图",
    kind = "modal", ❸
    buttons = ModalButtons
)

class EmployeeHandler(Handler): ❹
    def init(self, info):
        super(EmployeeHandler, self).init(info)
        print "init called"

    def init_info(self, info):
        super( EmployeeHandler, self).init_info(info)
        print "init info called"

    def position(self, info):
        super(EmployeeHandler, self).position(info)
        print "position called"

    def setattr(self, info, obj, name, value):
        super(EmployeeHandler, self).setattr(info, obj,
name, value)

```

```

        print "setattr called:%s.%s=%s" % (obj, name, value)

def apply(self, info):
    super(EmployeeHandler, self).apply(info)
    print "apply called"

def close(self, info, is_ok):
    super(EmployeeHandler, self).close(info, is_ok)
    print "close called: %s" % is_ok
    return True

def closed(self, info, is_ok):
    super(EmployeeHandler, self).closed(info, is_ok)
    print "closed called: %s" % is_ok

def revert(self, info):
    super(EmployeeHandler, self).revert(info)
    print "revert called"

zhang = Employee(name="Zhang")
print "zhang is ", zhang

                                zhang.configure_traits(view=view1,
handler=EmployeeHandler()) ⑤

zhang is <Employee at 0x91efcf0>
init info called
init called
position called

```

```
<Employee at 0x96223c0> department changed to 开发
setattr called:<Employee at 0x96223c0>.department=开发
<Employee at 0x96223c0> department changed to 开发部门
setattr called:<Employee at 0x96223c0>.department=开发部门
<Employee at 0x91efcf0> department changed to 开发部门
apply called
close called: True
closed called: True
True
```

❶在Employee模型类中，定义了department属性的事件处理方法 `_department_changed()`。❷覆盖标准的字符串转换方法 `__str__()`，用以显示模型对象所占用的地址。

❸为了显示对话框的标准按钮，在创建视图对象时设置View的 `kind`和`button`参数分别为'modal'和ModalButtons。这样界面所显示的对话框是“模式窗口、非即时更新”，并且有Apply、Revert、OK、Cancel、Help等按钮，如图7-10所示。

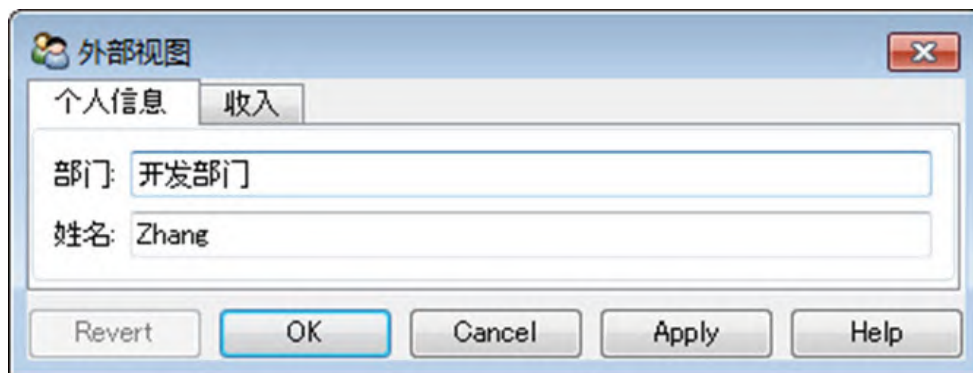


图7-10 带标准按钮的模式对话框

④EmployeeHandler从Handler继承，它覆盖了Handler中的init、init_info、position、setattr、apply、close、closed、revert等方法。如果close()返回True，则窗口会被关闭；如果它返回False，则不会关闭窗口。在这些方法中首先调用父类中被覆盖的方法，从而实现缺省的控制器的功能。实际上父类Handler中的大部分方法不执行任何任务，因此也可以不运行它们，请读者阅读Handler类的源代码以了解每个方法的缺省功能。

⑤最后创建控制器对象，并将它传递给configure_traits()的handler参数。

在对话框的“部门”文本输入框中输入“开发部门”，然后单击Apply按钮，最后单击OK按钮关闭对话框。程序在命令行窗口中输出控制器的各个方法的调用情况。

首先，对象zhang所表示对象的地址为0x91efcf0：

```
zhang is <Employee at 0x91efcf0>
```

调用configure_traits()之后，在窗口显示之前，运行了init_info()、init()、position()这三个方法：

```
init info called  
init called  
position called
```

接下来输入“开发部门”，每次文本输入框内的内容发生改变时，都会修改模型对象的department属性，从而调用模型对象的

`_department_changed()`，接着会调用控制器的`setattr()`。因此控制器的`setattr()`是在模型数据更新之后被调用的。

```
<Employee at 0x96223c0> department changed to 开发  
setattr called:<Employee at 0x96223c0>.department=开发  
<Employee at 0x96223c0> department changed to 开发部门  
setattr called:<Employee at 0x96223c0>.department=开发部门
```

`setattr()`的调用参数如下：

```
setattr(self, info, obj, name, value)
```

其中`info`是`UIInfo`对象，`obj`是被修改属性的模型对象，`name`是被修改的属性名，而`value`是被修改之后的值。仔细比较前面输出的对象地址就会发现：被修改属性的模型对象的地址并不是对象`zhang`的地址。这是因为“非即时更新”的对话框会对一个副本对象进行修改，在单击`Apply`或`OK`按钮时，才会将副本的内容写回原对象；而对副本对象的修改是“即时更新”的。

当单击`Apply`按钮时，程序输出了以下内容，可以看到原对象的`department`属性也被更新为“开发部门”了：

```
<Employee at 0x91efcf0> department changed to 开发部门  
apply called
```

最后单击`OK`按钮，程序输出下面两行，其中的`True`是`is_ok`参数的值，`True`表示用户单击的是`OK`按钮，如果单击`Cancel`按钮则为`False`：

```
close called: True  
closed called: True
```

7.4.2 Controller和UIInfo对象

Handler类的每个事件处理方法的第一个参数都是UIInfo对象，通过它可以获得控制器对应的模型对象和视图对象所产生的界面。但是有时我们希望通过控制器的属性访问它们。TraitsUI提供了从Handler继承的Controller类，它有两个Trait属性：model和info，分别保存模型对象和UIInfo对象。

在下面的程序中，模型和视图采用上节的定义，为了在显示窗口之后，在Notebook中继续运行命令，这里将view1.kind属性修改为非模式窗口。创建模型对象、控制器以及显示界面的代码如下：

```
from traitsui.api import Controller

view1.kind = "nonmodal"

zhang = Employee(name="Zhang")

c = Controller(zhang)

c.edit_traits(view=view1)
```

在创建Controller控制器时把模型对象传递给它，就可以通过c.model访问此模型对象。而调用Controller对象的edit_traits()不会显示控制器本身的Trait属性编辑窗口，而是显示模型对象的属性编辑窗口。

由于无论是控制器类、视图类还是模型类，最终都从HasTraits类继承，因此可以调用get()来快速查看其内容：

```
c.get()
{'_ipython_display_': None,
 '_repr_html_': None,
 '_repr_javascript_': None,
 '_repr_jpeg_': None,
 '_repr_json_': None,
 '_repr_latex_': None,
 '_repr_pdf_': None,
 '_repr_png_': None,
 '_repr_svg_': None,
 'info': <traitsui.ui_info.UIInfo at 0x5614810>,
 'model': <__main__.Employee at 0x55b71e0>}
```

```
c.info.get()
{'initialized': True, 'ui': <traitsui.ui.UI at 0x55b7570>}
```

```
c.info.ui.get()
{'_active_group': 0,
 '_checked': [],
 '_context': {'controller': <traitsui.handler.Controller at
0x5665870>,
 'handler': <traitsui.handler.Controller at 0x5665870>,
 ...}
```

c.info是一个UIInfo对象，而UIInfo对象中最重要的内容就是UI对象c.info.ui。在UI对象中保存了用户界面中的各种信息。对UI对象

的详细介绍已超出了本书的范围，请感兴趣的读者自行查看源代码。
下面简要地查看UI对象的几个属性：

```
ui = c.info.ui
ui.context
{'controller': <traitsui.handler.Controller at 0x56143f0>,
 'handler': <traitsui.handler.Controller at 0x360b090>,
 'object': <__main__.Employee at 0x5b58030>}
```

```
ui.control # ui对象所表示的实际界面控件
<traitsui.qt4.ui_base._StickyDialog at 0x5658780>
```

```
ui.view
( Group(
    Item( 'department'
        object      = 'object',
        label       = u'\u90e8\u95e8',
        ...
```

```
ui._editors
[<traitsui.qt4.text_editor.SimpleEditor at 0x5abe480>,
 <traitsui.qt4.text_editor.SimpleEditor at 0x5b00510>,
 ...
```

7.4.3 响应Trait属性的事件

前面介绍过，从HasTraits继承的模型类中，可以通过定义`_traitname_changed()`来响应`traitname`属性值改变的事件。这是在模型类中响应事件，如果要在控制器类中响应，可以通过定义`setattr()`来响应模型对象的Trait属性的改变。

如果希望只响应模型对象中某个特定属性的事件，可以在控制器类中定义如下格式的事件响应方法：

```
extended_traitname_changed(self, info)
```

其中的`extended`是视图所产生的UI对象的`context`属性中与模型对象相对应的键，通常为`'object'`。

这样的事件响应方法在界面窗口初始化时，以及对应的属性改变时都会被调用。为了区分二者，可以使用`info`参数的`initialized`属性来判断。下面是一个例子：

```
from traits.api import HasTraits, Bool
from traitsui.api import View, Handler

class MyHandler(Handler):
    def setattr(self, info, object, name, value): ❶
        Handler.setattr(self, info, object, name, value)
        info.object.updated = True ❷
        print "setattr", name

    def object_updated_changed(self, info): ❸
        print "updated changed", "initialized=%s" %
```

```

info.initialized

    if info.initialized:
        info.ui.title += "*"

class TestClass(HasTraits):
    b1 = Bool
    b2 = Bool
    b3 = Bool
    updated = Bool(False)

view1 = View('b1', 'b2', 'b3',
             handler=MyHandler(),
             title = "Test",
             buttons = ['OK', 'Cancel'])

tc = TestClass()
tc.configure_traits(view=view1)
setattr b2
updated changed initialized=False

```

❶MyHandler类中定义了setattr()方法，在修改了模型对象的任何一个Trait属性之后，它都将被调用。❷在setattr()中修改模型对象的updated属性为True。

❸当模型对象的updated属性被修改时，它在控制器对象中所对应的object_updated_changed()将被调用。当用户通过界面上的单选框

或者通过程序修改模型对象的属性时，将调用该方法，在窗口的标题栏中添加一个“*”。

7.5 属性编辑器



与本节内容对应的Notebook为：`07-traits/traits-500-editors.ipynb`。

每个Trait类型都有一种缺省的界面编辑器(控件)与之对应，如果在视图对象中不指定编辑器，将使用缺省的编辑器生成界面。每种编辑器都可以有如下4种样式：

- “simple”：缺省值，使用一个比较简单的编辑器，尽量少占用界面空间。
- “custom”：使用较复杂的编辑器，尽量呈现更多的内容。
- “text”：使用一个文本编辑器。
- “readonly”：使用只读控件显示。

由于TraitsUI的编辑器种类繁多，本书不能一一详细介绍，请感兴趣的读者运行位于本书附盘的codes目录下的演示程序，运行界面如图7-11所示。

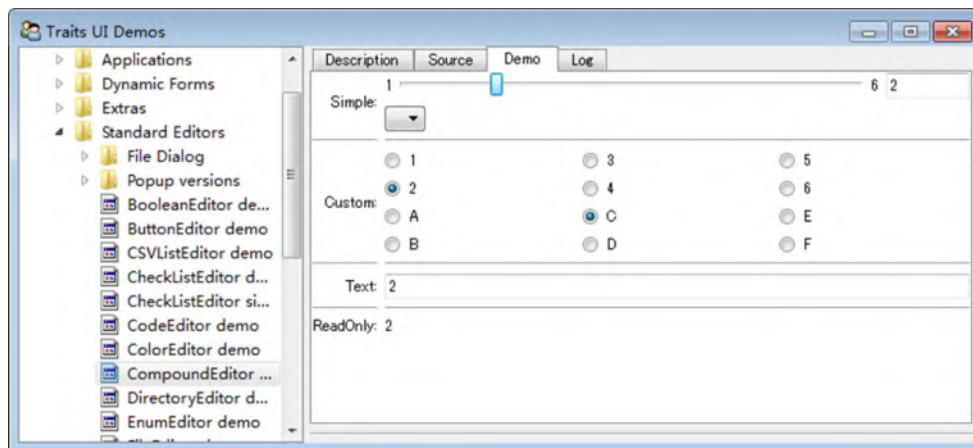


图7-11 TraitsUI演示程序的运行界面



traitsuidemo.demo: TraitsUI官方提供的演示程序。

下面以几个实例简单地介绍如何使用TraitsUI提供的编辑器。

7.5.1 编辑器演示程序

本节介绍一个能显示各种编辑器效果的演示程序，图7-12是它的界面截图。界面的左半部分是用来创建各种Trait属性的源程序列表，对于选中的某个Trait属性，在界面的右半部分使用4种样式创建属性编辑器。

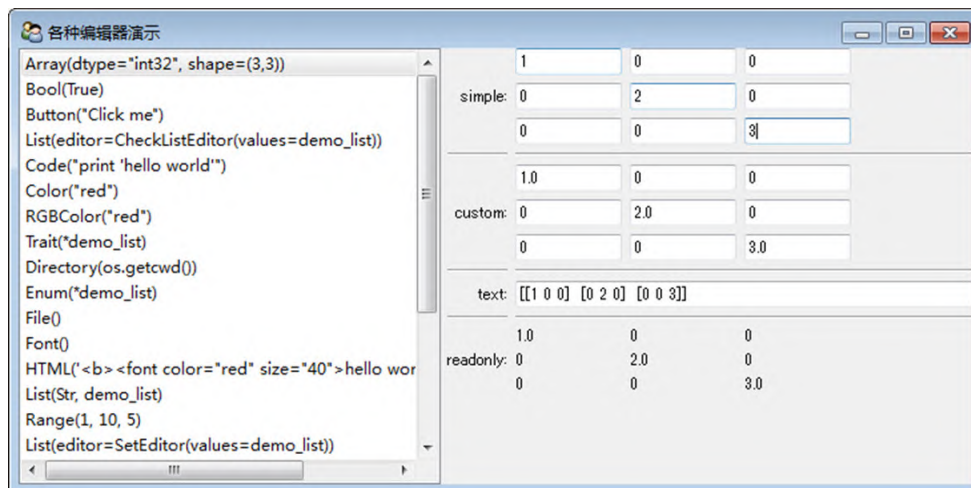


图7-12 演示TraitsUI提供的各种编辑器



scipy2.traits.traitsui_editors: 演示TraitsUI提供的各种编辑器的用法。

在下面的EditorDemoItem类的视图定义中，使用4种样式为item属性定义编辑器。请注意在EditorDemoItem类的定义中并没有item属性，但是由于视图中使用属性名字符串定义编辑器，因此只有在真正使用视图创建界面时，才会访问item属性，这时已经通过add_trait()为其添加了item属性。❶Item对象的width属性可以指定编辑器的宽度，以像素点为单位的长度用整数表示，负数表示强制设置其宽度。width属性还有多种设置宽度的用法，请读者查看Item类的源代码中的注释。❷使用下划线字符串在界面中创建分隔线。

```
class EditorDemoItem(HasTraits):
    code = Code()
    view = View(
```

```

        Group(
            Item("item", style="simple", label="simple",
width=-300), ❶
            "_", ❷
            Item("item", style="custom", label="custom"),
            "_",
            Item("item", style="text", label="text"),
            "_",
                                Item("item", style="readonly",
label="readonly"),
        ),
    )

```

在表示主界面的EditorDemo类中，codes属性保存一组用来创建各种Trait属性的字符串，selected_item属性是EditorDemoItem的对象。在EditorDemo类的视图定义中，使用HSplit将codes和selected_items所对应的编辑器水平隔开。❶用editor参数设置codes属性的编辑器为ListStrEditor，它是一个显示一组字符串的列表选择框控件。其editable属性为False，表示列表选择框中的字符串都是只读的。selected属性是保存被选中字符串的Trait属性名，在EditorDemo类中用selected_code属性保存列表选择框中被选中的字符串。可以通过editor参数设置Item对象的编辑器，这样界面中将使用指定的编辑器显示Trait属性。

❷当用户通过列表选择框选中了某个字符串时，selected_code属性将发生变化，因此_selected_code_changed()会被调用。在该方法中创建一个EditorDemoItem对象，并调用其add_trait()方法，动态地

为其创建一个名为item的Trait属性，其类型则通过eval()对selected_code字符串进行求值得到。

```
class EditorDemo(HasTraits):  
    codes = List(Str)  
    selected_item = Instance(EditorDemoItem)  
    selected_code = Str  
    view = View(  
        HSplit(  
            Item("codes", style="custom", show_label=False,  
①  
                editor=ListStrEditor(editable=False,  
selected="selected_code")),  
            Item("selected_item", style="custom",  
show_label=False),  
        ),  
        resizable=True,  
        width = 800,  
        height = 400,  
        title=u"各种编辑器演示"  
    )  
  
    def _selected_code_changed(self):  
        item = EditorDemoItem(code=self.selected_code)  
        item.add_trait("item", eval(self.selected_code)) ②  
        self.selected_item = item
```

最后是定义各种Trait类型的程序。可以在定义Trait类型时，通过editor参数设置对应的编辑器，这样就不需要在视图的Item对象中定义了。请读者自行研究每个Trait类型的定义以及它们所创建的界面控件，这里就不再进行详细说明了。

```
employee = Employee()
demo_list = [u"低通", u"高通", u"带通", u"带阻"]

trait_defines = """
    Array(dtype="int32", shape=(3,3))
    Bool(True)
    Button("Click me")
    List(editor=CheckListEditor(values=demo_list))
    Code("print 'hello world'")
    Color("red")
    RGBColor("red")
    Trait(*demo_list)
    Directory(os.getcwd())
    Enum(*demo_list)
    File()
    Font()
    HTML('<b><font color="red" size="40">hello world</font>
</b>')
    List(Str, demo_list)
    Range(1, 10, 5)
    List(editor=SetEditor(values=demo_list))
    List(demo_list, editor=ListStrEditor())
```

```

        Str("hello")
        Password("hello")
        Str("Hello", editor=TitleEditor())
        Tuple(Color("red"), Range(1,4), Str("hello"))
        Instance(EditorDemoItem, employee)
        Instance(EditorDemoItem, employee, editor=ValueEditor())
        Instance(time, time(), editor=TimeEditor())
    """
    demo = EditorDemo()
    demo.codes = [s.split("#")[0].strip() for s in
trait_defines.split("\n") if s.strip()!=""]
    demo.configure_traits()

```

7.5.2 对象编辑器

随着程序开发的进行，界面中的控件数目会逐渐增多，功能会越来越复杂，这意味着与界面对应的模型类也会变得复杂起来。为了便于代码的理解、管理以及重用，我们需要对模型类及其对应的界面视图对象进行重构。将程序中重复使用、相对独立的部分作为组件分离出来，单独为其设计模型类和视图对象，最终的应用程序由一系列这样的组件构成。这些组件可以在程序的不同地方重复使用，从而起到功能分离、代码重用等多方面的作用。TraitsUI的MVC模式非常适合这种组件开发方式，下面让我们通过一些实例深入理解MVC模式所带来的便利。

下面介绍的程序创建如图7-13所示的界面，用户可以通过上方的下拉选择框选择一种形状，选择框下面的控件会自动根据所选的形状发生变化。当通过这些控件输入形状数据时，界面下方的信息栏会自动更新。由于程序较长，下面将它分为几个部分进行分析。

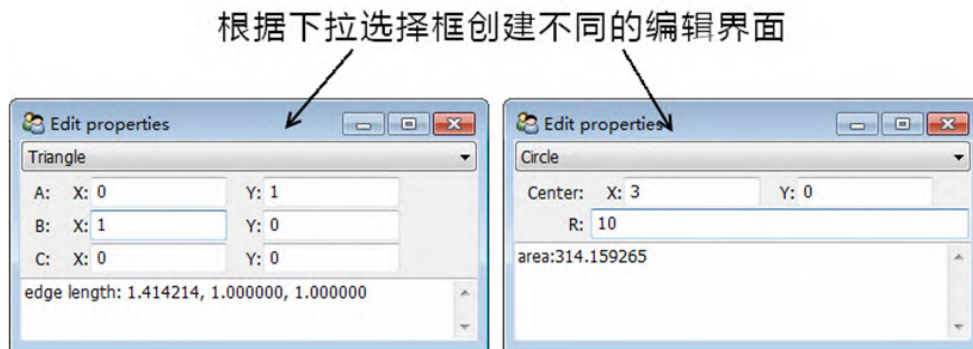


图7-13 组件演示，根据下拉选择框创建不同的编辑界面



scipy2.traits.traitsui_component: TraitsUI的组件演示程序。

```
class Point(HasTraits):  
    x = Int  
    y = Int  
    view = View(HGroup(Item("x"), Item("y")))
```

上面的程序定义了用于保存平面上点的坐标的Point类。我们还为它指定了一个视图对象，视图中X和Y轴的坐标值输入框是横向排列的。运行Point().configure_traits()即可看到Point对象所创建的界面效果。

我们可以将Point类当作组件使用，将它嵌入更复杂的界面中。在下面的程序中定义了一个基类Shape及其两个派生类Triangle和Circle，使用Point类定义所有表示二维坐标点的属性：

```
class Shape(HasTraits):
    info = Str ❶

    def __init__(self, **traits):
        super(Shape, self).__init__(**traits)
        self.set_info() ❷

class Triangle(Shape):
    a = Instance(Point, ()) ❸
    b = Instance(Point, ())
    c = Instance(Point, ())

    view = View(
        VGroup(
            Item("a", style="custom"), ❹
            Item("b", style="custom"),
            Item("c", style="custom"),
        )
    )

    @on_trait_change("a.[x,y],b.[x,y],c.[x,y]")
    def set_info(self):
```

```

        a,b,c = self.a, self.b, self.c

        l1 = ((a.x-b.x)**2+(a.y-b.y)**2)**0.5
        l2 = ((c.x-b.x)**2+(c.y-b.y)**2)**0.5
        l3 = ((a.x-c.x)**2+(a.y-c.y)**2)**0.5

        self.info = "edge length: %f, %f, %f" % (l1,l2,l3)

class Circle(Shape):
    center = Instance(Point, ())
    r = Int

    view = View(
        VGroup(
            Item("center", style="custom"),
            Item("r"),
        )
    )

    @on_trait_change("r")
    def set_info(self):
        from math import pi
        self.info = "area:%f" % (pi*self.r**2)

```

❶在Shape类中定义info属性，❷在初始化方法中调用派生类的set_info()以修改info属性。

❸在Triangle类中使用Instance(Point, ())定义了表示三角形三个顶点坐标的属性：a、b和c。在Circle类中使用同样的方式定义了表

示圆心坐标的center属性，这些属性都是Point对象。Instance的第二个参数指定创建缺省对象时所用的参数，当没有第二个参数时，它所定义的属性的缺省值为None。这里用一个空元组表示与之对应的属性的缺省值是通过调用Point()得到的，即缺省为Point()创建的Point对象。

④如果Trait属性是Instance类型，并且它在视图中对应的编辑器为“custom”样式，则属性对象的视图将直接嵌入当前的视图中。因此在Triangle和Circle对象的编辑界面中将嵌入多个Point对象的编辑器。在IPython中运行下面的程序可以看到所创建的界面效果：

```
Triangle().configure_traits()
Circle().configure_traits()
```

接下来，使用上面的形状类制作最终的形状选择类ShapeSelector：

```
class ShapeSelector(HasTraits):
    select = Enum(*[cls.__name__ for cls in
Shape.__subclasses__()]) ①
    shape = Instance(Shape) ②

    view = View(
        VGroup(
            Item("select"),
            Item("shape", style="custom"), ③
            Item("object.shape.info", style="custom"), ④
            show_labels = False
```

```

        ),
        width = 350, height = 300, resizable = True
    )

    def __init__(self, **traits):
        super(ShapeSelector, self).__init__(**traits)
        self._select_changed()

    def _select_changed(self): ❸
        klass = [c for c in Shape.__subclasses__() if
c.__name__ == self.select][0]
        self.shape = klass()

```

❶ 下拉选择框所对应的select属性为枚举类型，为了让程序显得更自动化一些，这里不直接指定枚举类型的候选值，而是通过Shape的派生类名创建候选值列表。这样当添加其他的Shape派生类时，不需要修改这段代码。

❷ shape属性的类型是Shape，由于不需要创建缺省的Shape对象，因此不用指定Instance的第二个参数。❸ 当select属性发生变化时，在事件处理方法_select_changed()中创建select属性所对应的类的实例，并赋值给shape属性。❹ shape属性对应的编辑器是“custom”样式，因此它的编辑界面将作为组件嵌入ShapeSelector的界面中。并且它能根据当前的shape属性值，动态更新界面上的编辑器。也就是说，当shape属性是Triangle对象时将使用Triangle类的视图创建编辑器，而当shape属性是Circle对象时将使用Circle类的视图创建编辑器。

④通过`object.shape.info`可以为`shape`属性的`info`属性在界面中创建编辑器。当为某个`Trait`属性的属性创建编辑器时，注意需要在属性名之前添加“`object.`”。

读者也许会认为这种为属性的属性创建编辑器的做法有些混乱，一个比较简单的解决方法就是从`ShapeSelector`的视图中删除`object.shape.info`的`Item`对象，并分别给`Triangle`和`Circle`的视图添加显示`info`属性的编辑器：`Item("info", style="custom")`。这种做法的缺点是需要给每个从`Shape`派生的类的视图添加`info`属性的编辑器，而当我们不想显示`info`属性时，代码的修改量也会随着`Shape`的派生类的增加而增加。

还有一种使用多个视图对象的方法，它充分体现了MVC模式将模型和视图完全分离的优点。



`scipy2.traits.traitsui_component_multi_view`: 使用多个视图显示组件。

由于程序的改动不大，下面只介绍它和`traitsui_component.py`的不同之处：

```
class Shape(HasTraits):
    info = Str
    view_info = View(Item("info", style="custom",
show_label=False))
```

```
def __init__(self, **traits):
    super(Shape, self).__init__(**traits)
    self.set_info()
```

首先为Shape类添加一个view_info视图专门用于显示其info属性。这样Shape的派生类Triangle和Circle都具有两个视图：view和view_info。如果模型类有多个视图，将其嵌入其他视图中时需要指定使用哪个视图创建编辑器。因此ShapeSelector类的视图需要做如下修改：

```
view = View(
    VGroup(
        Item("select", show_label=False),
        VSplit( ❶
            Item("shape", style="custom",
editor=InstanceEditor(view="view")), ❷
            Item("shape", style="custom",
editor=InstanceEditor(view="view_info")),
            show_labels = False
        )
    ),
    width = 350, height = 300, resizable = True
)
```

❶为了和前面的例子有所区别，这里用一个垂直分隔容器将形状数据输入界面和显示形状信息的控件分隔开。❷shape属性的编辑器样式仍然为“custom”，但是为了指定编辑器所使用的视图，需要通过

editor参数传递一个InstanceEditor对象，而通过InstanceEditor对象的view参数可以指定创建界面时所使用的视图名。实际上，Instance类型的Trait属性缺省就是使用InstanceEditor作为“custom”样式的编辑器，因此前面的程序中都没有通过editor参数指定。当需要修改InstanceEditor对象的一些缺省值时，就需要手工创建它了。

下面总结一下本节的内容：

- 通过将Instance类型的Trait属性的编辑器样式指定为“custom”，可以实现界面的层层嵌套，即组件功能。
- 当模型类有多个视图对象时，通过InstanceEditor的view参数可以选择其中的某个视图来创建编辑此模型对象的控件。

TraitsUI的组件并不局限于界面上的某一块区域，我们可以在界面中的不同位置用不同的视图，为同一个模型对象创建多个不同的编辑器，因此使用TraitsUI创建的界面是非常灵活的。

7.5.3 自定义编辑器

Enthought的官方绘图库采用的是Chaco，不过如果读者对matplotlib更为熟悉，也可以在界面中使用matplotlib的绘图控件。为了实现这个目的，需要自己编写一个Trait编辑器，用它包装matplotlib的绘图控件。

由于TraitsUI库和matplotlib都支持wx和Qt界面库，因此下面的程序首先根据ETSConfig.toolkit选择载入matplotlib中对应的绘图控

件FigureCanvas和工具条Toolbar，并从traits对应的后台库中载入所有编辑器的父类Editor：

```
import matplotlib
from traits.api import Bool
from traitsui.api import toolkit
from traitsui.basic_editor_factory import BasicEditorFactory
from traits.etsconfig.api import ETSConfig

if ETSConfig.toolkit == "wx":
    # matplotlib采用WXAgg为后台，这样才能将绘图控件嵌入以wx为后台界面库的traitsUI窗口中
    import wx
    matplotlib.use("WXAgg")
    from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as FigureCanvas
    from matplotlib.backends.backend_wx import NavigationToolbar2Wx as Toolbar
    from traitsui.wx.editor import Editor

elif ETSConfig.toolkit == "qt4":
    matplotlib.use("Qt4Agg")
    from matplotlib.backends.backend_qt4agg import FigureCanvasQTAgg as FigureCanvas
    from matplotlib.backends.backend_qt4agg import NavigationToolbar2QT as Toolbar
```

```
from traitsui.qt4.editor import Editor
from pyface.qt import QtGui
```

对于每个界面库都需要编写从Editor类继承的包装matplotlib图表的编辑器类，下面是从traitsui.qt4.editor.Editor继承的编辑器类：

```
class _QtFigureEditor(Editor):
    scrollable = True

    def init(self, parent): ❶
        self.control = self._create_canvas(parent)
        self.set_tooltip()

    def update_editor(self):
        pass

    def _create_canvas(self, parent):

        panel = QtGui.QWidget()

        def mousemoved(event):
            if event.xdata is not None:
                x, y = event.xdata, event.ydata
                name = "Axes"
            else:
                x, y = event.x, event.y
```

```

name = "Figure"

        panel.info.setText("%s: %g, %g" % (name, x, y))

    panel.mousemoved = mousemoved
    vbox = QtGui.QVBoxLayout()
    panel.setLayout(vbox)

    mpl_control = FigureCanvas(self.value) ❷
    vbox.addWidget(mpl_control)
    if hasattr(self.value, "canvas_events"):
        for event_name, callback in
self.value.canvas_events:
            mpl_control.mpl_connect(event_name,
callback)

            mpl_control.mpl_connect("motion_notify_event",
mousemoved)

    if self.factory.toolbar: ❸
        toolbar = Toolbar(mpl_control, panel)
        vbox.addWidget(toolbar)

    panel.info = QtGui.QLabel(panel)
    vbox.addWidget(panel.info)

    return panel

```


❶在初始化编辑器控件时会调用`init()`方法，在该方法中调用`_create_canvas()`以创建控件。❷该编辑器对象的`value`属性保存对应的模型对象，即`matplotlib`中的`Figure`对象。在创建编辑器时，可以根据模型对象属性执行初始化工作，这里判断模型对象是否有`canvas_events`属性。如果有，就对其中定义的事件进行绑定。

❸在创建编辑器时传递的参数可以通过`factory`属性获得，这里根据其中的`toolbar`属性判断是否创建工具栏。

最后还需要编写一个编辑器工厂类`MPLFigureEditor`，它从`BasicEditorFactory`类继承：

```
class MPLFigureEditor(BasicEditorFactory):  
    """  
    相当于traits.ui中的EditorFactory，它返回真正创建控件的类  
    """  
    if ETSToolkit == "wx":  
        klass = _WxFigureEditor  
    elif ETSToolkit == "qt4":  
        klass = _QtFigureEditor ❶  
  
    toolbar = Bool(True) ❷
```

❶类属性`klass`为编辑器类，这里根据当前的界面库选择编辑器类。❷最后定义工厂类中的`Traits`属性`toolbar`，它的缺省值为`True`。

下面是使用`MPLFigureEditor`将`matplotlib`的图表嵌入`TraitUI`界面中的例子：

```

import numpy as np

from matplotlib.figure import Figure

from scipy2.traits import MPLFigureEditor


class SinWave(HasTraits):

    figure = Instance(Figure, ())

    view = View(
        Item("figure", editor=MPLFigureEditor(toolbar=True),
show_label=False),
        width = 400,
        height = 300,
        resizable = True)

    def __init__(self, **kw):
        super(SinWave, self).__init__(**kw)
        self.figure.canvas_events = [
            ("button_press_event",
self.figure_button_pressed)
        ]

        axes = self.figure.add_subplot(111)
        t = np.linspace(0, 2*np.pi, 200)
        axes.plot(np.sin(t))

    def figure_button_pressed(self, event):
        print event.xdata, event.ydata

```

```
model = SinWave()  
  
model.edit_traits()
```

7.6 函数曲线绘制工具



与本节内容对应的Notebook为：07-traits/traits-600-example.ipynb。

作为本章的最后一节，让我们用学到的内容编写一个绘制函数曲线的小程序，界面如图7-14所示。界面上方是显示曲线的图表，左下方为代码编辑器，右下方为显示数据点的表格。

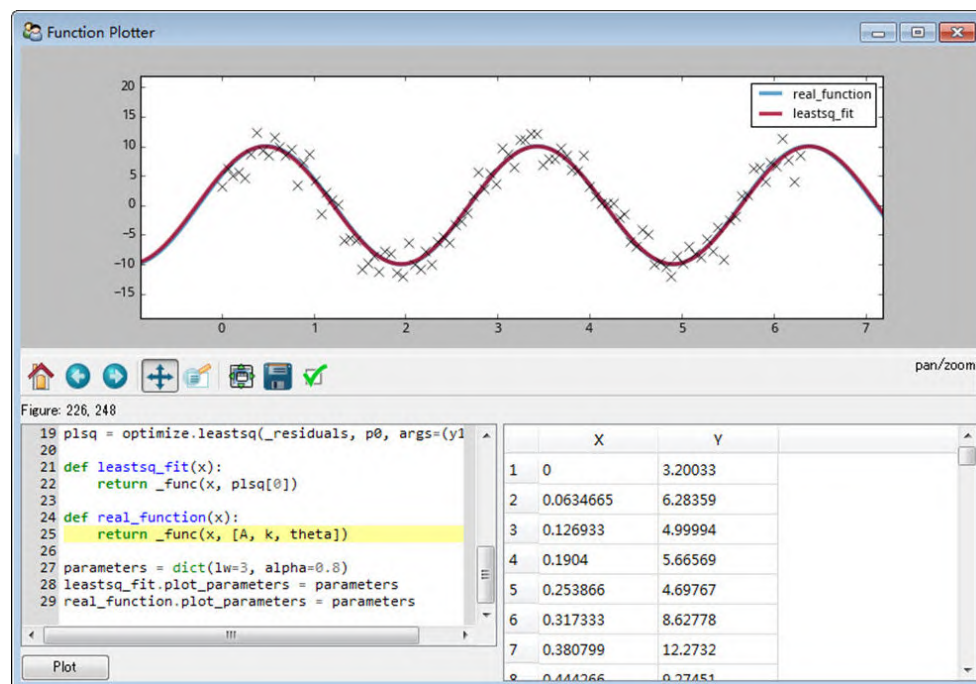


图7-14 函数曲线绘制工具的界面



`scipy2.traits.traitsui_function_plotter` : 采用 TraitsUI 编写的函数曲线绘制工具。

本函数曲线绘制工具有如下功能：

- 在图表上单击鼠标左键添加点，单击鼠标右键删除最后添加的点。也可以通过表格添加点或者修改点的坐标。
- 在运行代码之前，将表格中的点转换为形状为 $(N, 2)$ 的二维数组，该数组可以在用户程序中使用 `points` 访问。
- 程序运行之后，如果运行环境中名为 `points`、形状为 $(N, 2)$ 的二维数组，将该数组的内容显示在左侧的表格中，在图表中这些数据点使用黑色的叉点表示。
- 将运行环境中所有返回值为数组的单参数函数显示为曲线，如果函数名以下划线开头，则忽略该函数。
- 如果曲线函数的 `plot_parameters` 属性为字典，该属性将作为关键字参数传递给 `plot()` 函数。
- 如果图表的 X 轴范围发生变化，自动调用曲线对应的函数并更新曲线。

图 7-14 所示的界面由三个控件组成。❶ `figure` 属性是一个 `matplotlib` 的 `Figure` 对象，它对应的编辑器为上节介绍的 `MPLFigureEditor`。❷ `code` 属性为 `Code` 类型，它从 `Str` 继承，其缺省的编辑器为带高亮显示的代码编辑器。❸ `points` 属性是一个 `Point` 对象的列表，其对应的编辑器为 `point_table_editor`。



Code 对应的编辑器代码存在 BUG，请读者将
patches\pygments_highlighter.py 复制到 site-
packages\pyface\ui\qt4\code_editor 下以覆盖原有的文件。

```
class FunctionPlotter(HasTraits):  
    figure = Instance(Figure, ()) ❶  
    code = Code() ❷  
    points = List(Instance(Point), []) ❸  
    draw_button = Button("Plot")  
  
    view = View(  
        VSplit(  
            Item("figure",  
editor=MPLFigureEditor(toolbar=True), show_label=False),  
            HSplit(  
                VGroup(  
                    Item("code", style="custom"),  
                    HGroup(  
                        Item("draw_button",  
show_label=False),  
                    ),  
                    show_labels=False  
                ),  
                Item("points", editor=point_table_editor,  
show_label=False)
```

```

        )
    ),
    width=800, height=600, title="Function Plotter",
resizable=True
)

```

下面是Point类和point_table_editor的定义。TableEditor为表格数据的编辑器类，表格的每一列与一个ObjectColumn对象对应。row_factory为创建新行时调用的对象。

```

class Point(HasTraits):
    x = Float()
    y = Float()

point_table_editor = TableEditor(
    columns=[ObjectColumn(name='x', width=100, format="%g"),
             ObjectColumn(name='y', width=100,
format="%g")],
    editable=True,
    sortable=False,
    sort_model=False,
    auto_size=False,
    row_factory=Point
)

```

下面是FunctionPlotter类的初始化函数。❶在调用__init__()时，Figure对象已经创建，但是对应的后台界面库中的绘图控件

canvas 尚未创建，因此这里无法使用 `self.figure.canvas.mpl_connect()` 设置事件响应函数。在上节介绍的 `MPLFigureEditor` 编辑器中，在创建绘图控件时会绑定 `Figure` 对象的 `canvas_events` 属性中定义的事件。❷创建子图对象 `axe` 之后，调用 `callbacks.connect()` 以绑定子图的 `xlim_changed` 事件，该事件在子图的 X 轴显示范围发生变化时触发。

```
def __init__(self, **kw):
    super(FunctionPlotter, self).__init__(**kw)
    self.figure.canvas_events = [ ❶
        ("button_press_event", self.memory_location),
        ("button_release_event", self.update_location)
    ]
    self.button_press_status = None #保存鼠标按键按下时的状态
    self.lines = [] #保存所有曲线
    self.functions = [] #保存所有的曲线函数
    self.env = {} #代码的执行环境

    self.axe = self.figure.add_subplot(1, 1, 1)
        self.axe.callbacks.connect('xlim_changed',
self.update_data) ❷
        self.axe.set_xlim(0, 1)
        self.axe.set_ylim(0, 1)
        self.points_line, = self.axe.plot([], [], "kx",
ms=8, zorder=1000) #数据点
```

在图表中使用鼠标进行平移和缩放时，也会触发鼠标按键按下和释放的响应函数。为了区分这些操作与鼠标按键的单击操作，在 `memory_location()` 中记录下鼠标按键按下时的状态，并在 `update_location()` 中与鼠标释放时的状态进行比较，如果满足：❶鼠标按键按下的时间少于0.5秒，❷鼠标的移动距离小于4个像素，就认为是鼠标单击操作。

❸当鼠标左键释放时，创建一个Point对象，其x和y属性被设置为鼠标释放时在子图中的坐标，并将该对象添加进points列表。❹若按下右键，则调用 `points.pop()` 删除列表中的最后一个元素。

```
def memory_location(self, evt):
    if evt.button in (1, 3):
        self.button_press_status = time.clock(), evt.x,
evt.y
    else:
        self.button_press_status = None

def update_location(self, evt):
    if evt.button in (1, 3) and self.button_press_status
is not None:
        last_clock, last_x, last_y =
self.button_press_status
        if time.clock() - last_clock > 0.5: >❶
            return
        if ((evt.x - last_x) ** 2 + (evt.y - last_y) **
2) ** 0.5 > 4: >❷
```



```

        return

    if evt.button == 1:
        if evt.xdata is not None and evt.ydata is not
None:

            point = Point(x=evt.xdata, y=evt.ydata) ❸

            self.points.append(point)

    elif evt.button == 3:
        if self.points:
            self.points.pop() ❹

```

当points列表本身或者其中的元素发生增减时，将调用 `_points_changed()`，其new参数为新添加进列表的元素。❶当通过表格编辑坐标点的位置时，Point对象的x或y属性将发生变化，为了捕捉到这些变化并更新图表中的坐标点，需要对新添加进points列表的对象进行事件绑定。

❷在 `update_points()` 中，从points属性创建二维数组，并调用 `points_line.set_data()` 更新图表中的坐标点，调用 ❸ `update_figure()` 重新绘制图表。由于这一系列的触发事件可能在图表的绘图控件创建之前发生，因此❹在调用 `canvas.draw_idle()` 重绘制图表之前需要判断canvas是否为None。

```

@on_trait_change("points[]")
def _points_changed(self, obj, name, new):
    for point in new:
        point.on_trait_change(self.update_points,
name="x, y") ❶

```

```

        self.update_points()

    def update_points(self): ❷
        arr = np.array([(point.x, point.y) for point in
self.points])
        if arr.shape[0] > 0:
            self.points_line.set_data(arr[:, 0], arr[:, 1])
        else:
self.points_line.set_data([], [])
        self.update_figure()

    def update_figure(self): ❸
        if self.figure.canvas is not None: ❹
            self.figure.canvas.draw_idle()

```

当Axes对象的X轴显示范围发生变化时，调用update_data()重新计算位于显示范围之内的曲线数据：

```

def update_data(self, axe):
    xmin, xmax = axe.get_xlim()
    x = np.linspace(xmin, xmax, 500)
    for line, func in zip(self.lines, self.functions):
        y = func(x)
        line.set_data(x, y)
    self.update_figure()

```

最后，当Plot按钮按下时_draw_button_fired()会被调用，在其中调用plot_lines()运行代码编辑器中的程序，并绘制函数曲线。

❶ 调用 `axe.get_xlim()` 获得子图的 X 轴显示范围，并调用 `linspace()` 创建等分此区间的数组 `x`。❷ 创建代码的运行环境 `env`，它是一个字典，其 `points` 键对应由 `points` 列表转换而来的二维数组，并调用 `exec` 运行 `code` 中保存的代码。

对一些属性进行初始化操作之后，❸ 对 `env` 中的所有键-值对进行循环，找到其中符合条件的函数并调用，然后把结果保存到 `results` 列表中。❹ 如果曲线函数有 `plot_parameters` 属性，则将之作为绘图参数传递给 `plot()`。

❺ 最后将执行环境中的 `points` 数组转换为 `Point` 对象的列表。

```
def _draw_button_fired(self):
    self.plot_lines()

def plot_lines(self):
    xmin, xmax = self.axe.get_xlim() ❶
    x = np.linspace(xmin, xmax, 500)
    self.env = {"points": np.array([(point.x, point.y) for point
in self.points])} ❷
    exec self.code in self.env

    results = []
    for line in self.lines:
        line.remove()

        self.axe.set_color_cycle(None) #重置颜色循环
    self.functions = []
    self.lines = []
```

```

        for name, value in self.env.items(): ❸
            if name.startswith("_"): #忽略以_开头的名字
                continue
            if callable(value):
                try:
                    y = value(x)
                    if y.shape != x.shape: #输出数组应该与输入
数组的形状一致
                        raise ValueError("the return shape
is not the same as x")
                except Exception as ex:
                    import traceback
                    print "failed when call function
{} \n".format(name)
                    traceback.print_exc()
                    continue

            results.append((name, y))
            self.functions.append(value)

        for (name, y), function in zip(results,
self.functions):
            #如果函数有plot_parameters属性,则用其作为plot()的参数
            kw = getattr(function, "plot_parameters", {}) ❹
            label = kw.get("label", name)
            line, = self.axe.plot(x, y, label=label, **kw)
            self.lines.append(line)

```

```
points = self.env.get("points", None) ❸  
if points is not None:  
    self.points = [Point(x=x, y=y) for x, y in  
np.asarray(points).tolist()]  
  
self.axe.legend()  
self.update_figure()
```


第8章 TVTK与Mayavi-数据的三维可视化

VTK是一套功能十分强大的三维数据可视化库，它使用C++编写，其中包含了近千个类。它在Python下有标准的扩展库，不过由于其Python扩展库的API和C++的API相同，不能体现出Python作为动态语言的优势；因此Enthought公司开发了一套名为TVTK的扩展库来对VTK进行包装，提供了Python风格的API，并支持Trait属性和NumPy数组。本章以TVTK的API为例介绍如何在Python中使用VTK进行数据的三维可视化。

由于TVTK库十分庞大，为了方便用户查询文档，TVTK库提供了一个显示TVTK文档的工具。可以通过下面的语句运行它：

```
from tvtk.tools import tvtk_doc
tvtk_doc.main()
```



scipy2.tvtk.tvtk_class_doc: 更方便的TVTK文档查询工具。

TVTK库提供的工具并不太好用，本书为读者提供了一个更方便的TVTK文档查询工具，其界面如图8-1所示。

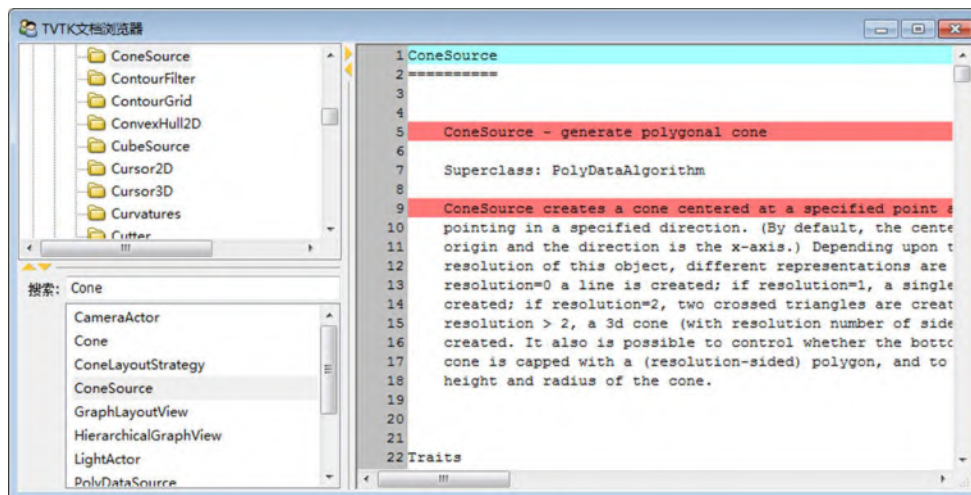


图8-1 TVTK文档浏览器（使用wx库时的界面截图）

第一次运行此文档工具时，它将对TVTK库中所有的类进行扫描，并将类的继承关系和文档全部保存在tvtk_classes.cache中，这个过程可能需要等待较长的时间。界面的左上部分使用一个树状控件显示TVTK中各个类之间的继承关系。在中间的文本框中输入搜索文本，其下方的列表框中会实时显示搜索结果。输入全小写字母进行忽略大小写的搜索，而输入带大写字母的文本则进行精确搜索。

8.1 VTK的流水线(Pipeline)



与本节内容对应的 Notebook 为：08-tvtk_mayavi/tvtk_mayavi-200-pipeline.ipynb。

VTK是一个十分复杂的系统，为了方便用户使用，它使用流水线技术将VTK中的各个对象串联起来。每个对象只需要实现相对简单的任

务，整个流水线则能够根据用户的需求实现十分复杂的数据可视化处理。

8.1.1 显示圆锥

作为第一例子，让我们首先看一个显示圆锥的小程序，它的运行效果如图8-2所示。

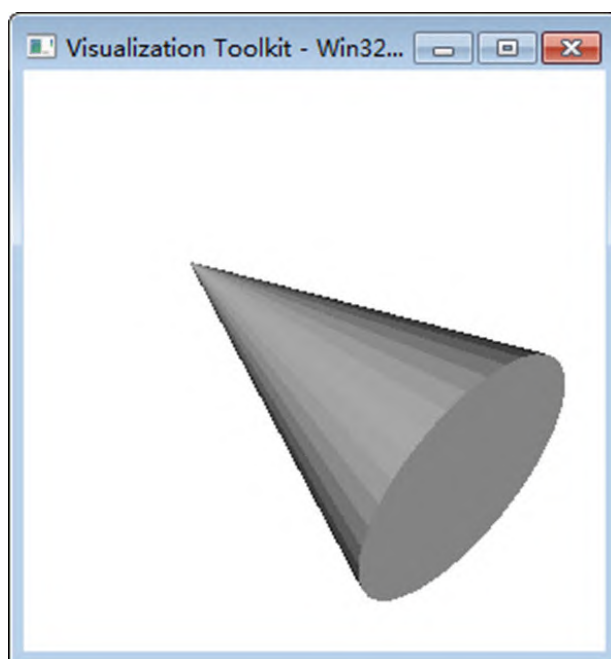


图8-2 使用TVTK绘制简单的圆锥

由于在IPython Notebook中显示VTK的窗口之后将无法关闭，因此下面使用`%%python`魔法命令在新的Python进程中运行显示圆锥的程序：

```
%%python
```

```

#coding=utf-8

from tvtk.api import tvtk ❶

# 创建一个圆锥数据源，并且同时设置其高度、底面半径和底面圆的分辨率(用36
边形近似)

cs = tvtk.ConeSource(height=3.0, radius=1.0, resolution=36)
❷

# 使用PolyDataMapper将数据转换为图形数据
m = tvtk.PolyDataMapper(input_connection=cs.output_port) ❸
# 创建一个Actor
a = tvtk.Actor(mapper=m) ❹
# 创建一个Renderer，将Actor添加进去
ren = tvtk.Renderer(background=(1, 1, 1)) ❺
ren.add_actor(a)
# 创建一个RenderWindow(窗口)，将Renderer添加进去
rw = tvtk.RenderWindow(size=(300,300)) ❻
rw.add_renderer(ren)
# 创建一个RenderWindowInteractor(窗口的交互工具)
rwi = tvtk.RenderWindowInteractor(render_window=rw) ❼
# 开启交互
rwi.initialize()
rwi.start()

```

❶首先载入tvtk对象，它帮助我们创建TVTK库中的各种对象。❷创建了一个ConeSource对象，它是计算圆锥形状的数据源对象。在TVTK中，所有类都从HasTraits继承，因此可以在创建对象的同时，使

用关键字参数直接设置Trait属性的值。在这个例子中，同时设置了圆锥的高度、底面半径和底面圆的边数等属性。

事实上，我们载入的tvtk并不是一个模块，而是某个类的实例。之所以如此设计，是因为VTK库有近千个类，而TVTK对所有这些类都进行了包装。如果一次性载入这么多类，会极大地影响库的载入速度。

我们载入的tvtk虽然是某个实例对象，但是用起来就和模块一样：

- 通过它可以使用所有的TVTK类。
- 它不需要载入近千个TVTK类就能支持类名的自动补全。
- 只有在真正使用时，TVTK类才会被载入。

所有对VTK进行包装的类全部保存在tvtk_classes.zip文件中，而tvtk对象的类则在此压缩文件里的tvtk_helper.py中定义。对于TVTK中的每个类，tvtk对象都有一个同名的属性与之对应。

下面查看ConeSource对象的所有Traits属性名，并显示height、radius和resolution等属性的值：

```

from tvtk.api import tvtk
cs = tvtk.ConeSource(height=3.0, radius=1.0, resolution=36)
m = tvtk.PolyDataMapper(input_connection=cs.output_port)
a = tvtk.Actor(mapper=m)
ren = tvtk.Renderer(background=(1, 1, 1))
ren.add_actor(a)

cs.trait_names()
['number_of_output_ports',
 'abort_execute_',
 'class_name',
 'executive',
 ...
cs.height cs.radius cs.resolution
-----
3.0      1.0      36

```

为了将原始数据转换为屏幕上的一幅图像，需要经过许多处理步骤。这些步骤由众多的VTK对象分步实现，就好像生产线上加工零件一样，每位工人都负责一部分工作，整条生产线就能将原材料制作成产品。在VTK中，这种在各个对象之间协调完成工作的过程被称作流水线(Pipeline)。

原始数据被加工成图像要经过两条流水线：

- 可视化流水线(Visualization Pipeline)：它的工作是将原始数据加工成图形数据。一般来说，我们需要进行可视化展示的数据本身并不是图形数据，例如可能是某个零件内部各个部分的温度，或是流体中各个坐标点上的速度等。
- 图形流水线(Graphics Pipeline)：它的工作是将图形数据加工为我们所看到的图像。可视化流水线所产生的图形数据通常是三维空间的数据，图形流水线将这些三维数据加工成能在二维屏幕上显示的图像。

③映射器 (Mapper) 是可视化流水线的终点、图形流水线的起点，它的各种派生类能将众多的数据映射为图形数据以供图形流水线加工。在本例中，ConeSource对象输出一个描述圆锥的顶点和面的PolyData对象，然后PolyData对象通过PolyDataMapper映射器转换为图形数据。因此在本例中，可视化流水线由ConeSource和PolyDataMapper对象组成。

可视化流水线中的对象经由input_connection和output_port属性连接起来。在本例中，ConeSource对象产生一个表示圆锥的PolyData对象，并转交给PolyDataMapper对象进行处理。可以通过output或input属性查看在流水线中实际传递的PolyData对象：

```
print type(cs.output), cs.output is m.input
<class 'tvtk.tvtk_classes.poly_data.PolyData'> True
```

然后图形数据再依次通过Actor、Renderer最终在RenderWindow中显示出来，这一部分就是图形流水线。④Actor对象代表场景中的一个实体，它的mapper属性是表示图形数据的PolyDataMapper对象，Actor对象还有许多属性可以控制实体的位置、方向、大小等。

```
print a.mapper is m
print a.scale # Actor对象的scale属性表示各个轴的缩放比例
True
[ 1.  1.  1.]
```

⑤Renderer对象表示三维场景，它可以包含多个Actor对象，这些Actor对象都保存在actors列表属性中。在本例中，它只包含一个显示圆锥的Actor对象：

```
ren.actors  
['<tvtk.tvtk_classes.actor.Actor object at 0x0D7C7BD0>']
```

❹RenderWindow对象表示包含场景的窗口，它可以同时包含多个场景。在本例中，它只有一个Renderer对象。

❺RenderWindowInteractor对象为图形窗口提供一些用户交互功能，例如平移、旋转和缩放。这些交互式操作并不改变场景中的各个实体(Actor对象)，也不改变图形数据的属性，它们只是修改场景中照相机(Camera)的设置，从不同的角度和距离观察场景中的实体。

8.1.2 用ivtk观察流水线

为了方便对流水线进行观察和操作，本书在scipy2.tvtk.tvtkhelp模块中提供了ivtk_scene()和event_loop()两个函数。使用它们可以交互式地对各种TVTK对象的属性进行编辑，下面是使用这两个函数显示圆锥的程序，运行画面如图8-3所示。

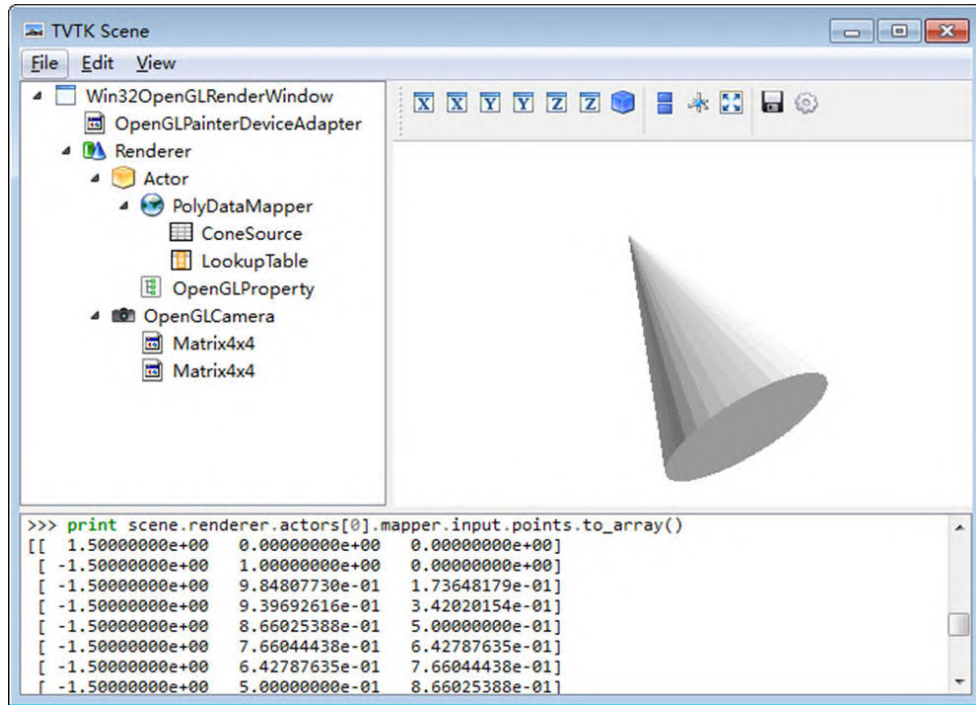


图8-3 带流水线浏览器和Python命令行的ivtk界面

```
from tvtk.api import tvtk

from scipy2.tvtk.tvtkhelp import ivtk_scene, event_loop

cs = tvtk.ConeSource(height=3.0, radius=1.0, resolution=36)
m = tvtk.PolyDataMapper(input_connection=cs.output_port)
a = tvtk.Actor(mapper=m)

window = ivtk_scene([a]) ❶
window.scene.isometric_view()
event_loop() ❷
```

❶将Actor对象列表传递给ivtk_scene()，创建并显示一个包含所有Actor对象的TVTK Scene窗口。❷调用event_loop()开始界面消息循

环，若在Notebook中通过%gui命令启动了界面消息循环，则可以省略该行。下面看看TVTK Scene窗口的各个组成部分：

- 场景：用于显示可视化的结果，这里只显示了一个圆锥。
- 场景工具条：位于场景的上方，主要提供了各种视角、全屏显示、保存图像等功能。
- 流水线浏览器：场景左边是一个表示流水线的树状控件。从子节点ConeSource开始逐步向上层直到根节点RenderWindow，是显示圆锥的整个流水线。
- Python命令行：界面下方提供了一个Python命令行，方便用户直接输入命令来操作各个对象。例如图中显示了通过场景对象scene获取ConeSource对象所输出的PolyData对象的points属性，即构成圆锥图形的各个顶点的三维坐标。

流水线浏览器中显示各个对象的类都从HasTraits继承，因此它们可以提供一个用户界面以交互式地修改其Trait属性。图8-4是双击流水线中的ConeSource对象之后弹出的属性编辑界面。通过此界面可以直接修改height、radius、resolution等属性，并且修改之后场景中的圆锥会根据最新的属性值立即更新显示。

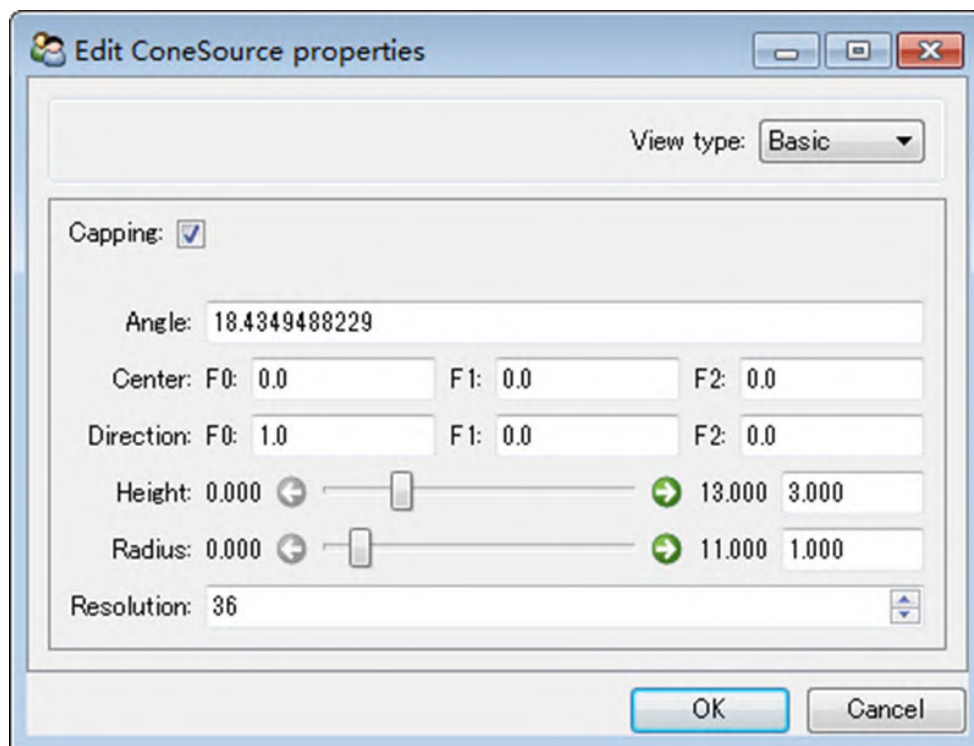


图8-4 编辑ConeSource对象的属性的对话框

1. 照相机

在ivtk的窗口左侧的流水线浏览器中可以找到场景中的照相机对象OpenGLCamera，双击它会弹出如图8-5所示的编辑照相机对象属性的窗口。

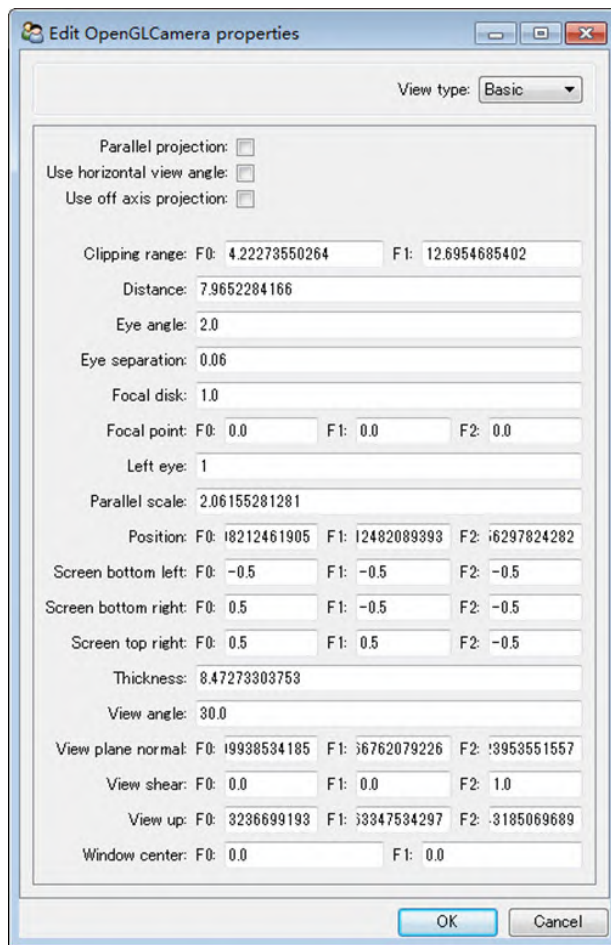


图8-5 编辑照相机属性的对话框

也可以使用下面的程序从窗口对象window获得照相机对象，然后查看或修改它的属性：

```
camera = window.scene.renderer.active_camera
print camera.clipping_range
camera.view_up = 0, 1, 0
camera.edit_traits() # 显示编辑照相机属性的窗口
[ 4.2227355 12.69546854]
```

下面列出照相机对象的一些常用属性：

- `clipping_range`: 它有两个元素，分别表示照相机到近远两个裁剪平面的距离。在这两个平面之外的对象将不会显示。
- `position`: 照相机在三维空间中的坐标。
- `focal_point`: 照相机所聚焦的焦点坐标。
- `view_up`: 照相机的上方向矢量。
- `parallel_projection`: `True`表示采用平行透视，即在三维场景中平行的直线在屏幕上也是平行的。

这些属性虽然可以完全控制照相机的位置和方向，但是实际操作起来并不方便。如果已经将照相机的焦点固定在某个位置，可以调用照相机对象的如下两个方法，在以焦点为原点的球面坐标系中对照相机进行操作。它们保持照相机的`view_up`属性不变。

- `azimuth(angle)`: 沿着纬度线旋转指定角度，即水平旋转，改变其经度。
- `elevation(angle)`: 沿着经度线方向旋转指定角度，即垂直旋转，改变其纬度。

2. 光源

在ivtk窗口中，单击场景上方的工具栏中的最后一个齿轮形状的图标，将打开如图8-6所示的编辑场景和光源的对话框。在此对话框中可以添加和删除光源以及修改它们的一些属性。

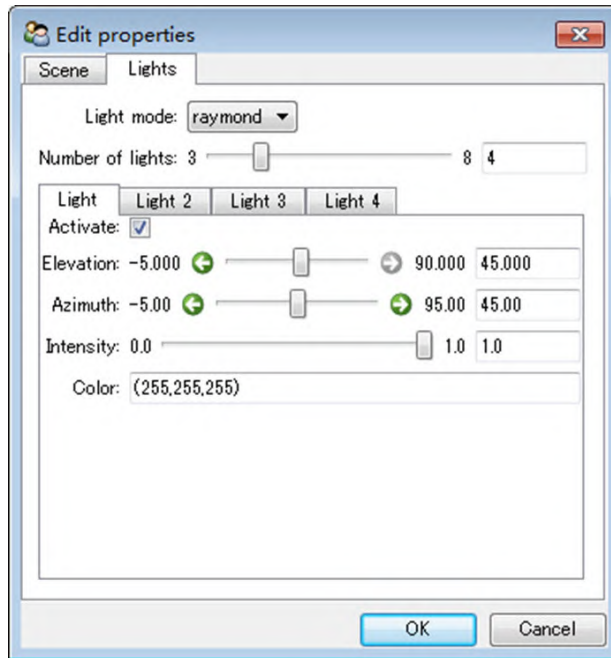


图8-6 设置场景和光源的对话框

场景中的光源可以通过Renderer对象的lights属性获得，它是一个光源对象的列表。Renderer对象还有add_light()和remove_light()等方法用于添加或删除光源对象。

```
lights = window.scene.renderer.lights
lights[0].edit_traits() # 显示编辑光源属性的窗口
```

下面的程序在照相机所在处添加一个红色的光源，它的照射方向和照相机的方向相同，朝向focal_point点。如果设置光源对象的positional属性为True，它将变成一个探照灯光源，这时照射方向有效。并且可以通过cone_angle属性设置探照灯的光锥角度，如果光锥为180度，它是无方向光源。

```
camera = window.scene.renderer.active_camera
light = tvtk.Light(color=(1,0,0))
light.position=camera.position
```

```
light.focal_point=camera.focal_point  
window.scene.renderer.add_light(light)
```

3. 实体

Actor对象表示场景中的实体，在圆锥的流水线浏览器中，可以看到一个表示圆锥的Actor对象，双击它会打开如图8-7所示的对话框。

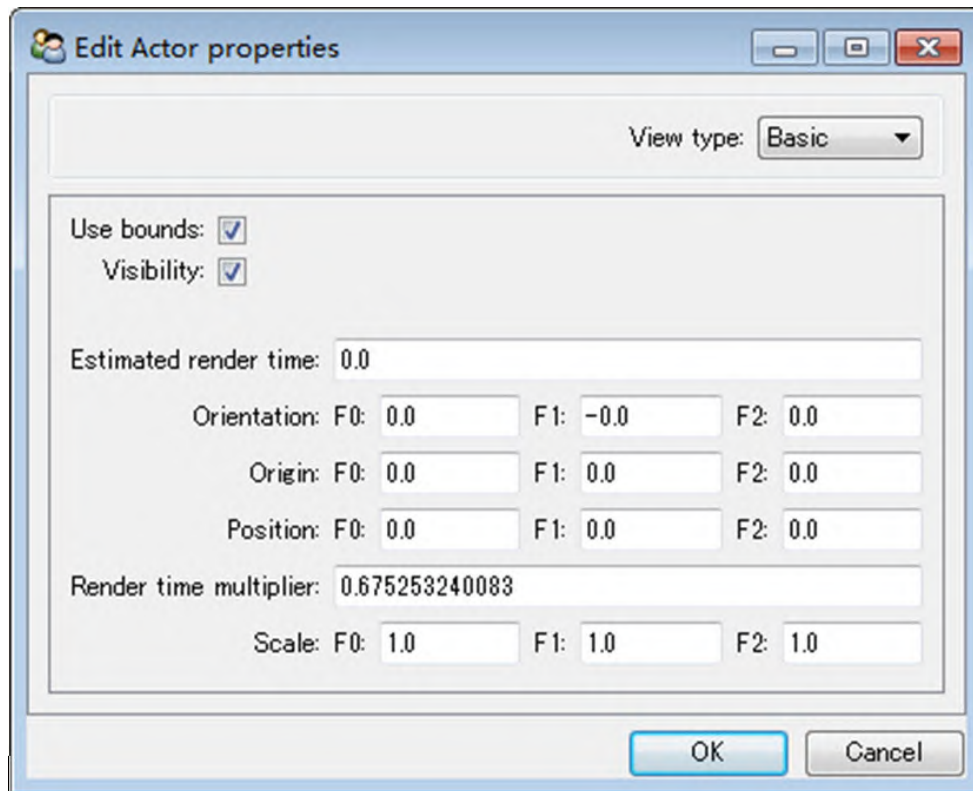


图8-7 Actor对象的编辑对话框

也可以使用下面的程序打开此对话框：

```
a.edit_traits() # a是表示圆锥的Actor对象  
window.scene.renderer.actors[0].edit_traits()
```

在此对话框中可以编辑Actor对象的origin、position、orientation和scale等属性，修改场景中实体的位置、方向以及大小。这4个属性通过一系列复杂的计算得到一个 4×4 的三维空间的变换矩阵。变换步骤如下：

(1) 以origin为中心，使用scale对物体在三个轴上进行缩放。

(2) 以origin为中心，使用rotate对物体在三个轴上进行旋转，旋转的顺序是Y轴→X轴→Z轴。

(3) 将物体放到position处。

我们通过下面的例子理解坐标变换的步骤。首先运行下面的程序，在场景中添加一个坐标轴：

```
axe = tvtk.AxesActor(total_length=(3,3,3)) # 在场景中添加坐标轴
window.scene.add_actor( axe )
```

可以看到屏幕的横轴方向是X轴，纵轴方向是Y轴，而从屏幕里往外是Z轴方向。整个圆锥的长度为3，它的底面在 $X=-1.5$ 的平面之上。双击流水线对话框中的表示圆锥的Actor，打开编辑其属性的对话框。

我们将origin修改为 $(-1.5, 0, 0)$ ，这样将以圆锥的底面圆心为中心进行缩放和旋转。依次按照图8-8的顺序修改各个属性的值。对话框中的“F0”、“F1”和“F2”等标签分别表示X、Y和Z轴的分量。

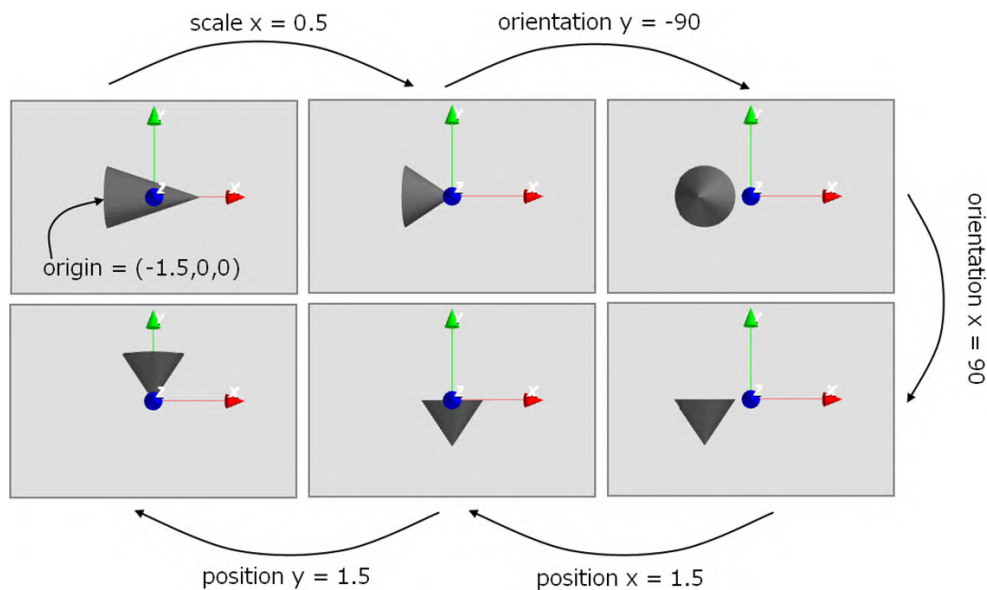


图8-8 依次修改圆锥的scale、orientation和position属性

请读者仔细观察每两幅图之间的变化，分析并理解前述坐标变换步骤。旋转的正方向按照右手法则决定：右手握拳，并伸出大拇指让它指向某个轴的正方向，则其余4指的方向为绕此轴旋转的正方向。

Actor对象的property属性是一个OpenGLProperty对象，它包含了对实体进行着色时所使用的各种配置，例如color属性是实体的颜色，opacity属性是实体的不透明度。输入下面的语句可以打开编辑这些属性的对话框：

```
a.property.edit_traits() # a是表示圆锥的Actor对象
```

由于OpenGLProperty对象的属性太多，这里不一一进行介绍。在后面的实例中用到时再对其进行说明。

8.2 数据集



与 本 节 内 容 对 应 的 Notebook 为 : 08-tvtk_mayavi/tvtk_mayavi-300-dataset.ipynb。

数据可视化的第一步是用合适的数据结构表示数据，VTK提供了多种表示不同种类数据的数据集(Dataset)。数据集包括点(Point)和数据(Data)两部分。点之间可以是连接的或非连接的，多个相关的点组成单元(Cell)，而点之间的连接可以是显式或隐式的。数据可以是标量或矢量，数据可以属于点或单元。下面让我们通过一些实例逐步理解数据集的构造。

为了帮助读者更形象地了解数据集的结构，我们使用Mayavi将数据集的结构绘制成三维图。请读者在学习的过程中运行这些程序，以加深对数据集的理解。在学习Mayavi时，也可以将这些程序作为实例，了解Mayavi的一些高级用法。

8.2.1 ImageData

最容易理解的数据集是ImageData，它是表示二维或三维图像的数据结构。可以简单地将其理解为二维或三维数组。数组中存放的是数据，由于点位于正交且等间距的网格之上，因此不需要给出点的坐标，而点之间的连接关系也由它们在数组中的位置决定，因此连接也是隐式的。

下面的程序创建了一个ImageData对象，并且设置了它的spacing、origin和dimensions属性：


```
img = tvtk.ImageData(spacing=(0.1,0.1,0.1), origin=(0.1,0.2,0.3), dimensions=(3,4,5))
```

origin属性为三维网格数据的起点坐标，spacing属性为三维网格在X、Y和Z轴上的间距，dimensions属性为X、Y和Z轴上的网格数。img.get_point(n)可以获得网格中第n个点的坐标值，n为点的序号，起始点的序号为0，序号依次沿着X、Y和Z轴递增。下面的程序输出前6个点的坐标：

```
for n in range(6):
    print "%.1f, %.1f, %.1f" % img.get_point(n)
0.1, 0.2, 0.3
0.2, 0.2, 0.3
0.3, 0.2, 0.3
0.1, 0.3, 0.3
0.2, 0.3, 0.3
0.3, 0.3, 0.3
```

与每个点对应的数据都保存在point_data属性中，它是一个PointData对象：

```
img.point_data
<tvtk.tvtk_classes.point_data.PointData at 0xa187780>
```

PointData对象可以保存多组数据，它的scalars属性是VTK库中的一个数组对象，用来保存与每个点对应的标量值。当将一个NumPy数组赋值给它时，TVTK将自动创建VTK中相应的数组对象，并保存NumPy

数组的内容。例如下面的程序运行之后，scalars属性从None变成一个DoubleArray数组。程序为每个点添加了一个与其序号相同的标量值：

```
print img.point_data.scalars # 没有数据
img.point_data.scalars = np.arange(0.0,
img.number_of_points)
print type(img.point_data.scalars)
img.point_data.scalars
None
<class 'tvtk.tvtk_classes.double_array.DoubleArray'>
[0.0, ..., 59.0], length = 60
```

DoubleArray数组只能以整数为下标，不支持切片以及其他高级下标运算。可以通过它的to_array()方法获得与其共享数据存储空间的NumPy数组，通过此NumPy数组可以进行更高级的数据存取操作：

```
a = img.point_data.scalars.to_array()
print a
a[:2] = 10, 11
print img.point_data.scalars[0], img.point_data.scalars[1]
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11.
12. 13. 14.
15. 16. 17. 18. 19. 20. 21. 22. 23. 24. 25. 26.
27. 28. 29.
30. 31. 32. 33. 34. 35. 36. 37. 38. 39. 40. 41.
42. 43. 44.
45. 46. 47. 48. 49. 50. 51. 52. 53. 54. 55. 56.]
```

```
57.  58.  59.]  
10.0 11.0
```

VTK的数组对象有许多属性，可以用`print_traits()`方法查看其各个属性的值，例如`number_of_tuples`属性表示数组的长度：

```
img.point_data.scalars.number_of_tuples  
60
```

每个VTK数组都有一个`name`属性用于保存其名字，下面的语句将数组的名字设置为`'scalars'`：

```
img.point_data.scalars.name = 'scalars'
```

`PointData`对象可以保存多个数组，它所包含的数组的个数可以通过`number_of_arrays`属性获得，可以通过`add_array()`方法添加新的数组对象，`remove_array()`方法用于删除数组对象，`get_array()`和`get_array_name()`分别用于获得数组对象及其名字。下面演示这些方法的使用。首先创建一个TVTK的数组对象，并调用其`from_array()`方法通过NumPy数组设置其内容，数组的长度为`ImageData`对象的点数：

```
data = tvtk.DoubleArray() # 创建一个空的DoubleArray数组  
data.from_array(np.zeros(img.number_of_points))
```

接下来将TVTK数组对象的名字设置为`"zerodata"`：

```
data.name = "zerodata"
```

然后调用`PointData`对象的`add_array()`方法，将所创建的数组添加进`PointData`对象。当`PointData`对象已经有相同名字的数组时，将

会覆盖原来的数组。add_array() 方法的返回值是新数组的序号，可以通过此序号获得数组或数组名：

```
print img.point_data.add_array(data)
print repr(img.point_data.get_array(1)) # 获得第1个数组
print img.point_data.get_array_name(1) # 获得第1个数组的名字
print repr(img.point_data.get_array(0)) # 获得第0个数组
print img.point_data.get_array_name(0) # 获得第0个数组的名字
1
[0.0, ..., 0.0], length = 60
zerodata
[10.0, ..., 59.0], length = 60
scalars
```

最后，remove_array() 方法通过数组名删除数组：

```
img.point_data.remove_array("zerodata") # 删除名为"zerodata"的数组
img.point_data.number_of_arrays
1
```

可以用上述方法对PointData对象中的多个数组进行管理，同时为了方便使用，它的scalars属性也可用于保存数组。与每个点所对应的值除了标量之外，还可以是矢量或张量(矩阵)，矢量数组可以使用vectors属性保存，而张量数组可以使用tensors属性保存。下面的程序将一个形状为(N, 3)的NumPy数组赋值给vectors属性，其中N为点数，这样ImageData对象中的每个点都对应三维空间中的一个矢量：

```

    vectors = np.arange(0.0, img.number_of_points*3).reshape(-1,
3)

    img.point_data.vectors = vectors
    print repr(img.point_data.vectors)
    print type(img.point_data.vectors)
    print img.point_data.vectors[0]

[(0.0, 1.0, 2.0), ..., (177.0, 178.0, 179.0)], length = 60
<class 'tvtk.tvtk_classes.double_array.DoubleArray'>
(0.0, 1.0, 2.0)

```

我们看到所创建的仍然是一个DoubleArray对象，但它是二维数组。number_of_tuples 属性获得其第0轴的长度，而number_of_components属性获得其第1轴的长度：

img.point_data.vectors.number_of_tuples	img.point_data.vectors.number_of_components
60	3

同样，直接使用DoubleArray对象的方法或属性对其进行操作比较烦琐，因此建议读者仍然使用to_array()方法将其转换为NumPy数组之后再进行操作。

在ImageData对象中，点的坐标是通过spacing、origin和dimensions等属性隐式定义的。与之类似，点和单元之间的关系也是隐式定义的。单元和点之间的关系如图8-9所示。单元是由8个邻近的点构成的立方体，图中使用半透明灰色立方体标识出第0个单元。

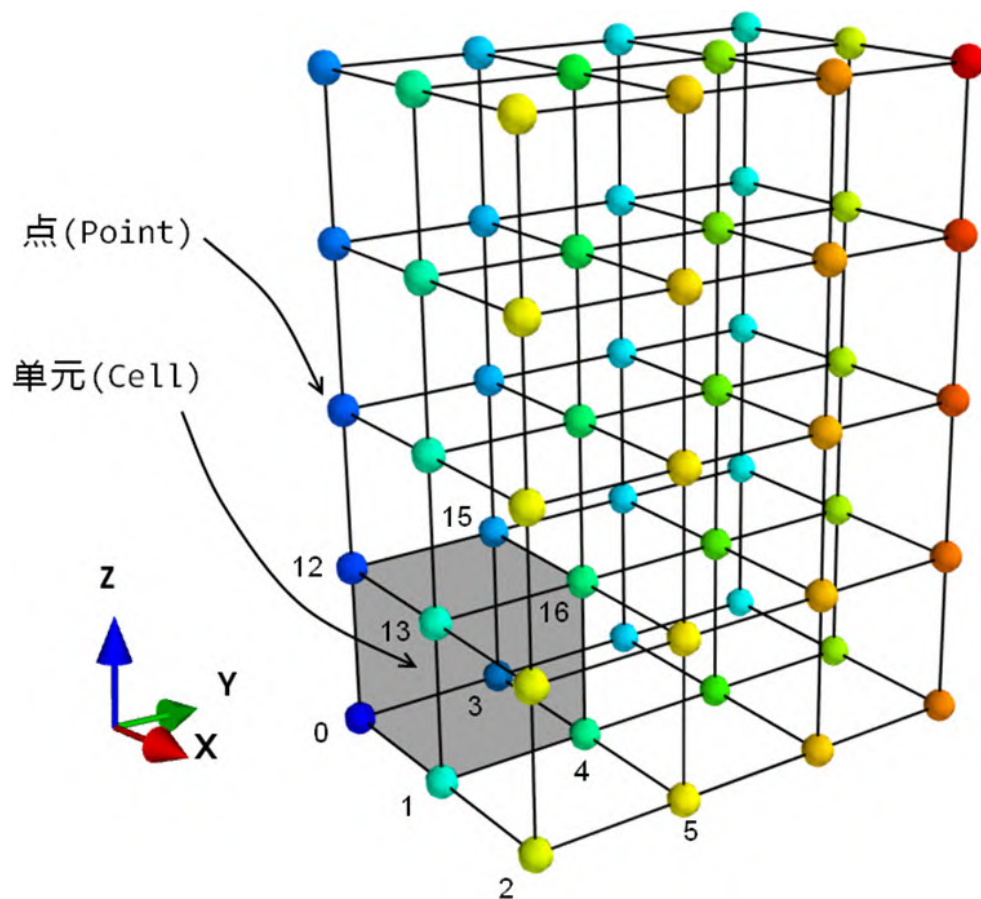


图8-9 单元和点之间的关系



`scipy2.tvtk.figure_imagedata`: 用于绘制图8-9的程序。

通过`get_cell()`方法可以获得表示单元的Voxel(体素)对象，它的`number_of_points`、`number_of_edges`和`number_of_faces`等属性分别是构成体素对象的点数、边数以及面数。

```

cell = img.get_cell(0)
print repr(cell)
<tvtk.tvtk_classes.voxel.Voxel object at 0x0A184C30>
cell.number_of_points  cell.number_of_edges  cell.number_of_faces
-----
8                      12                  6

```

point_ids属性可以获得构成体素的点的序号列表，而points属性则获得构成体素的点的坐标：

```

print repr(cell.point_ids)
cell.points.to_array()
[0, 1, 3, 4, 12, 13, 15, 16]
array([[ 0.1,  0.2,  0.3],
       [ 0.2,  0.2,  0.3],
       [ 0.1,  0.3,  0.3],
       [ 0.2,  0.3,  0.3],
       [ 0.1,  0.2,  0.4],
       [ 0.2,  0.2,  0.4],
       [ 0.1,  0.3,  0.4],
       [ 0.2,  0.3,  0.4]])

```

ImageData对象的number_of_cells属性是数据集中的单元数，也就是图8-9中小立方体的数目。

```

img.number_of_cells
24

```

数据集提供了许多获取点和单元之间关系的方法。例如get_point_cells()获得包含某个点的所有单元的序号，而get_cell_points()则获得某个单元包含的所有点的序号。由于这两个

方法将结果写入一个IdList对象中，因此需要先创建一个空的IdList对象用于保存结果：

```
a = tvtk.IdList()
img.get_point_cells(3, a)
print "cells of point 3:", repr(a)
img.get_cell_points(0, a)
print "points of cell 0:", repr(a) # 和cell.point_ids的值相同
cells of point 3: [2, 0]
points of cell 0: [0, 1, 3, 4, 12, 13, 15, 16]
```

IdList是VTK中管理序号的列表对象，它和Python的标准列表一样支持append()和extend()方法，并且可以通过from_array()将列表或数组转换为IdList对象：

```
a = tvtk.IdList()
a.from_array([1,2,3])
a.append(4)
a.extend([5,6])
print repr(a)
[1, 2, 3, 4, 5, 6]
```

与每个单元对应的数据都保存在cell_data属性中，它是一个CellData对象，用法和PointData对象类似，这里就不再多做介绍了。

```
img.cell_data
<tvtk.tvtk_classes.cell_data.CellData at 0xa285c60>
```


8.2.2 RectilinearGrid

ImageData是最简单的数据集，它的所有点都在一个等间距的三维网格之上，因此只需要起始坐标、网格大小以及网格间距等信息就可以计算出网格上所有点的坐标。如果要表示间距不均匀的网格，可以使用如图8-10所示的RectilinearGrid数据集。

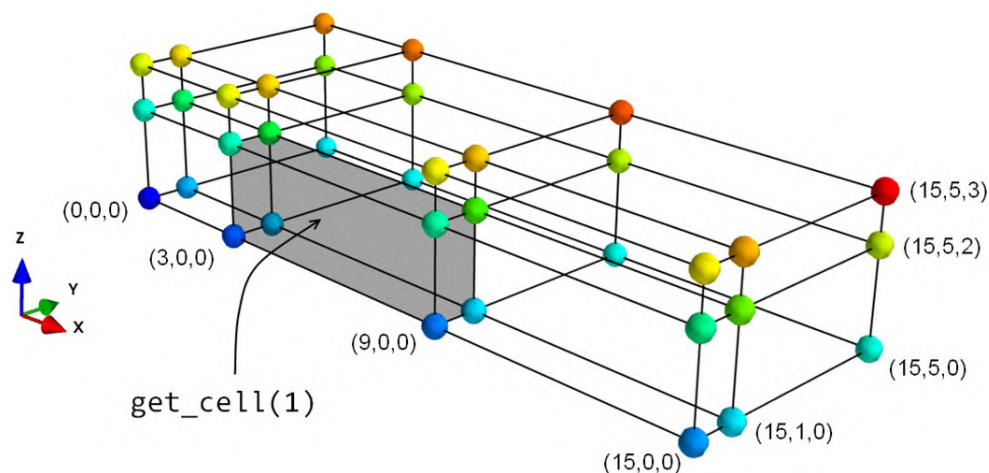


图8-10 使用RectilinearGrid创建分布不均匀的网格



scipy2.tvtk.figure_rectilineargrid: 绘制图8-10的程序。

```
x = np.array([0,3,9,15])
y = np.array([0,1,5])
z = np.array([0,2,3])
r = tvtk.RectilinearGrid()
r.x_coordinates = x ❶
```

```
r.y_coordinates = y
r.z_coordinates = z
r.dimensions = len(x), len(y), len(z) ❷

r.point_data.scalars = np.arange(0.0, r.number_of_points) ❸
r.point_data.scalars.name = 'scalars'
```

RectilinearGrid和ImageData相同，所有点都在一个正交的网格之上，所不同的是网格的分布是不均匀的，因此需要通过一些属性设置X、Y和Z轴的各个网格平面的位置。❶通过RectilinearGrid对象的x_coordinates、y_coordinates和z_coordinates等属性，分别设置网格中与X轴、Y轴和Z轴垂直的平面的位置。RectilinearGrid对象中的点就是所有这些平面的交点。❷由于RectilinearGrid对象不会根据这三个数组的长度自动调整dimensions属性，因此需要根据数组的长度设置dimensions属性。❸最后通过point_data属性设置每个点所对应的数据。

和ImageData对象一样，点的序号依次沿着X、Y和Z轴递增，例如：

```
for i in xrange(6):
    print r.get_point(i)
(0.0, 0.0, 0.0)
(3.0, 0.0, 0.0)
(9.0, 0.0, 0.0)
(15.0, 0.0, 0.0)
(0.0, 1.0, 0.0)
(3.0, 1.0, 0.0)
```

单元和点之间的关系也和ImageData一样：

```
c = r.get_cell(1)
print "points of cell 1:", repr(c.point_ids)
print c.points.to_array()
points of cell 1: [1, 2, 5, 6, 13, 14, 17, 18]
[[ 3.  0.  0.]
 [ 9.  0.  0.]
 [ 3.  1.  0.]
 [ 9.  1.  0.]
 [ 3.  0.  2.]
 [ 9.  0.  2.]
 [ 3.  1.  2.]
 [ 9.  1.  2.]]
```

8.2.3 StructuredGrid

比RectilinearGrid更进一步，StructuredGrid需要我们指定每个点的坐标。而点和单元之间的关系仍然由点在网格中的位置决定。图8-11显示了两种用StructuredGrid创建的网格结构。

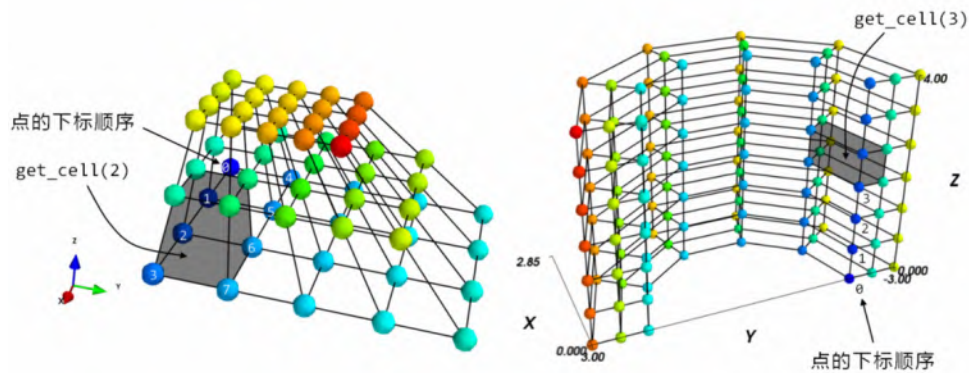


图8-11 用StructuredGrid创建的网格结构

下面对创建这两个网格的程序进行分析：



scipy2.tvtk.figure_structuredgrid：绘制图8-11的程序。

```
def make_points_array(x, y, z):
    return np.c_[x.ravel(), y.ravel(), z.ravel()]

z, y, x = np.mgrid[:3.0, :5.0, :4.0] ❶
x *= (4-z)/3 ❷
y *= (4-z)/3
s1 = tvtk.StructuredGrid()
s1.points = make_points_array(x, y, z) ❸
s1.dimensions = x.shape[:::-1] ❹
s1.point_data.scalars = np.arange(0, s1.number_of_points)
s1.point_data.scalars.name = 'scalars'
```

❶首先用NumPy的mgrid对象创建了三个数组x、y和z。它们的形状都是(3, 5, 4)。其中数组x的数据在第2轴上变化，数组y的数据在第1轴上变化，数组z的数据在第0轴上变化。在这三个数组中，由对应下标的数值组成等间距网格上的点。❷将每个点的X和Y轴的坐标值，乘以由Z轴坐标值决定的系数。沿着Z轴正方向乘积系数逐渐变小，相当于将垂直Z轴的网格进行不同比例的收缩，最终形成一个如图8-11(左)所示的梯形网格。

❸StructuredGrid对象的points属性是数据集中每个点的坐标。它是一个表示N个点的坐标的数组，形状为(N, 3)。因此需要将三个形状为(3, 5, 4)的多维数组合并成一个形状为(3*5*4, 3)的二维数组。这个转换工作由make_points_array()完成。其中首先调用参数数组的ravel()方法，得到其平坦化之后的一维数组，然后通过np.c_对象将三个一维数组按列组合成二维数组。

❹将StructuredGrid对象的dimensions属性设置为(4, 5, 3)。在points属性中只保存点的坐标，点之间的关系由dimensions属性决定。dimensions和NumPy数组的shape属性类似，但是其中第0轴的变化最快，因此需要将数组的shape属性倒序之后再赋值给它。经过dimensions属性处理之后，各个点之间的关系如图8-12所示。图中每个小方块代表points属性中的一个点，方块上的数字表示点在points属性中的下标。

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19

20	21	22	23
24	25	26	27
28	29	30	31
32	33	34	35
36	37	38	39

40	41	42	43
44	45	46	47
48	49	50	51
52	53	54	55
56	57	58	59

图8-12 dimensions为(4, 5, 3)的StructuredGrid的点的结构

单元由网格中相邻的几个点构成，因此单元2由图8-12中8个灰色矩形表示的点构成：

```
s1.get_cell(2).point_ids
[2, 3, 7, 6, 22, 23, 27, 26]
```

由于单元的形状不是长方体，因此VTK采用Hexahedron对象表示单元，get_face()和get_edge()方法分别用于获得构成此单元的面和边：

```
c = s1.get_cell(2)
print "cell type:", type(c)
print "number_of_faces:", c.number_of_faces #单元的面数
f = c.get_face(0) #获得第0个面
print "face type:", type(f) #每个面用一个Quad对象表示
print "points of face 0:", repr(f.point_ids) #构成第0面的4个点的下标

print "edge count of cell:", c.number_of_edges # 单元的边数
e = c.get_edge(0) #获得第0个边
print "edge type:", type(e)
```

```

    print "points of edge 0:", repr(e.point_ids) #构成第0边的两个
点的下标

    cell type: <class 'tvtk.tvtk_classes.hexahedron.Hexahedron'>
    number_of_faces: 6
    face type: <class 'tvtk.tvtk_classes.quad.Quad'>
    points of face 0: [2, 22, 26, 6]
    edge count of cell: 12
    edge type: <class 'tvtk.tvtk_classes.line.Line'>
    points of edge 0: [2, 3]

```

使用StructuredGrid可以创建出任意形状的网络，例如下面的程序创建图8-11(右)所示的半个空心圆柱。程序中，首先在圆柱坐标系中创建等距网格，然后将各点坐标转换到直角坐标系中，具体的步骤留给读者自行分析。

```

r, theta, z2 = np.mgrid[2:3:3j, -np.pi/2:np.pi/2:6j, 0:4:7j]
x2 = np.cos(theta)*r
y2 = np.sin(theta)*r

s2 = tvtk.StructuredGrid(dimensions=x2.shape[:::-1])
s2.points = make_points_array(x2, y2, z2)
s2.point_data.scalars = np.arange(0, s2.number_of_points)
s2.point_data.scalars.name = 'scalars'

```

8.2.4 PolyData

PolyData数据集由一系列的点、点之间的连线以及由点构成的多边形面组成。这些信息都需要用户进行设置，因此用程序创建PolyData对象比较烦琐。TVTK中的许多三维模型类都输出PolyData对象。例如在第一节中介绍的创建圆锥数据的ConeSource类：

```
source = tvtk.ConeSource(resolution = 4)
source.update() # 让source计算其输出数据
cone = source.output
type(cone)
tvtk.tvtk_classes.poly_data.PolyData
```

ConeSource对象的输出数据是一个PolyData对象。PolyData对象的points属性是一个保存点的坐标的数组。为了方便查看各个点的坐标，我们用NumPy的array_str()函数输出此数组的内容，利用suppress_small参数将很小的数显示为0：

```
print np.array_str(cone.points.to_array(),
suppress_small=True)
[[ 0.5  0.  0. ]
 [-0.5  0.5  0. ]
 [-0.5  0.  0.5]
 [-0.5 -0.5  0. ]
 [-0.5 -0. -0.5]]
```

由于设置了ConeSource对象的resolution属性为4，因此圆锥的底面是一个正方形。由各个点的坐标很容易看出底面垂直于X轴，并且在X=-0.5的平面上，而圆锥的顶点在X轴上的X=0.5处。各个点之间的联系由polys属性决定：


```

print type(cone.polys)
print cone.polys.number_of_cells # 圆锥有5个面
print cone.polys.to_array()
<class 'tvtk.tvtk_classes.cell_array.CellArray'>
5
[4 4 3 2 1 3 0 1 2 3 0 2 3 3 0 3 4 3 0 4 1]

```

polys属性是一个CellArray对象，其中保存各个面和点之间的关系。我们创建的圆锥有5个面，因此CellArray对象的number_of_cells属性为5。CellArray对象内部使用一个一维整数数组保存构成各个面的点的下标。由于构成每个面的点数可能不同，因此还需要保存构成每个面的点数。我们可以把这个一维数组理解为如下所示的构造：其中每一行对应一个面，冒号前面的数值是构成此面的点数，冒号后面的一串数字是每个点在points属性中的下标。由下面的数据可知，方锥由4个三角形和1个四边形构成：

```

4 : 4, 3, 2, 1
3 : 0, 1, 2
3 : 0, 2, 3
3 : 0, 3, 4
3 : 0, 4, 1

```

下面看看如何直接创建PolyData对象，首先是一个简单的方锥的例子：

```

p1 = tvtk.PolyData()
p1.points = [(1,1,0), (1,-1,0), (-1,-1,0), (-1,1,0), (0,0,2)] ❶
faces = [

```

```

    4,0,1,2,3,
    3,4,0,1,
    3,4,1,2,
    3,4,2,3,
    3,4,3,0
]

cells = tvtk.CellArray() ❷
cells.set_cells(5, faces) ❸
p1.polys = cells

    p1.point_data.scalars      =      np.linspace(0.0,      1.0,
len(p1.points))

```

❶在介绍StructuredGrid时，我们将一个形状为(N, 3)的数组赋值给points属性，这里使用坐标列表进行赋值，效果和使用数组相同。❷为了给polys属性赋值，需要首先创建一个新的CellArray对象。❸然后调用CellArray对象的set_cells()设置其内容，第一个参数为面(单元)的个数，第二个参数是描述各个面的构成的数组(或列表)。所创建的方锥如图8-13(左)所示，图中标出了各个点的序号。PolyData对象中的各个面可以通过get_cell()方法获得，图中标出了第0个和第1个面。

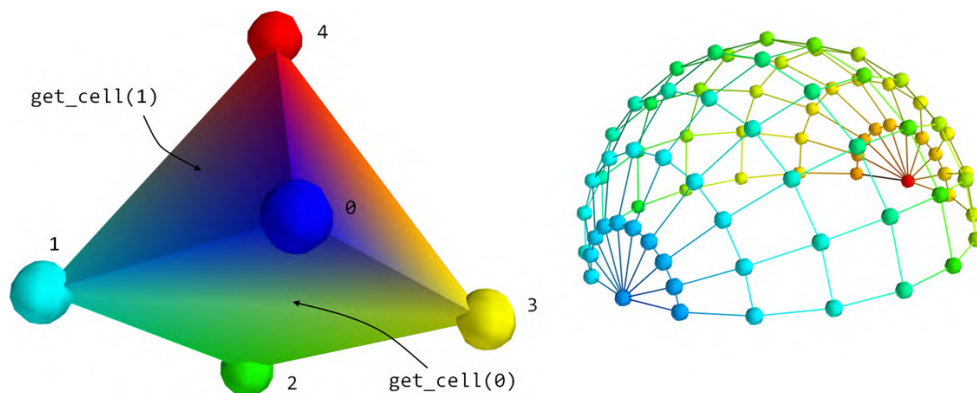


图8-13 用PolyData创建的多面体



scipy2.tvtk.figure_polydata: 绘制图8-13的程序。

下面获取第0面和第1面上的点的序号:

```
print repr(p1.get_cell(0).point_ids)
print repr(p1.get_cell(1).point_ids)

[0, 1, 2, 3]
[4, 0, 1]
```

下面是使用PolyData创建半球面的程序, 图8-13(右)是半球面的显示效果。为了显示出整个半球上的点, 图中将面隐藏, 仅显示各个面的边框。

```
N = 10
a, b = np.mgrid[0:np.pi:N*1j, 0:np.pi:N*1j]
x = np.sin(a)*np.cos(b)
y = np.sin(a)*np.sin(b)
z = np.cos(a)

points = make_points_array(x, y, z) ❶
faces = np.zeros(((N-1)**2, 4), np.int) ❷
t1, t2 = np.mgrid[:(N-1)*N:N, :N-1]
faces[:,0] = (t1+t2).ravel()
faces[:,1] = faces[:,0] + 1
```

```

faces[:,2] = faces[:,1] + N
faces[:,3] = faces[:,0] + N

p2 = tvtk.PolyData(points = points, polys = faces)
    p2.point_data.scalars      =      np.linspace(0.0,      1.0,
len(p2.points))

```

首先将球坐标系中的点转换为直角坐标系中的坐标。❶然后调用 `make_points_array()` 将这些坐标值转换为形状为 $(N, 3)$ 的数组。❷如果每个面的点数相同，可以用一个二维数组表示面和点之间的关系，其中第0轴的长度为面数，第1轴的长度为每个面的点数。可以将此二维数组直接赋值给 `polys` 属性，TVTK库会帮我们完成二维数组到 `CellArray` 对象之间的转换。

```

p2.polys.to_array()[:20]
array([ 4,  0,  1, 11, 10,  4,  1,  2, 12, 11,  4,  2,  3,
13, 12,  4,  3,
       4, 14, 13])

```

请读者根据程序思考 `faces` 数组的计算方法，这里就不再多做解释了。

8.3 TVTK的改进



与本节内容对应的 Notebook 为：08-tvtk_mayavi/tvtk_mayavi-400-tvtk_and_vtk.ipynb。

VTK拥有详细的C++ API说明文档，而Python的VTK扩展库的用法和C++的用法基本相同。为了让读者能有效地用TVTK替代VTK扩展库，本节对TVTK的一些改进进行总结。首先看一个使用标准的VTK扩展库显示圆锥的例子：

```
%%python

# -*- coding: utf-8 -*-
import vtk

# 创建一个圆锥数据源
cone = vtk.vtkConeSource( )
cone.SetHeight( 3.0 )
cone.SetRadius( 1.0 )
cone.SetResolution(10)
# 使用PolyDataMapper将数据转换为图形数据
coneMapper = vtk.vtkPolyDataMapper( )
coneMapper.SetInputConnection( cone.GetOutputPort( ) )
# 创建一个Actor
coneActor = vtk.vtkActor( )
coneActor.SetMapper ( coneMapper )
# 用线框模式显示圆锥
coneActor.GetProperty( ).SetRepresentationToWireframe( )
# 创建Renderer和窗口
ren1 = vtk.vtkRenderer( )
```

```
ren1.AddActor( coneActor )
ren1.SetBackground( 0.1 , 0.2 , 0.4 )
renWin = vtk.vtkRenderWindow( )
renWin.AddRenderer( ren1 )
renWin.SetSize(300 , 300)
# 创建交互工具
iren = vtk.vtkRenderWindowInteractor( )
iren.SetRenderWindow( renWin )
iren.Initialize( )
iren.Start( )
```

此程序和C++程序的区别仅仅是没有声明变量的类型，其他的用法完全和C++的VTK API相同。官方所提供的VTK-Python包和C++语言的接口相似，许多地方没有体现出Python作为动态语言的优势，可以说标准的VTK-Python库不是Python风格的。为了弥补这些不足之处，Enthought公司开发的TVTK库进一步对VTK-Python进行包装。它具有如下优点：

- 支持Trait属性
- 支持元素的Pickle操作
- API更接近Python风格
- 能自动处理NumPy数组或列表对象
- 流水线浏览器ivtk

8.3.1 TVTK的基本用法

下面是前面介绍过的用TVTK显示圆锥的程序：

```
from tvtk.api import tvtk

cs = tvtk.ConeSource(height=3.0, radius=1.0, resolution=36)
m = tvtk.PolyDataMapper(input_connection = cs.output_port)
a = tvtk.Actor(mapper=m)
ren = tvtk.Renderer(background=(1, 1, 1))
ren.add_actor(a)
rw = tvtk.RenderWindow(size=(300,300))
rw.add_renderer(ren)
rwi = tvtk.RenderWindowInteractor(render_window=rw)
rwi.initialize()
rwi.start()
```

可以看到它比标准VTK版本要简短许多，从中可以看到TVTK的一些重要改进：

- TVTK库中的类名除去了前缀“vtk”。有些类名在“vtk”之后是数字，TVTK库对这种类名进行特殊处理：如果首字符为数字，就用其英文单词代替，例如vtk3DSImporter变成ThreeDSImporter。
- 函数名按照Python的惯例，采用下划线连接单词，例如AddItem变成add_item。
- 许多VTK对象的方法在TVTK中用Trait属性替代，例如下面列出了圆锥实例中的两处改进：

```

m.SetInputConnection(cs.GetOutputPort())      # VTK
m.input_connection = cs.output_port            # TVTK

p.SetRepresentationToWireframe()               # VTK
p.representation = 'w'                        # TVTK

```

- Trait属性可以在创建对象的同时通过关键字参数进行设置，这样更便于程序的编写和阅读。

在TVTK库的内部实现中，所有的TVTK对象的内部都有一个VTK对象，对TVTK对象的函数调用将转给内部的VTK对象执行。如果返回值是VTK对象，它将被包装成TVTK对象返回。如果方法的参数是TVTK对象，其中的VTK对象将作为值进行参数传递。

8.3.2 Trait属性

所有的TVTK类都从HasStrictTraits继承，HasStrictTraits规定了它的子类的对象在创建之后不能对不存在的属性进行赋值。VTK中所有和基本状态有关的方法在TVTK中都使用Trait属性表示。调用set()方法可以一次设置多个Trait属性，例如：

```

p = tvtk.Property()
p.set(opacity=0.5, color=(1,0,0), representation="w")

```

调用edit_traits()或configure_traits()可以显示编辑属性的对话框：


```
p.edit_traits()
```

可以通过`tvtk.to_vtk()`得到任何TVTK对象所包装的VTK对象，在必要的时候直接对VTK对象进行操作：

```
print p.representation
p_vtk = tvtk.to_vtk(p)
p_vtk.SetRepresentationToSurface()
print p.representation
wireframe
surface
```

也可以通过TVTK对象的`_vtk_obj`属性获得其中的VTK对象，而`tvtk.to_vtk()`则可以将VTK对象包装成TVTK对象。

8.3.3 序列化

TVTK对象支持简单的序列化处理。单个TVTK对象的状态可以被序列化：

```
import cPickle
p = tvtk.Property()
p.representation = "w"
s = cPickle.dumps(p)
del p
q = cPickle.loads(s)
```

```
q.representation  
  
'wireframe'
```

但是序列化仅仅能保存对象的状态，对象之间的引用无法被保存。因此TVTK的整个流水线无法用序列化保存。通常`pickle.load()`将创建新的对象，如果我们希望更新某个已经存在的对象的状态，可以如下调用`__setstate__()`来实现：

```
p = tvtk.Property()  
p.interpolation = "flat"  
d = p.__getstate__()  
del p  
  
q = tvtk.Property()  
print q.interpolation  
q.__setstate__(d)  
print q.interpolation  
  
gouraud  
  
flat
```

8.3.4 集合迭代

从`tvtk.Collection`继承的对象可以像标准的Python序列对象一样使用，下面的例子演示了`ActorCollection`对象支持`len()`、`append()`以及`for`循环：

```
ac = tvtk.ActorCollection()  
  
print len(ac)
```

```

ac.append(tvtk.Actor())
ac.append(tvtk.Actor())
print len(ac)

for a in ac:
    print repr(a)

del ac[0]
print len(ac)
0
2
    <tvtk.tvtk_classes.open_gl_actor.OpenGLActor object at
0x0A24A690>
    <tvtk.tvtk_classes.open_gl_actor.OpenGLActor object at
0x0A174ED0>
1

```

对比一下VTK的相应程序，就能体会出TVTK库的优点了：

```

import vtk

ac = vtk.vtkActorCollection()

print ac.GetNumberOfItems()

ac.AddItem(vtk.vtkActor())

ac.AddItem(vtk.vtkActor())

print ac.GetNumberOfItems()

ac.InitTraversal()

```

```

for i in range(ac.GetNumberOfItems()):
    print repr(ac.GetNextItem())

ac.RemoveItem(0)
print ac.GetNumberOfItems()
0
2
(vtkOpenGLActor) 0A24AF90
(vtkOpenGLActor) 0A24AF00
1

```

8.3.5 数组操作

所有DataArray的派生类和Python的序列一样，支持迭代接口以及__getitem__()、__setitem__()、__repr__()、append()、extend()等。此外，还可以通过from_array()直接用NumPy数组或列表进行赋值，可以很方便地将其中保存的数据转换为NumPy数组。Points和IdList等对象也同样支持这些特性：

```

pts = tvtk.Points()
p_array = np.eye(3)
pts.from_array(p_array)
pts.print_traits()
pts.to_array()
_in_set: 0
_vtk_obj: (vtkPoints) 0A2D82D0

```

```

actual_memory_size:      1
bounds:                  (0.0, 1.0, 0.0, 1.0, 0.0, 1.0)
class_name:              'vtkPoints'
data:                    [(1.0, 0.0, 0.0), (0.0, 1.0, 0.0),
(0.0, 0.0, 1.0)]
data_type:               'double'
data_type_:              11
debug:                   0
debug_:                  0
global_warning_display:  1
global_warning_display_: 1
m_time:                  44927
number_of_points:        3
reference_count:          1
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])

```

如果TVTK对象的属性或方法能够接受DataArray、Points、IdList以及CellArray等对象，那么它也同时能够接受数组和列表：

```

points = np.array([[0,0,0],[1,0,0],[0,1,0],[0,0,1]], 'f')
triangles = np.array([[0,1,3],[0,3,2],[1,2,3],[0,2,1]])
values = np.array([1.1, 1.2, 2.1, 2.2])
mesh = tvtk.PolyData(points=points, polys=triangles)
mesh.point_data.scalars = values
print repr(mesh.points)

```

```

print repr(mesh.polys)
print mesh.polys.to_array()
print mesh.point_data.scalars.to_array()

[(0.0, 0.0, 0.0), (1.0, 0.0, 0.0), (0.0, 1.0, 0.0), (0.0,
0.0, 1.0)]

<tvtk.tvtk_classes.cell_array.CellArray object at
0x0A2E5360>

[3 0 1 3 3 0 3 2 3 1 2 3 3 0 2 1]

[ 1.1  1.2  2.1  2.2]

```

8.4 TVTK可视化实例



与本节内容对应的 Notebook 为：08-tvtk_mayavi/tvtk_mayavi-500-tvtk-examples.ipynb。

由于篇幅所限，本书不对VTK库的用法进行详细解释。在本节，我们将对几个比较典型的实例进行分析。希望读者在学习这几个实例之后能够融会贯通，掌握VTK开发的一般流程以及使用TVTK带来的便利。

从VTK的网站可以下载大量的C++和TCL的程序实例，由于TVTK的用法更加简洁，读者应该能很容易将它们转换成使用TVTK库的Python程序。

在本节的可视化实例程序中，使用了本书提供的辅助模块 `scipy2.tvtk.tvtkhelp`。其中的 `ivtk_scene()` 使用 `ivtk` 显示一组 Actor

对象。

8.4.1 切面

展示三维数据的一个比较简单的方法是使用切面，对切面经过的数据进行可视化，这样就把三维数据可视化问题转换成了二维数据的可视化。通过交互式地修改切面的位置和方向，用户能直观地对三维数据进行观察。下面是使用切片工具观察数据的一个实例，效果如图8-14所示。

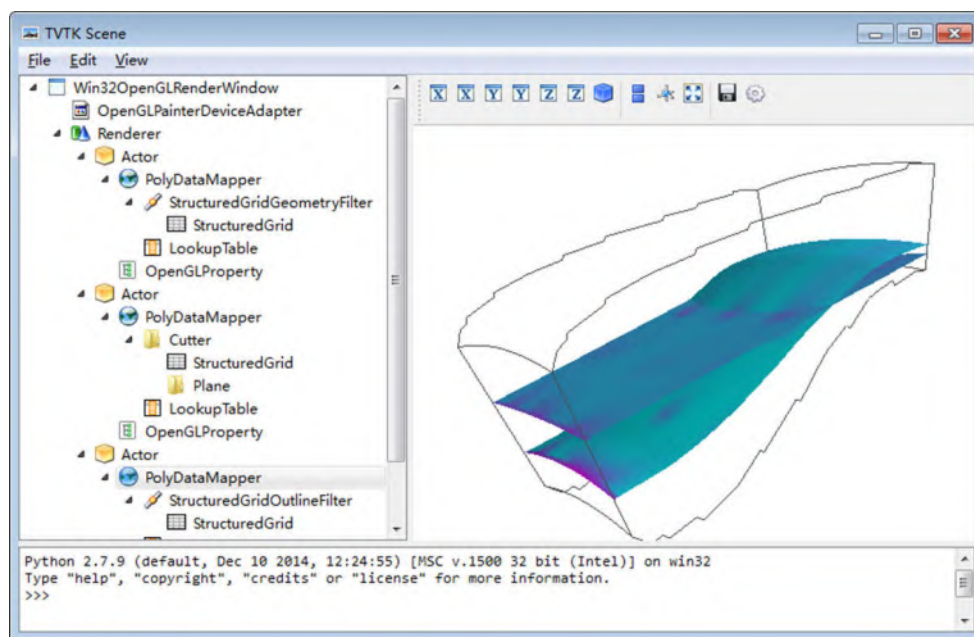


图8-14 使用切面观察StructuredGrid数据集

它由三个部分组成：一个曲面、一个平面和一个外框。在曲面和平面上，每个点使用颜色表示对应数值的大小。由于我们使用ivtk显示可视化结果，因此可以使用界面左侧的流水线浏览器观察组成整个场景的流水线。



scipy2.tvtk.example_cut_plane: 切面演示程序。

```
def read_data():
    # 读入数据

    plot3d = tvtk.MultiBlockPLOT3DReader( ❶
        xyz_file_name = "combxyz.bin",
        q_file_name = "combq.bin",
        scalar_function_number = 100, vector_function_number
= 200
    )

    plot3d.update() ❷

    return plot3d

plot3d = read_data()
grid = plot3d.output.get_block(0) ❸

# 创建颜色映射表

lut = tvtk.LookupTable() ❹

lut.table = pl.cm.cool(np.arange(0,256))*255
```

下面先看看曲面的制作过程。❶为了对可视化的方法进行重点介绍，我们直接使用一个MultiBlockPLOT3DReader对象从数据文件读入三维数据信息。❷为了让它真正从文件读入数据，需要调用其update()方法。运行此方法之后，❸就可以通过其output属性获取读入的数据集对象了。通过output属性获得的是一个MultiBlockDataSet

对象，通过`get_block(0)`可以获得其中下标0对应的`StructuredGrid`数据集。

```
print type(plot3d.output)
print type(plot3d.output.get_block(0))

                                     <class
'tvtk.tvtk_classes.multi_block_data_set.MultiBlockDataSet'>
    <class 'tvtk.tvtk_classes.structured_grid.StructuredGrid'>
```

`MultiBlockPLOT3DReader`读入`PLOT3D`格式的文件，`PLOT3D`是一种流体力学数据可视化的程序。程序中通过设置`scalar_function_number`和`vector_function_number`属性，分别指定标量数组和矢量数组为流体的密度和速度。各个参数的具体含义请读者查看VTK的相关文档。`MultiBlockPLOT3DReader`对象输出的是一个`StructuredGrid`数据集。下面观察它的一些属性：

```
print "dimensions:", grid.dimensions
print grid.points.to_array()
print "cell arrays:", grid.cell_data.number_of_arrays
print "point arrays:", grid.point_data.number_of_arrays

print "arrays name:"
for i in xrange(grid.point_data.number_of_arrays):
    print "    ", grid.point_data.get_array_name(i)

print "scalars name:", grid.point_data.scalars.name
print "vectors name:", grid.point_data.vectors.name
dimensions: [57 33 25]
```

```

[[ 2.66700006 -3.77476001 23.83292007]
 [ 2.94346499 -3.74825287 23.66555977]
 [ 3.21985817 -3.72175312 23.49823952]
 ...,
 [ 15.84669018  5.66214085 35.7493782 ]
 [ 16.17829895  5.66214085 35.7493782 ]
 [ 16.51000023  5.66214085 35.7493782 ]]
cell arrays: 0
point arrays: 4
arrays name:
    Density
    Momentum
    StagnationEnergy
    Velocity
scalars name: Density
vectors name: Velocity

```

由上面的各个属性可以得知，数据集grid是一个形状为(57, 33, 25)的网格。由于它是一个StructuredGrid对象，因此网格中的每个点的坐标保存在points属性中。网格中的单元没有数据，而每个点对应4种数据，它们的名字分别为“Density”、“Momentum”、“StagnationEnergy”和“Velocity”。其中通过point_data的scalars属性可以获得名为“Density”的数组，它是一个标量数组，而“Velocity”数组则可以通过point_data的vectors属性获得，它是一个三维矢量数组。切面将针对scalars属性的数据进行运算，因此切面所显示的是切面上每个点的密度值。

④我们需要把切面上的密度用某种颜色来表示，因此需要进行值到颜色的转换。在VTK中这种转换工作由LookupTable对象完成。它的table属性是一个保存颜色表的数组。这里我们使用matplotlib库的颜色映射对象计算LookupTable对象的颜色。

接下来显示StructuredGrid中某一层网格面上的数据：

```
# 显示StructuredGrid中的一个网格面

plane = tvtk.StructuredGridGeometryFilter(extent = (0, 100,
0, 100, 6, 6)) ①

plane.set_input_data(grid) ②

plane_mapper = tvtk.PolyDataMapper(lookup_table = lut,
input_connection = plane.output_port) ③

plane_mapper.scalar_range = grid.scalar_range ④

plane_actor = tvtk.Actor(mapper = plane_mapper) ⑤
```

①StructuredGridGeometryFilter对象能从StructuredGrid对象中提取部分网格。程序中通过extent属性指定了提取的网格的范围：第0轴和第1轴的范围是0到100，而第2轴的范围是6到6。其中第0轴、第1轴的范围大于网格在这两个方向上的长度，因此它们提取这两个轴上的整个范围，而在第2轴上只提取第6层的数据。②当直接把数据集对象当作输入时，应当调用VTK中的SetInputData()函数，在TVTK中该函数为set_input_data()。StructuredGridGeometryFilter对象输出一个PolyData数据集：



TVTK库没有将SetInputData() 转换成input_data属性，因此需要调用与之对应的set_input_data() 函数来设置输入数据集。

```
plane.update()
p = plane.output
type(p)
tvtk.tvtk_classes.poly_data.PolyData
```

下面查看这个PolyData对象的一些属性：

```
print p.number_of_points, grid.dimensions[0] *
grid.dimensions[1]
1881 1881
```

它由1881个点构成，因为它是StructuredGrid对象中第2轴上的一层，所以它的点数等于原始数据集的第0轴和第1轴长度的乘积。下面比较StructuredGrid对象和PolyData对象中点的坐标：

```
print grid.dimensions
points1 = grid.points.to_array().reshape((25,33,57,3))
points2 = p.points.to_array().reshape((33,57,3))
np.all(points1[6] == points2)
[57 33 25]
True
```

points1是将StructuredGrid对象的点还原成三维网格之后的数组，数组的第3轴表示每个点的X、Y和Z轴的坐标值。由于NumPy数组的shape属性和StructuredGrid对象的dimensions属性为倒序关系，因此dimensions属性中的第0轴相当于shape属性的第2轴。同样，我们将PolyData对象的点还原成表示二维网格数组的points2。因为我们提取的是StructuredGrid对象的第2轴的第6层数据，因此points1[6]应该和points2相等。

使用StructuredGridGeometryFilter对象不但从数据源提取了点的坐标，还提取了每个点所对应的数据，因此PolyData对象的每个点也对应4组数据，并且数组名也保持不变。

```
print p.point_data.number_of_arrays
print p.point_data.scalars.name
4
Density
```

③使用PolyDataMapper对象将PolyData对象转换为图形数据，同时用前面创建的颜色映射表lut对PolyData对象中的每个点进行着色。

④为了能使用颜色映射表中的所有颜色，我们将颜色映射表的范围设置成和密度数组的范围一致。⑤最后创建在场景中显示图形数据的Actor对象。

接下来是创建平面切面的程序：

```
lut2 = tvtk.LookupTable()
lut2.table = pl.cm.cool(np.arange(0,256))*255
cut_plane = tvtk.Plane(origin = grid.center, normal=(-0.287,
```

```
0, 0.9579)) ❶

cut = tvtk.Cutter(cut_function = cut_plane) ❷
cut.set_input_data(grid)

cut_mapper = tvtk.PolyDataMapper(input_connection =
cut.output_port, lookup_table = lut2)

cut_actor = tvtk.Actor mapper = cut_mapper)
```

❶首先创建表示平面的Plane对象。它是一个经过origin属性表示的坐标点，法线方向为normal属性的无限平面，通过这两个属性可以唯一确定平面。Plane对象的输出是一个PolyData对象：

```
type(plane.output)
tvtk.tvtk_classes.poly_data.PolyData
```

❷创建一个Cutter对象，它使用传递给cut_function属性的Plane对象，对set_input_data()设置的目标数据集进行切面插值。Cutter对象的输出也是PolyData对象，由于StructuredGrid对象中的点不一定能正好落在Plane对象所指定的平面之上，因此Cutter对象会对StructuredGrid对象中的数据进行插值计算。它所输出的PolyData对象中的点不再是数据源中的点。

```
cut.update()

cut.output.number_of_points
2537
```

PolyData对象中的每个点仍然与4组数据相对应，这些数据都是通过数据源进行插值计算得到的：

```
cut.output.point_data.number_of_arrays
```

```
4
```

为了在场景中显示切面，和前面所讲的曲面一样，使用PolyDataMapper和Actor将PolyData对象加工成场景中的物体。接下来创建StructuredGrid对象的外边框部分：

```
def make_outline(input_obj):  
    from tvtk.common import configure_input  
    outline = tvtk.StructuredGridOutlineFilter()  
    configure_input(outline, input_obj)  
    outline_mapper = tvtk.PolyDataMapper(input_connection =  
outline.output_port)  
    outline_actor = tvtk.Actor(mapper = outline_mapper)  
    outline_actor.property.color = 0.3, 0.3, 0.3  
    return outline_actor  
  
outline_actor = make_outline(grid)
```

在make_outline()中使用StructuredGridOutlineFilter对象计算出一个表示外边框的PolyData对象。其中调用tvtk.common.configure_input()来设置StructuredGridOutlineFilter对象的输入。该函数会根据输入对象的类型选择input_connection或set_input_data()。

请读者根据前面介绍的方法观察此PolyData对象的内容，这里就不再举例了。最后使用本书提供的tvtkhelp模块中定义的

ivtk_scene() 显示前面创建的三个Actor对象：plane_actor、cut_actor、outline_actor。

```
win = ivtk_scene([plane_actor, cut_actor, outline_actor])  
win.scene.isometric_view()
```

在界面左侧的流水线浏览器中，双击某个对象可以打开对应的编辑器，对其各种属性进行编辑。例如可以打开Plane对象的编辑器。在此编辑器中修改Plane对象的original和normal属性，从而改变切面的位置和方向，观察数据集中不同位置的密度分布情况。而打开StructuredGridGeometryFilter对象的编辑器，可以修改曲面切面的范围。图8-15显示了这两个编辑器，以及通过它们修改之后的切面。

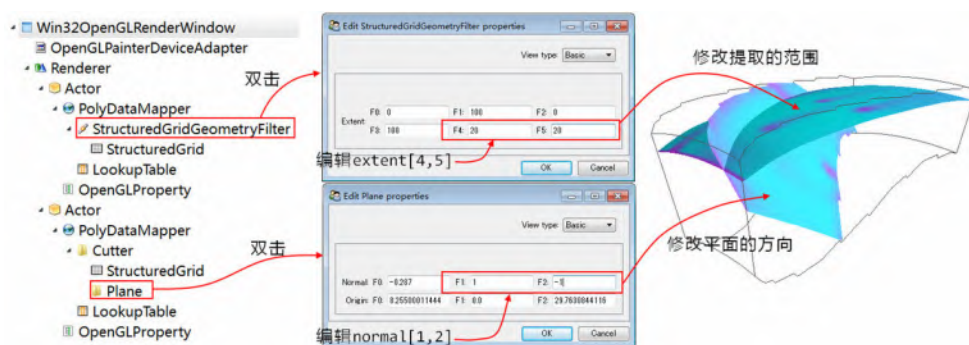


图8-15 通过编辑器修改切面的位置和方向

8.4.2 等值面

等值面是标量场中标量值相等的曲面，和地图中的等高线类似。在VTK中使用ContourFilter计算等值面。下面的程序对上节的流体数据进行等值面可视化，效果如图8-16所示。

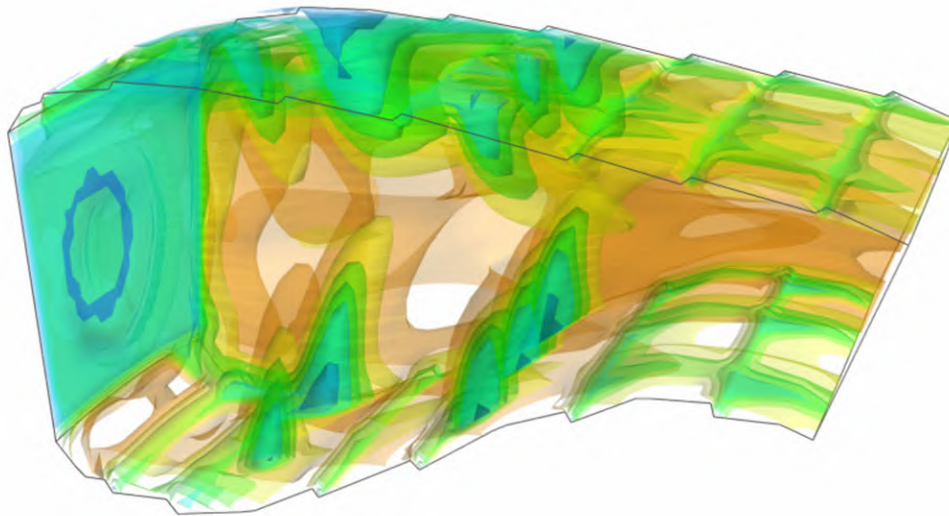


图8-16 使用等值面对标量场进行可视化



scipy2.tvtk.example_contours: 使用等值面可视化标量场。

```
contours = tvtk.ContourFilter()
contours.set_input_data(grid)
contours.generate_values(8, grid.point_data.scalars.range)
①
mapper = tvtk.PolyDataMapper(input_connection =
contours.output_port,
    scalar_range = grid.point_data.scalars.range) ②
actor = tvtk.Actor(mapper = mapper)
actor.property.opacity = 0.3 ③

outline_actor = make_outline(grid)
```

```
win = ivtk_scene([actor, outline_actor])  
win.scene.isometric_view()
```

❶ 创建一个ContourFilter对象，并且调用generate_values()方法来创建8个等值面，等值面的取值范围由标量值数组scalars的范围决定。❷ 等值面的颜色映射也由scalars数组的范围决定。由于没有设置映射器的颜色表，将使用系统缺省的映射表，它将最小值映射为红色，将最大值映射为蓝色。

❸ 由于8个等值面是嵌套的，我们需要修改等值面的Actor对象的透明度，以便观察等值面的内部构造。除了使用generate_values()创建等差等值面之外，还可以使用set_value()方法直接设置每个等值面的值，而get_value()方法可以获得等值面的值。下面的程序修改第0个等值面的值：

```
print contours.get_value(0)  
contours.set_value(0, 0.21)  
0.197813093662
```

在这个例子中，同一个等值面上所有点的颜色是相同的。因为等值面上的标量值(流体密度)相同，而对等值面进行着色时，缺省也使用标量值。有时候我们希望等值面的颜色由另外的标量值决定。下面的程序演示了如何使用别的标量值对等值面进行着色，效果如图8-17所示。

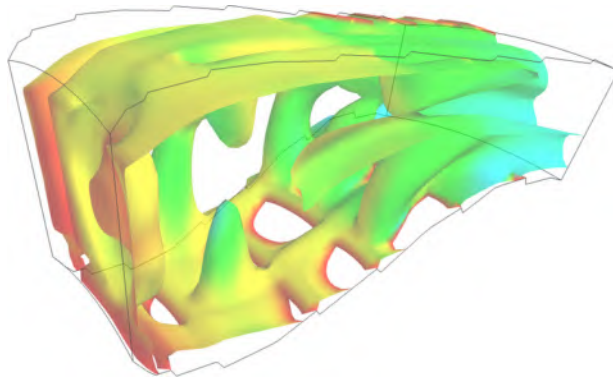


图8-17 在等值面上用颜色显示其他标量值

```

plot3d = read_data()
plot3d.add_function(153) ❶
plot3d.update()
grid = plot3d.output.get_block(0)

contours = tvtk.ContourFilter()
contours.set_input_data(grid)
contours.set_value(0, 0.30) ❷

mapper = tvtk.PolyDataMapper(input_connection =
contours.output_port,

    scalar_range = grid.point_data.get_array(4).range, ❸
    scalar_mode = "use_point_field_data") ❹
mapper.color_by_array_component("VelocityMagnitude", 0) ❺
actor = tvtk.Actor(mapper = mapper)
actor.property.opacity = 0.6

outline_actor = make_outline(grid)

```

```
win = ivtk_scene([actor, outline_actor])  
win.scene.isometric_view()
```

❶首先调用PLOT3DReader对象的add_function()方法，为每个点添加一组标量值。所增加的新数组名为“VelocityMagnitude”，表示每点所对应的速度大小，下面将使用此数组对等值面进行着色。

```
grid.point_data.get_array_name(4)  
'VelocityMagnitude'
```

❷调用ContourFilter对象的set_value()，创建一个值为0.3的等值面。❸设置映射器的标量范围属性scalar_range，将它设置为新增加的数组的取值范围。❹scalar_mode属性决定映射器所使用的标量数据类型，这里的“use_point_field_data”表示使用点数据中的数组。它有如下几种选择：

- “default”：使用point_data.scalars，如果不存在就使用cell_data.scalars。
- “use_point_data”：使用point_data.scalars。
- “use_cell_data”：使用cell_data.scalars。
- “use_point_field_data”：使用point_data中的某个数组，具体的数组需要另外指定。
- “use_cell_field_data”：使用cell_data中的某个数组，具体的数组需要另外指定。

❺通过color_by_array_component()指定着色所使用的数据。它的第一个参数是数组名，第二个参数是从数组中选择的列。由于“VelocityMagnitude”是一个标量数组，它只有一列数据，因此第二

个参数为0。如果第一个参数是矢量数组名，例如“Velocity”，那么可以通过第二个参数选择矢量数组中的不同分量。请读者修改这两个参数，使用其他的数据对等值面进行着色。

8.4.3 流线

在前面的实例中，我们使用切面和等值面对流体的密度分布进行了可视化。空间中每一点的密度可以用一个数值(标量)表示，因此可以将密度分布理解为一个标量场。而流体在每一点的速度是一个矢量，因此速度的分布情况需要使用矢量场来描述。本节介绍如何使用随机散布的矢量箭头和流线对矢量场进行可视化，效果如图8-18所示。由此图可知，整个场景由4个实体构成：外框、随机散布的矢量箭头、表示流线源的球体以及流线。

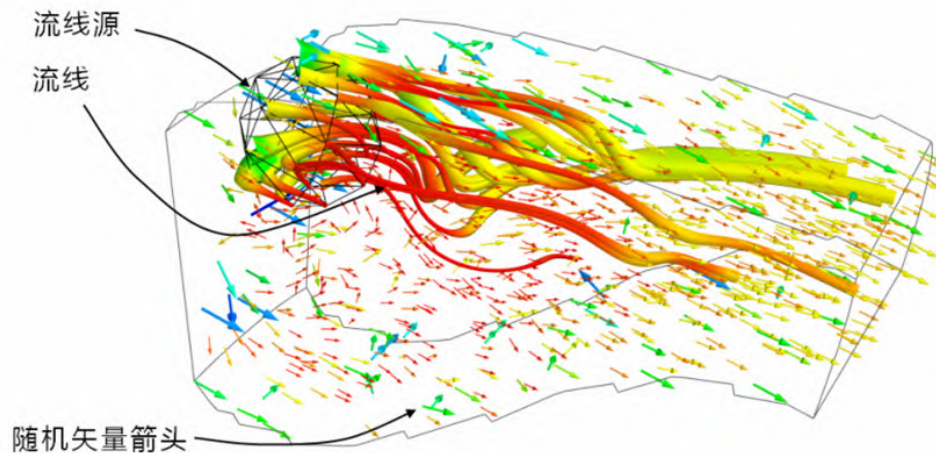


图8-18 矢量场的可视化



scipy2.tvtk.example_streamline: 使用流线和箭头可视化矢量场。

下面的程序创建随机散布的矢量箭头。这些箭头的起始点是数据集中的点的坐标，箭头的方向由点所对应的矢量决定，而箭头的大小和颜色则由点所对应的标量决定。本例中，箭头的方向表示速度的方向，而大小和颜色则表示密度。箭头越大表示该点的标量值(密度)越大，箭头的颜色也同时表示标量值的大小，红色对应的标量值最小，而蓝色对应的标量值最大。

```
# 矢量箭头

mask = tvtk.MaskPoints(random_mode=True, on_ratio=50) ❶
mask.set_input_data(grid)

arrow_source = tvtk.ArrowSource() ❷
arrows = tvtk.Glyph3D(input_connection = mask.output_port,
❸

scale_factor=2/np.max(grid.point_data.scalars.to_array()))
arrows.set_source_connection(arrow_source.output_port)
arrows_mapper = tvtk.PolyDataMapper(input_connection =
arrows.output_port,

    scalar_range = grid.point_data.scalars.range)
arrows_actor = tvtk.Actor(mapper = arrows_mapper)
```

❶由于原始数据集中的点数很多，如果在所有点的位置都描绘箭头，将十分耗时并且无法区分众多的箭头。因此首先使用MaskPoints对象对数据集中的数据进行随机选取，某点被选中的概率为1/50，即每50个点选择一个点。下面的程序查看MaskPoints对象输出的数据类型和点数，以及每个点所对应的数据：

```
print grid.number_of_points
mask.update()
print type(mask.output)
print mask.output.number_of_points
print mask.output.point_data.number_of_arrays
47025
<class 'tvtk.tvtk_classes.poly_data.PolyData'>
952
5
```

❷ArrowSource对象创建表示箭头的PolyData数据集。❸Glyph3D对象对箭头数据进行复制，在masks输出的PolyData数据集的每个点之上都放置一个箭头。箭头的方向、长度和颜色由与点对应的矢量和标量数据决定。scale_factor参数是所有箭头共同的缩放系数，这里使用标量数组的最大值对缩放系数进行正规化。Glyph3D对象可以对任意的PolyData数据进行复制，读者可以尝试将箭头数据源ArrowSource改为圆锥数据源ConeSource。



由于TVTK没有提供source_connection属性，因此只能通过set_source_connection()设置Glyph3D对象的输入。

```
arrows.update()

print arrow_source.output.number_of_points # 一个箭头有31个点
print arrows.output.number_of_points # 箭头被复制了N份，因此有
N*31个点

31

29512
```

还可以使用流线直观地观察矢量场。流线上每一点的切线方向就是矢量场在该点的方向。下面是显示流线的程序：

```
center = grid.center
sphere = tvtk.SphereSource( ❶
    center=(2, center[1], center[2]), radius=2,
    phi_resolution=6, theta_resolution=6)

    sphere_mapper =
tvtk.PolyDataMapper(input_connection=sphere.output_port)
sphere_actor = tvtk.Actor(mapper=sphere_mapper)
sphere_actor.property.set(
    representation = "wireframe", color=(0,0,0))

# 流线
streamer = tvtk.StreamLine( ❷
    step_length=0.0001,
    integration_direction="forward",
    integrator=tvtk.RungeKutta4()) ❸
streamer.set_input_data(grid)
```



```

streamer.set_source_connection(sphere.output_port)

tube = tvtk.TubeFilter( ❷
    input_connection=streamer.output_port,
    radius=0.05,
    number_of_sides=6,
    vary_radius="vary_radius_by_scalar")

tube_mapper = tvtk.PolyDataMapper(
    input_connection=tube.output_port,
    scalar_range=grid.point_data.scalars.range)
tube_actor = tvtk.Actor(mapper=tube_mapper)
tube_actor.property.backface_culling = True

outline_actor = make_outline(grid)

win = ivtk_scene([outline_actor, sphere_actor, tube_actor,
arrows_actor])

win.scene.isometric_view()

```

❶ SphereSource 对象创建表示球体的 PolyData 数据集。通过 center 和 radius 属性指定球体的球心位置和半径。phi_resolution 和 theta_resolution 属性指定球体的经度和纬度方向上的等分次数，值越大输出的 PolyData 数据集越接近球体。

❷ StreamLine 对象在矢量场中计算流线。通过 set_source_connection() 设置决定流线起点的数据集的输入端口，这里使用 SphereSource 对象输出的球面上的点作为流线的起点。如果通

过start_position属性设置起始点的坐标，则只计算一条流线。step_length属性决定了流线上的点的间隔，此值越小流线上的点越多，流线越平滑，计算所需的时间也越长。integration_direction属性决定流线的计算方向，值可以为'backward'、'forward'和'integrate_both_directions'。'forward'表示计算起点之后的流线，'backward'表示计算起点之前的流线。

③流线的计算需要使用由integrator属性指定的数值积分算法，这里使用RungeKutta4，它是一个4阶Runge-Kutta积分算法。

StreamLine对象的输出是一个PolyData数据集，它的所有单元都是表示流线的路径，因此边(line)数为23，而面数为0：

```
streamer.update()
print streamer.output.number_of_points
print streamer.output.number_of_polys
print streamer.output.number_of_lines
5528
0
23
```

④使用TubeFilter对象可以将流线路径转换为有粗细的圆管。radius属性为圆管的粗细，而number_of_side属性指定圆管的切面圆的边数。圆管的粗细可以根据点的数据发生变化，这里使用"vary_radius_by_scalar"指定圆管的粗细由标量(密度)决定。

TubeFilter对象的输出也是PolyData数据集，它由众多的面构成，但是它的number_of_polys属性却等于0：

```
tube.update()

tube.output.number_of_polys

0
```

这里为了节省内存空间和计算时间，PolyData对象使用TriangleStrip对象表示三角面。一个TriangleStrip对象由一组点构成。每连续的三个点构成一个三角形面：

```
print tube.output.number_of_strips
t = tube.output.get_cell(0)
print type(t)
print t.number_of_points
138
<class 'tvtk.tvtk_classes.triangle_strip.TriangleStrip'>
498
```

上面的例子中，箭头的大小和颜色、流线的粗细和颜色所表示的是流体的密度。有时候我们希望用这些可视化元素表示矢量的长度，即流体的速度的大小。我们可以直接计算vectors数组中各个矢量的长度，并且将其写入数据集的scalars数组中。例如，在读入数据之后如下添加两行程序：

```
point_data = grid.point_data
point_data.scalars =
np.sqrt(np.sum(point_data.vectors.to_array()**2, axis=-1))
```

或者使用TVTK库中的VectorNorm。它计算输入数据的vectors数组中各个矢量的长度，并且保存到scalars数组中。只需将之前程序中的

grid修改为vnorm.output_port即可：



grid是数据集对象，而vnorm为Algorithm对象。为了让程序兼容这两种不同的输入类型，可以使用tvtk.common.configure_input()。

```
vnorm = tvtk.VectorNorm()  
vnorm.set_input_data(grid)
```

8.4.4 计算圆柱的相贯线

两个互相垂直的圆管相交会产生8条相贯线，下面我们用PolyData的布尔运算功能生成两个互相垂直的圆管，并计算它们的相贯线。程序的运行结果如图8-19所示：

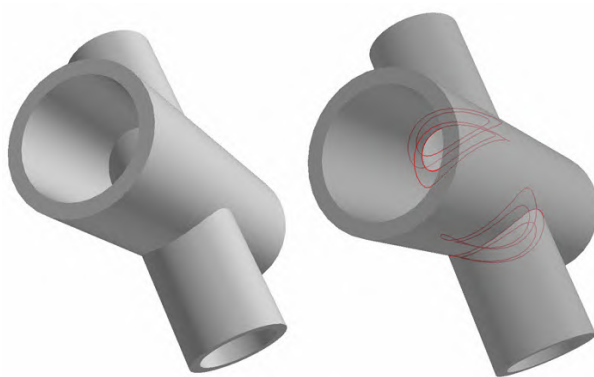


图8-19 两个互相垂直的圆管(左)，打通圆管并显示相贯线(右)



scipy2.tvtk.example_tube_intersection: 计算两个圆管的相贯线，可通过界面中的滑块控件修改圆管的内径和外径。

首先定义make_tube()函数，它创建指定方向和大小的圆管，生成圆管的流水线如图8-20所示。

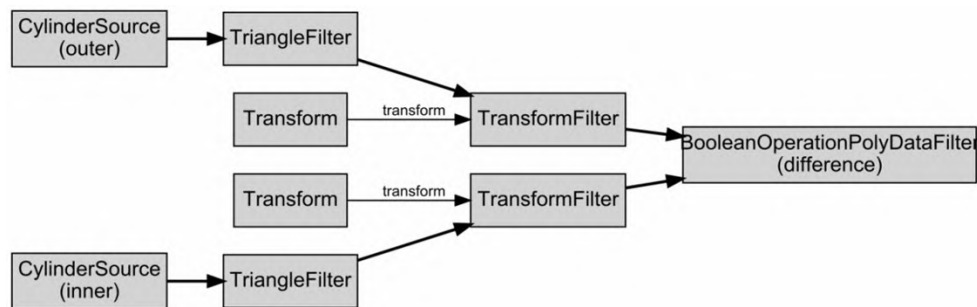


图8-20 生成圆管的流水线

```

def make_tube(height, radius, resolution, rx=0, ry=0, rz=0):
    cs1 = tvtk.CylinderSource(height=height, radius=radius[0],
resolution=resolution) ❶
    cs2 = tvtk.CylinderSource(height=height+0.1,
radius=radius[1], resolution=resolution)

    triangle1 =
tvtk.TriangleFilter(input_connection=cs1.output_port) ❷
    triangle2 =
tvtk.TriangleFilter(input_connection=cs2.output_port)

    tr = tvtk.Transform()
    tr.rotate_x(rx)
    tr.rotate_y(ry)
  
```

```

        tr.rotate_z(rz)

        tf1      =      tvtk.TransformFilter(transform=tr,
input_connection=triangle1.output_port) ③

        tf2      =      tvtk.TransformFilter(transform=tr,
input_connection=triangle2.output_port)

        bf = tvtk.BooleanOperationPolyDataFilter() ④
        bf.operation = "difference"
        bf.set_input_connection(0, tf1.output_port)
        bf.set_input_connection(1, tf2.output_port)

        m = tvtk.PolyDataMapper(input_connection=bf.output_port,
scalar_visibility=False)

        a = tvtk.Actor(mapper=m)
        a.property.color = 0.7, 0.7, 0.7

        return bf, a, tf1, tf2

    tubel, tubel_actor, tubel_outer, tubel_inner = make_tube(5,
[1, 0.8], 32)

    tube2, tube2_actor, tube2_outer, tube2_inner = make_tube(5,
[0.7, 0.55], 32, rx=90)

    win = ivtk_scene([tubel_actor, tube2_actor])
    win.scene.isometric_view()

```

上面的程序创建了两个表示圆管的PolyData对象tubel和tube2，以及构成这两个圆管的外圆柱和内圆柱PolyData对象：tubel_outer、tubel_inner、tube2_outer和tube2_inner。

❶使用CylinderSource创建两个圆柱面，它的output属性是表示圆柱面的PolyData对象。该PolyData对象中的每个面都是四边形，为了后续的布尔运算能正确运行，❷需要通过TriangleFilter将其转换为完全由三角形面构成的PolyData对象。

❸使用TransformFilter对PolyData对象进行旋转，它的transform属性是一个描述旋转、偏移以及缩放操作的Transform对象。❹最后使用BooleanOperationPolyDataFilter计算粗圆柱体与细圆柱体的差分布尔运算，得到表示圆管的PolyData对象。为了让差分布尔运算能正常工作，在创建圆柱时让细圆柱比粗圆柱略微长一些。由于布尔运算需要两个输入对象，因此需要调用set_input_connection()来指定每个输入端口的输入对象。

由图8-19(左)可以看到两个圆管互相并不是相通的。下面我们使用布尔运算将两个圆管相交的部分打通。这相当于用圆管1减去圆管2的内圆柱，以及用圆管2减去圆管1的内圆柱。这个操作仍然使用BooleanOperationPolyDataFilter来完成。为了高亮显示两个圆管相交的曲线，用IntersectionPolyDataFilter计算圆管1和圆管2的相交线，图8-21是这部分的流水线：

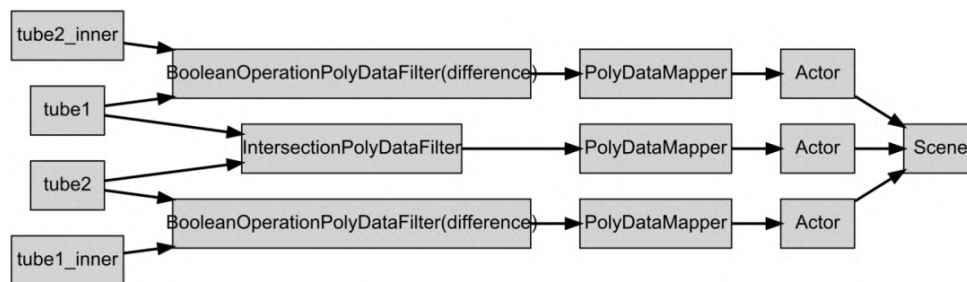


图8-21 计算相贯线的流水线

下面是具体的代码，❶为了清晰显示相贯线，将颜色设置为红色，并将线宽设置为两个像素：

```
def difference(pd1, pd2):
    bf = tvtk.BooleanOperationPolyDataFilter()
    bf.operation = "difference"
    bf.set_input_connection(0, pd1.output_port)
    bf.set_input_connection(1, pd2.output_port)
    m = tvtk.PolyDataMapper(input_connection=bf.output_port,
    scalar_visibility=False)
    a = tvtk.Actor(mapper=m)
    return bf, a

def intersection(pd1, pd2, color=(1.0, 0, 0), width=2.0):
    ipd = tvtk.IntersectionPolyDataFilter()
    ipd.set_input_connection(0, pd1.output_port)
    ipd.set_input_connection(1, pd2.output_port)
    m = tvtk.PolyDataMapper(input_connection=ipd.output_port)
    a = tvtk.Actor(mapper=m)
    a.property.diffuse_color = 1.0, 0, 0 ❶
    a.property.line_width = 2.0
    return ipd, a

tubel_hole, tubel_hole_actor = difference(tubel,
tubel_hole_actor, tube2_inner)
```



```

        tube2_hole, tube2_hole_actor = difference(tube2,
tube1_inner)

        intersecting_line, intersecting_line_actor =
intersection(tube1, tube2)

        tube1_hole_actor.property.opacity = 0.8
        tube2_hole_actor.property.opacity = 0.8
        tube1_hole_actor.property.color = 0.5, 0.5, 0.5
        tube2_hole_actor.property.color = 0.5, 0.5, 0.5

        win = ivtk_scene([tube1_hole_actor, tube2_hole_actor,
intersecting_line_actor])
        win.scene.isometric_view()

```

intersecting_line是由1624条线段构成的PolyData对象，如果希望获得8条表示相贯线的曲线，还需要做进一步处理。

```

print intersecting_line.output.points.number_of_points
print intersecting_line.output.lines.number_of_cells
1624
1650

```

IntersectionPolyDataFilter输出的PolyData对象中，有一些点的坐标非常接近，它们应该是一个点，但是由于计算误差，在PolyData中被表示为多个点。下面用scipy.spatial.distance中的pdist()和squareform()计算所有点之间的距离，并显示最小的10个距离：

```

from scipy.spatial import distance

dist = distance.squareform( distance.pdist(intersecting_line.output.points.to_array()) )
dist[np.diag_indices(dist.shape[0])] = np.inf
dist = dist.ravel()
print np.sort(dist)[:10]

[ 1.60932541e-06  1.60932541e-06  1.66893005e-06  1.66893005e-06
  3.23077305e-06  3.23077305e-06  3.23077305e-06  3.23077305e-06
  3.72555819e-06  3.72555819e-06]

```

下面使用 ClearPolyData 对 PolyData 进行清理工作，tolerance_is_absolute为True时，使用绝对阈值absolute_tolerance对坐标点进行合并，可以看到合并之后的点数减少了：

```

cpd = tvtk.CleanPolyData(tolerance_is_absolute=True,
                          absolute_tolerance=1e-5,
                          input_connection=intersecting_line.output_port)

cpd.update()

cpd.output.points.number_of_points

1578

```

下面的connect_line()根据PolyData.lines属性中的信息，将线段合并成曲线，并使用matplotlib的三维绘图功能绘制合并之后的8条曲线，如图8-22所示。请感兴趣的读者自行分析connect_line()的算法，这里就不多做解释了。

```

from collections import defaultdict

def connect_lines(lines):
    edges = defaultdict(set)

```

```

        for _, s, e in lines.to_array().reshape(-1,
3).tolist():

            edges[s].add(e)

            edges[e].add(s)

while True:

    if not edges:

        break

    poly = [edges.iterkeys().next()]

    while True:

        e = poly[-1]

        neighbours = edges[e]

        if not neighbours:

            break

        n = neighbours.pop()

        try:

            edges[n].remove(e)

        except:

            pass

        poly.append(n)

    yield poly

    edges = {k:v for k,v in edges.iteritems() if v}

from mpl_toolkits.mplot3d import Axes3D

fig = pl.figure(figsize=(5, 5))

ax = fig.gca(projection='3d')

```

```

points = cpd.output.points.to_array()
for line in connect_lines(cpd.output.lines):
    x, y, z = points[line].T
    ax.plot(x, y, z, label='parametric curve')

ax.auto_scale_xyz([-1, 1], [-1, 1], [-1, 1])

```

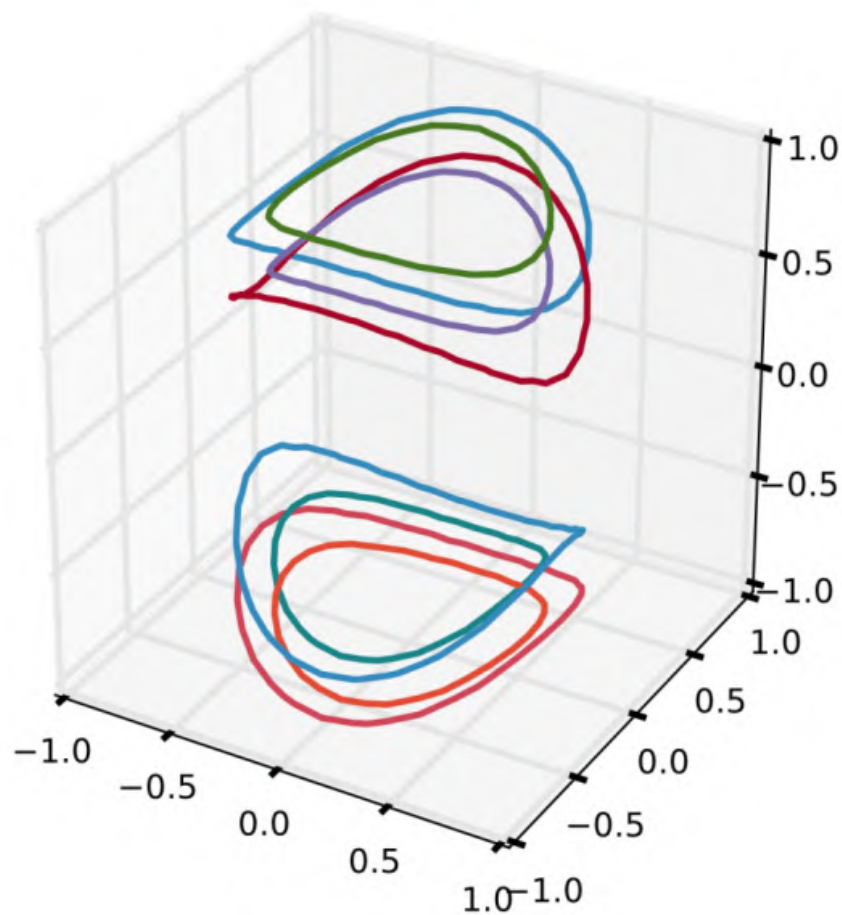


图8-22 用matplotlib绘制提取出的相贯线

8.5 用m1ab快速绘图



与 本 节 内 容 对 应 的 Notebook 为 : 08-tvtk_mayavi/tvtk_mayavi-600-mlab.ipyn。



最新版本的Mayavi 4.4.0中存在GUI操作不更新3D场景的问题，可以通过本书提供的`scipy.tvtk.fix_mayavi_bugs()`修复这些问题。

虽然VTK可视化软件包的功能很强大，Python的TVTK库也很方便简洁，但是用这些工具快速编写实用的三维可视化程序仍然是非常具有挑战性的。因此基于VTK开发出了许多可视化软件，例如ParaView、VTKDesigner2、Mayavi2等。

Mayavi2完全用Python编写，它不但是一个方便实用的可视化软件，而且可以用Python编写扩展，嵌入到用户编写的Python程序中，并且提供了面向脚本的mlab模块，以方便用户快速绘制三维图。和matplotlib的pylab一样，Mayavi的mlab模块提供了方便快捷的三维绘制函数。只要将数据准备好，通常只需要调用一次mlab模块的绘图函数，就可以看到数据的三维显示效果，非常适合在IPython中交互使用。

8.5.1 点和线

三维空间中独立的点使用`point3d()`绘制，而`plot3d()`则将一系列的点连接起来绘制三维曲线。它们可以有3个或4个数组参数。这些数组的形状完全相同，前3个参数`x`、`y`、`z`对应点的X、Y和Z轴的坐标值；而如果有第4个参数`s`，它指定每个坐标点所对应的数值(标量值)。`point3d()`的第4个参数还可以是通过坐标计算标量值的函数。下面是这两个函数的调用格式：

```
plot3d(x, y, z, ...)
plot3d(x, y, z, s, ...) #s为保存每个点对应的标量值的数组

points3d(x, y, z...)
points3d(x, y, z, s, ...) #s为保存每个点对应的标量值的数组
points3d(x, y, z, f, ...) #f为计算每个点对应的标量值的函数
```

`x`、`y`、`z`参数决定了三维空间中每个点的坐标，而每个点所对应的数值则可以用点的颜色、大小、线的粗细等直观地表现。

每个函数还有许多关键字参数用于设置各种绘图属性，例如点的形状、线的宽度、颜色、颜色映射等。所有这些参数能够设置的绘图属性，都可以在显示出图形窗口之后，在流水线对话框中交互式地修改。因此本书不对这些参数进行详细讲解，读者可以阅读函数文档，了解每个关键字参数的含义。

我们以`plot3d()`绘制洛伦茨吸引子轨迹为例，介绍如何绘制三维空间中的曲线，并且使用流水线对话框对图形的各种属性进行调整。下面是绘制洛伦茨吸引子的程序，算法请参照SciPy的相关章节，缺省的绘图结果如图8-23所示。

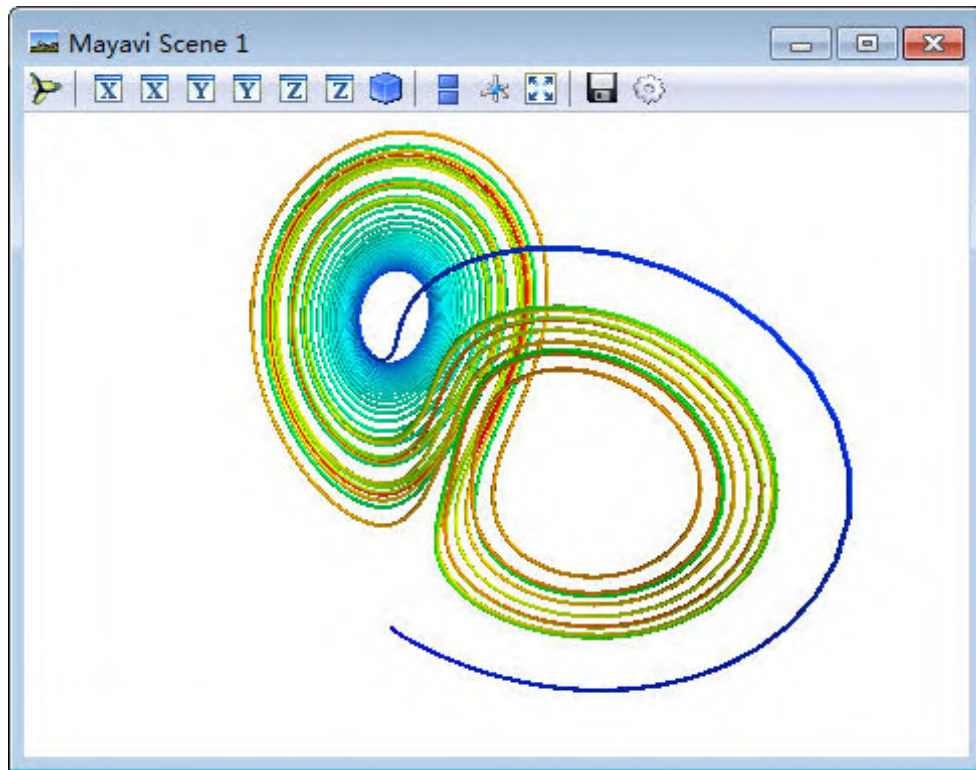


图8-23 plot3d() 绘制的洛伦茨吸引子，曲线使用很细的圆管绘制

```
from scipy.integrate import odeint

def lorenz(w, t, p, r, b):
    x, y, z = w
    return np.array([p*(y-x), x*(r-z)-y, x*y-b*z])

t = np.arange(0, 30, 0.01)
track1 = odeint(lorenz, (0.0, 1.00, 0.0), t, args=(10.0,
28.0, 3.0)) ❶

from mayavi import mlab
X, Y, Z = track1.T
```

```
mlab.plot3d(X, Y, Z, t, tube_radius=0.2) ❷  
mlab.show()
```

❶使用`odeint()`得到的轨迹数据`track1`是一个二维数组，它的第0轴的长度是轨迹的点数，第1轴的长度为3，分别为轨迹上每点的X、Y、Z轴坐标。

❷将`track1`拆分为三个一维数组之后，调用`plot3d()`绘制轨迹曲线。这里用时间数组`t`作为标量值数组，因此轨迹上每个点所对应的标量值就是到达此点的时间。`tube_radius`参数设置曲线的粗细，曲线实际上采用极细的圆管绘制。

在三维场景中可以使用键盘和鼠标对场景照相机或场景中的物体进行操作：

- 照相机模式：此模式为缺省模式，在此模式下所有的鼠标操作都是针对照相机进行，按“C”键切换到此模式。
- 角色模式：通过鼠标可以修改场景中物体的方向和位置，按“A”键切换到此模式。

在照相机模式下：

- 旋转场景：鼠标左键拖动或用键盘的方向键。
- 平移场景：鼠标中键拖动或按住Shift键并使用左键拖动，或者使用Shift+方向键。
- 缩放场景：鼠标右键上下拖动或使用“+”和“-”按键。
- 滚动照相机：按住Ctrl按键并用左键拖动。

窗口中的工具栏还提供了从坐标轴的6个方向观察场景、等角投影、切换平行透视和成角透视等功能的按钮。

8.5.2 Mayavi的流水线

工具栏中最左边的图标是打开Mayavi pipeline对话框的按钮，单击此按钮将弹出如图8-24所示的对话框。左侧用树状控件显示了构成场景的流水线。此流水线是Mayavi在TVTK的流水线之上进行包装的结果。选中流水线中的某个对象之后，窗口右边的部分将显示设置选中对象用的界面。让我们看看`plot3d()`生成的流水线中都有哪些对象，如图8-24所示：

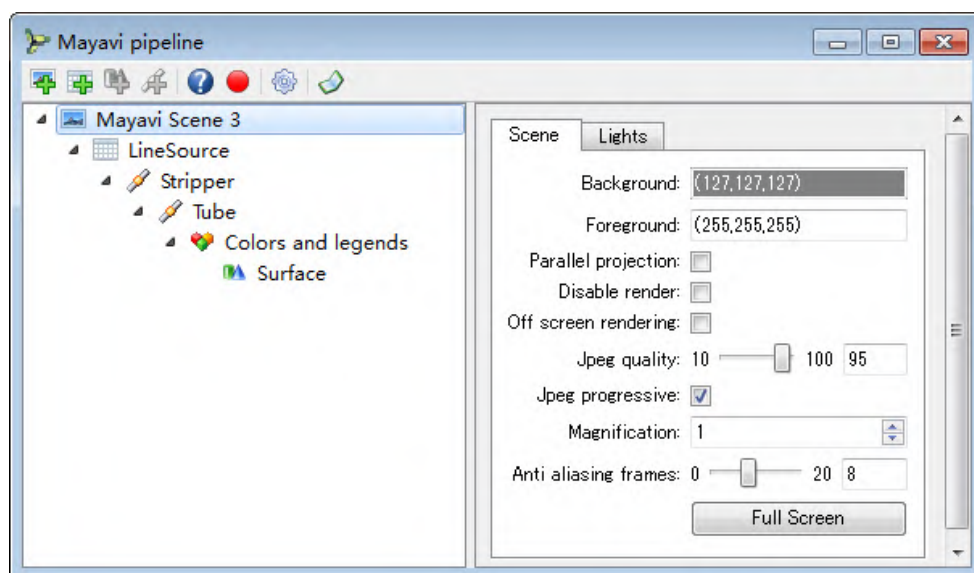


图8-24 `plot3d()` 绘制的洛伦茨吸引子的流水线对话框

Mayavi Scene：处于树的最顶层的对象表示场景。在其配置界面中可以设置场景的背景和前景色、场景中的灯光以及其他一些选项。例如，将背景色“Background”改为灰色，将前景色

“Foreground” 改为白色。也可以用下面的程序获取场景对象的背景色：

```
s = mlab.gcf() # 首先获得当前的场景
print s
print s.scene.background
<mayavi.core.scene.Scene object at 0x13A320F0>
(0.5, 0.5, 0.5)
```

LineSource：线数据源。在其配置界面中，第一项为每个点对应的标量数据的名称，在本例中只有一个名为scalars的标量数据，它就是我们传递给plot3d()的第4个数组：表示轨迹中每点的时间的数组t。下面的语句从场景中获取LineSource对象，并且获取其中的各种数据：

```
source = s.children[0] # 获得场景的第一个子节点，也就是
LineSource
print repr(source)
print source.name # 节点的名字，也就流水线中显示的文字
print repr(source.data.points) # LineSource中的坐标点
print repr(source.data.point_data.scalars) #每个点所对应的标量
数组
<mayavi.sources.vtk_data_source.VTKDataSource object at
0x13A06CF0>
LineSource
      [(0.0, 1.0, 0.0), ..., (0.021550891680468726,
1.6938271906706417, 20.31711497016887)], length = 3000
      [0.0, ..., 29.99], length = 3000
```

Stripper : 根据内部filter对象的maximum_length属性对线数据源进行分段处理。在本例中，输入的线数据源有3000个点，而maximum_length属性为1000，即每1000个点将对应一条线。因此stripper对象输出的PolyData对象中有3条线：

```
stripper = source.children[0]
print stripper.filter.maximum_length
print stripper.outputs[0].number_of_points
print repr(stripper.outputs[0])
print stripper.outputs[0].number_of_lines
1000
3000
<tvtk.tvtk_classes.poly_data.PolyData object at 0x0CD527B0>
3
```

Tube : 将输入的PolyData数据中的每条线转换为表示三维圆管的PolyData数据。它的配置界面中有许多参数可以改变生成圆管的方式。例如：将“Vary radius”设置为“vary_radius_by_scalar”，则圆管的粗细由每个点对应的标量值决定。而加粗的比例则由“Radius factor”参数决定，我们将其设置为3，于是此圆管最粗处(终点)的半径是最细处(起点)的3倍。下面的语句获得Tube对象，并查看输出对象的类型：

```
tube = stripper.children[0] # 获得Tube对象
print repr(tube.outputs[0]) # tube的输出是一个PolyData对象，它是一个三维圆管
<tvtk.tvtk_classes.poly_data.PolyData object at 0x0CD52210>
```

Colors and legends : 在其配置界面中的“Scalar LUT”选项卡中可以设置将标量值转换为颜色的查询表(Look Up Table)。例如, 将“Lut mode”改为Blues, 于是场景中的圆管完成了从白色到深蓝色的渐变。勾选“Show legend”选择框, 在场景中将添加一个颜色条来显示颜色和标量值之间的关系。也可以通过下面的程序修改这两个选项:

```
manager = tube.children[0]
manager.scalar_lut_manager.lut_mode = 'Blues'
manager.scalar_lut_manager.show_legend = True
```

Surface : 它将Tube所输出的PolyData数据转换为最终在场景中显示的三维实体。通过其配置界面的“Actor”选项卡, 可以对实体进行配置。例如, 将“Representation选择为“wireframe”, 并将“Line width”设置为0, 则实体采用细线框模型显示。将“Opacity”设置为0.6, 则实体变为半透明状态。也可以通过下面的程序修改这两个配置:

```
surface = manager.children[0]
surface.actor.property.representation = 'wireframe'
surface.actor.property.opacity = 0.6
```

修改之后的场景如图8-25所示。

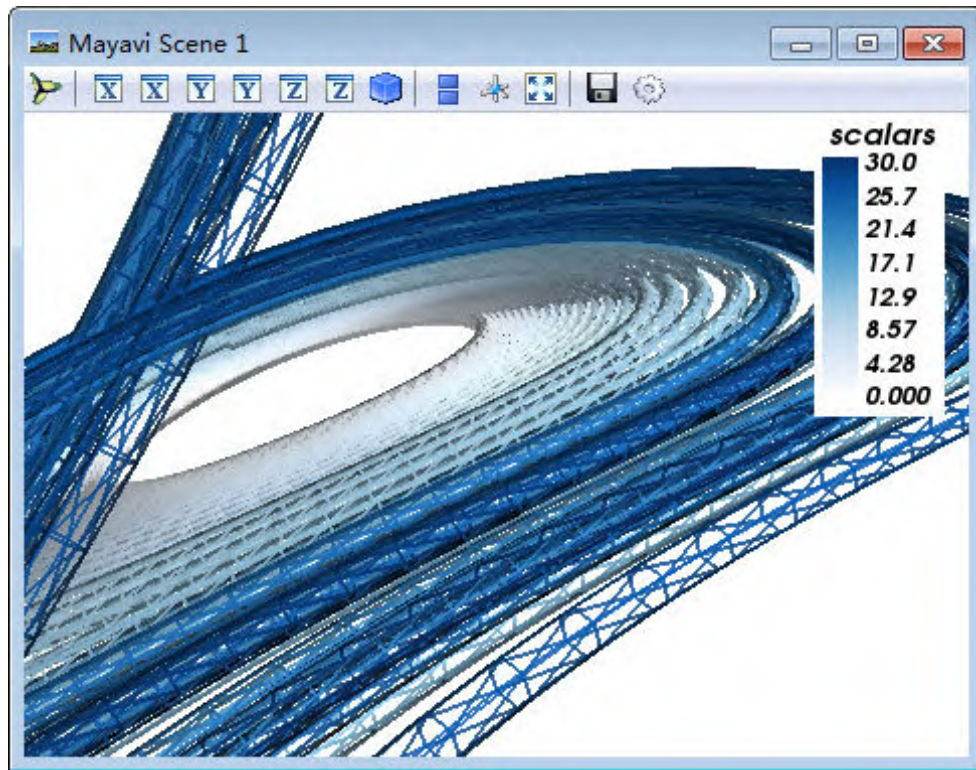


图8-25 在流水线对话框中修改了许多配置之后的洛伦茨吸引子轨迹

下面是用程序配置这些属性的步骤：

- (1) 先获得场景对象，例如用`mlab.gcf()`。
- (2) 通过每个对象的`children`属性，在流水线中找到需要修改的对象。
- (3) 当其配置窗口有多个选项卡或多个配置分组框时，意味着其属性可能需要一级一级地获得。对象的属性名和界面上的文字之间有很简单的转换关系：首字变大写、下划线变空格。例如：Surface对象的Actor选项卡中的Property分组框中的“Line width”选项，用程序描述就是：

```
surface.actor.property.line_width  
2.0
```

Mayavi还提供了脚本录制功能，以方便我们编写配置各种属性的程序。单击流水线对话框的工具栏中的红色圆形图标即可开始脚本录制，并且打开一个脚本对话框。之后的界面配置操作，都会被记录到此脚本对话框中。

8.5.3 二维图像的可视化

三维空间中的曲面可以用`surf()`绘制，它实际上是将二维图像(数组)绘制成三维的曲面，用曲面的高度表示图像中每点的值。下面的程序绘制图8-26所示的曲面。

```
x, y = np.ogrid[-2:2:20j, -2:2:20j] ❶  
z = x * np.exp( - x**2 - y**2) ❷  
  
face = mlab.surf(x, y, z, warp_scale=2) ❸  
axes = mlab.axes(xlabel='x', ylabel='y', zlabel='z', color=  
(0, 0, 0)) ❹  
outline = mlab.outline(face, color=(0, 0, 0))
```

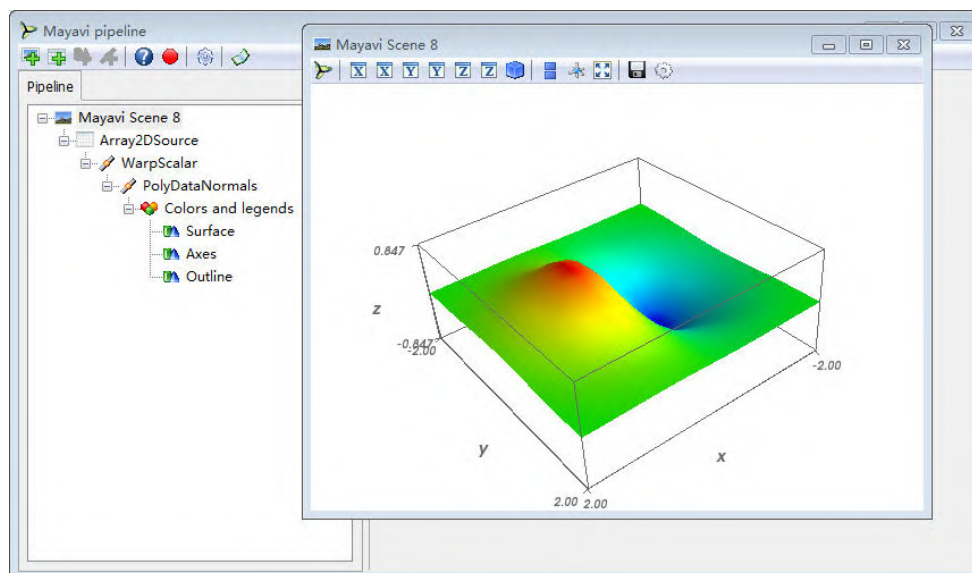


图8-26 surf() 绘制的曲面及流水线对话框

①先通过ogrid对象计算两个形状分别为(20, 1)和(1, 20)的数组x和y。②然后通过广播运算，计算出由x和y构成的等距网格上每点的函数值z，它是一个形状为(20, 20)的数组。③接着调用mlab.surf()，将数组z绘制成三维空间中的曲面，所绘制的曲面在X-Y平面上的投影是一个等距离网格。④最后通过mlab.axes()和mlab.outline()，分别在三维场景中添加坐标轴和曲面区域的外框。需要注意的是，与matplotlib相反，在Mayavi中二维数组的第0轴表示X轴，第1轴表示Y轴。

仔细观察曲面的颜色就会发现颜色和曲面的高度有一一对应的关系，曲线上的点越高，颜色越红，越低则颜色越蓝。打开流水线对话框，可以看到数据源是一个Array2DSource对象。它的输出是一个TVTK的ImageData对象：

```
data = mlab.gcf().children[0]
img = data.outputs[0]
```

```
img  
<tvtk.tvtk_classes.image_data.ImageData at 0x13a243f0>
```

ImageData对象是一个表示三维图像的数据集。在ImageData对象中，实际上不保存空间中每点的坐标，而是通过origin、spacing和dimensions等属性计算出一个三维空间中的等距离网格，网格中的每个点所对应的标量值保存在point_data.scalars中，这些值决定了曲面的高度和颜色。

```
print img.origin # X、Y、Z轴的起点  
print img.spacing # X、Y、Z轴上点的间隔  
print img.dimensions # X、Y、Z轴上点的个数  
print repr(img.point_data.scalars) # 每个点所对应的标量值  
[-2. -2.  0.]  
[ 0.21052632  0.21052632  1.          ]  
[20 20  1]  
[-0.000670925255805, ..., 0.000670925255805], length = 400
```

通过上面的分析可以看出，surf()的功能是将一个二维图像转换为三维空间中的曲面。因此曲面上每个点的X、Y轴的坐标都是通过网格配置计算出来的。

由于曲面的高度和其X-Y平面上的尺寸可能相差很大，因此流水线中，在Array2DSource对象的下面是一个WarpScalar对象，它将输入数据沿着Z轴方向进行缩放，可以看到其配置面板中的“Scale factor”为2，它是由surf()的warp_scale参数决定的。WarpScalar对象的输出是一个PolyData对象：


```
data.children[0].outputs[0]  
<tvtk.tvtk_classes.poly_data.PolyData at 0x14263720>
```

流水线中剩下的对象请读者自己研究，通过研究流水线可以了解Mayavi内部的组织构造，这有助于我们创建自己的流水线以对复杂的数据进行可视化。

如果数据在三个坐标轴上的范围相差很大，在进行可视化时需要调整坐标轴的显示比例，以达到更好的可视化效果。例如在下面的曲面函数中，X轴方向需要更大的显示范围：

```
x, y = np.ogrid[-10:10:100j, -1:1:100j]  
z = np.sin(5*((x/10)**2+y**2))
```

如果直接使用数据的范围进行显示，效果如图8-27(左)所示，虽然可以很直观地看出X轴的显示范围是Y轴的10倍，但是很难观察曲面的一些细节信息。

```
mlab.surf(x, y, z)  
mlab.axes()
```

通过surf()的extent参数可以修改坐标轴的数据范围：

```
mlab.surf(x, y, z, extent=(-1,1,-1,1,-0.5,0.5))  
mlab.axes(nb_labels=5)
```

extent参数是一个有6个元素的序列，分别指定X轴最小值、X轴最大值、Y轴最小值、Y轴最大值、Z轴最小值、Z轴最大值。这些值将修改数据范围，因此所绘制曲面的X、Y轴范围相同，而曲面在Z轴上的高度是X、Y轴范围的一半，效果如图8-27(中)所示。由于extent参数所

改变的是数据的范围，因此坐标轴上的刻度值也随之发生变化。为了解决这个问题，可以给`axes()`传递`ranges`参数：

```
mlab.surf(x, y, z, extent=(-1,1,-1,1,-0.5,0.5))  
  
mlab.axes(ranges=  
(x.min(),x.max(),y.min(),y.max(),z.min(),z.max()), nb_labels=5)
```

`ranges`参数也是一个有6个元素的序列，分别指定三个坐标轴上的刻度范围，效果如图8-27(右)所示。

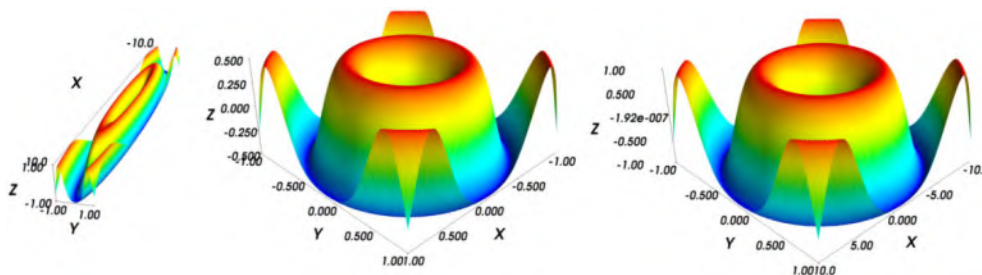


图8-27 修改坐标轴的显示比例

除`surf()`之外，`imshow()`和`contour_surf()`也是可视化二维图像的工具。`imshow()`将二维图像放在三维空间中显示，它与将`surf()`的`warp_scale`参数设置为0的效果一样。下面的语句将二维数组`z`用`imshow()`绘制成图，效果如图8-28(左)所示：

```
x, y = np.ogrid[-2:2:20j, -2:2:20j]  
z = x * np.exp( - x**2 - y**2)  
  
mlab.imshow(x, y, z)  
mlab.show()
```

而`contour_surf()`和`surf()`的参数类似，但可以通过`contours`参数指定等高线的数目或者等高值的列表。下面的语句将曲面以20条等高线表示，如图8-28(右)所示。

```
mlab.contour_surf(x,y,z,warp_scale=2,contours=20)
```

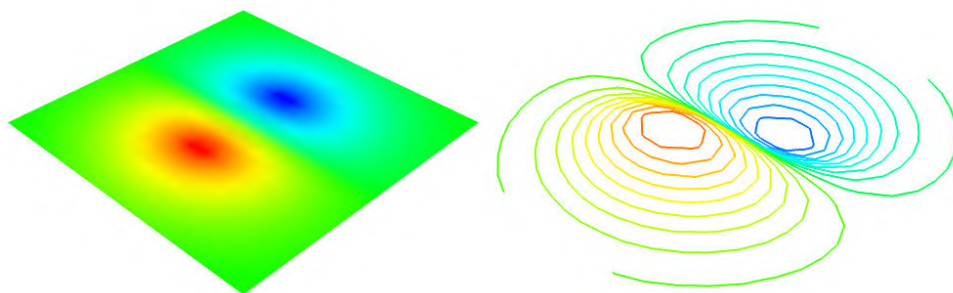


图8-28 用`imshow`绘制图像(左)，用`contour_surf`绘制等高线(右)

在用`surf()`绘制曲面之后，在流水线对话框中对`Surface`对象进行如下配置，也可以实现和`contour_surf()`一样的效果：

- 在“Contours”选项卡中，勾选“Enable Contours”。
- 勾选“Auto contours”选项，并且指定“Number of contours”为20，这样会自动产生20条等高线。
- 也可以不勾选“Auto contours”选项，然后手工添加等高线。

或者用下面的程序设置等高线，其中`face`为`surf()`返回的对象，也就是流水线中的`Surface`：

```
face.enable_contours = True  
face.contour.number_of_contours = 20
```

8.5.4 网格面mesh

如果需要绘制更复杂的三维曲面，可以使用`mesh()`。下面是使用`mesh()`绘制复杂曲面的程序，结果如图8-29所示：

```
from numpy import sin, cos
dphi, dtheta = np.pi/80.0, np.pi/80.0
phi, theta = np.mgrid[0:np.pi+dphi*1.5:dphi,
0:2*np.pi+dtheta*1.5:dtheta]
m0, m1, m2, m3, m4, m5, m6, m7 = 4,3,2,3,6,2,6,4
r = sin(m0*phi)**m1 + cos(m2*phi)**m3 + sin(m4*theta)**m5 +
cos(m6*theta)**m7 ❶
x = r*sin(phi)*cos(theta) ❷
y = r*cos(phi)
z = r*sin(phi)*sin(theta)
s = mlab.mesh(x, y, z) ❸

mlab.show()
```

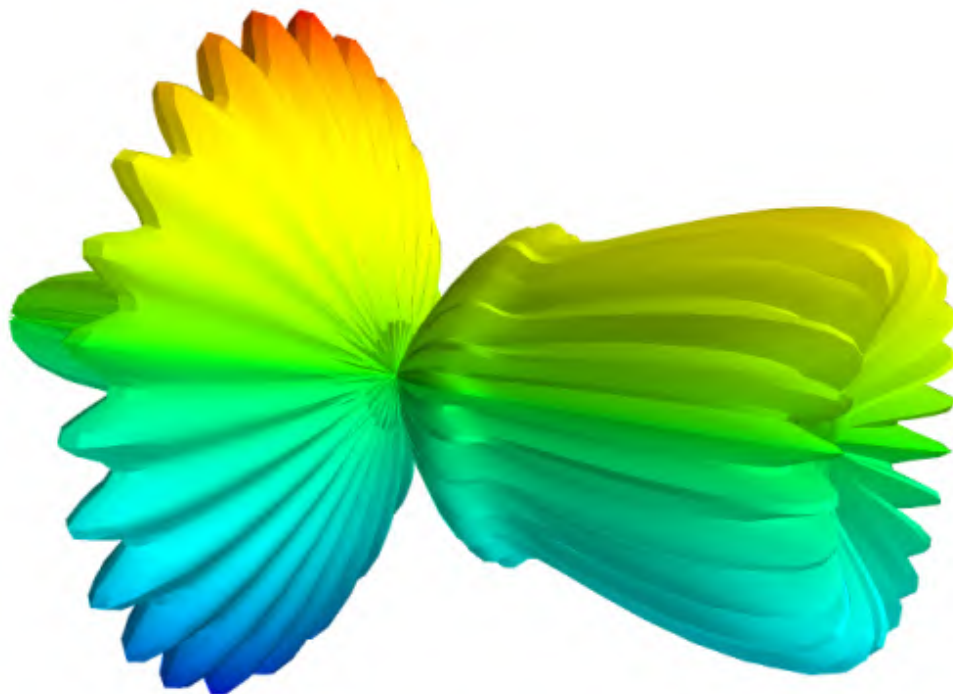


图8-29 使用mesh函数绘制的3D旋转体

③程序中调用mesh()绘制曲面，它和surf()类似，其三个数组参数x、y、z都是形状相同的二维数组。这些数组的相同下标的三个数值组成曲面上某点的三维坐标。点之间的连接关系(边和面)由其在x、y、z数组中的位置关系决定。①曲面上各点的坐标在球坐标系中计算，②然后按照坐标转换公式将球坐标系转换为笛卡尔坐标系。

为了方便读者理解mesh()是如何绘制出曲面的，下面通过手工输入坐标的方式，绘制如图8-30所示的立方体表面的一部分：

```
x = [[-1,1,1,-1,-1],  
      [-1,1,1,-1,-1]]  
  
y = [[-1,-1,-1,-1,-1],  
      [ 1, 1, 1, 1, 1]]
```

```

z = [[1,1,-1,-1,1],
      [1,1,-1,-1,1]]

box = mlab.mesh(x, y, z, representation="surface")
mlab.axes(xlabel='x', ylabel='y', zlabel='z')
mlab.outline(box)
mlab.show()

```

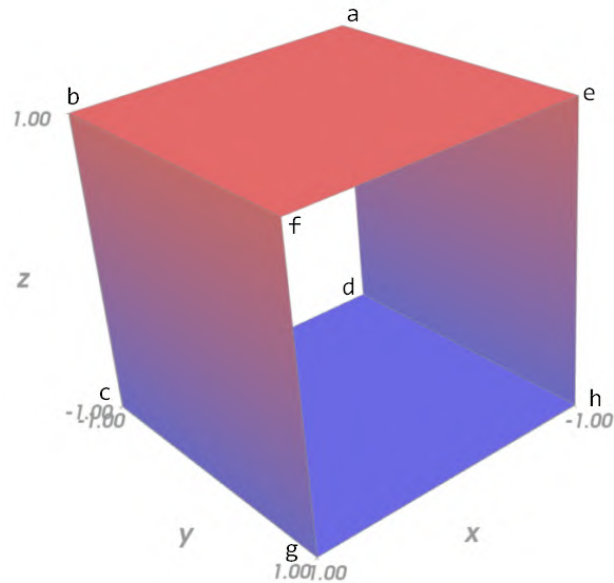


图8-30 组成立方体的各个面和顶点坐标

传递给`mesh()`的三个数组`x`、`y`、`z`中下标相同的三个数字组成一个三维坐标，因此这三个数组实际描述的坐标点为：

```

[
    [(-1, -1, 1), (1, -1, 1), (1, -1, -1), (-1, -1, -1),
     (-1, -1, 1)],
    [(-1, 1, 1), (1, 1, 1), (1, 1, -1), (-1, 1, -1),
     (-1, 1, 1)]
]

```

```
(-1, 1, 1)]  
]
```

为了理解方便，图中将上面的坐标点用字母表示：

```
[[a,b,c,d,a],  
 [e,f,g,h,e]]
```

坐标点之间的关系由其在数组中的位置决定，因此下面的4组坐标点构成4个正方形平面：

```
a, b, f, e ->顶面  
b, c, g, f ->左面  
c, d, h, g ->底面  
d, a, e, h ->右面
```

使用`mesh()`可以很方便地将二维平面上的曲线绕着对称轴进行旋转得到旋转面。下面的程序用`mesh()`绘制由抛物线旋转之后得到的旋转抛物面，如图8-31所示。

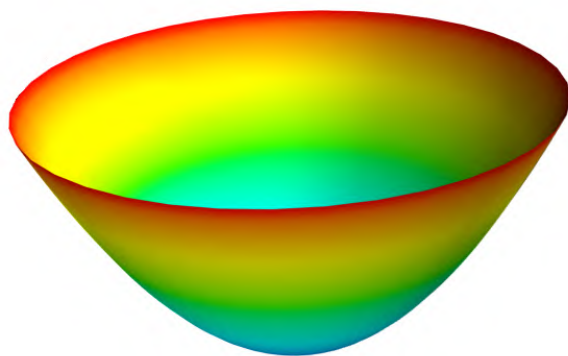


图8-31 用`mesh()`绘制旋转抛物面

```

rho, theta = np.mgrid[0:1:40j, 0:2*np.pi:40j] ❶

z = rho*rho ❷

x = rho*np.cos(theta) ❸
y = rho*np.sin(theta)

s = mlab.mesh(x, y, z)
mlab.show()

```

旋转面上点的坐标很容易在圆柱坐标系 (ρ , ϕ , z) 中计算。❶首先在 (ρ , ϕ) 平面中创建一个 40×40 的二维网格。❷通过 ρ 计算出抛物面上每点的高度 z 。由于是旋转面，因此高度和 ϕ 无关。❸将圆柱坐标系转换为直角坐标系，得到`mesh()`所需的三个数组。

还可以使用`mesh()`绘制出`surf()`所绘制的曲面。下面是用`mesh()`绘制曲面的程序：

```

x, y = np.mgrid[-2:2:20j, -2:2:20j] ❶

z = x * np.exp( - x**2 - y**2)

z *= 2

c = 2*x + y ❷

pl = mlab.mesh(x, y, z, scalars=c) ❸
mlab.axes(xlabel='x', ylabel='y', zlabel='z')
mlab.outline(pl)
mlab.show()

```


①这里使用mgrid对象产生数组x和y。这是因为用mesh()绘制曲面时，必须给出曲面上每点的坐标值，因此x和y参数不能是ogrid产生的数组。

②为了演示mesh()绘制曲面的优点，我们另外计算一个二维数组c，③并且把它传递给mesh()的scalars参数。于是曲面上每点所对应的标量值将使用数组c中相应下标的值。这样可以使用数组c对表面上的每点进行着色，得到一个颜色和高度无关的曲面，它能比surf()所绘制的曲面表达更多的信息。图8-32为程序绘制结果及对应的流水线。

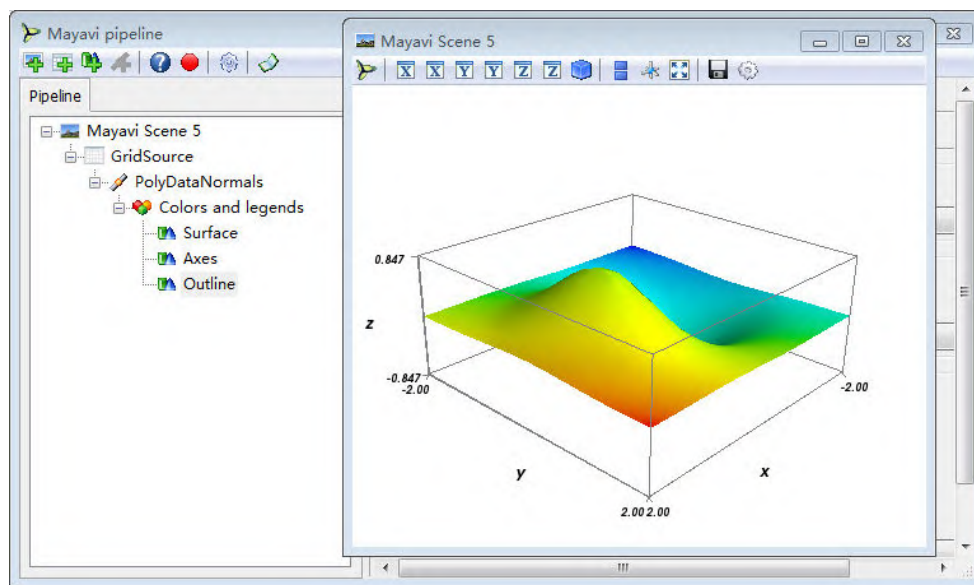


图8-32 用mesh()绘制高度和颜色不同的曲面

8.5.5 修改和创建流水线

用surf()也可以绘制高度和颜色不同的曲面，但是需要对流水线进行一些改动。首先使用之前介绍的方法用surf()绘制曲面：

```

x, y = np.ogrid[-2:2:20j, -2:2:20j]
z = x * np.exp( - x**2 - y**2)

face = mlab.surf(x, y, z, warp_scale=2)
mlab.axes(xlabel='x', ylabel='y', zlabel='z')
mlab.outline(face)

```

我们需要为每个点添加一个新的标量值以控制它们的颜色。因此，首先获得流水线中Array2DSource对象内部的ImageData对象。Array2DSource实际上是一个ArraySource对象，通过查看其源代码可知它用image_data属性保存所创建的ImageData对象。

```

source = mlab.gcf().children[0]
print source
img = source.image_data
print repr(img)

```

<mayavi.sources.array_source.ArraySource object at
 0x127FCE70>

<tvtk.tvtk_classes.image_data.ImageData object at
 0x127C5960>

然后给img添加一个标量数组，并且命名为“color”。注意数组c的第0轴对应X轴而第1轴对应Y轴，因此需要将其转置，而ravel()方法则将二维数组变为一维数组：

```

c = 2*x + y # 表示颜色的标量数组
array_id = img.point_data.add_array(c.T.ravel())
img.point_data.get_array(array_id).name = "color"

```

修改了ArraySource对象的输入数据之后，调用update() 计算其输出数据，并设置pipeline_changed事件属性，让流水线上后续的对象更新它们的输入输出。

```
source.update()

source.pipeline_changed = True
```

如果读者对为什么需要转置有疑问，查看一下数组z在ImageData对象中的保存顺序，可以看出二维数组z的数据是按照第0轴、第1轴的顺序保存在scalars数组中的：

```
print z[:3,:3] # 原始的二维数组中的元素
# ImageData中的标量值的顺序
print img.point_data.scalars.to_array()[:3] # 和数组z的第0列
的数值相同

[[-0.00067093 -0.00148987 -0.00302777]
 [-0.00133304 -0.00296016 -0.00601578]
 [-0.00239035 -0.00530804 -0.01078724]]
[-0.00067093 -0.00133304 -0.00239035]
```

接下来需要在流水线的PolyDataNormals和Colors and legends之间插入一个SetActiveAttribute对象，它将PolyDataNormals的输出数据中名为“color”的标量数组设置为当前标量数组。下面先获得PolyDataNormals对象：

```
normals = mlab.gcf().children[0].children[0].children[0]
```

通过下面的语句可以看到，PolyDataNormals输出的PolyData对象的当前标量数组为数组z：

```
normals.outputs[0].point_data.scalars.to_array()[:3]
array([-0.00067093, -0.00133304, -0.00239035])
```

接下来是插入操作。首先获得normals的下一级对象，并将其从children列表中删除：

```
surf = normals.children[0]
del normals.children[0]
```

然后调用 pipeline.set_active_attribute()，创建一个SetActiveAttribute对象并将其添加进normals.children列表。通过point_scalars参数将名为“color”的数组设置为缺省标量值：

```
active_attr = mlab.pipeline.set_active_attribute(normals,
point_scalars="color")
```

最后将surf对象添加进SetActiveAttribute对象的子列表：

```
active_attr.children.append(surf)
```

可以看到，现在PolyDataNormals输出的PolyData对象的当前标量数组已经变为数组c了：

```
normals.children[0].outputs[0].point_data.scalars.to_array()
[:3]
array([-6. , -5.57894737, -5.15789474])
```

于是其后的颜色查询表将使用数组c作为输入，从而使得曲面的高度和曲面的颜色分别使用不同的数据进行描绘。最终的绘图效果和相

应的流水线如图8-33所示。

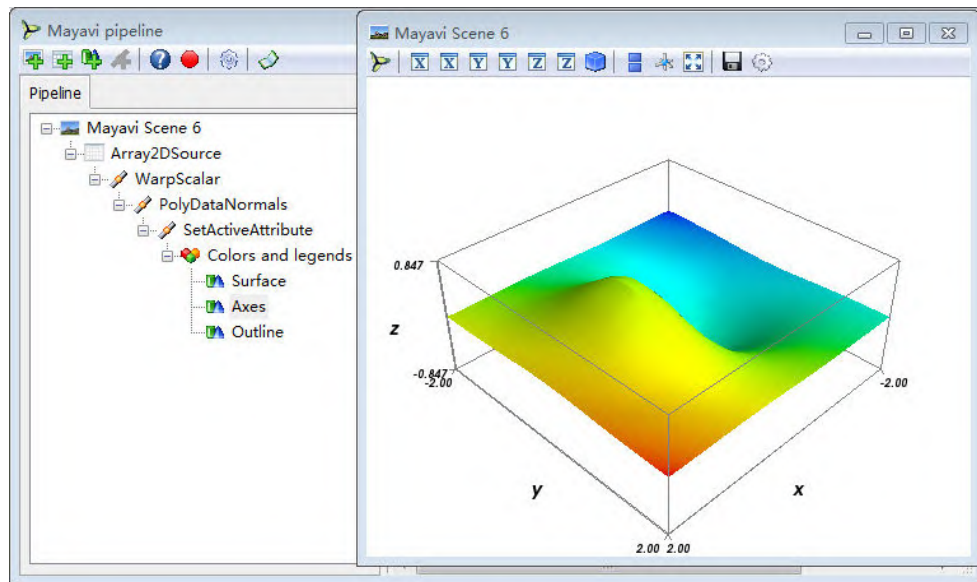


图8-33 用surf() 绘制高度和颜色不同的曲面

也可以不调用surf(), 而直接创建流水线中的每个对象。完整的程序如下:

```
src = mlab.pipeline.array2d_source(x, y, z) #创建ArraySource
数据源

#添加color数组
image = src.image_data
array_id = image.point_data.add_array(c.T.ravel())
image.point_data.get_array(array_id).name = "color"
src.update() #更新数据源的输出

# 创建流水线上的后续对象
warp = mlab.pipeline.warp_scalar(src, warp_scale=2.0)
normals = mlab.pipeline.poly_data_normals(warp)
```

```
active_attr = mlab.pipeline.set_active_attribute(normals,
    point_scalars="color")

surf = mlab.pipeline.surface(active_attr)

mlab.axes()

mlab.outline()

mlab.show()
```

直接创建流水线需要开发者对Mayavi的流水线上的各种对象十分了解，因此建议读者首先熟悉Mayavi的界面操作以及各种对象的作用，然后通过录制脚本逐步学习。

8.5.6 标量场

前面介绍了如何对一维和二维数据进行可视化，下面看看三维数据的可视化问题。最简单的三维数据就是三维图像，可以用一个三维数组表示。图像中每个点的三维坐标都由它在数组中的下标决定，而每个点对应的标量值则是数组中对应元素的值。这种三维图像可以用来描述标量场，例如房间中的温度分布、材料的密度分布或者通过X射线断层成像(CT)技术采集到的人体的内部构造。

标量场有三种可视化方法：

- 等值面：和二维图像的等高线类似，用标量值相等的曲面，显示标量场的形态。
- 体素呈像：利用透明度和颜色直接呈现标量场的形态。
- 切面：通过对标量场进行切面处理，显示在某个切面上场的形态。



scipy2.tvtk.mlab_scalar_field: 使用等值面、体素呈像和切面可视化标量场。

我们用下面的程序计算一个有两个点电荷的电势场，两个点电荷分别位于 $(-1, 0, 0)$ 和 $(1, 0, 0)$ 处。为了方便显示，只计算Z轴上 $(-2, 0)$ 区间的电势场：

```
x, y, z = np.ogrid[-2:2:40j, -2:2:40j, -2:0:40j]
s = 2/np.sqrt((x-1)**2 + y**2 + z**2) + 1/np.sqrt((x+1)**2
+ y**2 + z**2)
```

使用`contour3d()`可以快速绘制此电势场的等值面：

```
surface = mlab.contour3d(s)
```

缺省情况下，`contour3d()`绘制5个等分标量值范围的等值面，它不能很好地显示整个电势场的结构，因此用下面的语句修改等值面的范围、数目以及透明度等属性：

```
surface.contour.maximum_contour = 15 # 等值面的上限值为15
surface.contour.number_of_contours = 10 # 在最小值到15之间绘制
10个等值面
surface.actor.property.opacity = 0.4 # 透明度为0.4
```

这些属性也可以通过流水线对话框来修改，程序的绘制结果如图8-34所示。

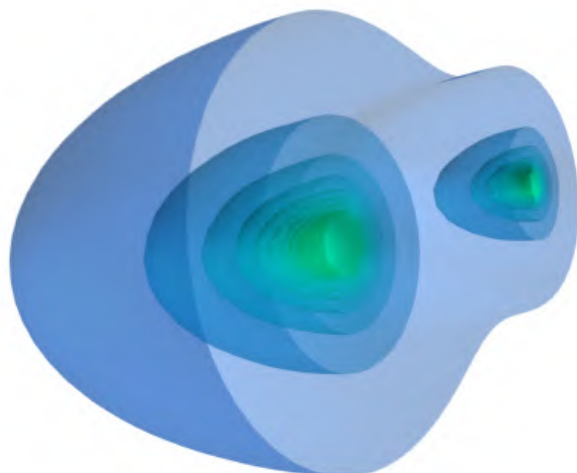


图8-34 用等值面可视化电势场

使用等值面对标量场进行可视化时，外面的等值面可能会完全包含内部的等值面，观察不到内部的状态。例如在本例中，如果将Z轴的计算范围改为 $(-2, 2)$ ，并且不设置等值面透明度，则无论绘制多少个等值面，都只能看到最外层的等值面。

体素呈像法用每个点的颜色和透明度对整个标量场进行润色，从而能够呈现更多的信息。体素呈像没有对应的函数，需要我们自己创建流水线：

```
field = mlab.pipeline.scalar_field(s)
mlab.pipeline.volume(field)
```

此程序首先通过`scalar_field()`在流水线中创建一个标量场数据源，然后通过`volume()`将此数据源用体素呈像进行可视化，效果如图8-35(左)所示。

由于电势强度随着距离的平方衰减，因此整体的润色效果并没有突出电势强的部分。为了解决这个问题，可以给`volume()`传递两个关

键字参数：`vmin`和`vmax`。它们指定标量值的润色范围，即只绘制标量值在`vmin`到`vmax`之间的区域：

```
mlab.pipeline.volume(field, vmin=1.5, vmax=10)
```

效果如图8-35(右)所示，它很清楚地表现出了电荷附近的电势情况。



图8-35 用体素呈像可视化电势场：(左)缺省效果，(右)通过`vmin`和`vmax`指定电势值的润色范围

还可以使用切片工具观察标量场在某个平面之上的数据，它通常和其他的工具同时使用，并且可以直接在三维场景中交互式地改变平面的位置和方向。下面的程序在流水线中添加一个标量切面，在流水线中它是“Colors and legends”的子节点。通过`plane_orientation`参数指定切面的法线方向为Y轴，即切面和Y轴垂直：

```
cut = mlab.pipeline.scalar_cut_plane(field.children[0],  
plane_orientation="y_axes")
```

然后通过下面的程序设置切面工具的一些属性：

```
cut.enable_contours = True # 开启等高线显示  
cut.contour.number_of_contours = 40 # 等高线的数目为40
```

切面工具的效果如图8-36所示，图中还显示了添加切面工具之后的流水线。在3D场景中可以对切面工具进行如下操作：

- 拖动切面的红色外框修改切面的位置。
- 拖动切面的法线箭头修改切面的方向。
- 拖动法线和切面相交的灰色圆球，改变切面的旋转中心。

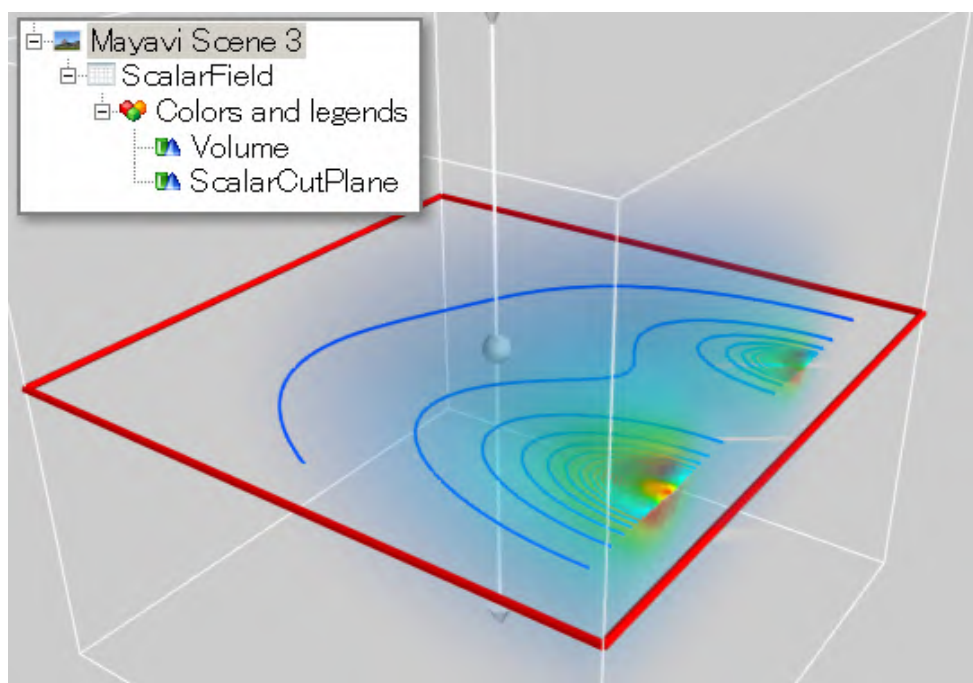


图8-36 用切面工具观察电势场

8.5.7 矢量场

如果场中每一点的属性都可以用矢量代表，那么这个场就是一个矢量场。对于三维空间中的场，每个点对应一个矢量，它由X、Y、Z轴

上的三个分量组成。因此需要3个三维数组表示矢量场，这些数组分别表示矢量在三个轴上的分量。下面以洛伦茨吸引子为例，介绍对矢量场进行可视化的一些基本方法。



scipy2.tvtk.mlab_vector_field: 使用矢量箭头、切片、等梯度面和流线显示矢量场。

下面的程序根据洛伦茨吸引子的公式计算出X、Y、Z轴三个方向上的速度分量u、v、w：

```
p, r, b = (10.0, 28.0, 3.0)
x, y, z = np.mgrid[-17:20:20j, -21:28:20j, 0:48:20j]
u, v, w = p*(y-x), x*(r-z)-y, x*y-b*z
```

这三个速度分量构成一个矢量场，下面调用quiver3d()将每个点的速度矢量用一个箭头表示：

```
vectors = mlab.quiver3d(x, y, z, u, v, w)
```

效果如图8-37(左)所示。由于矢量场数据的网格过密，无法看清矢量场的内部结构。此时可以用下面的语句修改Vectors对象的一些属性以减少箭头的数量并增加箭头的长度，效果如图8-37(右)所示。

```
vectors.glyph.mask_input_points = True # 开启使用部分数据的选择项
vectors.glyph.mask_points.on_ratio = 20 # 随机选择原始数据中的
```

1/20个点进行描绘

```
vectors.glyph.glyph.scale_factor = 5.0 # 设置箭头的缩放比例
```

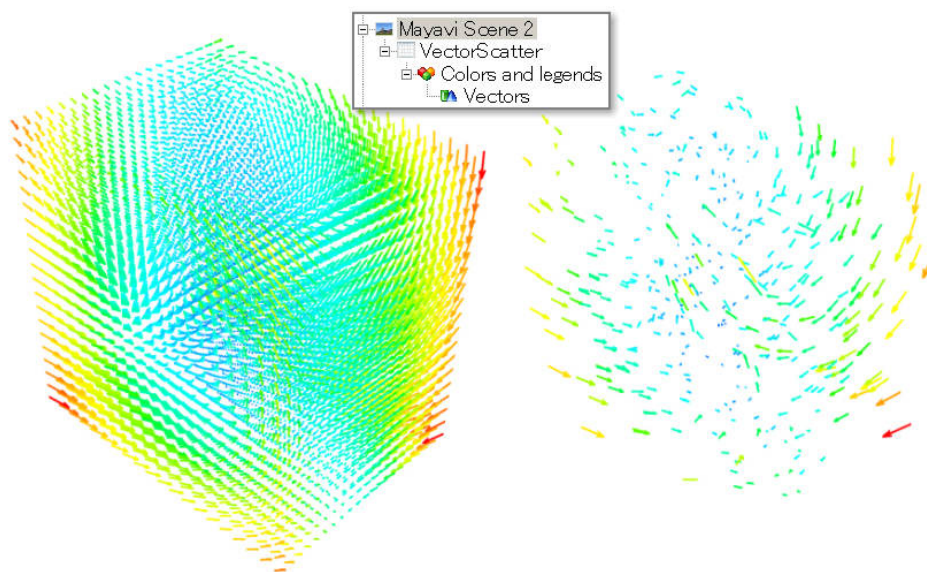


图8-37 用矢量箭头可视化矢量场

和标量场的切面工具一样，也可以对矢量场进行切面显示，这样可以观察矢量场在某个切面上的形态：

```
src = mlab.pipeline.vector_field(x, y, z, u, v, w)
    cut_plane = mlab.pipeline.vector_cut_plane(src,
scale_factor=3)
    cut_plane.glyph.mask_points.maximum_number_of_points =
10000
    cut_plane.glyph.mask_points.on_ratio = 2
    cut_plane.glyph.mask_input_points = True
```

还可以通过矢量场计算标量场。下面通过`extract_vector_norm()`在流水线中添加一个`ExtractVectorNorm`对象，它将每个点所对应的矢量的长度设置为此点的标量值：

```
magnitude = mlab.pipeline.extract_vector_norm(src)
```

于是可以对magnitude表示的标量场绘制等值面：

```
surface = mlab.pipeline.iso_surface(magnitude)
surface.actor.property.opacity = 0.3
```

图8-38(左)是矢量切面工具和等模值面的显示效果。下面的语句分别获取magnitude所输出的ImageData对象的标量数组和矢量数组：

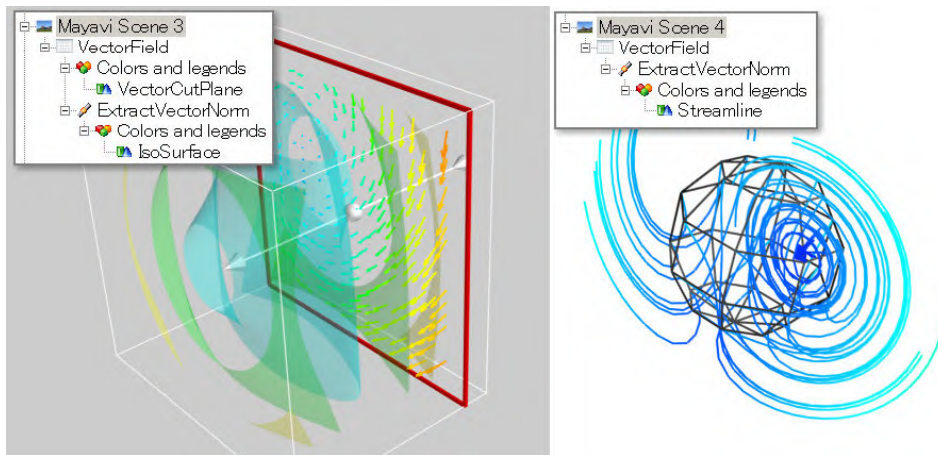


图8-38 用矢量切面和等模值面可视化矢量场(左)，用flow()观察轨迹(右)

```
print repr(magnitude.outputs[0].point_data.scalars)
print repr(magnitude.outputs[0].point_data.vectors)
[579.71887207, ..., 602.195983887], length = 8000
[(-40.0, -455.0, 357.0), ..., (80.0, -428.0, 416.0)],
length = 8000
```

最后，还可以使用flow()观察洛伦茨吸引子的轨迹。它相当于绘制矢量场的场线，空间中每点所对应的矢量等于过此点的场线的切线方向：

```
mlab.flow(x, y, z, u, v, w)
```

图8-38(右)是使用`flow()`绘制的洛伦茨吸引子轨迹。以图中球体上的每点为初始点计算它们所对应的场线轨迹。流水线中的Streamline对象有许多配置选项，请读者自行研究。

8.6 将TVTK和Mayavi嵌入界面

虽然可以使用TVTK和Mayavi提供的流水线控件修改流水线中各个对象的参数，但是在完整的三维可视化程序中，除了三维场景之外，通常还需要许多界面控件用来帮助用户与场景交互。TVTK和Mayavi都是建立在Trait库之上，因此可以很方便地使用TraitsUI界面库。

8.6.1 TVTK场景的嵌入

在下面的例子中，用户可以通过如图8-39所示界面中的两个滚动条控制场景中圆管的内外半径。

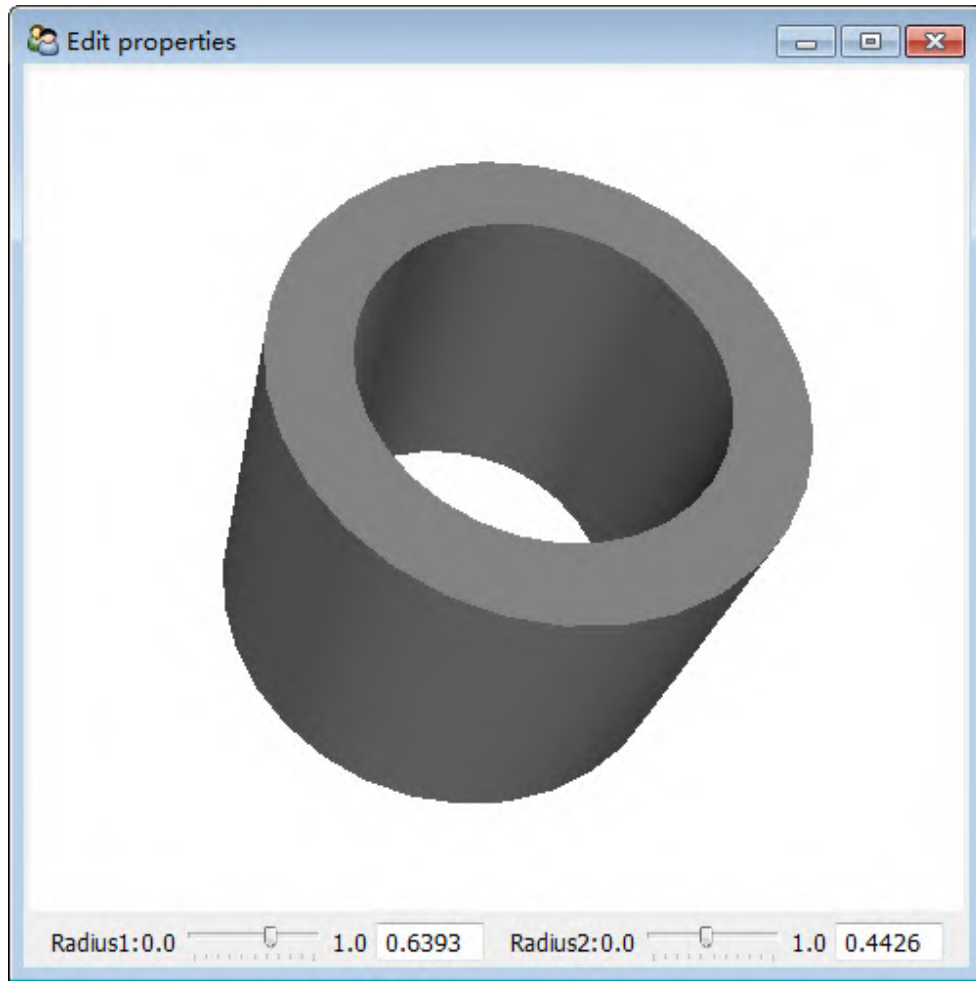


图8-39 将TVTK场景嵌入TraitsUI界面



`scipy2.tvtk.example_embed_tube`: 演示如何将TVTK场景嵌入TraitsUI界面，可通过界面中的控件调节圆管的内径和外径。

```
from traits.api import HasTraits, Instance, Range,
on_trait_change
from traitsui.api import View, Item, VGroup, HGroup,
Controller
```

```

from tvtk.api import tvtk

from tvtk.pyface.scene_editor import SceneEditor
from tvtk.pyface.scene import Scene
from tvtk.pyface.scene_model import SceneModel


class TVTKSceneController(Controller):

    def position(self, info):
        super(TVTKSceneController, self).position(info)
        self.model.plot() ❸


class TubeDemoApp(HasTraits):

    radius1 = Range(0, 1.0, 0.8)
    radius2 = Range(0, 1.0, 0.4)
    scene = Instance(SceneModel, ()) ❶
    view = View(
        VGroup(
            Item(name="scene",
editor=SceneEditor(scene_class=Scene)), ❷
            HGroup("radius1", "radius2"),
            show_labels=False),
        resizable=True, height=500, width=500)

    def plot(self):
        r1, r2 = min(self.radius1, self.radius2),
max(self.radius1, self.radius2)

```



```

        self.cs1 = cs1 = tvtk.CylinderSource(height=1,
radius=r2, resolution=32)

        self.cs2 = cs2 = tvtk.CylinderSource(height=1.1,
radius=r1, resolution=32)

        triangle1 =
tvtk.TriangleFilter(input_connection=cs1.output_port)

        triangle2 =
tvtk.TriangleFilter(input_connection=cs2.output_port)

        bf = tvtk.BooleanOperationPolyDataFilter()
        bf.operation = "difference"
        bf.set_input_connection(0, triangle1.output_port)
        bf.set_input_connection(1, triangle2.output_port)

        m =
tvtk.PolyDataMapper(input_connection=bf.output_port,
        scalar_visibility=False)

        a = tvtk.Actor(mapper=m)
        a.property.color = 0.5, 0.5, 0.5
        self.scene.add_actors([a])
        self.scene.background = 1, 1, 1
        self.scene.reset_zoom()

    @on_trait_change("radius1, radius2") ④
    def update_radius(self):
        self.cs1.radius = max(self.radius1, self.radius2)
        self.cs2.radius = min(self.radius1, self.radius2)
        self.scene.render_window.render()

```

```
if __name__ == "__main__":  
    app = TubeDemoApp()  
    app.configure_traits(handler=TVTKSceneController(app))
```

❶ SceneModel表示TVTK的场景模型，它在MVC模式中属于模型对象，因此程序中用它定义Trait属性scene。

❷ scene属性在界面中将呈现为一个TVTK的三维场景，因此在视图的Item定义中，用editor参数指定一个编辑器，让它能正确显示scene所代表的模型。SceneEditor是用来创建场景编辑器的工厂类，通过scene_class参数指定真正创建场景的对象类Scene。

❸ 在控制器的position()方法中调用模型对象的plot()方法以生成场景中的物体。position()方法在窗口显示之前调用，此时界面中的所有控件都已经被创建，因此能保证plot()中的语句正确运行。

❹ 通过on_trait_change()响应属性radius1和radius2的变化事件，修改流水线最上层的两个CylinderSource对象的radius属性，并调用self.scene.render_window.render()以刷新场景。因此当用户通过界面中的控件修改半径时，场景中的圆管会同步更新。

8.6.2 Mayavi场景的嵌入

下面看一个Mayavi场景嵌入的例子。用户输入一个由x、y、z等变量构成的表达式，例如 $x*x+y*y+z*z$ 。程序使用此表达式计算指定范围内的三维标量场，并且添加等值面和切面工具对标量场进行可视

化。等值面的数值可以自动计算，也可以通过界面上的滚动条配置；而切面的位置和方向则可以直接在场景中用鼠标操作。程序的界面如图8-40所示。

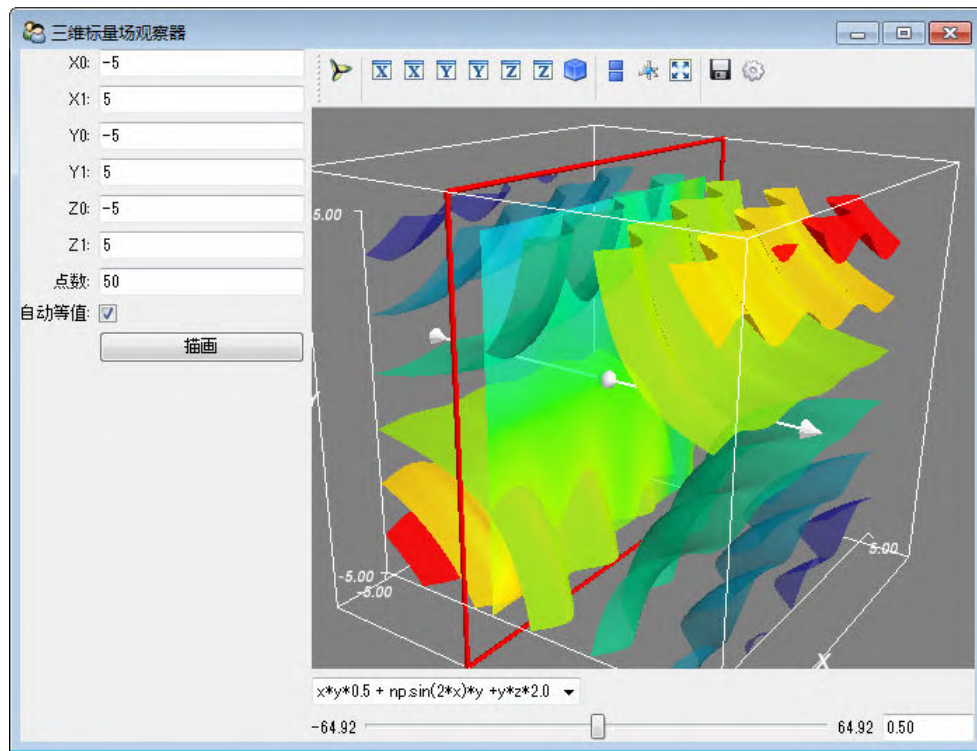


图8-40 三维标量场观察器



scipy2.tvtk.example_embed_fieldviewer: 标量场观察器，演示如何将Mayavi的场景嵌入TraitsUI界面。

```
import numpy as np

from traits.api import HasTraits, Float, Int, Bool, Range,
Str, Button, Instance
```

```

        from traitsui.api import View, HSplit, Item, VGroup,
EnumEditor, RangeEditor

        from tvtk.pyface.scene_editor import SceneEditor
        from mayavi.tools.mlab_scene_model import MlabSceneModel
        from mayavi.core.ui.mayavi_scene import MayaviScene
        from scpy2.tvtk import fix_mayavi_bugs

        fix_mayavi_bugs()

class FieldViewer(HasTraits):

    # 三个轴的取值范围
    x0, x1 = Float(-5), Float(5)
    y0, y1 = Float(-5), Float(5)
    z0, z1 = Float(-5), Float(5)
    points = Int(50) # 分割点数
    autocontour = Bool(True) # 是否自动计算等值面
    v0, v1 = Float(0.0), Float(1.0) # 等值面的取值范围
    contour = Range("v0", "v1", 0.5) # 等值面的值
    function = Str("x*x*0.5 + y*y + z*z*2.0") # 标量场函数
    function_list = [
        "x*x*0.5 + y*y + z*z*2.0",
        "x*y*0.5 + np.sin(2*x)*y + y*z*2.0",
        "x*y*z",
        "np.sin((x*x+y*y)/z)"
    ]

```

```

plotbutton = Button(u"描画")

scene = Instance(MlabSceneModel, ()) ❶

view = View(
    HSplit(
        VGroup(
            "x0", "x1", "y0", "y1", "z0", "z1",
            Item('points', label=u"点数"),
            Item('autocontour', label=u"自动等值"),
            Item('plotbutton', show_label=False),
        ),
        VGroup(
            Item('scene',

editor=SceneEditor(scene_class=MayaviScene), ❷
    resizable=True,
        height=300,
        width=350
    ),
    Item('function',
        editor=EnumEditor(name='function_list',
evaluate=lambda x:x)),
    Item('contour',
        editor=RangeEditor(format="%1.2f",
            low_name="v0", high_name="v1")
    ), show_labels=False
)

```

```

        ),
        width = 500, resizable=True, title=u"三维标量场观察器"
    )

    def _plotbutton_fired(self):
        self.plot()

    def plot(self):
        # 产生三维网格
        x, y, z = np.mgrid[ ③
            self.x0:self.x1:1j*self.points,
            self.y0:self.y1:1j*self.points,
            self.z0:self.z1:1j*self.points]

        # 根据函数计算标量场的值
        scalars = eval(self.function) ④
        self.scene.mlab.clf() # 清空当前场景

        # 绘制等值平面
        g = self.scene.mlab.contour3d(x, y, z, scalars, contours=8,
transparent=True) ⑤
        g.contour.auto_contours = self.autocontour

self.scene.mlab.axes(figure=self.scene.mayavi_scene) # 添加坐标轴

        # 添加一个x-y的切面
        s = self.scene.mlab.pipeline.scalar_cut_plane(g)

```

```

        cutpoint = (self.x0+self.x1)/2,
(self.y0+self.y1)/2, (self.z0+self.z1)/2
    s.implicit_plane.normal = (0,0,1) # x cut
    s.implicit_plane.origin = cutpoint

    self.g = g ❹
    self.scalars = scalars
    # 计算标量场的值的范围
    self.v0 = np.min(scalars)
    self.v1 = np.max(scalars)

def _contour_changed(self): ❺
    if hasattr(self, "g"):
        if not self.g.contour.auto_contours:
            self.g.contour.contours = [self.contour]

def _autocontour_changed(self): ❻
    if hasattr(self, "g"):
self.g.contour.auto_contours = self.autocontour
        if not self.autocontour:
            self._contour_changed()

if __name__ == '__main__':
    app = FieldViewer()
    app.configure_traits()

```

❶MlabSceneModel表示Mayavi的场景模型，❷对应的场景控件为MayaviScene。

用户单击描绘按钮之后，将调用plot()绘图。❸首先计算三维标量场的网格，这里使用mgrid快速产生三维网格，其中的x0、x1、y0、y1、z0、z1、points、function等都是模型类的Trait属性，可以通过界面上的控件直接修改这些属性的值。❹由于用户输入的三元函数是一个字符串，这里用eval()对字符串进行求值，在字符串中可以使用x、y、z等局域变量。

❺清空当前场景之后，调用mlab模块中的contour3d()、axes()、pipeline.scalar_cut_plane()等在场景中添加等值面、坐标轴和切面。mlab模块缺省对当前场景进行处理，如果应用程序有多个场景，就需要分别在其中绘图时，通过figure参数指定需要进行处理的场景，例如：

```
mlab.axes(figure=self.scene.mayavi_scene)
```

其中，self.scene是MlabSceneModel对象，mayavi_scene属性是真正表示场景的Scene对象。

❻最后更新模型对象的几个属性，其中变量g是contour3d()的返回值，它表示场景中的等值面。self.v0和self.v1是标量场的最小值和最大值，它们设置等值面滚动条的取值范围。

❼当与contour属性相对应的滚动条控件(位于三维场景的下方)的值发生变化时，将调用_contour_changed()方法修改保存等值面数值的列表。

⑧当“自动等值”选择框控件改变时，在autocontour属性的事件监听函数_autocontour_changed()中改变等值面对象g的自动等值面选项。

第9章 OpenCV-图像处理和计算机视觉

OpenCV是一套采用C/C++编写的开源跨平台计算机视觉库，它提供了两套Python调用接口。其中cv2模块是针对OpenCV 2.x API创建的，它直接采用NumPy的数组对象表示图像。为了兼容OpenCV 1.x API，在cv模块下提供了原来的OpenCV 1.x API的扩展库。本章所介绍的代码均采用如下方式载入这两套接口的模块：

```
import cv2  
from cv2 import cv
```

9.1 图像的输入输出



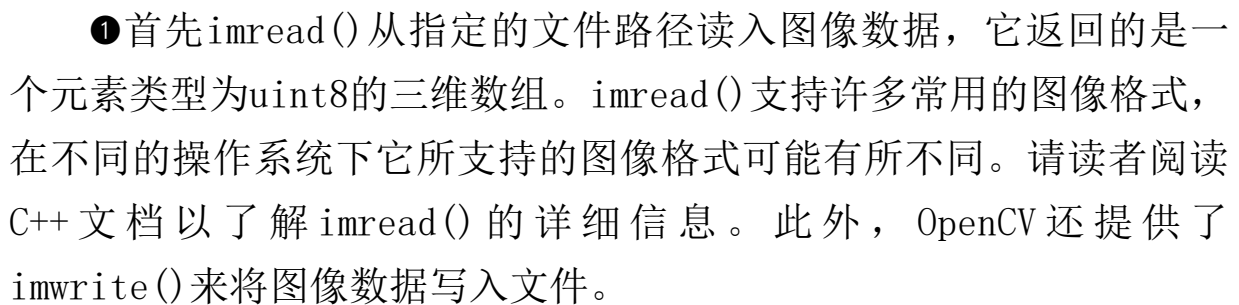
与本节内容对应的Notebook为：[09-opencv/opencv-100-input-output.ipynb](#)。

本节介绍如何使用图像的输入输出函数。由于cv2模块中的函数使用NumPy数组表示图像，因此很容易将从文件中读入的图像数据传递给其他基于NumPy数组的扩展库，也可以调用cv2提供的图像函数对NumPy数组进行处理。

9.1.1 读入并显示图像

让我们从读入并显示一幅图像开始，在IPython Notebook中运行如下语句，将会看到一个显示美女“lena”的窗口：

```
filename = "lena.jpg"
img = cv2.imread( filename ) ❶
print type(img), img.shape, img.dtype
cv2.namedWindow("demo1") ❷
cv2.imshow("demo1", img) ❸
cv2.waitKey(0) ❹
<type 'numpy.ndarray'< (512, 512, 3) uint8
```

❶首先 `imread()` 从指定的文件路径读入图像数据，它返回的是一个元素类型为uint8的三维数组。`imread()` 支持许多常用的图像格式，在不同的操作系统下它所支持的图像格式可能有所不同。请读者阅读C++ 文档以了解 `imread()` 的详细信息。此外，OpenCV 还提供了 `imwrite()` 来将图像数据写入文件。

为了方便用户快速观察图像处理的效果，OpenCV提供了一些简单的GUI功能。❷这里调用 `namedWindow()`，创建一个名为“demo1”的窗口，❸最后调用 `imshow()` 将图像显示到所创建的窗口中。`imshow()` 的第一个参数是窗口名，第二个参数是表示图像的数组。如果第一个参数指定的窗口不存在，`imshow()` 会自动创建一个新窗口，因此这里也可以不调用 `namedWindow()`。

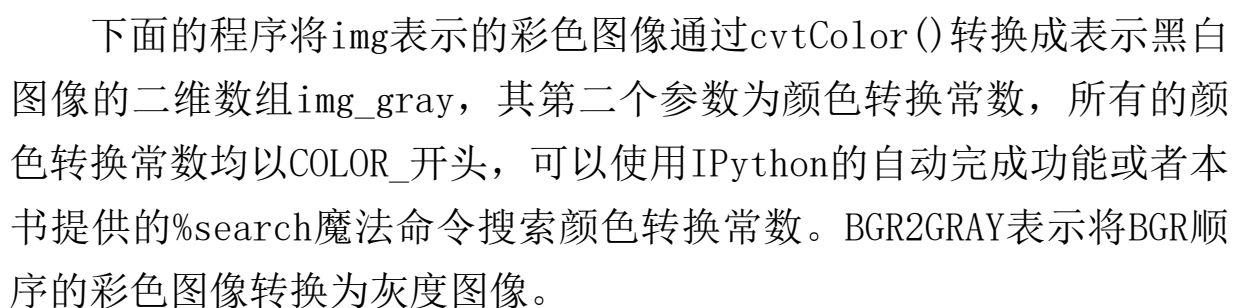
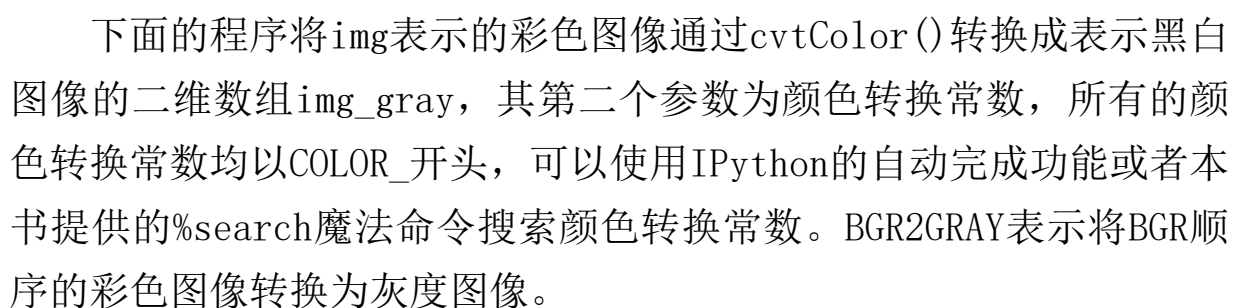


`imread()` 函数读入彩色图像时，第2轴按照蓝、绿、红的顺序排列。这种顺序与matplotlib的`imshow()` 函数所需的三通道的顺序正好相反。因此，若需要在matplotlib中显示图像，则需要反转第2轴：`img[:, :, ::-1]`。

④最后调用`cv.waitKey(0)` 等待用户按下按键，其参数为等待的毫秒数，0表示永远等待。



在IPython Notebook显示窗口时，`waitKey(0)` 会阻塞运算核，因此无法再运行其他命令。为了不阻塞运行，可以在新的线程中显示窗口，并等待窗口关闭，例如通过本书提供的`%%thread`魔法命令。

下面的程序将表示的彩色图像通过`cvtColor()` 转换成表示黑白图像的二维数组_gray，其第二个参数为颜色转换常数，所有的颜色转换常数均以COLOR_开头，可以使用IPython的自动完成功能或者本书提供的`%search`魔法命令搜索颜色转换常数。BGR2GRAY表示将BGR顺序的彩色图像转换为灰度图像。



cv2模块下的图像处理函数都能直接对NumPy数组进行操作，在这些函数内部会将NumPy数组转换成OpenCV中表示图像的对象，并传递给实际进行运算的C/C++函数。虽然在调用cv2模块下的函数时会进行类型转换，但是NumPy数组与OpenCV的图像对象能够共享内存，因此并不会太浪费内存和CPU运算时间。

```
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
print img_gray.shape
(512, 512)
```

9.1.2 图像类型

图像中的每个像素点可能有多个通道，例如用单通道可以表示灰度图像，而用红绿蓝三个通道表示彩色图像，用4个通道表示带透明度(alpha)的彩色图像。通道的数值类型可以有多种选择，例如通常的图像用8位无符号整数表示，而医学图像可能会用16位整数表示图像数据。因此像素点的类型由通道数和数值类型决定。

例如前面的img是一个三维数组，形状为(512, 512, 3)，第0轴为图像的高，第1轴为图像的宽，而第2轴为图像的通道数。因此图像img的宽为512个像素，高为512个像素，有3个通道，即图像中的每个像素的颜色用三个数值表示。根据dtype属性可知，每个通道的颜色值都用一个字节表示。而灰度图像img_gray则是一个二维数组，因为它只有一个通道。

在前面的例子中，没有设置imread()的第二个参数，其缺省值为IMREAD_COLOR，使用该参数读入的数据是三通道且每个通道8比特的数组。第二个参数有如下候选值：

- IMREAD_ANYCOLOR：转换成8比特的图像，通道数由图像文件决定，注意4通道图像会被转换成三通道图像。
- IMREAD_ANYDEPTH：转换为单通道，比特数由图像文件决定。
- IMREAD_COLOR：转换为三通道、8比特的图像。
- IMREAD_GRAYSCALE：转换成单通道、8比特的图像。
- IMREAD_UNCHANGED：使用图像文件的通道数和比特数。

表9-1显示了对各种通道数和比特数的图像文件使用上述标志读入之后的结果，其中文件名的格式为：“通道比特数_通道数.png”。

表9-1 图像文件读入后的结果

	IMREAD_ANYCOLOR	IMREAD_ANYDEPTH	IMREAD_COLOR	IMREAD_GRAYSCALE	IMREAD_UNCHANGED
uint16_1.png	uint8、1ch	uint16、1ch	uint8、3ch	uint8、1ch	uint16、1ch
uint16_3.png	uint8、3ch	uint16、1ch	uint8、3ch	uint8、1ch	uint16、3ch
uint16_4.png	uint8、3ch	uint16、1ch	uint8、3ch	uint8、1ch	uint16、4ch
uint8_1.png	uint8、1ch	uint8、1ch	uint8、3ch	uint8、1ch	uint8、1ch
uint8_3.png	uint8、3ch	uint8、1ch	uint8、3ch	uint8、1ch	uint8、3ch
uint8_4.png	uint8、3ch	uint8、1ch	uint8、3ch	uint8、1ch	uint8、4ch

9.1.3 图像输出

imwrite()将数组编码成指定的图像格式并写入文件，图像的格式由文件的扩展名决定。某些格式有额外的图像参数，例如JPEG格式的文件可以指定画质参数，这些参数都以IMWRITE_开头。图像参数以[参

数名, 参数值, 参数名, 参数值, ...]的形式传递给imwrite()的第三个参数。在下面的例子中, 把从lena.jpg读入的数据以各种画质保存为不同的JPEG文件:

```
img = cv2.imread("lena.jpg")
for quality in [90, 60, 30]:
    cv2.imwrite("lena_q{:02d}.jpg".format(quality), img,
                [cv2.IMWRITE_JPEG_QUALITY, quality])
```

当图像格式支持更高的通道比特数时, imwrite()会保持图像数组的精度。在下面的例子中, ❶通过NumPy计算出下面的二元函数的值:

$$f(x, y) = \frac{(x^2 - y^2) \sin\left(\frac{x + y}{a}\right)}{x^2 + y^2}$$

❷然后调用matplotlib的ScalarMappable(), 使用颜色映射表jet把函数值转换成彩色图像, 注意OpenCV的所有输入输出函数都采用蓝绿红的通道顺序。由于to_rgba()的输出为元素值在0到1之间的浮点数数组, 因此还需要将其转换成整数数组。

```
from matplotlib.cm import ScalarMappable
from IPython.display import Image

def func(x, y, a):
    return (x*x - y*y) * np.sin((x + y) / a) / (x*x + y*y)
def make_image(x, y, a, dtype="uint8"):
    z = func(x, y, a) ❶
    img_rgba = ScalarMappable(cmap="jet").to_rgba(z)
```



```

img = (img_rgba[:, :, 2::-1] *
np.iinfo(dtype).max).astype(dtype) ❷

return img

```

下面调用 `make_img()` 创建8比特数组 `img_8bit` 和16比特数组 `img_16bit`，然后分别将其保存为JPEG格式和PNG格式的图像。由于JPEG图像只支持8比特通道，因此将 `img_16bit` 保存为JPEG图像时会出现问题，请读者打开 `img_16bit.jpg` 来查看结果。而PNG格式则支持8比特和16比特通道，如果读者查看 `img_16bit.png` 的文件属性，就会看到它的“位深度”为48比特，即三个通道并且每个通道16比特。

```

y, x = np.ogrid[-10:10:250j, -10:10:500j]
img_8bit = make_image(x, y, 0.5, dtype="uint8")
img_16bit = make_image(x, y, 0.5, dtype="uint16")
cv2.imwrite("img_8bit.jpg", img_8bit)
cv2.imwrite("img_16bit.jpg", img_16bit)
cv2.imwrite("img_8bit.png", img_8bit)
cv2.imwrite("img_16bit.png", img_16bit)

```

9.1.4 字节序列与图像的相互转换

`imdecode()` 可以把图像文件数据解码成图像数组，`imencode()` 则把图像数组编码成图像文件数据。由于所有的运算都在内存中完成，因此可以使用这两个函数快速压缩和解压图像。例如把从摄像头读入的图像编码之后通过网络传递给其他计算机。

在下面的例子中，❶首先通过`frombuffer()`创建一个和字符串`png_str`共享内存的数组`png_data`。❷然后调用`imdecode()`将PNG文件的数据解压成表示图像的数组`img`。❸最后调用`imencode()`将图像数组压缩成JPG数据，第一个参数是表示图像类型的扩展名。它返回两个值，第一个值表示压缩是否成功，第二个值为压缩之后的数据，它是一个形状为(N, 1)的`uint8`数组。❹调用数组的`tobytes()`可以将数组中的二进制数据转换成字符串。

```
with open("img_8bit.png", "rb") as f:
    png_str = f.read()

png_data = np.frombuffer(png_str, np.uint8) ❶
img = cv2.imdecode(png_data, cv2.IMREAD_UNCHANGED) ❷
res, jpg_data = cv2.imencode(".jpg", img) ❸
jpg_str = jpg_data.tobytes() ❹
```

可以使用`Image`将编码之后的图像数据嵌入IPython Notebook中。在下面的例子中，将编码之后的字符串数据传递给`Image`的`data`参数，IPython会将字符串数据直接嵌入Notebook中，然后由浏览器将其显示为图像，效果如图9-1所示。

```
res, jpg_data = cv2.imencode(".jpg", img_8bit)
Image(data=jpg_data.tobytes())
```

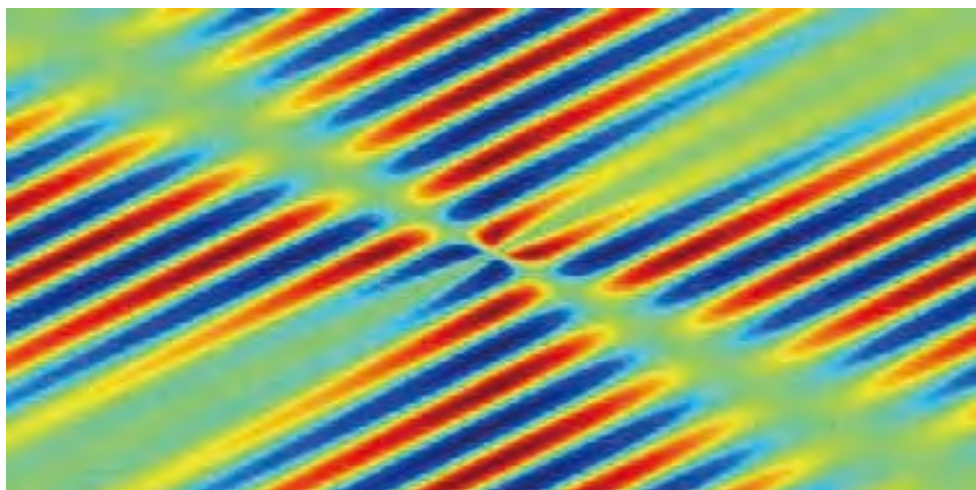


图9-1 使用Image将imencode()编码的结果直接嵌入Notebook中

9.1.5 视频输出

通过VideoWriter类可以将多个NumPy数组写入视频文件。下面的例子中以不同的参数调用make_image()并将结果写入fmp4.avi文件。

❶视频文件的编码由4个字符的FOURCC对象指定。❷使用指定的fourcc创建VideoWriter对象，其第3个参数为帧频，第4个参数为视频的宽度和高度，最后的参数为True时表示视频为彩色。❸调用write()方法将表示图像的数组写入视频，❹最后调用release()方法关闭视频文件。

```
def test_avi_output(fn, fourcc):  
    fourcc = cv.FOURCC(*fourcc) ❶  
    vw = cv2.VideoWriter(fn, fourcc, 15, (500, 250), True)  
❷  
    if not vw.isOpened():  
        return  
    for a in np.linspace(0.1, 2, 100):
```

```

img = make_image(x, y, a)

vw.write(img) ❸

vw.release() ❹

test_avi_output("fmp4.avi", "fmp4")

```

OpenCV自带的视频编码器无法设置码率之类的编码参数，如果读者希望控制视频文件的画质，可以通过fourcc代码指定操作系统中安装的VFW编码器，这些编码器通常都带有配置界面用于设置编码参数。为了获取各种编码器对应的fourcc代码，读者可以运行本书提供的fourcc.py，将弹出一个如图9-2所示的“视频压缩”对话框：

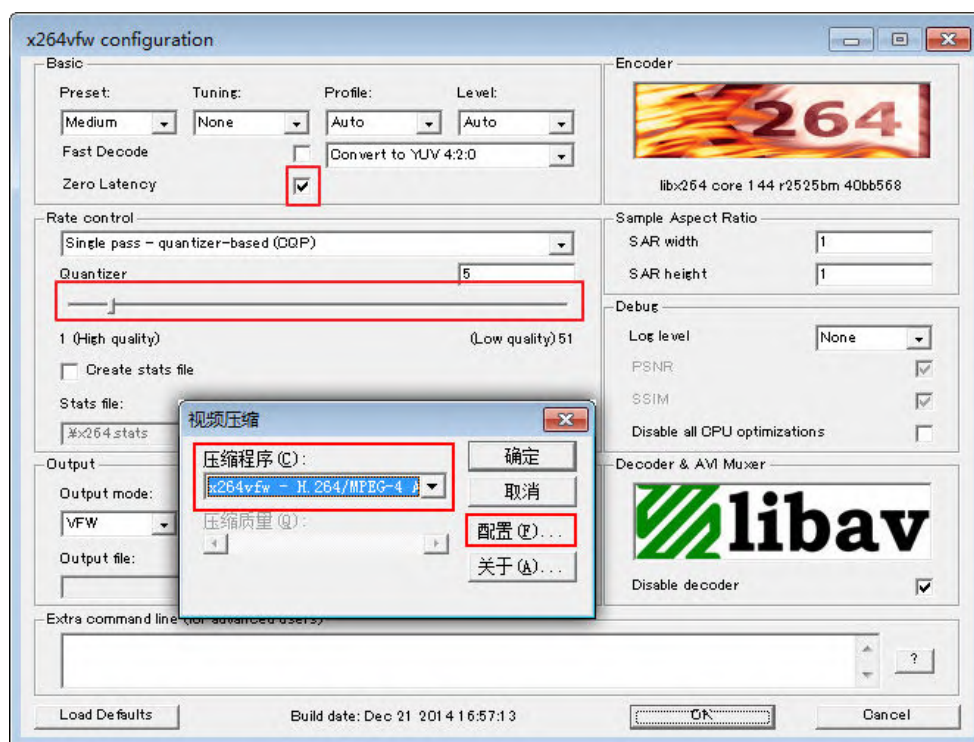


图9-2 编码选择以及x264编码设置对话框

在对话框中选择“x264vfw”，然后单击“配置”按钮，将打开图中显示的“x264vfw configuration”对话框，勾选“Zero

Latency”，并且通过“Quantizer”设置画质。然后单击“OK”和“确定”按钮完成设置。fourcc.py将输出与所选择的编码器对应的fourcc代码：x264。



scipy2.opencv.fourcc：查看与选中的视频编码器对应的fourcc代码。

如果读者的计算机中没有x264vfw编码器，可以通过下面的网址下载安装程序：



<http://sourceforge.net/projects/x264vfw/>

x264vfw编码器的下载地址。

x264vfw 编码器的配置保存在注册表的HKEY_CURRENT_USER\Software\GNU\x264路径下。本书提供了set_quantizer()来设置其编码画质，读者可以仿照该程序，根据需要修改其他项的值。下面的程序输出不同的画质选项所得到的视频文件大小：

```
from scipy2.opencv.x264_settings import set_quantizer
from os import path
```

```

for quantizer in [1, 10, 20, 30, 40]:
    set_quantizer(quantizer)
    fn = "x264_q{:02d}.avi".format(quantizer)
    test_avi_output(fn, "x264")
    fsize = path.getsize(fn)
    print "quantizer = {:02d}, size = {:07d}
bytes".format(quantizer, fsize)

quantizer = 01, size = 5686272 bytes
quantizer = 10, size = 2406912 bytes
quantizer = 20, size = 0932864 bytes
quantizer = 30, size = 0396288 bytes
quantizer = 40, size = 0189952 bytes

```

9.1.6 视频输入

VideoCapture类用于从视频文件或视频设备读入图像。在下面的例子中，使用它从视频文件读入相关的属性和帧。❶get()方法获得指定的属性，所有视频相关的属性名都在cv模块中以CV_CAP_PROP_开头。这里读入视频的帧频、总帧数、像素宽和高。❷调用read()方法读入一帧图像，它返回两个值：表示是否正确获得图像的布尔值和表示图像的数组。正常读入一帧图像之后，当前帧自动递增。❸还可以通过set()方法设置当前帧，从而直接读取视频中指定位置的图像。

```

video = cv2.VideoCapture("x264_q10.avi")
print "FPS:", video.get(cv.CV_CAP_PROP_FPS) ❶
print "FRAMES:", video.get(cv.CV_CAP_PROP_FRAME_COUNT)

```

```
print "WIDTH:", video.get(cv.CV_CAP_PROP_FRAME_WIDTH)
print "HEIGHT:", video.get(cv.CV_CAP_PROP_FRAME_HEIGHT)
print "CURRENT FRAME:", video.get(cv.CV_CAP_PROP_POS_FRAMES)
res, frame0 = video.read() ❷
print "CURRENT FRAME:", video.get(cv.CV_CAP_PROP_POS_FRAMES)
video.set(cv.CV_CAP_PROP_POS_FRAMES, 50) ❸
print "CURRENT FRAME:", video.get(cv.CV_CAP_PROP_POS_FRAMES)
res, frame50 = video.read()
print "CURRENT FRAME:", video.get(cv.CV_CAP_PROP_POS_FRAMES)
video.release()

FPS: 15.0
FRAMES: 100.0
WIDTH: 500.0
HEIGHT: 250.0
CURRENT FRAME: 0.0
CURRENT FRAME: 1.0
CURRENT FRAME: 50.0
CURRENT FRAME: 51.0
```

当传递给VideoCapture的参数是整数时，将打开该整数对应的视频设备。下面的程序从笔者的笔记本电脑自带的摄像头读取一帧图像：

```
camera = cv2.VideoCapture(0)
res, frame = camera.read()
camera.release()
```

```
print res, frame.shape  
  
True (480, 640, 3)
```

9.2 图像处理



与本节内容对应的Notebook为：[09-opencv/opencv-200-imgprocess.ipynb](#)。

OpenCV的图像处理功能十分丰富，本节以二维卷积、形态学图像处理、颜色填充、去瑕疵等为例简要地介绍OpenCV的图像处理功能。希望读者通过这些实例举一反三，能通过阅读OpenCV的文档尝试更多的图像处理功能。

9.2.1 二维卷积

图像处理中最基本的算法就是将图像和某个卷积核进行卷积，使用不同的卷积核可以得到各种不同的图像处理效果。OpenCV提供了`filter2D()`来完成图像的卷积运算，调用方式如下：

```
filter2D(src, ddepth, kernel[, dst[, anchor[, delta[,  
borderType]]]])
```

其中`src`参数是原始图像，`dst`参数是目标图像。若省略`dst`参数，将创建一个新的数组来保存图像数据。`ddepth`参数用于指定目标图像

的每个通道的数据类型，负数表示其数据类型和原始图像相同。`kernel`参数设置卷积核，它将与原始图像的每个通道进行卷积计算，并将结果存储到目标图像的对应通道中。`anchor`参数指定卷积核的锚点位置，当它为默认值 $(-1, -1)$ 时，以卷积核的中心为锚点。`delta`参数指定在将计算结果存储到`dst`中之前对数值的偏移量。

`filter2D()`的卷积运算过程如下：

(1) 对于图像`src`中的每个像素点 (x,y) ，让它和卷积核的锚点对齐。

(2) 对于图像`src`中与卷积核重叠的部分，计算像素值和卷积核的值乘积。

(3) 图像`dst`中的像素点 (x,y) 的值为上面所有乘积的总和。

显然当卷积核的尺寸很大时，上述方法的运算速度将会很慢。因此对于较大的卷积核，`filter2D()`将使用离散傅立叶变换相关的算法进行卷积运算。

下面的程序演示了使用不同卷积核对图像进行处理之后的效果，如图9-3所示。

```
src = cv2.imread("lena.jpg")

kernels = [
    (u"低通滤波器", np.array([[1, 1, 1], [1, 2, 1], [1, 1, 1]]) * 0.1),
    (u"高通滤波器", np.array([[0.0, -1, 0], [-1, 5, -1], [0, -1,
```

```

0]])),
    (u"边缘检测", np.array([[ -1.0,  -1,  -1], [-1,  8,  -1], [-1,
-1, -1]]))
]

index = 0
fig, axes = pl.subplots(1, 3, figsize=(12, 4.3))
for ax, (name, kernel) in zip(axes, kernels):
    dst = cv2.filter2D(src, -1, kernel)
    # 由于matplotlib的颜色顺序和OpenCV的顺序相反
    ax.imshow(dst[:, :, ::-1])
    ax.set_title(name)
    ax.axis("off")
fig.subplots_adjust(0.02, 0, 0.98, 1, 0.02, 0)

```



图9-3 使用filter2D()制作的各种图像处理效果



scipy2.opencv.filter2d_demo: 可通过图形界面自定义卷积核, 并实时查看处理结果。

有些特殊的卷积核可以表示成一个列矢量和一个行矢量的乘积，这时只需要将原始图像按顺序与这两个矢量进行卷积，所得到的最终结果和直接与卷积核进行卷积的结果相同。由于将一个 $N \times M$ 的矩阵分解成了两个 $N \times 1$ 和 $1 \times M$ 的矩阵，因此对于较大的卷积核能大幅度地提高计算速度。OpenCV提供了`sepFilter2D()`来进行这种分步卷积，调用参数如下：

```
sepFilter2D(src, ddepth, kernelX, kernelY[, dst[, anchor[,  
delta[, borderType]]]])
```

其中`kernelX`和`kernelY`分别为行卷积核和列卷积核。下面的程序比较`filter2D()`和`sepFilter2D()`的计算速度：

```
img = np.random.rand(1000,1000) ❶  
  
row = cv2.getGaussianKernel(7, -1) ❷  
col = cv2.getGaussianKernel(5, -1)  
  
kernel = np.dot(col[:,], row[:,].T) ❸  
  
%time img2 = cv2.filter2D(img, -1, kernel) ❹  
%time img3 = cv2.sepFilter2D(img, -1, row, col) ❺  
print "error=", np.max(np.abs(img2[:] - img3[:] ))  
Wall time: 31 ms  
Wall time: 25 ms  
error= 4.4408920985e-16
```

❶ 首先随机产生一幅比较大的图像 `img`。❷ 调用 `getGaussianKernel()` 分别获得长度为7和5的两个高斯模糊卷积核 `row` 和 `col`。❸ 计算 `row` 和 `col` 的矩阵乘积 `kernel`，它是一个形状为 (5, 7) 的二维数组。❹❺ 分别使用 `filter2D()` 和 `sepFilter2D()` 对图像 `img` 进行卷积，并测量它们的计算时间。

卷积核的尺寸越大，计算时间的差别越大。请读者更改 `row` 和 `col` 的值，观察计算时间的差别。

由于卷积计算很常用，因此OpenCV提供了一些高级函数来直接完成与某种特定卷积核的卷积计算。例如平均模糊 `blur()`、高斯模糊 `GaussianBlur()`、用于边缘检测的差分运算 `Sobel()` 和 `Laplacian()` 等。关于这些函数的用法请读者自行参考OpenCV的文档，这里就不再举例了。

9.2.2 形态学运算

在SciPy的图像处理章节中，我们介绍过如何使用SciPy的图像处理模块进行形态学图像的处理。OpenCV中也提供了类似的处理功能。例如 `dilate()` 对图像进行膨胀处理，而 `erode()` 则对图像进行腐蚀处理。另外，`morphologyEx()` 使用膨胀和收缩实现一些更高级的形态学处理。这些函数都可以对多值图像进行操作，对于多通道图像，它们将对每个通道进行相同的运算。`dilate()` 和 `erode()` 的调用参数相同：

```
dilate(src, kernel[, dst[, anchor[, iterations[,  
borderType[, borderValue]]]])
```

其中src参数是原始图像；kernel参数是结构元素，它指定针对哪些周围像素进行计算；anchor参数指定锚点的位置，其默认值为结构元素的中心；iterations参数指定处理次数。morphologyEx()的参数如下：

```
morphologyEx(src, op, kernel[, dst[, anchor[, iterations[,  
borderType[, borderValue]]]])
```

它比dilate()多了一个op参数，用于指定运算的类型。

膨胀运算可以用下面的公式描述：

$$\text{dst}(x,y) = \max_{\text{kernel}(x',y') \neq 0} \text{src}(x + x', y + y')$$

将结构元素的锚点与原始图像中的每个像素(x,y) 对齐之后，计算所有结构元素值不为0的像素的最大值，写入目标图像的(x,y) 像素点。而腐蚀运算则是计算所有结构元素不为0的像素的最小值。

morphologyEx()的高级运算包括：

- MORPH_OPEN：开运算，可以用来区分两个靠得很近的区域。算法为先腐蚀再膨胀：dst=dilate(erode(src))。
- MORPH_CLOSE：闭运算，可以用来连接两个靠得很近的区域。算法为先膨胀再腐蚀：dst=erode(dilate(src))。
- MORPH_GRADIENT：形态梯度，能够找出图像区域的边缘。算法为膨胀减去腐蚀：dst=dilate(src)- erode(src)。
- MORPH_TOPHAT：顶帽运算，算法为原始图像减去开运算：dst = src-open(src)。

- MORPH_BLACKHAT: 黑帽运算, 算法为闭运算减去原始图像:
 $\text{dst} = \text{close}(\text{src}) - \text{src}$ 。

下面的程序演示了上述形态学图像处理的效果。图9-4是界面截图。请读者通过此界面修改结构元素、处理类型以及迭代次数等参数, 并观察经过处理之后的图像, 理解各种运算的公式。

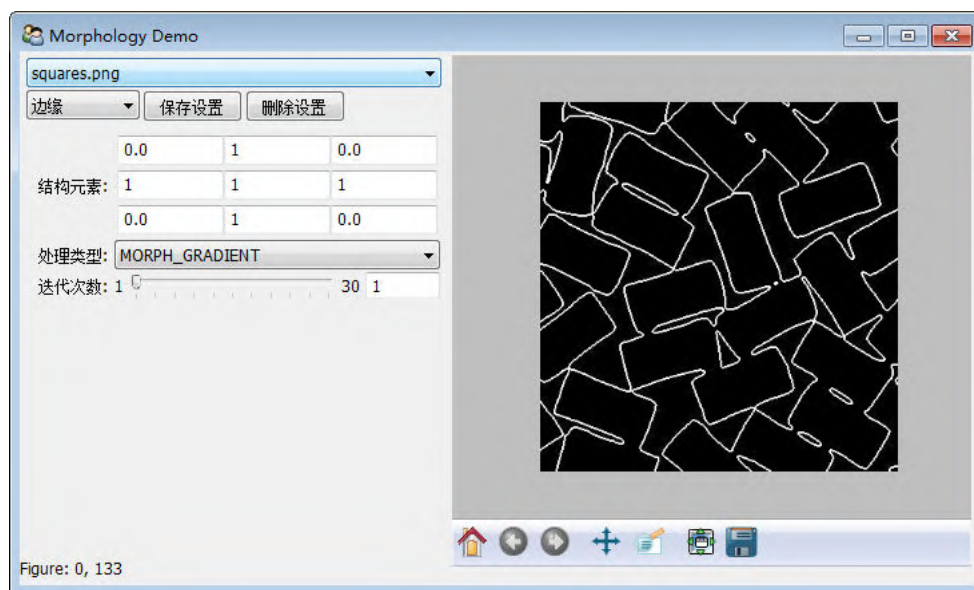


图9-4 形态学图像处理演示界面



scipy2.opencv.morphology_demo: 演示OpenCV中的各种形态学运算。

9.2.3 填充-floodFill

填充函数floodFill()在图像处理中经常用于标识或分离图像中的某些特定部分。它的调用方式为：

```
floodFill(image, mask, seedPoint, newVal[, loDiff[, upDiff[, flags]]])
```

其中image参数是需要填充的图像；seedPoint参数为填充的起始点，我们称之为种子点；newVal参数为填充所使用的颜色值；loDiff和upDiff参数是填充的下限和上限容差；flags参数是填充的算法标志。

填充从seedPoint指定的种子坐标开始，图像中与当前的填充区域颜色相近的点将被添加进填充区域，从而逐步扩大填充区域，直到没有新的点能添加进填充区域为止。颜色相近的判断方法有两种：

- 默认使用相邻点为基点进行判断。
- 如果开启了flags中的FLOODFILL_FIXED_RANGE标志位，则以种子点为基点进行判断。

假设图像中某个点(x,y) 的颜色为C(x, y)，C₀ 为基点颜色，则下面的条件满足时，(x,y) 将被添加进填充区域：

$$C_0 - \text{loDiff} \leq C(x, y) \leq C_0 + \text{hiDiff}$$

此外还可以通过flags指定相邻点的定义：四连通或八连通。

当mask参数不为None时，它是一个宽和高比image都大两个像素的单通道8位图像。image图像中的像素(x,y) 与mask中的(x + 1,y + 1) 对应。填充只针对mask中的值为0的像素进行。进行填充之后，mask中

所有被填充的像素将被赋值为1。如果只希望修改mask，而不对原始图像进行填充，可以开启flags标志中的FLOODFILL_MASK_ONLY。

在下面的例子中，第一次调用floodFill()时，由于设置了FLOODFILL_MASK_ONLY标志，因此填充只在mask中进行，并未修改img中的数据：

```
img = cv2.imread("coins.png")
seed1 = 344, 188
seed2 = 152, 126
diff = (13, 13, 13)
h, w = img.shape[:2]
mask = np.zeros((h+2, w+2), np.uint8)
    cv2.floodFill(img, mask, seed1, (0, 0, 0), diff, diff,
cv2.FLOODFILL_MASK_ONLY)
    cv2.floodFill(img, None, seed2, (0, 0, 255), diff, diff)

fig, axes = pl.subplots(1, 2, figsize=(9, 4))
axes[0].imshow(~mask, cmap="gray")
axes[1].imshow(img)
```

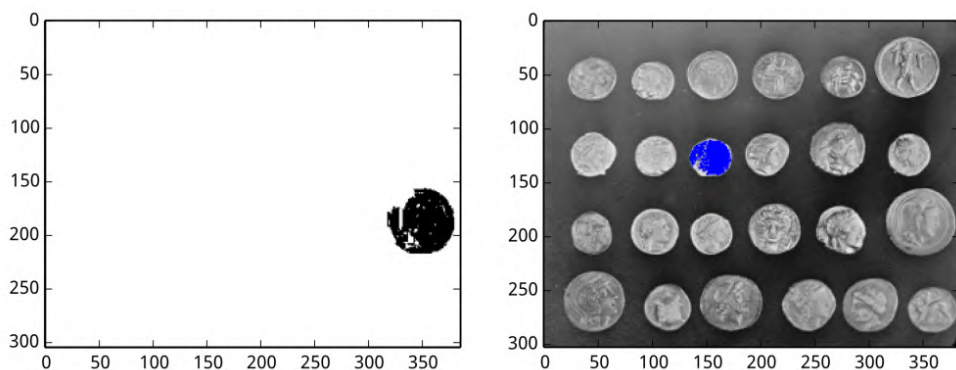


图9-5 演示floodFill()的填充效果

下面的程序演示了floodFill()的用法，界面如图9-6所示。在图像上用鼠标左键点选填充的种子点。通过界面上方的控件修改loDiff、upDiff和flags等参数。

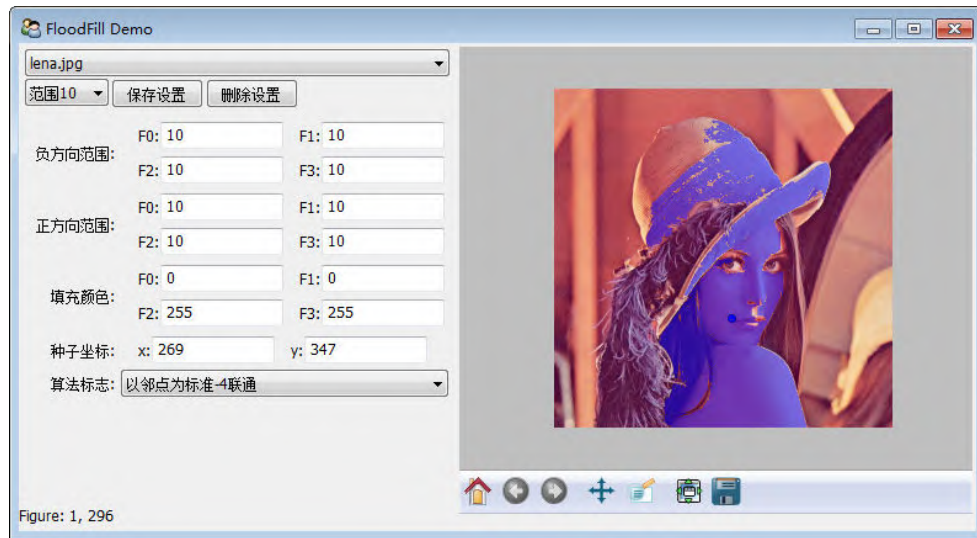


图9-6 填充演示程序的界面截图



scipy2.opencv.floodfill_demo : 演示填充函数floodFill()的各个参数的用法。

演示程序中使用两个叠加在一起的AxesImage对象显示图像，下层显示原始图像，上层半透明地显示填充之后的图像，因此可以观察被填充区域的原始图像。floodFill()的flags参数的选项如下：

```
Options = {  
    u"以种子为标准-4联通": cv2.FLOODFILL_FIXED_RANGE | 4,  
    u"以种子为标准-8联通": cv2.FLOODFILL_FIXED_RANGE | 8,  
    u"以邻点为标准-4联通": 4,
```

u"以邻点为标准-8联通": 8

}

9.2.4 去瑕疵-inpaint

使用inpaint()可以从图像上去除指定区域中的物体，可以用于去除图像上的水印、划痕、污渍等瑕疵。它的调用参数如下：

```
inpaint(src, inpaintMask, inpaintRadius, flags[, dst])
```

其中，src参数是原始图像，inpaintMask参数是大小和src相同的单通道8位图像，其中不为0的像素表示需要去除的区域。dst参数用于保存处理结果。inpaintRange参数是处理半径，半径越大处理时间越长，结果越平滑。flags参数选择inpaint的算法，目前有两个候选算法：INPAINT_NS和INPIANT_TELEA。

下面的程序演示inpaint()的用法，界面如图9-7(左)所示。右上图中用白色区域表示inpaintMask参数中不为0的像素，即需要处理的区域，右下图显示了对此区域进行处理之后的效果。

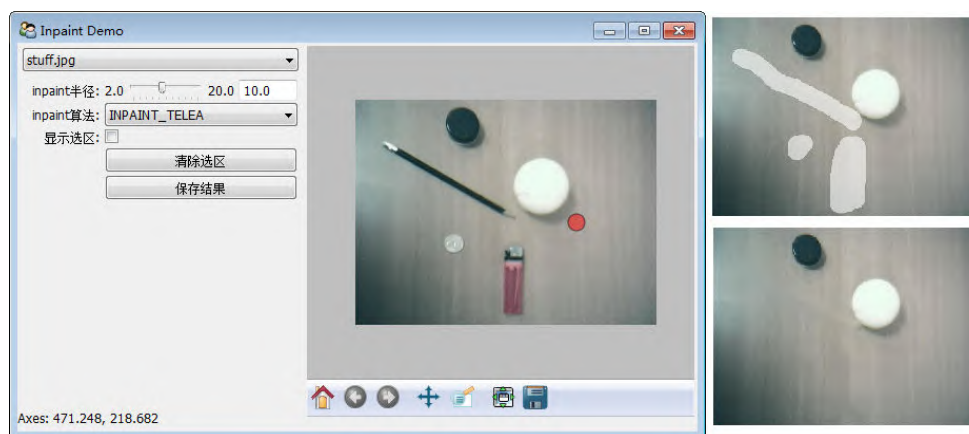


图9-7 使用inpaint去除图像中的物体



scipy2.opencv.inpaint_demo: 演示inpaint()的用法，用户用鼠标绘制需要去瑕疵的区域，程序实时显示运算结果。

在本书提供的inpaint_demo程序中，用鼠标绘制需要进行处理的区域之后，可以修改“inpaint半径”和“inpaint算法”等设置，实时观察它们对处理结果的影响。如果选区过大，处理可能需要较长时间，此时可以单击“保存结果”按钮，用当前的处理结果覆盖原始图像，并清除选区，以进行下一轮处理。

9.3 图像变换



与本节内容对应的Notebook为：09-opencv/opencv-300-transforms.ipynb

本节介绍一些常用的图像变换算法，其中包括：对图像中的像素坐标进行几何变换、对像素颜色进行转换、计算频域信息以及使用双目图像计算深度信息。

9.3.1 几何变换

我们可以对图像在二维平面上进行仿射变换，或者在三维空间中进行透视变换。仿射变换相当于将二维平面上的每个坐标点与一个 2×3 的矩阵相乘，得到新的坐标，而透视变换则是与 3×3 的矩阵相乘。原本平行的两条直线在经过仿射变换之后仍然是平行的，而经过透视变换之后，它们就可能不再平行了。

OpenCV中使用`warpAffine()`对图像进行仿射变换，调用参数如下：

```
warpAffine(src, M, dsize[, dst[, flags[, borderMode[,  
borderValue]]]])
```

其中`src`参数是变换的原始图像，`dsize`参数为返回图像的大小，返回图像的像素类型和`src`的相同。`M`参数是仿射变换的矩阵，它是一个形状为 $(2, 3)$ 的数组。`flags`参数是内插方式，`borderMode`是外插方式，`borderValue`为背景颜色。关于这些参数的含义请读者阅读OpenCV的文档。

假设矩阵`M`的各个元素如下：

$$\begin{pmatrix} a_{00} & a_{01} & b_0 \\ a_{10} & a_{11} & b_1 \end{pmatrix}$$

那么仿射变换可以用下面的公式表示：

$$\text{dst}(a_{00}x + a_{01}y + b_0, a_{10}x + a_{11}y + b_1) = \text{src}(x, y)$$

仿射变换矩阵中有6个参数，因此只需要指定变换前后3个坐标点的坐标，就可以通过解线性方程组获得变换矩阵。OpenCV提供了`getAffineTransform(src, dst)`来快速完成这种计算。`src`和`dst`参数

是变换前后的三个点的坐标，它们都是形状为(3, 2)的单精度浮点数数组。下面的程序演示了这两个函数的用法，效果如图9-8所示：

```
img = cv2.imread("lena.jpg")
h, w = img.shape[:2]
src = np.array([[0, 0], [w - 1, 0], [0, h - 1]],
dtype=np.float32) ❶
dst = np.array([[300, 300], [873, 78], [161, 923]],
dtype=np.float32) ❷

m = cv2.getAffineTransform(src, dst) ❸
result = cv2.warpAffine(
    img, m, (2 * w, 2 * h), borderValue=(255, 255, 255,
255)) ❹
```



图9-8 对图像进行仿射变换

❶src为图9-8中三角形的三个顶点坐标，这三个点分别为图像的左上、右上和左下三个顶点。❷dst为这三个顶点经过仿射变换之后的坐标，图中用三个箭头连接仿射变换前后的坐标点。❸调用getAffineTransform()得到仿射变换矩阵m，然后❹调用warpAffine()对图像img进行仿射变换，结果图像的大小为原始图像的两倍，背景采用白色填充。

warpPerspective()和warpAffine()类似，也对图像进行几何变换，不过它是在三维空间中进行透视变换，因此它的变换矩阵是 3×3 的矩阵。这个变换矩阵可以通过getPerspectiveTransform(src, dst)

计算。src和dst参数是变换前后的4个点的坐标，它们都是形状为(4, 2)的单精度浮点数数组。下面的程序演示了这两个函数的用法，结果如图9-9所示。

```
src = np.array(
    [[0, 0], [w - 1, 0], [w - 1, h - 1], [0, h - 1]],
    dtype=np.float32)

dst = np.array(
    [[300, 350], [800, 300], [900, 923], [161, 923]],
    dtype=np.float32)

m = cv2.getPerspectiveTransform(src, dst)
result = cv2.warpPerspective(
    img, m, (2 * w, 2 * h), borderValue=(255, 255, 255,
255))
```



图9-9 对图像进行透视变换

为了便于直观地理解仿射变换和透视变换，请读者运行下面的演示程序，界面如图9-10所示。



scipy2.opencv.warp_demo: 仿射变换和透视变换的演示程序，可以通过鼠标拖曳图中蓝色三角形和四边形的顶点，从而决定原始图像各个顶角经过变换之后的坐标。

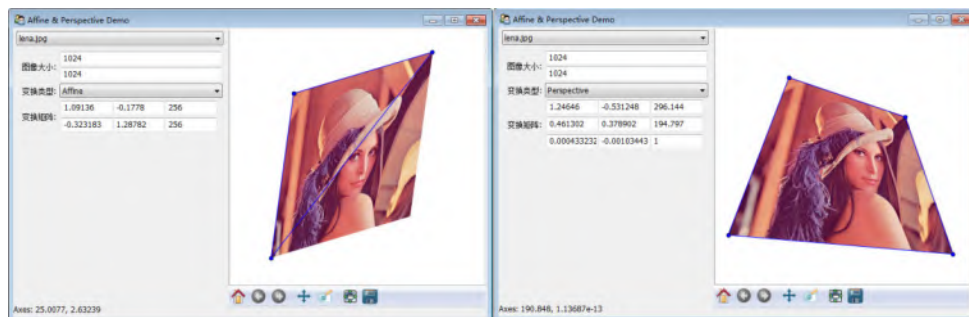


图9-10 仿射变换和透视变换演示程序

9.3.2 重映射-remap

对于图像的各种变换都有一个共同特点：它们从原始图像上的某个位置取出一个像素点，并把它绘制到目标图像上的另外一个位置。从原始坐标到目标坐标的映射不一定是一对一的关系。OpenCV提供了一个通用的图像映射函数`remap()`来完成这种计算，其调用参数如下：

```
remap(src, map1, map2, interpolation[, dst[, borderMode[,
borderValue]]])
```

其中`map1`和`map2`参数是两个大小与原始图像`src`相同的数组，它们的元素值是图像`dst`中对应下标的像素点在图像`src`中的坐标值，其元素可以是整数或单精度浮点数。`map1`中存储映射的X轴坐标，而`map2`中存储映射的Y轴坐标。下面的数学公式表示了这种映射关系，其中`x`和`y`是目标图像中每个像素的坐标，通过`map1`和`map2`分别获得它们在`src`中的坐标。

$$\text{dst}(x, y) = \text{src}(\text{map1}(x, y), \text{map2}(x, y))$$

下面的程序使用`remap()`将图像Y轴方向缩小为原来的三分之一，将X轴方向缩小为原来的一半，`img2[y, x]`的像素值与`img[mapy[y, x], mapx[y, x]]`的像素值相同。程序中通过`interpolation`参数指定采用线性插值`INTER_LINEAR`，读者可以通过IPython的自动完成功能查看其他的插值选项。

```
mapy, mapx = np.mgrid[0:h * 3:3, 0:w * 2:2]
img2 = cv2.remap(img, mapx.astype("f32"),
mapy.astype("f32"), cv2.INTER_LINEAR)
x, y = 12, 40 #用于验证映射公式的坐标点
assert np.all(img[mapy[y, x], mapx[y, x]] == img2[y, x])
```

缩小之后的图像`img2`的大小仍然和原始图像`img`相同，但是其中只有左上部分有图像数据。这里使用`mgrid`对象直接创建两个映射数组`mapx`和`mapy`。

为了演示`remap()`的强大功能，下面的程序让用户输入一个三维空间的曲面函数，程序将根据此函数所计算的曲面对图像进行变形。效果如图9-11所示，就像是将图像贴在曲面上一样。

```
def make_surf_map(func, r, w, h, d0):
    """计算曲面函数func在[-r:r]范围内的值，并进行透视投影。
    视点高度为曲面高度的d0倍+1"""
    y, x = np.ogrid[-r:r:h * 1j, -r:r:w * 1j]
    z = func(x, y) + 0 * (x + y) ❶
    d = d0 * np.ptp(z) + 1.0 ❷
    map1 = x * (d - z) / d ❸
    map2 = y * (d - z) / d
```

```
        return (map1 / (2 * r) + 0.5) * w, (map2 / (2 * r) +  
0.5) * h ④
```

```
def make_func(expr_str):  
    def f(x, y):  
        return eval(expr_str, np.__dict__, locals())  
    return f  
  
def get_latex(expr_str):  
    import sympy  
    x, y = sympy.symbols("x, y")  
    env = {"x": x, "y": y}  
    expr = eval(expr_str, sympy.__dict__, env)  
    return sympy.latex(expr)  
  
settings = [  
    ("sqrt(8 - x**2 - y**2)", 2, 1),  
    ("sin(6*sqrt(x**2+y**2))", 10, 10),  
    ("sin(sqrt(x**2+y**2))/sqrt(x**2+y**2)", 20, 0.5)  
]  
  
fig, axes = pl.subplots(1, len(settings), figsize=(12, 12.0  
/ len(settings)))  
  
for ax, (expr, r, height) in zip(axes, settings):  
    mapx, mapy = make_surf_map(make_func(expr), r, w, h,  
height)  
    img2 = cv2.remap(  

```

```

        img, mapx.astype("f32"), mapy.astype("f32"),
cv2.INTER_LINEAR)

ax.imshow(img2[:, :, ::-1])

ax.axis("off")

ax.set_title("${}\$".format(get_latex(expr)))

fig.subplots_adjust(0, 0, 1, 1, 0.02, 0)

```



图9-11 使用三维曲面和remap()对图片进行变形

程序中，先计算指定范围内的网格 x 、 y ，❶然后计算出网格上每一点的曲面的高度 z 。❷通过曲面的高度范围和 $d0$ 参数决定观察点的高度。❸然后利用投影变换公式，计算出表示坐标变换的两个数组 $mapx$ 和 $mapy$ 。其中投影公式的示意图如图9-12所示。❹最后将 $mapx$ 和 $mapy$ 的取值范围改为图像的范围之内。

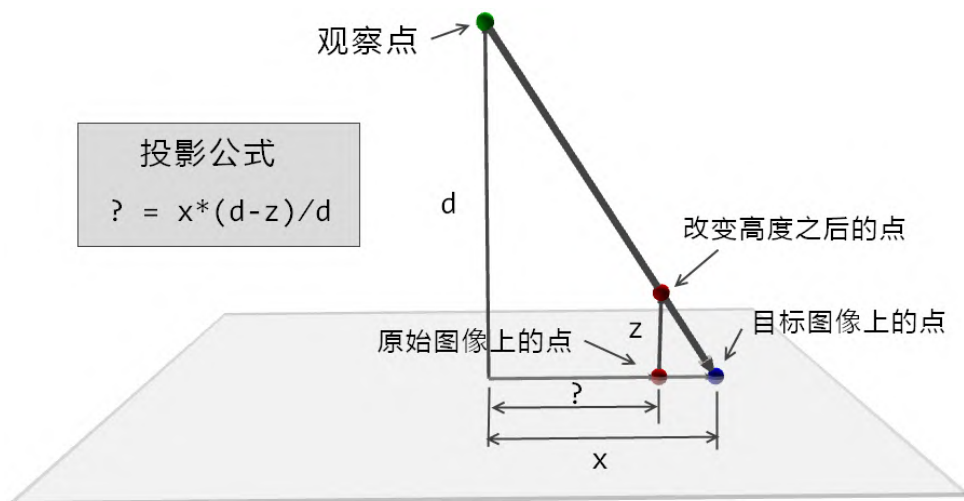


图9-12 投影公式示意图

还可以用`remap()`移动图像中指定的区域，使用该算法可以让用户通过鼠标拖曳图像的局部使其变形。在下面的例子中， (tx, ty) 为鼠标拖曳的起始坐标， (sx, sy) 为拖曳的终点坐标， r 是拖曳半径，半径越大被影响的像素越多。

为了方便程序的编写，我们将`remap()`的`map1`和`map2`两个参数分别分解为两个部分`gridx`和`offsetx`、`gridy`和`offsety`。①`gridx`和`gridy`为恒等映射，使用这两个数组不会对图像产生任何改变。②在终点坐标处创建一个半径为 r 的圆形遮罩数组`mask`。③将`offsetx`和`offsety`中与`mask`对应的值修改为拖曳的起点与终点的坐标差。如果使用这时的`gridx + offsetx`和`gridy + offsety`作为坐标映射，就会将以起点坐标为中心、半径为 r 的圆形区域复制到终点对应的区域。④为了让变形更加柔和，我们使用`GaussianBlur()`对两个`offset`数组进行高斯模糊处理，`sigma`参数决定模糊的程度，该值越大越模糊。

图9-13显示了变形之后的效果，其中虚线圆表示拖曳的起始位置，实线圆表示拖曳的终点位置。



scipy2.opencv.remap_demo: 演示remap()的拖曳效果。在图像上按住鼠标左键进行拖曳，每次拖曳完成之后，都将修改原始图像，可以按鼠标右键撤销上次的拖曳操作。

```
img = cv2.imread("lena.jpg")
h, w = img.shape[:2]
gridy, gridx = np.mgrid[:h, :w] ❶
tx, ty = 313, 316
sx, sy = 340, 332
r = 40.0
sigma = 20

mask = ((gridx - sx) ** 2 + (gridy - sy) ** 2) < r ** 2 <
❷

offsetx = np.zeros((h, w))
offsety = np.zeros((h, w))
offsetx[mask] = tx - sx ❸
offsety[mask] = ty - sy
offsetx_blur = cv2.GaussianBlur(offsetx, (0, 0), sigma) ❹
offsety_blur = cv2.GaussianBlur(offsety, (0, 0), sigma)
img2 = cv2.remap(img,
                  (offsetx_blur + gridx).astype("f4"),
                  (offsety_blur + gridy).astype("f4"),
                  cv2.INTER_LINEAR)
```



图9-13 使用remap()实现图像拖曳效果

9.3.3 直方图

在NumPy中有三个直方图统计函数：histogram()、histogram2d()和histogramdd()，分别对应一维数据、二维数据以及多维数据的情况。下面的程序用histogram()和histogram2d()对图像的颜色分布进行统计，输出如图9-14所示的统计结果。

```
img = cv2.imread("lena.jpg")
fig, ax = pl.subplots(1, 2, figsize=(12, 5))
colors = ["blue", "green", "red"]

for i in range(3):
    hist, x = np.histogram(img[:, :, i].ravel(), bins=256,
range=(0, 256)) ❶
```

```

        ax[0].plot(0.5 * (x[:-1] + x[1:]), hist,
label=colors[i], color=colors[i])

ax[0].legend(loc="upper left")
ax[0].set_xlim(0, 256)
hist2, x2, y2 = np.histogram2d( ❷
    img[:, :, 0].ravel(), img[:, :, 2].ravel(),
    bins=(100, 100), range=[(0, 256), (0, 256)])
ax[1].imshow(hist2, extent=(0, 256, 0, 256), origin="lower",
cmap="gray")
ax[1].set_ylabel("blue")
ax[1].set_xlabel("red")

```

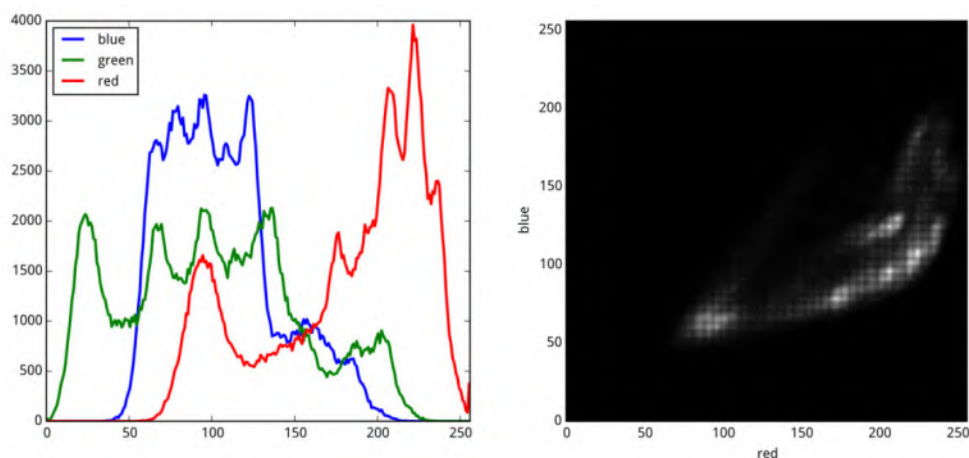


图9-14 lena.jpg的三个通道的直方图统计(左)，通道0和通道2的二维直方图统计(右)

❶通过`histogram()`对图像的三个通道分别进行一维直方图统计，由于被统计的数组必须是一维的，因此这里调用数组的`ravel()`方法将二维数组转换为一维数组。通过`range`参数指定统计区间为0~256，`bin`参数指定将统计区间等分为256份。`histogram()`返回两个数

组hist和x，其中hist为统计结果，长度为bin。而x为统计区间，长度为bin+1。hist[i]的值为数组中满足 $x[i] \leq v < x[i+1]$ 的元素v的个数。

②用histogram2()对通道0和通道2进行二维直方图统计。被统计的数组是两个一维数组，因此也需要用ravel()方法进行转换。它们分别为图像的通道0和通道2的数据。bins和range参数都变成了有两个元素的序列，分别与两个数组相对应。返回的统计结果hist2是一个二维数组，其形状由bins决定。第0轴与第一个数组相对应，第1轴与第二个数组相对应。它是由两个一维数组的对应元素所构成的二维矢量的分布统计结果。

观察图9-14(左)可知红色通道的值普遍较大，因此整个图片呈现暖色调；而从右图不但可以得出红色通道的值比蓝色通道较大的结论，还可以看到几处分布比较密集的区域。例如其中一块的中心坐标大约为(207, 125)，这说明图像中红色值在207附近、蓝色值在125附近的像素点较多。

OpenCV中的直方图统计函数为calcHist()。它支持对多幅图像进行N维直方图统计，因此其第一个参数为数组列表。下面使用calcHist()对img的三个通道(0, 1, 2)进行三维直方图统计。每个通道的等分数分别为(30, 20, 10)，所有通道的取值范围都为(0, 256)。返回结果result是一个形状为(30, 20, 10)的数组：

```
result = cv2.calcHist([img],  
                      channels=(0, 1, 2),  
                      mask = None,  
                      histSize = (30, 20, 10),
```

```
ranges = (0, 256, 0, 256, 0, 256))

result.shape

(30, 20, 10)
```

1. 直方图反向映射

计算出直方图之后，可以用`calcBackProject()`将图像中的每点替换为它在直方图中所对应的值。于是在直方图中出现次数越高，图像中对应的像素就越亮。可以用这种方法找出图像中和直方图相匹配的区域。下面用一个实际的例子加以说明：

```
img = cv2.imread("fruits_section.jpg") ❶
img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

result = cv2.calcHist([img_hsv], [0, 1], None, ❷
                      [40, 40], [0, 256, 0, 256])

result /= np.max(result) / 255 ❸

img2 = cv2.imread("fruits.jpg") ❹
img_hsv2 = cv2.cvtColor(img2, cv2.COLOR_BGR2HSV)

img_bp = cv2.calcBackProject([img_hsv2], ❺
                             channels=[0, 1],
                             hist=result,
                             ranges=[0, 256, 0, 256],
```

```

                                scale=1)
_, img_th = cv2.threshold(img_bp, 180, 255,
cv2.THRESH_BINARY) ❹
struct = np.ones((3, 3), np.uint8)
img_mp = cv2.morphologyEx(img_th, cv2.MORPH_CLOSE, struct,
iterations=5) ❺

```

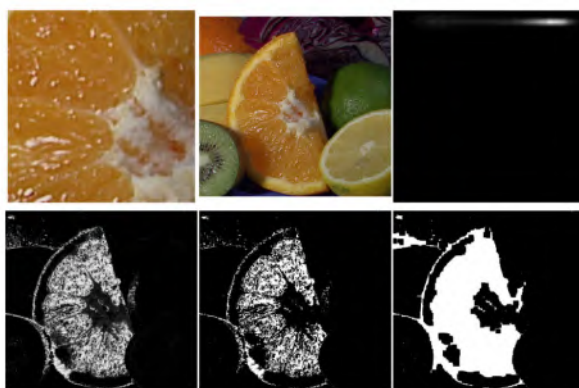


图9-15 使用`calcBackProject()`寻找图像中的橙子部分

程序的输出如图9-15所示。它在图像`fruits.jpg`(上中图)中寻找和图像`fruits_section.jpg`(上左图)的颜色近似的部分。

❶首先载入颜色匹配的模板图像，并通过`cvtColor()`将图像的三通道的数据从蓝绿红变换为色相、饱和度和明度。❷调用`calcHist()`对模板图像的色相与饱和度进行二维直方图计算。在色相与饱和度空间进行颜色匹配，能够得到较好的匹配结果。❸为了后续的`calcBackProject()`计算不越界，这里将直方图的最大值缩小到255。图9-15(上右)为所计算的直方图。

❹载入目标图像，然后也通过`cvtColor()`进行颜色转换。❺调用`calcBackProject()`将目标图像中的每个像素的颜色变换为其在直方图中所对应的值。它的第一个参数是一个图像列表，`hist`参数指定直方

图，返回值是一幅单通道的形状和数值类型与输入图像相同的图像。`channels`、`ranges`参数和`calcHist()`的参数含义相同。图9-15(下左)为`calcBackProject()`的计算结果。

❶调用`threshold()`对`calcBackProject()`的输出图像进行二值化处理，参数`THRESH_BINARY`指定了二值化处理的方法，它将图像中值小于等于180的点都设置为0，将大于180的设置为255。图9-15(下中)为二值化的结果。

❷最后对二值化之后的图像进行形态学图像处理。使用5次开运算将图像中的分散的区域连接成一个大区域。图9-15(下右)为最终结果，它的白色区域正好对应目标图像中的橙子部分。还可以对这个结果进行一些处理：例如使用闭运算消除一些杂点，然后找出图像中面积最大的区域。

这种方法也可以用于在视频中跟踪一个颜色鲜明的物体。在跟踪物体之前，首先对一幅物体充满整个画面的图像进行直方图统计，然后对视频后续的帧进行`calcBackProject()`计算。

2. 直方图匹配

直方图可以表示图像的颜色分布情况，而通过直方图匹配算法可以将一幅图像的直方图分布复制给另一幅图像，从而让目标图像拥有原图像的直方图信息。基本步骤如下：

❶计算原图像`src`与目标图像`dst`的归一化之后的直方图统计，得到的结果为概率密度分布。❷用`cumsum()`对概率密度分布进行累加，

得到累计分布。❸在原图像的累计分布中搜索目标图像的累计分布所对应的下标index，由于累计分布是递增函数，因此可以用searchsorted()进行二分查找。❹调用clip()将index的取值范围限制在0~255之间，❺然后用它对目标图像进行映射，即将目标图像中每个像素值v替换为index[v]。

```
def histogram_match(src, dst):
    res = np.zeros_like(dst)
    cdf_src = np.zeros((3, 256))
    cdf_dst = np.zeros((3, 256))
    cdf_res = np.zeros((3, 256))
    kw = dict(bins=256, range=(0, 256), normed=True)

    for ch in (0, 1, 2):
        hist_src, _ = np.histogram(src[:, :, ch], **kw) ❶
        hist_dst, _ = np.histogram(dst[:, :, ch], **kw)
        cdf_src[ch] = np.cumsum(hist_src) ❷
        cdf_dst[ch] = np.cumsum(hist_dst)
        index = np.searchsorted(cdf_src[ch], cdf_dst[ch],
side="left") ❸
        np.clip(index, 0, 255, out=index) ❹
        res[:, :, ch] = index[dst[:, :, ch]] ❺
        hist_res, _ = np.histogram(res[:, :, ch], **kw)
        cdf_res[ch] = np.cumsum(hist_res)

    return res, (cdf_src, cdf_dst, cdf_res)
```

下面调用`histogram_match()`将一幅秋景的直方图复制给夏景图像。图9-16显示了对夏景图进行直方图匹配处理前后的图像，图中下层的三个图表显示图像的蓝绿红三个通道的累计分布曲线。其中点线为处理之前夏景图的累计分布，实线为处理之后的累计分布，而虚线为秋景图的累计分布。可以看到图中实线与虚线几乎完全重合，因此完美地将秋景图的直方图复制给了夏景图：

```
src = cv2.imread("autumn.jpg")  
dst = cv2.imread("summer.jpg")  
  
res, cdfs = histogram_match(src, dst)
```

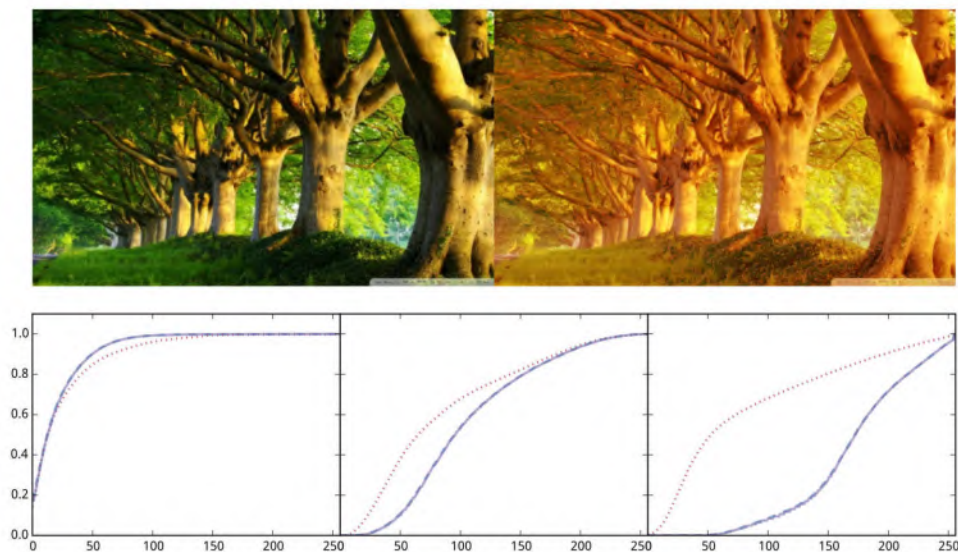


图9-16 直方图匹配结果

9.3.4 二维离散傅立叶变换

图像数据可以看作二维离散信号，对其进行二维离散傅立叶变换，能将其转换为频域信号，将原始图像分解为众多二维正弦波的叠加。由于NumPy已经提供了二维离散傅立叶变换的函数，因此本节主要使用NumPy的相关函数进行说明。

为了更好地理解本节所介绍的内容，需要读者掌握离散傅立叶变换相关的知识。在本书最后一章有一维离散傅立叶变换的详细论述。

对一维的 N 点实数信号 x 进行快速傅立叶变换(FFT)之后，得到表示频域信号的 N 个复数的数组 X 。但是数据的信息量并没有增加，这是因为：

- 下标为0和 $N/2$ 的两个复数的虚数部分为0。
- 下标为 i 和 $N-i$ 的两个复数共轭，也就是虚数部分数值相同、符号相反。

同样，对于一个 $N*N$ 的二维实数信号 x 进行二维快速傅立叶变换之后，得到表示频域信号的 $N*N$ 个复数元素的数组 X 。其中 $X[i, j]$ 和 $X[N-i, N-j]$ 共轭，并且 $X[0, 0]$ 、 $X[0, N/2]$ 、 $X[N/2, 0]$ 、 $X[N/2, N/2]$ 这4个元素的虚部为0。下面我们用程序验证一下：

```
from numpy import fft
x = np.random.rand(8, 8)
X = fft.fft2(x)
print np.allclose(X[1:, 1:], X[7:0:-1, 7:0:-1].conj()) # 共轭复数
print X[:, :4, :, :4] # 虚数为零
True
```

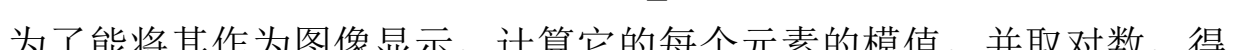
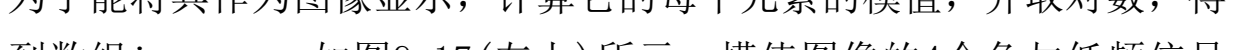
```
[[ 31.48765415+0.j   -2.80563949+0.j]
 [  0.75758598+0.j   -0.53147589+0.j]]
```

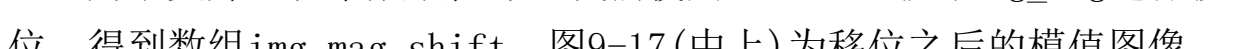
频域信号通过`ifft2()`可以转换回空域信号，结果和原始的空域信号完全相等。但是`ifft()`所得到的仍然是一个复数数组，只是每个元素的虚部都十分接近于0。

```
x2 = fft.ifft2(X) # 将频域信号转换回空域信号
np.allclose(x, x2) # 和原始信号进行比较
True
```

频域信号中的每个元素都对应空域信号中的一个二维正弦波，如果只选择频域信号中的一部分转换回空域信号，就相当于对空域信号进行了滤波处理。下面演示将频域信号中的不同区域转换回空域信号之后的滤波效果，输出如图9-17所示。

首先载入一幅彩色图像，并将其转换为灰度图像。由于FFT运算的最佳大小为2的整数次幂，因此使用`resize()`将图像的大小改为256*256。

然后计算图像的频域信号，由于它是一个复数数组，为了能将其作为图像显示，计算它的每个元素的模值，并取对数，得到数组，如图9-17(左上)所示。模值图像的4个角与低频信号对应，中心与高频信号对应。由于4个角附近较亮，这说明原始图像的低频成分较多，这符合一般图像信号的规律。

为了更好地观察频域信号，我们使用`fftshift()`对进行移位，得到数组。图9-17(中上)为移位之后的模值图像。

fftshift()将两个对角线上的方块对调，即1、3象限对调，2、4象限对调。这样图像的中部与低频对应，而4角与高频信号对应。

```
N = 256

img = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)
img = cv2.resize(img, (N, N))
img_freq = fft.fft2(img)
img_mag = np.log10(np.abs(img_freq))
img_mag_shift = fft.fftshift(img_mag)

rects = [(80, 125, 85, 130), (90, 90, 95, 95),
          (150, 10, 250, 250), (110, 110, 146, 146)]
```

最后选择频域信号中的一部分，将其转换回空域信号。图9-17的右上图和下排的图，分别显示中上图中4个矩形领域所对应的空域图像。

❶mask是一个布尔数组，其形状和频域信号数组一样。❷将其中坐标在指定的矩形范围内的元素设置为True。❸同时选择共轭对称的部分，否则通过ifft2()转换回空域信号时虚部将不会为0。❹通过fftshift()对mask数组进行移位，使得它和频域信号img_freq匹配。

❺接下来将频域信号img_freq与mask相乘，得到在频域进行滤波之后的频域信号img_freq2。❻然后调用ifft2()将img_freq2转换回空域信号。



scipy2.opencv.fft2d_demo: 演示二维离散傅立叶变换，用户在左侧的频域模值图像上用鼠标绘制遮罩区域，右侧的图像为频域信号经过遮罩处理之后转换成的空域信号。

```
filtered_results = []
for i, (x0, y0, x1, y1) in enumerate(rects):
    mask = np.zeros((N, N), dtype=np.bool) ❶
    mask[x0:x1 + 1, y0:y1 + 1] = True ❷
    mask[N - x1:N - x0 + 1, N - y1:N - y0 + 1] = True ❸
    mask = fft.fftshift(mask) ❹
    img_freq2 = img_freq * mask ❺
    img_filtered = fft.ifft2(img_freq).real ❻
    filtered_results.append(img_filtered)
```

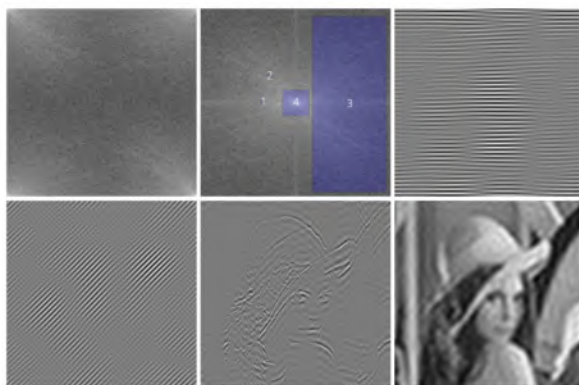


图9-17 (左上)用fft2()计算的频域信号，(中上)使用fftshift()移位之后的频域信号，(其他)各个领域所对应的空域信号

9.3.5 用双目视觉图像计算深度信息

所谓双目视觉是指模拟人眼处理场景的方式，用两台照相机从不同视点观察同一场景获得两幅图像。通过在左右两幅图像中匹配场景对应的点，计算出场景中各点距离照相机的距离，从而重建三维信息，原理如图9-18所示。

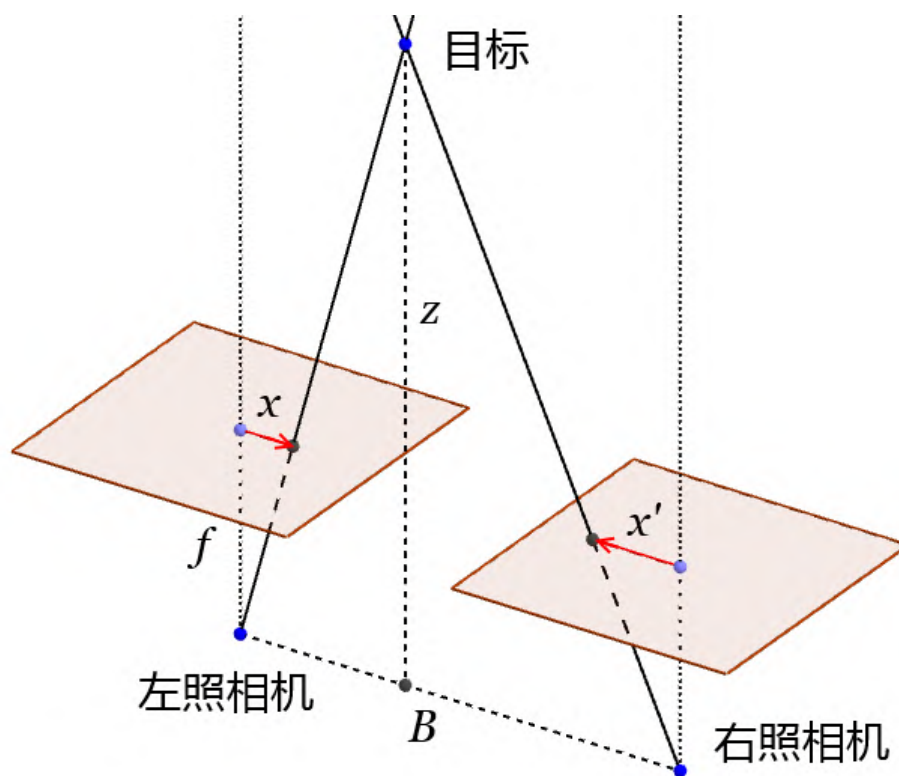


图9-18 双目视觉图像计算深度示意图

图9-18中，两台照相机的焦距均为 f ，距离为 B 。场景中目标点在左相机图像中的位置为 x' ，在右相机图像中的位置为 $x-x'$ 。同一点在两幅图中的视差(disparity)为，由相似三角形可知两照相机的连线的中心点到目标点的距离为：

$$z = \frac{B \cdot f}{x - x'}$$

由上面的公式可知视差越大，目标点到照相机的距离越小，读者可以试着轮流使用左右单眼观察场景，可以发现距离越远的物体偏移量越小。

视差信息可由OpenCV提供的StereoSGBM类计算。它有相当多的参数可供调节，为了获得较好的效果，需要仔细调节这些参数。下面的程序中的参数可用于通常情况。StereoSGBM.compute() 计算视差信息，得到的是一个双字节的整型数组，将其中的值除以16就可以得到以像素为单位的视差数据。

```
img_left = cv2.pyrDown(cv2.imread('aloeL.jpg'))
img_right = cv2.pyrDown(cv2.imread('aloeR.jpg'))

img_left = cv2.cvtColor(img_left, cv2.COLOR_BGR2RGB)
img_right = cv2.cvtColor(img_right, cv2.COLOR_BGR2RGB)

stereo_parameters = dict(
    SADWindowSize = 5,
    numDisparities = 192,
    preFilterCap = 4,
    minDisparity = -24,
    uniquenessRatio = 1,
    speckleWindowSize = 150,
    speckleRange = 2,
    disp12MaxDiff = 10,
```

```

        fullDP = False,

        P1 = 600,

        P2 = 2400)

    stereo = cv2.StereoSGBM(**stereo_parameters)

    disparity = stereo.compute(img_left,
img_right).astype(np.float32) / 16

```

若能完美地计算视差数组 `disparity`，则能满足等式：
`left_img[y, x] == right_img[y, x+disparity[y, x]]`。下面我们用前面介绍过的 `remap()` 将右眼图像中的像素移到与其对应的左眼图像的坐标之上，结果如图9-19所示。其中，左图为直接将左右两幅图像叠加之后的结果，可以看出不同的位置有不同的视差错位。中图以灰度图像显示视差信息，颜色越浅表示视差越大，距离越近。右图是将右眼图像经过 `remap()` 处理之后再叠加到左眼图像之上，可以看到两幅图几乎完美地重合了。

```

h, w = img_left.shape[:2]
ygrid, xgrid = np.mgrid[:h, :w]
ygrid = ygrid.astype(np.float32)
xgrid = xgrid.astype(np.float32)

res = cv2.remap(img_right, xgrid - disparity, ygrid,
cv2.INTER_LINEAR)

fig, axes = pl.subplots(1, 3, figsize=(9, 3))
axes[0].imshow(img_left)
axes[0].imshow(img_right, alpha=0.5)

```

```

axes[1].imshow(disparity, cmap="gray")
axes[2].imshow(img_left)
axes[2].imshow(res, alpha=0.5)
for ax in axes:
    ax.axis("off")
fig.subplots_adjust(0, 0, 1, 1, 0, 0)

```

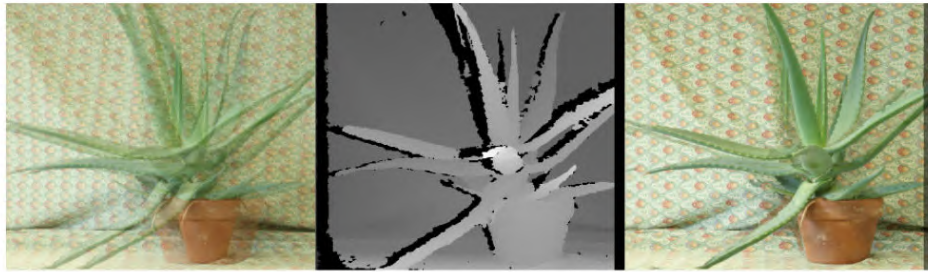


图9-19 用remap重叠左右两幅图像

下面设置焦距与相机间距的乘积 Bf ，根据视差信息计算每个点在三维空间中的位置。该三维坐标以两台照相机的连线为原点，连线方向为X轴，图像的上方为Y轴，照相机的前方为Z轴。

```

Bf = w * 0.8
x = (xgrid - w * 0.5)
y = (ygrid - h * 0.5)
d = (disparity + 1e-6)
z = (Bf / d).ravel()
x = (x / d).ravel()
y = -(y / d).ravel()

```

为了剔除一些噪声数据，我们只显示距离相机在30以内的坐标点，每个点对应的颜色从img_left图像获取。

```
mask = (z > 0) & (z < 30)
points = np.c_[x, y, z][mask]
colors = img_left.reshape(-1, 3)[mask]
```

有了点的坐标和颜色，就可以使用TVTK的PolyData将这些数据显示为点云。所创建的PolyData对象的points属性为所有点的坐标，为了对每个点进行着色，设置point_data.scalars属性为每个点的颜色，注意这里需要使用单字节的数组表示RGB颜色。该PolyData对象只有顶点(verts)，没有边线和面，每个顶点与points中的一个坐标相对应。

```
from tvtk.api import tvtk
from tvtk.tools import ivtk
from pyface.api import GUI

poly = tvtk.PolyData()
poly.points = points #所有坐标点
poly.verts = np.arange(len(points)).reshape(-1, 1) #所有顶点
与坐标点对应

poly.point_data.scalars = colors.astype(np.uint8) #坐标点的颜色，必须使用uint8

m = tvtk.PolyDataMapper()
m.set_input_data(poly)
a = tvtk.Actor(mapper=m)

from scipy2 import vtk_scene, vtk_scene_to_array
```

```
scene = vtk_scene([a], viewangle=22)
scene.camera.position = (0 , 20, -60)
scene.camera.view_up = 0, 1, 0
%array_image vtk_scene_to_array(scene)
```



图9-20 使用VTK显示三维点云

下面的程序在Notebook中用%gui启动GUI线程，打开三维窗口来观察点云：



scipy2.opencv.stereo_demo：使用双目视觉图像计算深度信息的演示程序。


```
%gui qt

from scipy2.tvtk.tvtkhelp import ivtk_scene

scene = ivtk_scene([a])

scene.scene.isometric_view()

scene.scene.camera.position = (0 , 20, -50)

scene.scene.camera.view_up = 0, 1, 0
```

9.4 图像识别



与本节内容对应的Notebook为：[09-opencv/opencv-400-identify.ipynb](#)。

OpenCV除了能够对图像进行各种处理和变换之外，还提供了大量的图像识别函数。本节介绍一些常用的图像识别和分割算法。

9.4.1 用霍夫变换检测直线和圆

用霍夫变换 (Hough transform) 能够找出图像中的直线和圆。OpenCV提供了如下三种霍夫变换相关的函数：

- `HoughLines`：检测图像中的直线。
- `HoughLinesP`：检测图像中的直线段。
- `HoughCircles`：检测图像中的圆。

下面的程序演示了使用HoughLinesP()和HoughCircles()进行线段和圆的检测。它们的各种参数均可以在控制面板中进行调整，运行界面如图9-21所示。

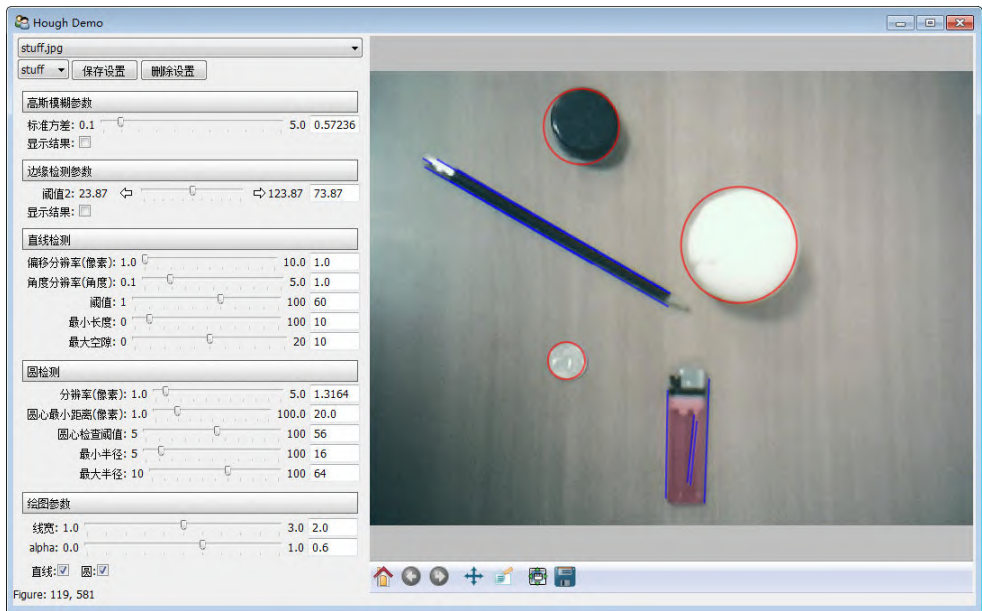


图9-21 用霍夫变换寻找图像中的直线和圆



scipy2.opencv.hough_demo: 霍夫变换演示程序，可通过界面调节函数的所有参数。

1. 检测线段

为了了解HoughLinesP()的每个参数的含义，让我们先学习直线霍夫变换的原理。

对于图像中的每条直线都可以用方程 $y=kx+m$ 表示。由于参数 k 和直线与 X 轴的夹角之间并不是线性关系，因此我们将直线方程改写为以直线到原点的距离 r 和直线与 X 轴的夹角 θ 为参数，如图9-22所示。

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right)$$

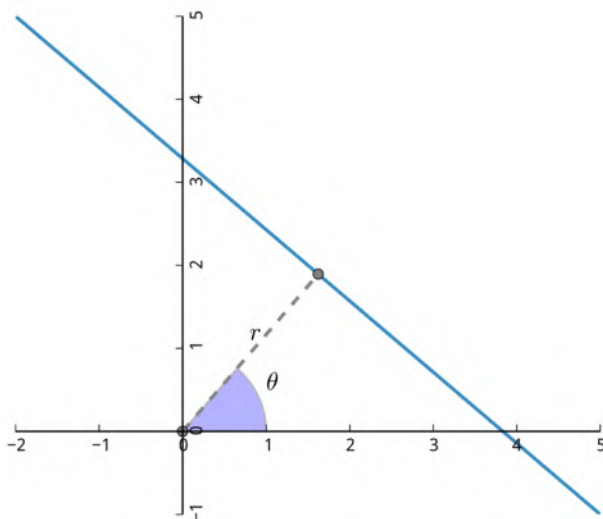


图9-22 用 r 和 θ 表示的直线

经过图像中某个白色的点 (x_0, y_0) 的直线参数 r 和 θ 满足下面的关系：

$$r = x_0 \cdot \cos\theta + y_0 \cdot \sin\theta$$

它是一条 $\theta - r$ 空间中的正弦曲线。所谓霍夫变换，就是指对于原始图像中的每个白色点 (x_0, y_0) ，绘制它们在 $\theta - r$ 空间中所对应的正弦曲线。众多正弦曲线的相交点 (θ_0, r_0) 就是原始图像中的一条直线。图9-23是一个简单的例子。其中，左图中的4个圆点构成一条直线，右图中与它们对应的正弦曲线(图中的实线)相交于一点，而与三角点对应的正弦曲线(虚线)则不经过此点。

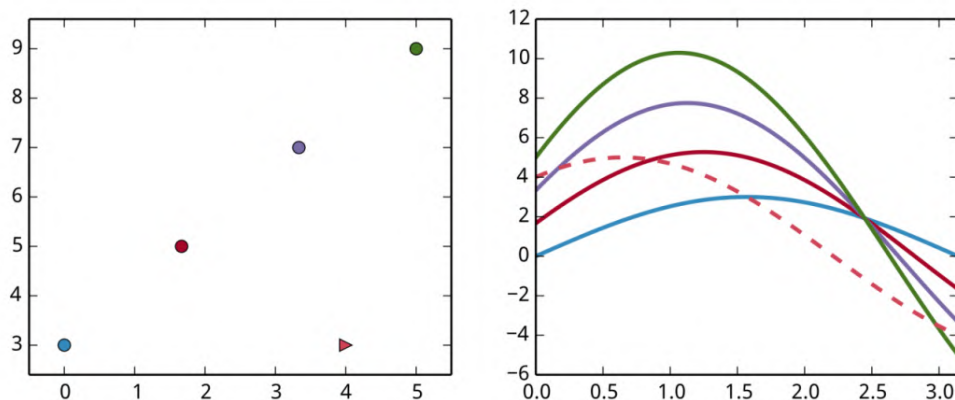


图9-23 霍夫变换示意图

在实际计算时，我们使用一幅表示 $\theta - r$ 空间的灰度图像作为累加器，用其中每个点对经过此点的正弦曲线进行计数。然后通过阈值找出累加器中的所有峰值点，这些峰值点所对应的 $\theta - r$ 坐标就是原始图像中的直线参数。

HoughLinesP() 的调用参数如下：

```
HoughLinesP(image, rho, theta, threshold[, lines[,
minLineLength[, maxLineGap]]])
```

其中image参数为进行直线检测的图像，rho和theta参数分别为累加器中每个点所表示的r和 θ 的大小。其中rho的单位是像素点，而theta是以弧度表示的角度。值越小则累加器的尺寸越大，最后找出的直线的参数的精度越高，但是运算时间也越长。threshold参数是在累加器中寻找峰值时所使用的阈值，即只有大于此值的峰值点才被当作与某条直线相对应。由于HoughLinesP()检测的是图像中的线段，因此minLineLength参数指定线段的最小长度，而maxLineGap参数则指定线段的最大间隙。当有多条线段共线时，间隙小于此值的线段将被合并为一条线段。

HoughLinesP() 的返回值是一个形状为(1, N, 4)的数组，其中N为线段数，第二轴的4个元素为线段的起点和终点：x0、y0、x1、y1。在下面的程序中使用matplotlib的LineCollection绘制这些线段，效果如图9-24所示。

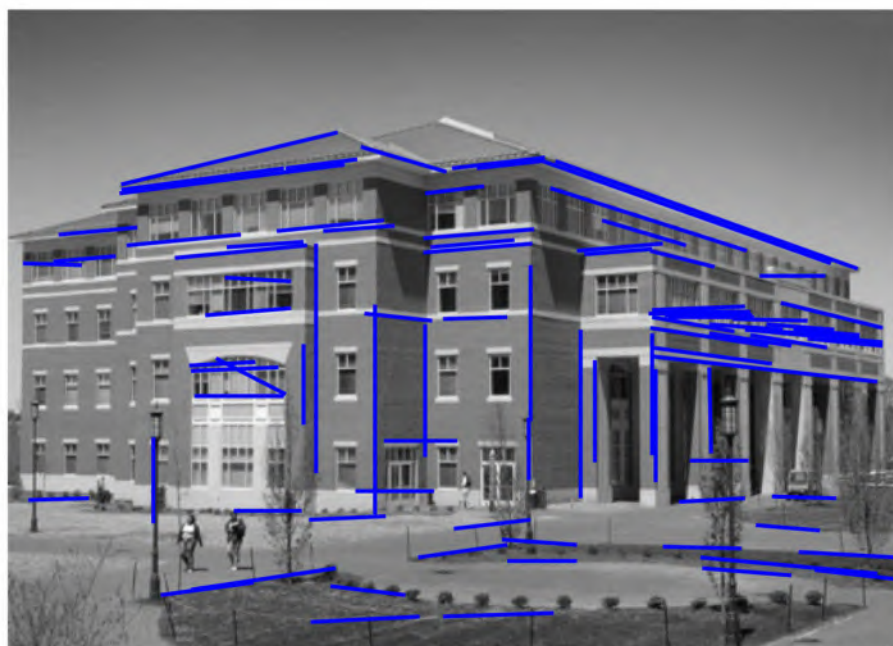


图9-24 使用HoughLinesP() 检测图像中的直线

由于HoughLinesP() 需要针对二值图像进行操作，因此先用Canny() 对灰度图像进行边缘检测，得到一幅二值图像img_binary。Canny() 有两个阈值参数，它们直接影响边缘检测的结果。阈值越小，从图像中检测出来的边缘细节越多。

```
img = cv2.imread("building.jpg", cv2.IMREAD_GRAYSCALE)
img_binary = cv2.Canny(img, 100, 255)
lines = cv2.HoughLinesP(img_binary, rho=1,
theta=np.deg2rad(0.1),
```

```
threshold=96, minLineLength=33,  
maxLineGap=4)  
  
fig, ax = plt.subplots(figsize=(8, 6))  
plt.imshow(img, cmap="gray")  
from matplotlib.collections import LineCollection  
lc = LineCollection(lines.reshape(-1, 2, 2))  
ax.add_collection(lc)  
ax.axis("off")
```

2. 检测圆形

检测圆形的HoughCircles()的参数如下:

```
HoughCircles(image, method, dp, minDist  
              [, circles[, param1[, param2[, minRadius[,  
maxRadius]]]])
```

其中method参数为圆形检测的算法，目前OpenCV中只实现了一种检测算法：CV_HOUGH_GRADIENT。dp参数和直线检测中的rho参数类似，决定了检测的精度，dp=1时累加器的分辨率和输入图像相同，而dp=2时累加器的分辨率为输入图像的一半。minDist参数是检测到的所有圆的圆心之间的最小距离，当它过小时会检测出很多近似的圆形，若过大则可能会漏掉一些结果。

param1 和 param2 参数是和检测算法相关的参数。在HoughCircles()内部会进行边缘检测，其中param1参数相当于边缘检

测Canny()的第二个阈值，Canny()的第一个阈值自动设置为它的一半。param2参数是累加器上的阈值，它的值越小检测出的圆形越多。minRadius和maxRadius参数指定圆形的半径范围，缺省都为0表示范围不限。

由于圆形有三个参数——圆心坐标和半径，如果直接使用三维累加器，则计算效率太低。并且由于累加器中每个点的累计次数不够多，会出现很多局部峰值，也会影响检测结果。因此HoughCircles()使用一种被称作霍夫梯度的算法进行圆形检测。它的计算步骤如下：

(1) 首先将原始图像经过边缘检测算法获得一张边缘图像，这里使用Canny()进行边缘检测，并使用param1参数指定阈值。

(2) 对于边缘图像中每个白色的点(x0, y0)计算其局部梯度，这里使用Sobel()进行梯度计算。假设白色点为圆周上的某点，经过(x0, y0)沿着梯度方向的直线将通过圆心。

(3) 对梯度直线上离点(x0, y0)的距离在minRadius和maxRadius之间的所有点，在累加器中进行计数。

(4) 累加器中大于阈值param2的局部峰值为图像中所检测出的圆形的中心。

然后对于每个检测出的圆心，在边缘图像中寻找离它距离相同的白色点的集合，并计算出半径。如果此圆心有足够多的白色点支持，那么它就是真正的圆心。

HoughCircles()返回一个形状为(1, N, 3)的数组，其中N为圆形的个数，第2轴上的三个元素分别为圆心的X轴坐标、Y轴坐标和圆形的

半径。

在下面的程序中(效果如图9-25所示)，❶为了获得较好的边缘检测结果，调用 `GaussianBlur()` 对图像进行模糊处理。❷使用 `EllipseCollection` 快速绘制多个圆形。

```
img = cv2.imread("coins.png", cv2.IMREAD_GRAYSCALE)
img_blur = cv2.GaussianBlur(img, (0, 0), 1.8) ❶
circles = cv2.HoughCircles(img_blur, cv.CV_HOUGH_GRADIENT,
dp=2.0, minDist=20.0,
                                param1=170, param2=44, minRadius=16,
maxRadius=40)

x, y, r = circles[0].T

fig, ax = plt.subplots(figsize=(8, 6))
plt.imshow(img, cmap="gray")
from matplotlib.collections import EllipseCollection
ec = EllipseCollection(widths=2*r, heights=2*r, angles=0,
units="xy", ❷
                                facecolors="none", edgecolors="red",
                                transOffset=ax.transData,
offsets=np.c_[x, y])
ax.add_collection(ec)
ax.axis("off")
```




图9-25 使用HoughCircles()检测图像中的圆形

9.4.2 图像分割

一般的图像中颜色丰富、信息繁杂，不利于计算机进行图像识别。因此通常会使用图像分割技术，将图像中相似的区域进行合并，使得图像更容易理解和分析。本节将介绍OpenCV中提供的两种常见的图像分割算法。

1. Mean-Shift算法

`pyrMeanShiftFiltering()` 使用Mean-Shift算法对图像进行分割。它的调用参数如下：

```
pyrMeanShiftFiltering(src, sp, sr[, dst[, maxLevel[,  
termcrit]]])
```

`pyrMeanShiftFiltering()` 以src中的每个点(x, y)为初始点，寻找与它邻近的点。这里的邻近点必须满足下面两个条件：

- 在以(x, y)为中心的边长为2*sp的正方形范围内。
- 和点(x, y)的颜色距离小于sr参数，也就是将3个颜色通道当作三维向量，在此颜色空间中，两个点的距离小于sr。

然后计算邻近点的坐标平均值和颜色平均值，并以此平均点再次寻找图像中的邻近点。如此迭代下去，直到达到迭代终止条件。将迭代终止时的颜色平均值写进图像dst的坐标点(x, y)。

在OpenCV中，所有的迭代算法都可以通过termcrit设置迭代相关的参数，它是一个有三个元素的元组：(type, maxCount, epsilon)。maxCount为最大迭代次数；epsilon为迭代终止时的误差，即两次迭代结果的差小于此值时将结束计算；type指定哪种终止条件有效，3表示两种终止条件都有效。

max_level参数指定使用图像金字塔进行计算。当使用图像金字塔时，先对低分辨率的图像进行分割计算，然后利用此结果对高分辨率的图像进行分割。

下面是使用pyrMeanShiftFiltering()进行图像分割的演示程序，效果如图9-26所示。

```

fig, axes = pl.subplots(1, 3, figsize=(9, 3))

img = cv2.imread("fruits.jpg")

srs = [20, 40, 80]
for ax, sr in zip(axes, srs):
    img2 = cv2.pyrMeanShiftFiltering(img, sp=20, sr=sr,
maxLevel=1)

    ax.imshow(img2[:, :, ::-1])
    ax.set_axis_off()
    ax.set_title("sr = {}".format(sr))

fig.subplots_adjust(0.02, 0, 0.98, 1, 0.02, 0)

```



图9-26 使用pyrMeanShiftFiltering()进行图像分割，从左到右参数sr分别为20、40、80

2. 分水岭算法

分水岭算法 (Watershed) 的基本思想是将图像的梯度当作地形图。图像中变化小的区域相当于地形图中的山谷，而变化大的区域相当于山峰。从指定的几个初始区域同时开始向地形灌不同颜色的水，水面上升逐渐淹没山谷，并且范围逐渐扩大。请注意这里所说的“颜色”是用来区分不同区域的一个数值，和图像的颜色没有任何关系。当所有区域的水面连接到一起时，所得到的不同颜色的灌溉区域就是最终的图像分割结果，最终的分割区域数和初始区域数相同。watershed() 实现此算法，它的调用形式如下：

```
watershed(image, markers)
```

image 参数是需要进行分割处理的图像，它必须是一个3通道8位图像。markers 参数是一个32位整数数组，其大小必须和image相同。markers 中值大于0的点构成初始灌溉区域，其值可以理解为水的颜色。调用watershed() 之后，markers 中几乎所有的点都将被赋值为某个初始区域的值，而在两个区域的境界线上的点将被赋值为-1。

下面用watershed() 对如图9-27(左)所示的药丸图片进行分割。图中每片药丸上的蓝色区域为使用下面的程序找到的局域最亮像素。我们使用这些像素作为每片药丸的初始灌溉区域。为了让每片药丸只有一个局域最亮像素，❶先调用blur() 对药丸的灰度图像进行模糊处理，这样可以有效消除噪声，减少局域最值的个数。❷我们只希望找到药丸区域对应的局域最亮像素，因此对灰度图像进行二值化处理，img_binary 中白色区域与药丸对应。❸局域最亮像素就是比周边临近的像素都亮的像素，使用dilate() 对灰度图像进行膨胀处理，将每个像素都设置为邻近像素中的最大值，然后与原始图像中的值比较，如果值保持不变，该像素即为局域最亮像素。

```

img = cv2.imread("pills.png")
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_gray = cv2.blur(img_gray, (15, 15)) ❶
_, img_binary = cv2.threshold(img_gray, 150, 255,
cv2.THRESH_BINARY) ❷
peaks = img_gray == cv2.dilate(img_gray, np.ones((7, 7)), 1)
❸
peaks &= img_binary
peaks[1, 1] = True ❹

from scipy.ndimage import label
markers, count = label(peaks) ❺
cv2.watershed(img, markers)

```

❺接下来需要对peaks中的每块区域进行编号，OpenCV中没有提供相应的函数，我们可以使用SciPy的图像处理一节中介绍的labels()对每块区域进行编号。然后使用编号之后的结果作为watershed()的markers参数。❹由于分水岭算法需要我们在每块分割区域中都设置初始值，❹因此为了区分背景与药丸，在背景区域中的某点之上也需要设置初始值。

watershed()采用分水岭算法将markers中值为零的元素设置为某个初始区域的值。在markers中-1表示区域边界，而1到count则为分割之后的每个区域的编号。图9-27(右)显示了对这些编号涂色之后的结果。图中将markers[1, 1]对应的编号设置为白色，将-1对应的编号设置为黑色，将其余的编号设置为随机颜色。

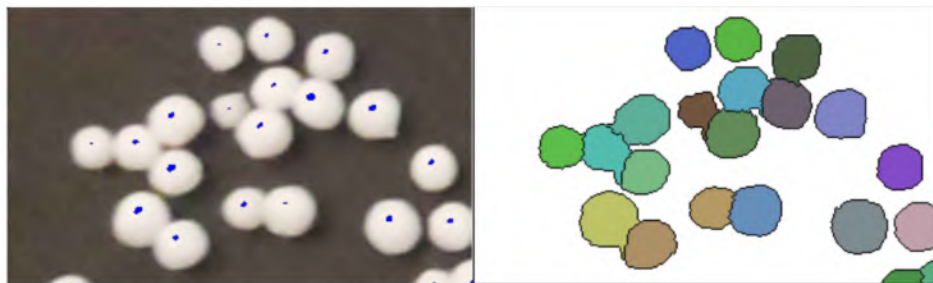


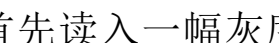
图9-27 使用watershed分割药丸



`scipy2.opencv.watershed_demo`: 分水岭算法的演示程序。用鼠标在图像上绘制初始区域，初始区域将使用“当前标签”填充，按鼠标右键切换到下一个标签。每次绘制初始区域之后，将显示分割结果。

9.4.3 SURF特征匹配

SURF是一种快速提取图像特征点的算法，它所提取的图像特征具有缩放和旋转的不变性，而且它对图像的灰度变化也不敏感。对SURF算法的详细介绍超出了本书的范围，这里只简单地介绍OpenCV中SURF类的用法。

在下面的程序中，❶首先读入一幅灰度图像, ❷然后创建SURF()对象，并设置hessianThreshold和nOctaves参数，这两个参数为计算关键点的参数，hessianThreshold越大则关键点的个数越少，nOctaves越大则关键点的尺寸范围越大。❸调用detect()方法从

灰度图像中检测出关键点。它返回一个列表key_points，其中每个元素都是一个保存关键点信息的KeyPoint对象。

④根据关键点的size属性按照从大到小的顺序排列关键点。⑤调用drawKeypoints()可以在图像上绘制关键点，这里为了使用红色绘制关键点，将灰度图像通过cvtColor()转换成灰色的RGB格式的图像，结果如图9-28(左)所示。图中红色圆圈的大小显示关键点的size属性，而半径线段的方向显示了关键点的方向属性angle。

图9-28(右)显示了前25个最大的关键点，每个关键点对应的小图块已经根据angle属性旋转过。由此可以大致观察SURF算法检测出的关键点对应的图像模式。

```
img_gray1 = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE) ①
surf = cv2.SURF(2000, 2) ②
key_points1 = surf.detect(img_gray1) ③
key_points1.sort(key=lambda kp:kp.size, reverse=True) ④

img_color1 = cv2.cvtColor(img_gray1, cv2.COLOR_GRAY2RGB)
cv2.drawKeypoints(img_color1, key_points1[:25], img_color1,
color=(255, 0, 0), ⑤

flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```



图9-28 SURF() 找到的关键点和每个关键点的局部图像

通过比对关键点对应的图块，可以在两幅图像中寻找相似的点。但是直接比较图块的像素值并不明智，SURF算法为每个关键点计算128个特征，这些特征与关键点的大小和方向无关。下面调用SURF.compute() 计算关键点列表key_points1对应的特征向量features1：

```
_, features1 = surf.compute(img_gray1, key_points1)
features1.shape
(145, 128)
```

还可以调用SURF.detectAndCompute() 直接计算关键点以及与之对应的特征向量：

```
img_gray2 = cv2.imread("lena2.jpg", cv2.IMREAD_GRAYSCALE)
img_color2 = cv2.cvtColor(img_gray2, cv2.COLOR_GRAY2RGB)
surf2 = cv2.SURF(2000, 2)
key_points2, features2 = surf2.detectAndCompute(img_gray2,
None)
```


通过比对features1和features2，可以找到两幅图像中最接近的关键点。在本例中由于关键点数不多，可以使用穷举法计算所有关键点对的距离。如果关键点数很多，可以使用OpenCV提供的FlannBasedMatcher寻找匹配的特征向量。



<http://docs.opencv.org/modules/flann/doc/flann.html>

OpenCV关于FLANN的文档。

在下面的程序中，❶创建FlannBasedMatcher对象时传递两个字典index_params和search_params，分别设置其索引参数和搜索参数。这两个参数在C++语言中分别为IndexParams和SearchParams对象。索引参数的algorithm属性决定搜索算法，候选值如表9-2所示。

表9-2 搜索算法及说明

算法名	编号	说明
FLANN_INDEX_LINEAR	0	穷举法
FLANN_INDEX_KDTREE	1	使用多棵随机Kd树
FLANN_INDEX_KMEANS	2	使用分层k-means树
FLANN_INDEX_COMPOSITE	3	结合使用上述两种算法
FLANN_INDEX_LSH	6	使用multi-probe LSH算

		法
FLANN_INDEX_AUTOTUNED	255	自动选择合适的算法

每种算法都有一些可调节的参数，在本例中使用多棵随机Kd树，通过trees参数设置树的个数为5。在搜索参数中checks参数决定搜索次数，该值越大结果越精确。

❷调用knnMatch()对features1中的每个特征向量在features2中搜索k个最近的特征向量。这里设置参数k为1，只搜索最接近的特征向量。

```
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
search_params = dict(checks=100)

fbm = cv2.FlannBasedMatcher(index_params, search_params) ❶
match_list = fbm.knnMatch(features1, features2, k=1) ❷
```

match_list 是一个嵌套列表，它的长度等于features1.shape[0]，其中每个子列表的长度等于k。子列表中的元素类型为DMatch对象，其distance属性保持两个关键点特征之间的距离，queryIdx属性保存features1中的下标，trainIdx属性保存features2中的下标。由下面的结果可知与features1[0]最近的向量为features2[21]，距离为0.41472：

```
m = match_list[0][0]
    m.distance      m.queryIdx m.trainIdx
-----
```

下面的程序通过列表推导式将关键点的坐标、匹配的下标以及距离都转换成数组，❶然后获取距离最小的50对关键点的坐标，在图9-29中用直线连接这些匹配的关键点。由图9-29可以看出大多数匹配点是正确的，但也有少数匹配错误的关键点。❷为了找到两幅图像之间的变换矩阵，可以使用`findHomography()`，它的前两个参数为原坐标点和变换之后的坐标点，第三个参数选择算法。这里使用RANSAC算法，它可以将匹配错误的关键点自动剔除。`findHomography()`返回变换矩阵和遮罩用的数组`mask`，其中0表示被剔除的点，注意形状为 $(N, 1)$ ，在实际使用时还需要将其转换成一维数组。在图9-29中用蓝色直线显示被剔除的匹配点。

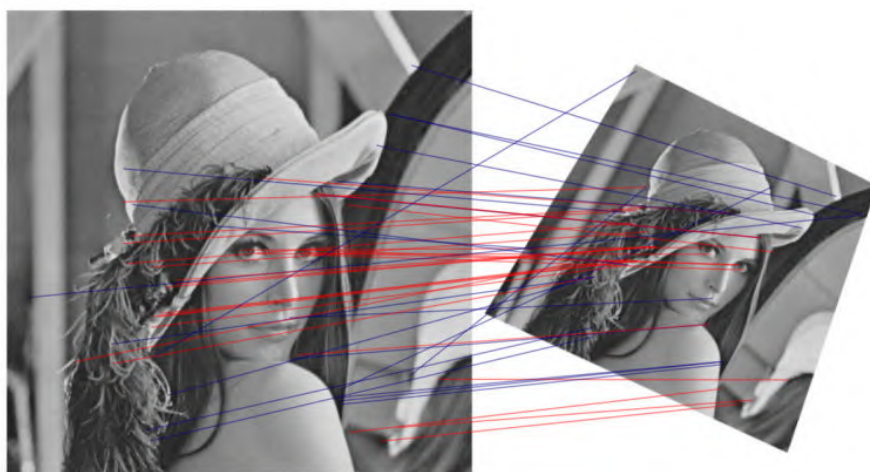


图9-29 显示特征匹配的关键点

? 请读者思考如何利用如下程序得到的`matrix`矩阵将变形之后的图像还原成原始图像。

```
key_positions1 = np.array([kp.pt for kp in key_points1])
key_positions2 = np.array([kp.pt for kp in key_points2])

index1 = np.array([m[0].queryIdx for m in match_list])
index2 = np.array([m[0].trainIdx for m in match_list])

distances = np.array([m[0].distance for m in match_list])

best_index = np.argsort(distances)[:50] ❶
matched_positions1 = key_positions1[index1[best_index]]
matched_positions2 = key_positions2[index2[best_index]]

matrix, mask = cv2.findHomography(matched_positions1,
matched_positions2, cv2.RANSAC) ❷
```



scipy2.opencv.surf_demo: SURF图像匹配演示程序。用鼠标修改右侧图像的4个角的位置计算出透视变换之后的图像，然后在原始图像和变换之后的图像之间搜索匹配点，并计算透视变换的矩阵。

9.5 形状与结构分析



与本节内容对应的Notebook为：09-opencv/opencv-500-shapes.ipynb。

从二值图像提取形状信息有助于对图像进行更高级的处理和识别。本节介绍如何使用`findContours()`搜索图像中的轮廓，并对其进行处理和计算。

9.5.1 轮廓检测

`findContours()`用于在二值图像中寻找黑白区域的边界轮廓，并返回描述轮廓的多边形，它的调用形式如下：

```
findContours(image, mode, method[, contours[, hierarchy[,  
offset]]])
```

`mode`参数为搜索模式，有以下4种选项：

- `RETR_EXTERNAL`：只返回最外层的轮廓。
- `RETR_LIST`：返回所有的轮廓，但是不建立边界的嵌套信息。
- `RETR_CCOMP`：建立一层嵌套信息。
- `RETR_TREE`：建立完整嵌套信息。

`method`参数为轮廓多边形的近似方法。由于轮廓信息是从像素点阵获得，因此不够平滑，可以指定多种近似方法来计算更平滑、点数更少的轮廓多边形。有以下几种选项：

- CHAIN_APPROX_NONE: 获取轮廓上的所有点, 不做任何近似处理
- CHAIN_APPROX_SIMPLE: 对水平线、垂直线以及对角线做近似处理
- CHAIN_APPROX_TC89_L1, CHAIN_APPROX_TC89_KCOS: 对整个轮廓进行近似处理, 从而获得点数更少的多边形。



scipy2.opencv.findcontours_demo: 轮廓检测演示程序。

在下面的例子中, 首先读入一幅灰度图像, 经过高斯模糊、边缘检测和形态学闭运算之后得到如图9-30(左)所示的二值图像。

```
img_coin = cv2.imread("coins.png", cv2.IMREAD_COLOR)
img_coin_gray = cv2.cvtColor(img_coin, cv2.COLOR_BGR2GRAY)
img_coin_blur = cv2.GaussianBlur(img_coin_gray, (0, 0), 1.5,
1.5)
img_coin_binary = cv2.Canny(img_coin_blur.copy(), 60, 60)
img_coin_binary = cv2.morphologyEx(img_coin_binary,
cv2.MORPH_CLOSE,
np.ones((3, 3), "uint8"))
```

由于图像中没有嵌套结构, 在下面的程序中使用RETR_EXTERNAL模式搜索轮廓, 并比较各种轮廓近似选项所得到的多边形的总点数:

```
for approx in ["NONE", "SIMPLE", "TC89_KCOS", "TC89_L1"]:
    approx_flag = getattr(cv2,
```

```

"CHAIN_APPROX_{}".format(approx))

    coin_contours, hierarchy =
cv2.findContours(img_coin_binary.copy(),
                 cv2.RETR_EXTERNAL, approx_flag)

    print "{}: {}".format(approx, sum(contour.shape[0] for
contour in coin_contours)),

    NONE: 3179    SIMPLE: 1579    TC89_KCOS: 849    TC89_L1: 802

```

findContours() 返回两个值：表示轮廓的多边形列表和轮廓的嵌套信息。其中每个多边形都用一个形状为(N, 1, 2)的32位整数数组表示。其中N为多边形的点数，第1轴为多余的轴。

图9-30(左)中右上下角处的噪声也会被检测出轮廓，可以通过下面的公式计算每个轮廓的圆度C，从而删除噪声轮廓，其中p为轮廓的周长，a为轮廓所包围的面积：

$$C = \frac{p^2}{4\pi a}$$

轮廓的周长和面积可以通过arcLength() 和contourArea() 计算，为了计算封闭轮廓的周长，需要设置arcLength() 的第二个参数closed为True。下面的程序找到所有圆度在0.8~1.2之间的轮廓，并调用drawContours() 将这些轮廓绘制在彩色图像img_coin之上。其第三个参数为所绘制轮廓的序号，负数表示绘制所有的轮廓。

```

def circularity(contour):
    perimeter = cv2.arcLength(contour, True)
    area = cv2.contourArea(contour) + 1e-6
    return perimeter * perimeter / (4 * np.pi * area)

```

```

coin_contours = [contour for contour in coin_contours
                  if 0.8 < circularity(contour) < 1.2]
cv2.drawContours(img_coin, coin_contours, -1, (255, 0, 0))

```

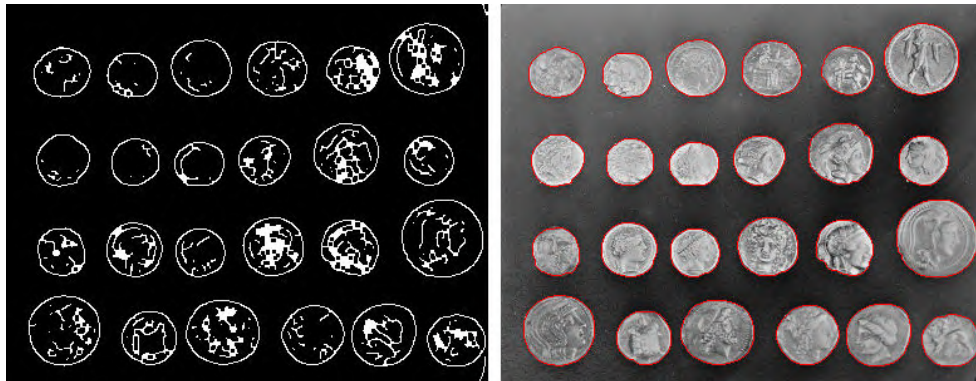


图9-30 显示所有圆度在0.8到1.2之间的轮廓

使用RETR_TREE搜索模式可以获取轮廓的嵌套信息。在下面的程序中，findContours()返回的第二个数组中保存轮廓的嵌套信息，它是一个形状为(1, N, 4)的数组，其第0轴的长度为1，为多余的轴，N为轮廓的个数。hierarchy[0, i, :]中保存与轮廓contours[i]对应的嵌套信息。其最后一个轴的4个数据的含义为：下一个同级别轮廓的下标、上一个同级别轮廓的下标、第一个子轮廓的下标、父轮廓的下标。其中-1表示无效下标。

```

img_pattern = cv2.imread("nested_patterns.png")
img_pattern_gray = cv2.cvtColor(img_pattern,
cv2.COLOR_BGR2GRAY)
_, img_pattern_binary = cv2.threshold(img_pattern_gray, 100,
255, cv2.THRESH_BINARY)
contours, hierarchy =
cv2.findContours(img_pattern_binary.copy(),

```



```
cv2.RETR_TREE, cv2.CHAIN_APPROX_TC89_L1)

hierarchy.shape = -1, 4
```

所有父轮廓下标为-1的轮廓就是图像中最外层的轮廓，下面的程序找到所有最外层轮廓的下标：

```
root_index = [i for i in range(len(hierarchy)) if
hierarchy[i, 3] < 0]

root_index
[0, 7, 19]
```

使用下面的`get_children()`可以获取`hierarchy`中`index`对应的轮廓的所有子轮廓的下标，而`get_descendant()`则可以获得所有嵌套轮廓的层次和下标。下面显示下标为0的轮廓的所有嵌套轮廓的级别和下标。图9-31显示了所有的轮廓，并用颜色映射表显示各个轮廓对应的层次。

```
def get_children(hierarchy, index):
    first_child = hierarchy.item(index, 2)
    if first_child >= 0:
        yield first_child
        brother = hierarchy.item(first_child, 0)
        while brother >= 0:
            yield brother
            brother = hierarchy.item(brother, 0)

def get_descendant(hierarchy, index, level=1):
    for child in get_children(hierarchy, index):
```

```

        yield level, child
        for item in get_descendant(hierarchy, child, level +
1):
            yield item

print list(get_descendant(hierarchy, 0))
[(1, 1), (2, 2), (3, 3), (2, 4), (3, 5), (3, 6)]

```

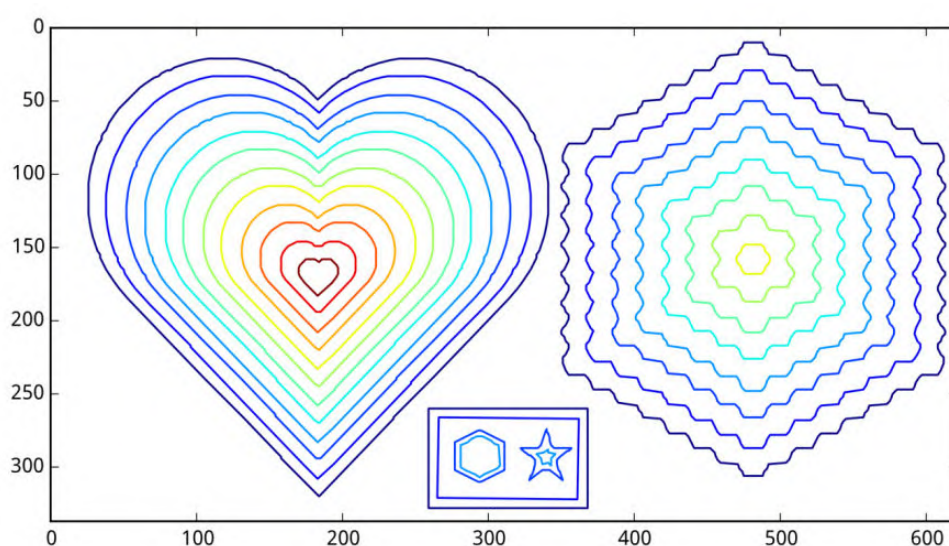


图9-31 显示轮廓的层次结构

9.5.2 轮廓匹配

通过`findContours()`获取轮廓之后，可以使用`approxPolyDP()`对其进行简化，然后通过`matchShapes()`比较两个简化之后的轮廓之间的近似程度。

在下面的例子中，我们要从 patterns.png 图像中找到与 targets.png 中最匹配的轮廓。首先获取轮廓信息，❶并将所有轮廓的坐标最小值都修改为0，这样便于使用matplotlib绘制轮廓。❷然后调用approxPolyDP()对轮廓进行近似处理。它的第二个参数为近似的误差允许范围，该值越大，近似之后的轮廓的点数越少。第三个参数指示轮廓是否为封闭形状。由于patterns.png中的轮廓都是标准图形，而targets.png中的轮廓为手绘图形，这里将手绘图形的近似误差参数设置得更大一些。

```
img_patterns = cv2.imread("patterns.png",
cv2.IMREAD_GRAYSCALE)
patterns, _ = cv2.findContours(img_patterns,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
img_targets = cv2.imread("targets.png",
cv2.IMREAD_GRAYSCALE)
targets, _ = cv2.findContours(img_targets,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

patterns = [pattern - np.min(pattern, 0, keepdims=True) for
pattern in patterns] ❶
targets = [target - np.min(target, 0, keepdims=True) for
target in targets]

patterns_simple = [cv2.approxPolyDP(pattern, 5, True) for
pattern in patterns] ❷
targets_simple = [cv2.approxPolyDP(target, 8, True) for
target in targets]
```

`matchShapes()` 比较两个形状的近似程度。它的第二个参数指定比较算法，有三种算法可选：`CV_CONTOURS_MATCH_I1`、`CV_CONTOURS_MATCH_I2`和`CV_CONTOURS_MATCH_I3`，这些算法都比较轮廓的7个Hu不变量，可以通过`HuMoments()`查看这些不变量的值。其中I3的比较公式选择各个不变量之间的最大误差作为近似程度的评分，在本例中使用该方法能得到最佳匹配效果。具体的公式请读者参照OpenCV的帮助文档。

下面调用 `matchShapes()` 计算 `targets_simple[0]` 和 `patterns_simple`中的所有轮廓之间的近似程度。图9-32显示了每两组轮廓之间的近似评分，并用红色标出最佳匹配。图中用黑色粗线描绘近似之后的轮廓，用填充图形显示原始轮廓。由结果可知，形状的旋转方向和大小不影响轮廓匹配结果。

```
for method in [1, 2, 3]:
    method_str = "CV_CONTOURS_MATCH_I{}".format(method)
    method = getattr(cv, method_str)
    scores = [cv2.matchShapes(targets_simple[0], patterns_simple[pidx], method, 0)
              for pidx in range(5)]
    print method_str, ", ".join("{: 8.4f}".format(score) for score in scores)
```

CV_CONTOURS_MATCH_I1	11.3737,	0.3456,	0.0289,	1.0495,	0.0020
CV_CONTOURS_MATCH_I2	4.8051,	2.2220,	0.0179,	0.3624,	0.0013
CV_CONTOURS_MATCH_I3	0.9164,	0.4778,	0.0225,	0.4552,	0.0016

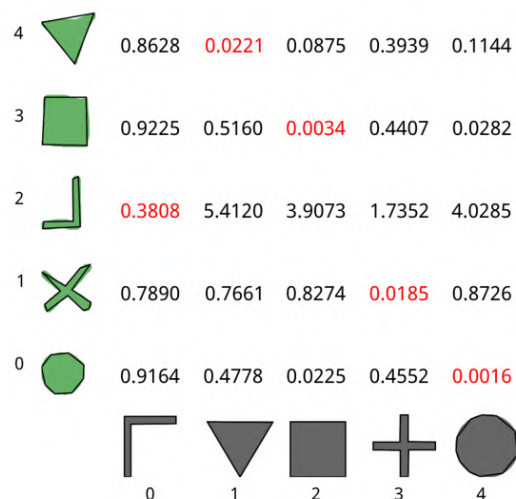


图9-32 使用matchShapes() 比较由approxPolyDP() 近似之后的轮廓

9.6 类型转换



与本节内容对应的Notebook为：09-opencv/opencv-600-type-convert.ipynb。

cv2中的函数所需的参数类型尽量使用数组或Python的标准数据类型，因此在cv2模块中并没有OpenCV的C++ API中的Mat、Point、Size、Vec等各种数据类型，而是用列表、元组或数组表示这些类型。调用cv2模块中的函数比使用OpenCV的C++ API更加便捷，然而需要了解cv2模块的数据类型转换规则，才能把正确的对象传递给函数。作为本章的最后一节，让我们看看在cv2模块内部如何实现Python对象和C++对象之间的相互转换。

9.6.1 分析cv2的源程序

为了了解cv2模块的数据转换工作，需要分析其源程序。将OpenCV的源程序解压之后，可以在opencv\modules\python\src2路径下找到相关的源程序。cv2中各个包装函数是通过cv2.py自动生成的。在命令行中切换到src2目录下，并运行命令“python cv2.py .”，即可在该目录下生成OpenCV的包装函数。



codes\pyopencv_src: 为了方便读者查看cv2模块的源代码，本书提供了自动生成的源代码。若读者遇到参数类型不确定的情况，可以查看这些文件中相应的函数。

所有的包装函数都在自动生成的pyopencv_generated_funcs.h中定义，而这些包装函数会调用cv2.cpp中的众多pyopencv_to()和pyopencv_from()函数以实现Python和OpenCV的各种类型转换工作。若不能确定包装函数使用何种Python数据类型，可以查看包装函数的内容。例如下面是OpenCV文档中关于line()的说明文档：

```
C++: void line(Mat& img, Point pt1, Point pt2, const Scalar& color,
               int thickness=1, int lineType=8, int shift=0)

Python:  cv2.line(img, pt1, pt2, color[, thickness[, lineType[, shift]]]) -> None
```

我们需要知道cv2.line()在调用C++的line()函数时做了哪些类型转换工作。下面是pyopencv_generated_funcs.h中该函数的源代码：

```
static PyObject* pyopencv_line(PyObject* , PyObject* args,
PyObject* kw)
{
    PyObject* pyobj_img = NULL;
    Mat img;
    PyObject* pyobj_pt1 = NULL;
    Point pt1;
    PyObject* pyobj_pt2 = NULL;
    Point pt2;
    PyObject* pyobj_color = NULL;
    Scalar color;
    int thickness=1;
    int lineType=8;
    int shift=0;

    const char* keywords[] = { "img", "pt1", "pt2", "color",
"thickness", "lineType", "shift", NULL };
    if( PyArg_ParseTupleAndKeywords(args, kw,
"OOOO|iii:line",
(char**)keywords, &pyobj_img, &pyobj_pt1, &pyobj_pt2,
&pyobj_color, &thickness, &lineType, &shift) &&
        pyopencv_to(pyobj_img, img, ArgInfo("img", 1)) &&
        pyopencv_to(pyobj_pt1, pt1, ArgInfo("pt1", 0)) &&
        pyopencv_to(pyobj_pt2, pt2, ArgInfo("pt2", 0)) &&
```

```

        pyopencv_to(pyobj_color, color, ArgInfo("color", 0))
    )

    {
        ERRWRAP2( cv::line(img, pt1, pt2, color, thickness,
lineType, shift));
        Py_RETURN_NONE;
    }

    return NULL;
}

```

C++函数`cv::line()`所需的4个参数类型为`Mat`、`Point`、`Point`和`Scalar`，程序中调用4次`pyopencv_to()`将Python的数据转换为这些类型。`pyopencv_to()`有众多重载函数，例如上述类型转换实际上会调用`cv2.cpp`中的如下三个函数：

```

static int pyopencv_to(const PyObject* o, Mat& m, const
ArgInfo info, bool allowND=true);

static inline bool pyopencv_to(PyObject* obj, Point& p,
const char* name = "<unknown>");

static bool pyopencv_to(PyObject *o, Scalar& s, const char
*name = "<unknown>");

```

下面是其中`Point`类型对应的转换函数：

```

static inline bool pyopencv_to(PyObject* obj, Point& p,
const char* name = "<unknown>")
{

```



```

(void)name;
if(!obj || obj == Py_None)
    return true;
if(!PyComplex_CheckExact(obj))
{
    Py_complex c = PyComplex_AsCComplex(obj);
    p.x = saturate_cast<int>(c.real);
    p.y = saturate_cast<int>(c.imag);
    return true;
}
return PyArg_ParseTuple(obj, "ii", &p.x, &p.y) >0;
}

```

分析该程序可知它可将Python的复数和元组转换为Point对象，例如100+200j或(100, 200)。

 请读者使用同样的方法找到与Scalar类型对应的pyopencv_to()函数，并分析它可将何种类型的对象转换成Scalars对象。

在调用C++的函数之后，还需要将其返回值转换为Python的对象。这种转换由pyopencv_from()函数实现。例如minAreaRect()函数找到一个包含所有点的最小矩形。帮助文档中的调用说明如下：

```
C++: RotatedRect minAreaRect(InputArray points)
```

```
Python: cv2.minAreaRect(points) → retval
```

只看该函数说明无法知道`cv2.minAreaRect()`会返回何种对象来表示C++中的`RotatedRect`对象。在`cv2.cpp`中搜索对应的类型转换函数为：

```
static inline PyObject* pyopencv_from(const RotatedRect&
src)
{
    return Py_BuildValue("((ff)(ff)f)", src.center.x,
src.center.y,
                                src.size.width, src.size.height,
src.angle);
}
```

分析该函数可知`minAreaRect()`返回一个有三个元素的元组，而其中第0、第1个元素是有两个元素的元组，因此可以写出如下测试程序，其中`x`和`y`为矩形的中心坐标，`w`和`h`为矩形的宽和高，`angle`为旋转角度：

```
points = np.random.rand(20, 2).astype(np.float32)
(x, y), (w, h), angle = cv2.minAreaRect(points)
```

9.6.2 Mat对象

在C++ API中用Mat对象表示图像，在调用cv2模块中的函数时，会自动将整数、浮点数、元组以及数组转换为Mat对象。为了说明cv2对Mat的自动转换，需要了解Mat对象的内部构造。

Mat对象可以看作一个二维像素阵列，我们可以使用cv模块的CreateMat()函数创建Mat对象。以下程序创建的cvmat是一幅高为200、宽为100、3个通道、像素类型编号为18的图像。step属性为图像中相邻两行像素的字节偏移量。由于 $600 == 100 * 3 * 2$ ，因此cvmat中的图像数据是连续存储的。与NumPy数组的strides不同，step属性只能表示第0轴方向上两个相邻元素之间的偏移量，因此Mat对象无法与某些非连续的数组共享内存。

cvmat = cv.CreateMat(200, 100, cv2.CV_16UC3)				
cvmat.height	cvmat.width	cvmat.channels	cvmat.type	cvmat.step
-----	-----	-----	-----	-----
200	100	3	18	600

在cv2模块中，可以通过其中的全局变量获得表示数值类型和像素类型的常数值。数值类型名由三部分组成：

- 固定部分CV_
- 数值的比特数：8、16、32、64
- 一个描述类型的字母：U表示无符号整数、S表示符号整数、F表示浮点数

像素类型则在数值类型的基础上，添加C1、C2、C3、C4等，分别表示1到4个通道的像素类型。因此CV_16U表示16位的无符号整数，而CV_16UC3表示三个通道的16位无符号整数：

cv2.CV_16U	cv2.CV_16UC3
-----	-----
2	18

将数组转换成Mat对象时，数组的第0轴为图像的纵轴方向，第1轴为图像的横轴方向。若数组有第2轴，则第2轴为图像的通道数；若无第2轴，则图像的通道数为1。数组的dtype类型与Mat的数值类型的对应关系如表9-3所示。

表9-3 dtype类型与Mat数值类型的对应关系

dtype类型	Mat数值类型	说明
uint8	CV_8U	单字节无符号整数
int8	CV_8S	单字节符号整数
uint16	CV_16U	双字节无符号整数
int16	CV_16S	双字节符号整数
int32	CV_32S	4字节符号整数
float32	CV_32F	单精度浮点数
float64	CV_64F	双精度浮点数

调用cv2模块中的函数时，NumPy数组会被转换为Mat对象，它尽可能与原数组共享内存，但如果数组的dtype类型不在表9-3中，或者由于数组的数据存储区的非连续性，在把数组转换为Mat对象时会复制数

据存储区。为了保险起见，建议读者传递给cv2模块的数组都是C语言连续的。

除了数组之外，cv2还能自动将整数和浮点数转换成高为4、宽为1、单通道的双精度浮点数Mat对象，而元组则会被转换为高为元组长度、宽为1、单通道的双精度浮点数Mat对象。例如normalize()函数对Mat对象进行正规化运算，它先将参数转换为Mat对象，把正规化之后的Mat对象转换为数组返回。根据上述规则可以得知下面的程序返回一个形状为(4, 1)的双精度浮点数组：

```
cv2.normalize (1)
array([[ 1.],
       [ 0.],
       [ 0.],
       [ 0.]])
```

9.3.3 在cv和cv2之间转换图像对象

在cv模块中使用cvmat和iplimage对象表示图像。如果需要混用cv2和cv两套API中的函数，就需要在它们与NumPy数组之间进行转换。表9-4列出了这些类型之间的转换方法：

表9-4 类型转换方法

类型转换	方法

array→cvmat	cv.fromarray(array)
cvmat→array	np.asarray(cvmat)
cvmat→iplimage	cv.GetImage(cvmat)
iplimage→cvmat	iplimage[:]或 cv.GetMat(iplimage)

下面通过cv.LoadImage() 和cv.LoadImageM() 分别将图像读入为iplimage和cvmat对象：

```
img = cv2.imread("lena.jpg")
iplimage = cv.LoadImage("lena.jpg")
cvmat = cv.LoadImageM("lena.jpg")
print iplimage
print cvmat
<iplimage(nChannels=3 width=512 height=512 widthStep=1536
)>
<cvmat(type=42424010 8UC3 rows=512 cols=512 step=1536 )>
```

如果需要在NumPy数组和iplimage之间转换，可以通过cvmat作为桥梁，例如：

```
import numpy as np
np.all(img == np.asarray(iplimage[:]))
True
```

由于iplimage类型需要数据保存在连续的内存空间中，因此使用切片获得的数组需要复制之后才能转换为iplimage对象：

```
iplimage2 =  
cv.GetImage(cv.fromarray(img[:, :, :].copy()))
```


第10章 Cython-编译Python程序

Python的动态特性虽然方便了程序的开发，但也会极大地降低运行速度，特别是对于计算密集型的程序。用Python开发这类程序时，通常会调用编译型语言编写的扩展库，例如NumPy、SciPy等，尽量避免直接在Python中进行大量的循环和数值计算。如果这些现成的库无法完成计算要求，就需要用更高效的语言编写核心计算部分，并为之提供Python的调用接口，从而同时实现高效开发和高效运算。

然而如果采用C语言编写扩展库，就会几乎失去Python带来的所有便利：函数的参数需要手工解析、对象的引用计数需要手工维护、大量的Python API需要记忆。所有这些困难使得我们无法把注意力集中到解决实际问题之上。

Cython是为了减轻用C语言编写Python扩展模块的负担而开发出来的一种编程语言。它的语法基本与Python相同，但增加了直接定义和调用C语言函数、定义变量类型等功能。通过Cython的编译器可以将Cython的源程序编译成C语言的源程序，再通过C语言编译器编译成扩展模块。Cython程序既能实现C语言的运算速度，也能使用Python的所有动态特性，极大地方便了扩展库的编写。

10.1 配置编译器



与本节内容对应的Notebook为：`10-cython/cython-100-compiler.ipynb`

为了使用Cython，首先要选择好C语言编译器。在WinPython和Anaconda中已经自带了mingw32编译器，为了让Python使用它作为缺省的编译器，需要编辑Python安装路径下Lib/distutils/distutils.cfg文件，添加如下内容：

```
[build]
compiler = mingw32
```

读者可以使用本书提供的show_compiler()显示默认的C语言编译器。使用set_compiler()可以快速设置distutils.cfg中的compiler选项：

```
from scpy2.utils import show_compiler, set_compiler
set_compiler("mingw32")
show_compiler()
mingw32    defined by C:\WinPython-32bit-2.7.9.2\python-
2.7.9\lib\distutils\distutils.cfg
```

如果未通过distutils.cfg文件指定编译器，distutils会尝试使用编译Python 2.7的编译器Visual C++ 2008，如果操作系统中未安装该编译器，会出现如下错误：

```
DistutilsPlatformError: Unable to find vcvarsall.bat
```

由于现在Visual C++ 2008已经很难入手，因此微软为Python 2.7提供了专门的Visual C++编译器供下载：



<http://www.microsoft.com/en-us/download/confirmation.aspx?id=44266>

微软提供的编译Python 2.7的编译器。

该编译器缺省安装在用户路径之下，为了让Python的distutils模块能正确找到它，需要运行下面的命令来更新setuptools模块：

```
pip install --upgrade setuptools
```

先载入setuptools模块就可以让distutils正确找到Visual C++编译器：

```
import setuptools #先载入setuptools
import distutils
from distutils.msvc9compiler import find_vcvarsall
find_vcvarsall(9.0)

u'C:\\Users\\RY\\AppData\\Local\\Programs\\Common\\Microsoft\\Visual C++ for Python\\9.0\\vcvarsall.bat'
```

然后就可以使用%%cython命令在Notebook中编译Cython程序，测试编译器是否正确设置了：



%%cython魔法命令缺省没有载入到IPython的运算核中，需要先通过%load_ext cython命令载入该命令。

```
%%cython

def add(a, b):
    return a + b
```

此外，使用新版本的Visual C++也可以编译扩展模块，在编译一些较新的函数库(例如笔者用Cython包装Kinect2的API)时可能需要最新版本的Visual C++编译器。但distutils只搜索编译Python时所使用的编译器，因此无法使用最新版本的编译器。

distutils从sys.version获取编译Python的编译器版本，可通过本书提供的set_msvc_version()修改该信息，从而实现选择新版本的编译器：

```
from scipy2.utils import set_msvc_version
set_compiler("msvc")
set_msvc_version(12)
show_compiler()

msvc 12.0 defined by C:\WinPython-32bit-2.7.9.2\python-
2.7.9\lib\distutils\distutils.cfg
```

下面将编译器设置还原成mingw32，接下来的章节中均使用此编译器：

```
set_msvc_version(9)
set_compiler("mingw32")
```

10.2 Cython入门



与本节内容对应的Notebook为：[10-cython/cython-200-intro.ipynb](#)。

当需要在Python中对数组的元素进行大量循环计算时，Python的运行效率会比C语言的效率低上百倍。让我们首先通过一个数组运算函数的实例，演示Cython如何实现数组的高速运算。

10.2.1 计算矢量集的距离矩阵

我们要实现计算一组矢量中每两对矢量的距离的函数。下面的数组X的形状为(200, 3)，可以把它看作200个三维空间中的点：

```
import numpy as np
np.random.seed(42)
X = np.random.rand(200, 3)
```

为了计算每对点之间的距离，可以用NumPy的广播功能，或者使用SciPy中的`scipy.spatial.distance.pdist()`来实现。不过这里为了比较Python和Cython的性能，我们用三层循环逐个元素进行计算：

```

def pairwise_dist_python(X):
    m, n = X.shape
    D = np.empty((m, m), dtype=np.float)
    for i in xrange(m):
        for j in xrange(i, m):
            d = 0.0
            for k in xrange(n):
                tmp = X[i, k] - X[j, k]
                d += tmp * tmp
            D[i, j] = D[j, i] = d ** 0.5
    return D

```

为了验证程序的结果是否正确，下面用SciPy的pdist()进行同样的计算，并比较二者的运算速度：可以看出二者的运算速度相差近200倍。

```

from scipy.spatial.distance import pdist, squareform
%timeit squareform(pdist(X))
%timeit pairwise_dist_python(X)
np.allclose(squareform(pdist(X)), pairwise_dist_python(X))
1000 loops, best of 3: 443 µs per loop
10 loops, best of 3: 92.7 ms per loop
True

```

Cython程序需要编译，通常需要编写一个setup.py程序，稍后会详细介绍这方面的内容。现在我们先用IPython Notebook中的%%cython魔法命令快速编译运行Cython程序。

%%cython魔法命令是一个单元命令，整个单元中的程序都会调用Cython编译成扩展模块，并自动从编译之后的模块中载入所有对象。由于Cython程序是一个单独的模块，因此需要在此模块中重新载入numpy库。下面的程序完全和pairwise_dist_python()相同，其运算速度也和pairwise_dist_python()相当，只有很小的进步：

```
%%cython
import numpy as np

def pairwise_dist_cython(X):
    m, n = X.shape
    D = np.empty((m, m), dtype=np.float)
    for i in xrange(m):
        for j in xrange(i, m):
            d = 0.0
            for k in xrange(n):
                tmp = X[i, k] - X[j, k]
                d += tmp * tmp
            D[i, j] = D[j, i] = d ** 0.5
    return D
```

下面测试其运算速度：

```
%timeit pairwise_dist_cython(X)
np.allclose(pairwise_dist_cython(X),
pairwise_dist_python(X))
10 loops, best of 3: 72.9 ms per loop
True
```

下面的程序用上Cython的所有优化手段，其中cdef为声明变量类型的关键字，@cython为指导Cython编译的命令。可以看出Cython编译出比pdist()更快的代码。

```
%%cython
import numpy as np
import cython
from libc.math cimport sqrt

@cython.boundscheck(False)
@cython.wraparound(False)
def pairwise_dist_cython2(double[:, ::1] X):
    cdef int m, n, i, j, k
    cdef double tmp, d
    m, n = X.shape[0], X.shape[1]
    cdef double[:, ::1] D = np.empty((m, m),
dtype=np.float64)
    for i in range(m):
        for j in range(i, m):
            d = 0.0
            for k in range(n):
                tmp = X[i, k] - X[j, k]
                d += tmp * tmp
            D[i, j] = D[j, i] = sqrt(d)
    return np.asarray(D)
```



```
%timeit pairwise_dist_cython2(X)
np.allclose(pairwise_dist_cython2(X),
pairwise_dist_python(X))
10000 loops, best of 3: 196 µs per loop
True
```

10.2.2 将Cython程序编译成扩展模块

在Notebook中测试了Cython程序的速度与正确性之后，我们希望将其编译成扩展模块，以供其他的Python程序调用。实际上，`%%cython`魔法命令自动将Cython代码编译成扩展模块，并从该模块载入所有的全局对象。下面通过 `sys.modules` 找到 `pairwise_dist_cython2()` 函数所在模块的文件路径。如果读者希望立即使用该扩展模块，可将其复制到自己的Python程序目录下。然后就可以像使用一般模块一样使用它了：

```
import _cython_magic_f9e6211d48d0b874fa7ae6ce345d297b as
fast_pdist
fast_pdist.pairwise_dist_cython2(X)
import sys
sys.modules[pairwise_dist_cython2.__module__]
<module ' _cython_magic_f9e6211d48d0b874fa7ae6ce345d297b'
from
```

```
'C:\Users\RY\Dropbox\scipybook2\settings\.ipython\cython\_cython_magic_f9e6211d48d0b874fa7ae6ce345d297b.pyd'>
```



可以通过`%%cython`命令的`-n`参数指定编译之后的扩展模块名，例如`%%cython -n fast_pdist`。

如果希望使用`distutils`库自动编译扩展模块，可以编写如下`setup_fast_pdist.py`安装文件。在其中定义了一个表示扩展模块的`Extension`对象，名为“`fast_pdist`”，包含一个Cython源程序“`fast_pdist.pyx`”，该文件的内容与前面定义`pairwise_dist_cython2()`的程序相同。由于它使用了NumPy的功能，因此需要通过`numpy.get_include()`指定NumPy的头文件的路径。

```
%%file setup_fast_pdist.py
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext
import numpy as np

ext_modules = [
    Extension("fast_pdist", ["fast_pdist.pyx"],
              include_dirs = [np.get_include()]),
]

setup(
```

```
name = 'a faster version of pdist',  
cmdclass = {'build_ext': build_ext},  
ext_modules = ext_modules  
)
```

在命令行中运行如下命令，即可对Cython文件进行编译，生成扩展模块fast_pdist.pyd：

```
python setup_fast_pdist.py build_ext --inplace
```

接下来即可载入该扩展模块，并调用其中的函数：

```
import fast_pdist  
  
    np.allclose(fast_pdist.pairwise_dist_cython2(X),  
pairwise_dist_python(X))  
  
True
```

除了使用setup.py之外，还可以使用cythonize命令。读者可以在PATH环境变量的搜索路径(例如Python的Scripts文件夹)下创建一个名为cythonize.bat的批处理文件，并添加如下内容：

```
@echo off  
  
python -m Cython.Build.Cythonize %*
```

然后就可以在命令行下输入cythonize -i fast_pdist.pyx，将指定的Cython文件编译成扩展模块，并保存在相同的路径之下。

10.2.3 C语言中的Python对象类型

使用Cython语言编写的程序可通过Cython编译器编译成C语言程序，然后再通过C语言编译器编译为扩展模块供Python调用。在Cython语言中可以对Python的对象类型和C语言类型进行处理。为了让读者更深入地理解Cython语言，本节先介绍在C语言中是如何表示Python对象的。

Python采用C语言编写，它的所有对象都采用C语言的结构体表示，无论何种对象的结构体，其头两个字段的含义是固定的：

- `ob_refcnt`：该对象的引用次数，当引用次数为0时，该结构体所占据的内存将被释放。
- `ob_type`：指向类型对象的指针。

在Python的C语言程序中定了`PyObject`结构体，它仅拥有上述两个字段。而其他对象类型的结构体则在之后添加新的字段，例如`float`类型对应的结构体如下：

```
typedef struct {  
    Py_ssize_t ob_refcnt;  
    struct _typeobject *ob_type;  
    double ob_fval;  
} PyFloatObject;
```

在32位系统中，Py_ssize_t和指针都使用4个字节表示，而double类型的长度为8个字节，所以Python中的一个float对象占据16个字节。下面通过sys.getsizeof()获得浮点数对象1.0的字节数：

```
import sys

sys.getsizeof(1.0)

16
```

由于所有Python对象的头两个字段类型是相同的，因此在C语言中使用PyObject *类型的指针表示Python对象。在Cython中，所有没有声明类型的变量都表示Python对象，因此编译成C语言之后，这些变量都是PyObject *类型的指针。Python提供了Python/C API以方便用户使用C语言编写扩展模块。

下面我们为%%cython魔法命令添加-a参数，显示Cython程序及其编译之后的C语言程序之间的关系。在Notebook中运行下面的程序将输出一段如图10-1所示的可折叠的代码。通过颜色显示每行Cython代码编译之后的C语言代码的多少。颜色越深表示该行代码的运行速度可能会越慢。请注意由于这些C语言代码都是Cython自动生成的，因此阅读起来可能有些困难，读者无须完全理解这些代码，只需要大致了解工作原理即可。

```

+1: a = 1.0
+2: b = 2.0
+3: c = a + b
    __pyx_t_1 = __Pyx_GetModuleGlobalName(__pyx_n_s_a); if (u
[0]; __pyx_lineno = 3; __pyx_clineno = __LINE__; goto __pyx
    __Pyx_GOTREF(__pyx_t_1);
    __pyx_t_2 = __Pyx_GetModuleGlobalName(__pyx_n_s_b); if (u
[0]; __pyx_lineno = 3; __pyx_clineno = __LINE__; goto __pyx
    __Pyx_GOTREF(__pyx_t_2);
    __pyx_t_3 = PyNumber_Add(__pyx_t_1, __pyx_t_2); if (unlik
__pyx_lineno = 3; __pyx_clineno = __LINE__; goto __pyx_L1_
    __Pyx_GOTREF(__pyx_t_3);
    __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
    __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
    if (PyDict_SetItem(__pyx_d, __pyx_n_s_c, __pyx_t_3) < 0)
; __pyx_clineno = __LINE__; goto __pyx_L1_error;}
    __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;

```

图10-1 使用“-a”参数查看编译之后的C语言程序

```

%%cython -a

a = 1.0

b = 2.0

c = a + b

```

在上面的程序中，变量a、b、c均为Python对象，因此c = a + b被编译成如图10-1所示的C语言代码。其中__Pyx_GOTREF和__Pyx_DECREF是为了保证垃圾回收机制能正常运行，而对对象的引用计数器进行操作。

__pyx_n_s_a 为 字 符 串 对 象 “a” 。

__Pyx_GetModuleGlobalName(__pyx_n_s_a)从全局字典中获取“a”表示的Python对象。PyNumber_Add()为Python/C API函数，它将两个Python对象当作数值相加，并返回一个新的Python对象__pyx_t_3。最后调用PyDict_SetItem()将__pyx_t_3添加进全局字典__pyx_d中，对应的键为__pyx_n_s_c所表示的字符串对象“c”。

因此一个简单的加法运算需要两次查询全局字典，获取变量a和b所表示的对象，然后调用一次API函数来完成加法运算，最后再将结果写入全局字典。

10.2.4 使用cdef关键字声明变量类型

为了大幅度地提高程序的运行速度，需要对Python程序中频繁用到的变量使用cdef关键字声明变量类型。用cdef关键字声明变量类型起到如下两个作用：

- 提高变量的存取速度：在Python中，全局变量和对象的属性与它们对应的值之间的关系保存在字典中，每次读写这些变量或属性都相当于对字典进行存取。而如果使用cdef关键字声明变量，那么该变量与其值的对应关系在编译期已经确定，可以省下字典查询所需的时间。
- 提高数值的处理速度：在Python中所有的对象都是C语言中的结构体，对其进行运算时需要调用Python提供的API函数。而当知道值的类型时，Cython会尽可能地使用C语言提供的运算功能，从而极大地提高计算速度。

在下面的例子中，使用cdef定义a、b、c三个变量为double类型。所生成的C语言代码大致如下：

```
static double a, b, c;  
...
```

```
PyMODINIT_FUNC init(){
    ...
    a = 1.0;
    b = 2.0;
    c = a + b;
    ...
}
```

该段代码创建了三个类型为double的全局变量，并在模块初始化函数中为这三个全局变量赋值。加法运算和变量赋值均为C语言中的操作，与上节中调用API函数的代码相比要简洁许多。



请注意这里使用cdef定义的三个全局变量为C语言的全局变量，并不能在Python中通过编译之后的扩展模块获取它们的值。

```
%%cython -a
cdef double a = 1.0
cdef double b = 2.0
cdef double c = a + b
```

当C语言的变量与Python对象进行运算时，会将C语言的变量转换成Python对象之后再调用API函数进行运算。在下面的例子中，s为C语言类型的变量，而a为Python的float对象。

❶s和a直接相加时，会大致运行如下C语言代码，其中_v_s为double类型变量，_t_1到_t_3为Python对象指针，_s_a为字符串对

象"a":

```
_t_1 = PyFloat_FromDouble(_v_s); //将s转换成Python对象
_t_2 = __Pyx_GetModuleGlobalName(_s_a); //获得全局字典中"a"对应的对象
_t_3 = PyNumber_Add(_t_1, _t_2); //加法运算
_v_s = __pyx_PyFloat_AsDouble(_t_3); //将加法运算的结果转换为double类型
```

❷使用<double>将Python对象强制转换为C语言的double类型，因此这里的加法运算采用C语言的加法操作符，大致相当于运行如下C语言代码，其中_t_1为Python对象指针，_t_2为double类型：

```
_t_1 = __Pyx_GetModuleGlobalName(_s_a);
_t_2 = __pyx_PyFloat_AsDouble(_t_1);
_v_s = _v_s + _t_2;
```

```
%%cython -a

cdef double s = 0
a = 3.0
s = s + a ❶
s = s + <double>a ❷
```

除了C语言的各种数据类型之外，Cython还能识别Python的许多内置的数据类型，例如列表、字典、元组等。在下面的例子中，使用list声明clist变量是一个列表对象，并且使用一个int类型的cindex变量作为下标获取列表中的值。为了进行比较，也使用两个动态变量pylist和pyindex进行同样的操作。

```

%%cython -a

cdef list clist = [1000, 2, 3]

cdef int cindex = 0

clist[cindex] ❶

pylist = [1000, 2, 3]

pyindex = 0

pylist[pyindex] ❷

```

❶由于声明了clist和cindex的类型，因此clist[cindex]被编译成如下代码，直接调用辅助函数__Pyx_GetItemInt_List()获取列表对象中某个下标对应的对象：

```

__Pyx_GetItemInt_List(clist, __pyx_v_4demo_cindex, int, 1,
__Pyx_PyInt_From_int, 1, 1, 1)

```

❷由于pylist和pyindex变量没有类型声明，因此把它们当作Python的对象进行处理。将pylist[pyindex]编译为如下代码：

```

__pyx_t_1 = __Pyx_GetModuleGlobalName(__pyx_n_s_pylist);
__Pyx_GOTREF(__pyx_t_1);
__pyx_t_2 = __Pyx_GetModuleGlobalName(__pyx_n_s_pyindex);
__Pyx_GOTREF(__pyx_t_2);
__pyx_t_3 = PyObject_GetItem(__pyx_t_1, __pyx_t_2);
__Pyx_GOTREF(__pyx_t_3);
__Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
__Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
__Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;

```

首先调用__Pyx_GetModuleGlobalName()从全局变量字典中获取pylist和pyindex对应的对象，然后调用Python的API函数PyObject_GetItem()获得下标对应的对象。

除了使用[]的下标存取操作之外，Cython还能识别许多内部类型的方法和属性。表10-1列出了目前Cython所能识别的Python数据类型以及相关的运算操作，当遇到表中已知的属性或方法时，Cython将生成高效代码。

表10-1 Cython所能识别的Python数据类型

数据类型	可识别的运算
bytes、str、unicode	join()、in
tuple	in
list	in、insert()、reverse()、append()、extend()
dict	in、get()、has_key()、keys()、values()、iter*()、clear()、copy()
set	in、clear()、add()、pop()
slice	start、stop、step
complex	cval、real、imag

读者可以在前面的程序中添加clist.append(pyindex)来查看append()编译之后的程序。如果遇到未知的属性或方法，将调用

Python对象提供的通用API接口，例如`clist.count(0)`。

10.2.5 使用def定义函数

通常为了提高程序的运算速度，我们在Cython中用`def`关键字定义函数，然后在Python中调用它们。由于需要在Python中调用，`def`所定义函数的参数和返回值均为Python对象。然而在Cython中可以为函数的参数添加类型定义，Cython会对这些参数进行类型检查并自动转换成对应的C语言类型。

下面的`py_square_add()`函数的两个参数均为`double`类型，当在Python中调用时，传递的是两个Python的`float`对象。在`py_square_add()`内部会将这两个`float`对象转换成C语言的`double`类型的变量，计算出`double`类型的结果之后，再将其转换成Python的`float`对象并返回。

对应的C语言代码大致如下，其中`x`、`y`和`_r`均为Python的对象指针类型，而`_v_x`和`_v_y`为C语言的`double`类型变量。通过API函数`PyFloat_AsDouble()`和`PyFloat_FromDouble()`在`float`对象和`double`类型变量之间进行转换。

```
double _v_x = PyFloat_AsDouble(x);
double _v_y = PyFloat_AsDouble(y);
PyObject * _r;
_r = PyFloat_FromDouble((( _v_x * _v_x) + ( _v_y * _v_y)));
```

```
%%cython -a

def py_square_add(double x, double y):
    return x*x + y*y
```

当使用Python的类型(例如list)声明参数时, Cython会进行类型检查, 并将可识别的运算转换成更高效的代码。在下面的例子中, 声明 `alist` 参数为 `list` 类型, 因此 `len(alist)` 将被编译为 `PyList_GET_SIZE(__pyx_v_alist)`, 而 `alist[i]` 则被编译为调用 `__Pyx_GetItemInt_List()` 函数:

```
%%cython -a

def sum_list(list alist):
    cdef double s = 0
    cdef int i = 0
    for i in range(len(alist)):
        s += <double>alist[i]
    return s
```

10.2.6 使用cdef定义C语言函数

`def`函数采用Python的调用接口, 即使是在Cython程序内部调用这些函数, 也需要对参数和返回值进行类型转换。在大量循环中调用这样的函数会有较大的损耗。可以使用`cdef`关键字定义只能在Cython程序内部调用的函数, 调用的开销和C语言的函数相同。

下面的`c_square_add()`使用`cdef`定义，参数和返回值均为`double`类型。其中`a = c_square_add(1.0, 2.0)`被编译为如下代码：

```
__pyx_v_a = __pyx_f_c_square_add(1.0, 2.0);
```



如果不声明`cdef`函数的返回值类型，则其类型为Python对象。

```
%%cython -a

cdef double c_square_add(double x, double y):
    return x*x + y*y

cdef double a = c_square_add(1.0, 2.0)
```

如果希望同一个函数能在Cython中快速调用，并且能在Python中调用，可以使用`cpdef`关键字。它将同时生成一个C语言函数和一个供Python调用的包装函数。在Cython中调用`cpdef`函数时会比`cdef`函数稍微耗时一些，但要比调用`def`函数快得多。

```
%%cython -a

cpdef double cp_square_add(double x, double y):
    return x*x + y*y

cp_square_add(1.0, 2.0)
```

10.3 高效处理数组



与本节内容对应的Notebook为：10-cython/cython-300-memoryview.ipynb。

科学计算程序中存在大量的数组操作，前面已经简要地介绍过如何使用Cython的内存视图(MemoryView)快速访问数组的元素，本节详细介绍用法。

10.3.1 Cython的内存视图

Cython中的内存视图采用如下语法进行声明：

```
cdef 元素类型[维数声明] 变量名
```

其中维数声明的形式如下，使用:代表轴，::1表示对应轴上的元素被连续存储：

- [:]: 一维数组
- [::1]: 一维连续存储的数组
- [:, :]: 二维数组
- [:, ::1]: 二维数组，第1轴的元素被连续存储

内存视图和NumPy数组的结构类似，保存有shape、base、stride等信息，它本身并不拥有数据存储区，而是从其他的Python对象或C语言数组获取数据存储区。

在Cython中内存视图有两种形式，根据使用方式的不同，Cython会在这两种形式之间自动切换。在C语言级别它是一个结构体，当需要将其作为Python对象使用时，Cython会自动将其转换成一个MemoryView对象。下面我们通过一些例子演示内存视图的用法。

在下面的程序中，❶buf是C语言中的一个全局二维数组，❷view是一个内存视图，并将其数据存储区初始化为全局二维数组buf。

❸numpy.asarray()是Python的一个函数，它可以把MemoryView对象转换成NumPy数组，然后就可以调用数组的方法和函数了。❹此外，MemoryView对象也可以直接传递给NumPy的函数，因为在这些函数内部都会调用类似asarray()的函数来将参数转换为数组。

❺当将内存视图返回到Python环境时，Cython也会将其转换成MemoryView对象。因此通过get_view()可获取转换之后的MemoryView对象，方便我们查看其中的内容。

❻在Cython中获取内存视图的shape属性相当于获取内存视图结构体中shape字段的数据，而对内存视图的下标操作，都会被编译成对数据存储区相应地址的直接访问，因此都是十分高效的。

```
%%cython
import numpy as np

cdef double buf[3][4]
```

❶


```

cdef double[:, ::1] view = buf ❷

def fill_value(double value):
    np.asarray(view).fill(value) ❸

def sum_view():
    return np.sum(view) ❹

def get_view():
    return view ❺

def square_view(): ❻
    cdef int i, j
    for i in range(view.shape[0]):
        for j in range(view.shape[1]):
            view[i, j] *= view[i, j]

```

下面通过`get_view()`获取`MemoryView`对象，并查看其各种属性，可以看到它们和`NumPy`数组是完全一样的：

```

view = get_view()
view.shape  view.strides  view.itemsize  view.nbytes
-----
(3, 4)      (32, 8)      8             96

```

下面使用`fill_view()`、`square_view()`、`sum_view()`等函数对C语言中的全局数组进行操作，`MemoryView`对象也支持下标运算：

```
fill_value(3.0)
print sum_view()
square_view()
print sum_view()
view[1, 2] = 10
print sum_view()
36.0
108.0
109.0
```

下面将MemoryView对象转换成NumPy数组，可以看出二者是共享数据存储区的：

```
arr = np.asarray(view)
arr[1, 0] = 11
print sum_view()
print arr
111.0
[[ 9.  9.  9.  9.]
 [11.  9. 10.  9.]
 [ 9.  9.  9.  9.]]
```

通过内存视图的base属性可以获得保存实际数据的对象，由于我们的数据保存在C语言的全局数组中，因此Cython创建了一个array对象来表示该全局数组，而通过其唯一的memview属性可获取一个新的MemoryView对象。

```
print view.base.__class__
print view.base.memview

<type
'_cython_magic_1118ba0f43c15a1b4a16015476c9c6fa.array'>
<MemoryView of 'array' object>
```

下面看看使用内存视图操作NumPy数组：

❶使用`double[:]`声明`x`为一维内存视图，由于没有指定其元素为连续存储，因此`x[i]`被编译为`x.data + i * x.strides[0]`，其中`x.data`为`x`的数据存储区的首地址，类型为单字节指针，`x.strides[0]`为第0轴上元素之间的字节间隔数。

❷由于声明了`res`的元素为连续存储，因此`res[i]`被编译为`((double *) res.data) + i`。因为省略了下标变量`i`与元素间隔字节数的乘法运算，所以运算速度更快。

❸通过`res.base`返回NumPy数组，注意`base`属性不是表示内存视图的C语言结构体中的字段，因此Cython会先将其转换成MemoryView对象，然后通过Python的`getattr()`函数获取其`base`属性。

如果在函数中对数组的元素进行逐个循环，可以将`boundscheck`和`wraparound`两个编译选项关闭。这样所生成的C语言代码不会对数组下标进行越界检查，也不支持负数下标，从而提高数组元素的访问速度。

```
%%cython -a
import numpy as np
import cython
```

```

@cython.boundscheck(False)
@cython.wraparound(False)
def square(double[:] x): ❶
    cdef int i
    cdef double[:,1] res = np.empty_like(x)
    for i in range(x.shape[0]):
        res[i] = x[i] * x[i] ❷
    return res.base ❸

```

此外，内存视图支持与NumPy数组相同的切片下标存取功能，当使用切片存取内存视图时，将创建一个与原内存视图共享内存的新结构体，因此会有一定的运算损耗，但是比数组的切片操作要快许多。

在下面的例子中用cpdef定义了一个可在Cython中快速调用，且能在Python中调用的norm()函数，它对一维向量进行原地归一化。而norm_axis()则可以对二维数组的指定轴进行归一化，如果inplace参数为True，则进行原地归一化，否则返回一个新的归一化之后的数组。

❶内存视图支持切片赋值运算，这时会创建一个临时的内存视图结构体，然后在两个内存视图结构体之间进行数据复制。❷对内存视图使用切片下标读取时将创建一个新的视图，并将此视图传递给norm()函数。



在这个例子中我们通过注释#cython设置boundscheck和wraparound编译选项。

```

%%cython

#cython: boundscheck=False, wraparound=False

import numpy as np
import cython

cpdef norm(double[:] x):
    cdef double s
    cdef int i
    s = 0
    for i in range(x.shape[0]):
        s += x[i]*x[i]
    s = 1 / s**0.5
    for i in range(x.shape[0]):
        x[i] *= s

    def norm_axis(double[:, :] x, int axis=0, bint
inplace=True):
        cdef int i
        cdef double[:, :] data
        if not inplace:
            data = np.empty_like(x)
            data[:] = x ❶
        else:
            data = x

```

```

    if axis == 1:
        for i in range(data.shape[0]):
            norm(data[i, :]) ❷
    elif axis == 0:
        for i in range(data.shape[1]):
            norm(data[:, i])

    return data.base

```

还可以通过地址运算符获得内存视图中数据的地址，将其作为指针传递给C语言的函数。在下面的例子中，❶通过cimport从C语言的标准头文件string.h中载入memcpy()函数，其函数原型如下：它将由src指向地址为起始地址的连续n个字节的数据复制到以dst指向地址为起始地址的空间内。

```
void *memcpy(void *dst, const void *src, size_t n);
```

❷通过地址运算符&获得指向内存视图第一个元素的指针，&dst[0]得到的是一个double *类型的指针。由于memcpy()接收的是字节长度，因此这里使用sizeof(double)计算双精度浮点数的字节数并乘以内存视图第0轴的长度。

请注意这里的内存视图是连续的，因此可以直接使用memcpy()进行内存的复制。如果不是连续的，需要将strides属性一起传递给C语言函数，这样才能在其中正确访问内存视图的元素。

```

%%cython
from libc.string cimport memcpy ❶

```

```
def copy_memview(double[:,1] src, double[:,1] dst):  
    memcpy(&dst[0], &src[0], sizeof(double)*dst.shape[0]) ❷
```

```
a = np.random.rand(10)  
b = np.zeros_like(a)  
copy_memview(a, b)  
assert np.all(a == b)
```

还可以通过类型转换操作将C语言的指针转换成内存视图。例如，如果addr是一个指针，可以通过<double[:10]>addr将其转换为一个长度为10的双精度浮点数的内存视图，长度可以使用变量表示。不过在进行这种转换时，要十分注意内存的分配和释放，否则可能会出现野指针，造成整个程序崩溃。

10.3.2 用降采样提高绘图速度

当使用matplotlib显示一条拥有大量数据点的曲线时，绘图速度会明显降低。由于屏幕的分辨率有限，绘制大量的线段并不能增加图表显示的信息，因此一般在显示大量数据时都会对其进行降采样运算。由于这种运算需要对数组中的每个元素进行迭代，因此需要使用Cython提高运算速度。

下面演示如何使用Cython进行快速降采样运算。首先创建测试数据，在正弦波信号中夹杂着1%的脉冲信号。

```

import numpy as np

def make_noise_sin_wave(period, n):
    np.random.seed(42)

    x = np.random.uniform(0, 2*np.pi*period, n)
    x.sort()
    y = np.sin(x)
    m = int(n*0.01)
    y[np.random.randint(0, n, m)] += np.random.randn(m) *
0.4

    return x, y

x, y = make_noise_sin_wave(10, 10000)

```

无论怎样降低取样频率，我们都希望能够显示这些脉冲信号，因此不能简单地使用每N个点取一个点的方法。图10-2显示了一种能尽量保持所有局域最值的方法。将X轴的范围等分为N个区间，找到每个区间的最小值和最大值的X轴坐标，对这些坐标按照从小到大的顺序排序之后，就得到了降采样之后的X轴坐标。

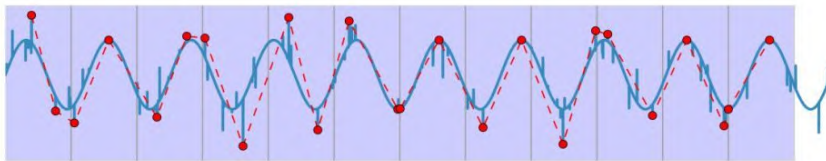


图10-2 降低取样频率示意图

下面首先使用Python实现上述算法，`get_peaks_py()`的`x`和`y`参数分别是曲线上各点的X轴和Y轴坐标，`x`中的值必须是递增且等间隔的。`n`为降低取样频率之后的数据点数，`x0`和`x1`为目标区域。如果为`None`，就对整条曲线进行处理。

为了测试程序是否运行正确，读者可以将由`x`、`y`表示的原始曲线和由`xr`、`yr`表示的降取样曲线绘制在同一个图表中，比较二者的差别。

```
def get_peaks_py(x, y, n, x0=None, x1=None):  
    if x0 is None:  
        x0 = x[0]  
  
    if x1 is None:  
        x1 = x[-1]  
  
    index0, index1 = np.searchsorted(x, [x0, x1])  
  
    index1 = min(index1, len(x) - 1)  
  
    x0, x1 = x[index0], x[index1]  
  
    dx = (x1 - x0) / n  
  
    i = index0  
  
    x_min = x_max = x[i]  
    y_min = y_max = y[i]  
    x_next = x0 + dx  
  
    xr, yr = np.empty(2 * n), np.empty(2 * n)  
    j = 0
```

```

while True:
    xc, yc = x[i], y[i]
    if xc >= x_next or i == index1:
        if x_min > x_max:
            x_min, x_max = x_max, x_min
            y_min, y_max = y_max, y_min
            xr[j], xr[j + 1] = x_min, x_max
            yr[j], yr[j + 1] = y_min, y_max
            j += 2

            x_min = x_max = xc
            y_min = y_max = yc
            x_next += dx
            if i == index1:
                break
        else:
            if y_min > yc:
                x_min, y_min = xc, yc
            elif y_max < yc:
                x_max, y_max = xc, yc
            i += 1

    return xr[:j], yr[:j]

```

```

xr, yr = get_peaks_py(x, y, 200)

```

下面对上述代码添加Cython的类型声明，就可以将其编译成能快速执行的C语言程序。

❶为了获得最佳运行速度，将wraparound和boundscheck设置为False，❷由于禁止了wraparound，因此不能使用负数作为下标，所以这里将x[-1]改为x[len(x)-1]。

❸当将内存视图传递给Python函数时，它将被转换为MemoryView对象，而在searchsorted()中会通过asarray()将其转换成NumPy数组。当然此处也可以使用x.base直接将数组对象传递给searchsorted()。

❹为了让函数返回数组，需要通过base属性获得内存视图对应的NumPy数组，然后对该数组进行切片运算。

```
%%cython
import numpy as np
import cython

@cython.wraparound(False) ❶
@cython.boundscheck(False)
def get_peaks(double[:,1] x, double[:,1] y, int n, x0=None,
x1=None):
    cdef int i, j, index0, index1
    cdef double x_min, x_max, y_min, y_max, xc, yc, x_next,
dx
    cdef double[:,1] xr, yr
```

```

if x0 is None:
    x0 = x[0]
if x1 is None:
    x1 = x[len(x) - 1] ❷

index0, index1 = np.searchsorted(x, [x0, x1]) ❸
index1 = min(index1, len(x) - 1)
x0, x1 = x[index0], x[index1]
dx = (x1 - x0) / n

i = index0
x_min = x_max = x[i]
y_min = y_max = y[i]
x_next = x0 + dx

xr = np.empty(2 * n)
yr = np.empty(2 * n)
j = 0

while True:
    xc, yc = x[i], y[i]
    if xc >= x_next or i == index1:
        if x_min > x_max:
            x_min, x_max = x_max, x_min
            y_min, y_max = y_max, y_min
        xr[j], xr[j + 1] = x_min, x_max
        yr[j], yr[j + 1] = y_min, y_max

```

```

        j += 2

        x_min = x_max = xc
        y_min = y_max = yc
        x_next += dx
        if i == index1:
            break
    else:
        if y_min > yc:
            x_min, y_min = xc, yc
        elif y_max < yc:
            x_max, y_max = xc, yc
    i += 1

    return xr.base[:j], yr.base[:j] ④

```

下面比较`get_peaks_py()`和`get_peaks()`的运算结果和运行速度。可以看到结果相同，而`get_peaks()`的运行速度提高了100多倍。

```

xr, yr = get_peaks_py(x, y, 200)
xr2, yr2 = get_peaks(x, y, 200)
print np.allclose(xr, xr2), np.allclose(yr, yr2)

%timeit get_peaks_py(x, y, 200)
%timeit get_peaks(x, y, 200)

True True

```

```
100 loops, best of 3: 9.55 ms per loop
10000 loops, best of 3: 78.7 µs per loop
```



`scipy2.cython.fast_curve_draw`演示了使用降采样提高matplotlib的曲线绘制速度。降采样函数为`scipy2.cython.get_peaks()`。

`scipy2.cython.fast_curve_draw`使用matplotlib演示`get_peaks()`的效果，在子图对象的`xlim_changed`事件中对曲线位于当前X轴的显示范围内的部分进行降取样，并更新曲线对象的数据，即可提高曲线的绘制速度。读者可以用鼠标右键拖动缩放显示范围时可以看到响应速度提高了很多。

10.4 使用Python标准对象和API



与本节内容对应的Notebook为：`10-cython/cython-400-python-api.ipynb`。

在10.2节中已经介绍过Cython可以识别Python的许多内置类型的常用操作，提高运算速度。此外还可以在Cython中调用Python/C API函数，实现一些只有在C语言中才能完成的操作。

10.4.1 操作list对象

在Python中可先创建一个空列表，然后通过`append()`方法往其中添加元素，或者通过`[None]*n`创建一个拥有n个元素的列表，然后通过下标设置列表的内容。在Cython中可以通过调用操作列表的API函数，提高创建列表的速度。

下面的代码对比使用API函数和`append()`方法创建列表的速度。在`my_range()`中调用如下三个API函数：

- `object PyList_New(Py_ssize_t len)`：创建指定大小的列表，列表的元素设置为NULL。注意这里的NULL是C语言的空地址，而不是Python的None对象。也就是说，这个列表虽然已经被创建了，但是其中的内容还没有初始化，无法在Python中使用。
- `void PyList_SET_ITEM(object list, Py_ssize_t i, object o)`：设置列表list的指定下标i的内容为o。它实际上是C语言中的一个宏，可以用于快速设置元素为NULL的列表中的元素。
- `void Py_INCREF(object o)`：将对象o的引用计数器加1。当调用`PyList_SET_ITEM()`将对象o添加进列表时，该列表就应该增加对象o的引用计数器，然而`PyList_SET_ITEM()`并不会自动增加被添加的对象的引用计数器，因此需要调用`Py_INCREF()`。这种不增加引用计数的函数在Python API说明文档中会被注明为：“steals a reference”。

```

%%cython

#cython: boundscheck=False, wraparound=False

from cpython.list cimport PyList_New, PyList_SET_ITEM ❶
from cpython.ref cimport Py_INCREF

def my_range(int n):
    cdef int i
    cdef object obj ❷
    cdef list result
    result = PyList_New(n)
    for i in range(n):
        obj = i
        PyList_SET_ITEM(result, i, obj)
        Py_INCREF(obj)
    return result

def my_range2(int n):
    cdef int i
    cdef list result
    result = []
    for i in range(n):
        result.append(i)
    return result

```

❶ 使用 `cimport` 从 Cython 头文件中载入 `PyList_New` 和 `PyList_SET_ITEM` 这两个函数声明。Cython 的头文件以 `.pxd` 为扩展名，和 C 语言的头文件类似，可以包含各种函数和类型的定义。

可以在Cython的安装目录之下的Includes目录下找到所有的Cython头文件。其中numpy目录下包含NumPy相关的类型和API函数的声明，libc目录下包含C语言标准库中的各个函数的声明，cpython目录下包括Python/C API的函数声明。

❷obj变量用于临时保存将C语言整数变量i转换成的Python整数对象。

由于my_range()直接创建目标大小的列表，省去了逐步扩容带来的损耗，因此速度是使用append()的两倍多。

```
%timeit range(100)
%timeit my_range(100)
%timeit my_range2(100)

1000000 loops, best of 3: 1.24 µs per loop
1000000 loops, best of 3: 1.04 µs per loop
100000 loops, best of 3: 2.29 µs per loop
```

10.4.2 创建tuple对象

在Python中tuple对象是不可变的，但是可以在Cython中调用API函数，在创建tuple对象时设置其内容，从而实现快速创建tuple对象。下面的to_tuple_list()通过调用API函数将二维数组转换成元组列表。

和列表相同，元组也有对应的初始化函数：PyTuple_New和PyTuple_SET_ITEM。它们的用法和列表的相同，这里不再重复了。

```

%%cython

#cython: boundscheck=False, wraparound=False

from cpython.list cimport PyList_New, PyList_SET_ITEM
from cpython.tuple cimport PyTuple_New, PyTuple_SET_ITEM
from cpython.ref cimport Py_INCREF

def to_tuple_list(double[:, :] arr):
    cdef int m, n
    cdef int i, j
    cdef list result
    cdef tuple t
    cdef object obj

    m, n = arr.shape[0], arr.shape[1]
    result = PyList_New(m)
    for i in range(m):
        t = PyTuple_New(n)
        for j in range(n):
            obj = arr[i, j]
            PyTuple_SET_ITEM(t, j, obj)
            Py_INCREF(obj)
        PyList_SET_ITEM(result, i, t)
        Py_INCREF(t)
    return result

```

下面比较NumPy数组的tolist()方法和to_tuple_list()的速度:

```

import numpy as np

arr = np.random.randint(0, 10, (5, 2)).astype(np.double)
print to_tuple_list(arr)

arr = np.random.rand(100, 5)
%timeit to_tuple_list(arr)
%timeit arr.tolist()

[(0.0, 4.0), (5.0, 7.0), (7.0, 0.0), (5.0, 5.0), (5.0,
9.0)]

100000 loops, best of 3: 13 µs per loop
10000 loops, best of 3: 20.5 µs per loop

```

10.4.3 用array.array作为动态数组

在NumPy一章中我们曾介绍过，可以将Python的标准模块array中的array对象当作一维动态数组使用。在Cython代码中，也同样可以用array.array对象作为动态数组。然而如果在Cython代码中调用array.append()添加元素，则需要调用Python的函数对象，起不到提速的作用。在Cython的cpython/array.pxd头文件中提供了快速操作array对象的C函数。

下面的in_circle()收集二维坐标数组points中所有位于由cx、cy和r表示的圆形内部的坐标。其中(cx, cy)为圆心坐标，r为圆的半径。

由于事先无法知道有多少点位于圆形内部，因此程序中使用 `array.array` 动态数组逐个添加满足条件的点。

```
%%cython -c-Ofast

#cython: boundscheck=False, wraparound=False

import numpy as np

from cpython cimport array

def in_circle(double[:, :] points, double cx, double cy,
double r):

    cdef array.array[double] res = array.array("d") ❶
    cdef double r2 = r * r
    cdef double p[2] ❷
    cdef int i
    for i in range(points.shape[0]):
        p[0] = points[i, 0]
        p[1] = points[i, 1]
        if (p[0] - cx)**2 + (p[1] - cy)**2 < r2:
            array.extend_buffer(res, <char*>p, 2) ❸
    return np.frombuffer(res, np.double).copy().reshape(-1,
2) ❹
```

❶用 `cimport` 关键字载入 `array` 的头文件之后，使用其中的 `array.array` 类型定义一个 Cython 变量 `res`，并创建一个新的 `array.array` 对象给它，“d”表示元素类型为双精度浮点数。❷`p`是有两个元素的C语言数组，我们用它临时保存当前处理的点的坐标。❸调用 `array.extend_buffer()` 将 `p` 添加进 `res` 中。`extend_buffer()` 的第一个

参数是array.array对象，第二个参数是一个char*指针，它指向待添加数据的首地址，第三个参数是待添加元素的个数(注意不是字节数)。④最后，通过numpy.frombuffer()创建与res共享内存的NumPy数组，我们复制该数组以便Python垃圾回收res对象。

下面比较in_circle()和使用NumPy相关方法的运算速度，当大多数点都位于圆形之外时，in_circle()的运算速度将更快一些。



本例的目的是为了演示array.array动态扩容，实际上使用布尔数组有可能得到更快的运算速度。

```
points = np.random.rand(10000, 2)
cx, cy, r = 0.3, 0.5, 0.05

%timeit points[(points[:, 0] - cx)**2 + (points[:, 1] - cy)**2 < r**2, :]
%timeit in_circle(points, cx, cy, r)
10000 loops, best of 3: 97.7 µs per loop
10000 loops, best of 3: 38.6 µs per loop
```

10.5 扩展类型



与本节内容对应的Notebook为：10-cython/cython-500-cdef-class.ipynb。

在Python中通过class定义的类采用字典保存实例的属性，然而为了提高属性的访问速度，Python内置的类型则直接将其属性保存在对象结构体的字段中。在Cython中则可以通过cdef class定义扩展类型。扩展类型和Python的内置类型一样，采用C语言的结构体保存对象的各个属性，因此在Cython程序中能够快速存取这些属性。扩展类型很适合用于包装C语言的函数库，提供面向对象的Python调用接口。

10.5.1 扩展类型的基本结构

下面的程序使用cdef class定义扩展类型Point2D，并使用cdef定义属性x和y。注意和Python的类不同，扩展类型的属性在类中定义，而不是在__init__()方法中生成。

```
%%cython

cdef class Point2D:
    cdef public double x, y
```

Cython将自动定义如下结构体来表示Point2D对象，由于ob_refcnt和ob_type两个字段是所有Python对象必须具备的，因此在Python的C语言代码中它们通常使用PyObject_HEAD宏定义。

```

struct __pyx_obj_Point2D {
    PyObject_HEAD
    double x;
    double y;
};

```

在Cython程序内部，当它明确知道对象p的类型为Point2D时，p.x将被转换成直接访问Point2D结构体的x字段，因此对扩展类型变量的属性存取是非常迅捷的。为了在Python中访问Point2D的x和y属性，需要在声明属性时使用public关键字。Cython能自动为整型、浮点型、字符串类型以及Python对象类型这4种cdef属性创建属性访问用的描述器。这些描述器包含__get__()和__set__()方法，用以获取和设置属性。对于只读属性，可以将public关键字替换为readonly。下面的代码查看x对应的属性访问描述器：

```

print type(Point2D.x)
print Point2D.x.__get__
print Point2D.x.__set__
<type 'getset_descriptor'>
    <method-wrapper '__get__' of getset_descriptor object at
0x097A3260>
    <method-wrapper '__set__' of getset_descriptor object at
0x097A3260>

```

和定义函数相同，扩展类型中可以使用def、cdef和cpdef定义对象的方法。所有方法都可以在Cython中调用，而只有def和cpdef定义的方法可以在Python中调用。在Cython中调用cdef和cpdef方法时，直接调用对应的C语言函数，因此效率比def方法要高很多。

扩展类型支持从其他扩展类型继承，例如下面的Point3D从Point2D继承，并增加了字段z：

```
%%cython -a

cdef class Point2D:
    cdef public double x, y

cdef class Point3D(Point2D):
    cdef public double z

cdef Point3D p = Point3D()
p.x = 1.0
p.y = 2.0
p.z = 3.0
```

Point3D对象对应的C语言结构体如下：

```
struct __pyx_obj_Point3D {
    struct __pyx_obj_Point2D __pyx_base;
    double z;
};
```

它的第一个字段__pyx_base是其基类Point2D对应的结构体。在Cython中通过__pyx_base.x访问基类中定义的属性x，而在Python中，则通过Point3D.__base__中x对应的描述器访问该属性。

10.5.2 一维浮点数向量类型

下面我们以一维浮点数向量Vector类型为例介绍扩展类型的用法。Vector对象拥有两个私有属性：count表示数组的长度，data为存储数组数据的首地址。

```
cdef class Vector:
    cdef int count
    cdef double * data
```

在创建一个新的Vector对象时，会从堆内存中分配一个结构体。在结构体的内存分配完毕之后，应该立即初始化其中的属性。这个初始化工作由__cinit__()完成，可以将之理解为C语言级别的__init__()。Vector对象支持两种初始化方式：分配指定大小的内存，或者使用一个序列对象初始化数组的内容。这里使用Python API中的PyMem_Malloc()从Python管理的堆中分配存储数组数据的内存。为了使用该API函数，需要使用from cpython cimport mem载入声明内存管理API函数的头文件。

```
def __cinit__(self, data):
    cdef int i
    if isinstance(data, int):
        self.count = data
    else:
        self.count = len(data)
    self.data = <double
*>mem.PyMem_Malloc(sizeof(double)*self.count)
```

```
if self.data is NULL:
    raise MemoryError

if not isinstance(data, int):
    for i in range(self.count):
        self.data[i] = data[i]
```

当对象的引用计数变为0时，将被垃圾回收。在对象结构体被回收之前，Cython会调用`__dealloc__()`，在这里需要释放`data`属性指向的内存：

```
def __dealloc__(self):
    if self.data is not NULL:
        mem.PyMem_Free(self.data)
```

为了让Vector对象支持整数下标存取和循环迭代，需要定义`__len__()`、`__getitem__()`和`__setitem__()`等方法。这种以两个下划线开头和结尾的方法被称为魔法方法，通过定义这些方法可以改变对象在某种特定语法中的行为。例如查看对象长度的内置函数`len(obj)`实际上返回`obj.__len__()`的值，而`obj[index]`会调用`obj.__getitem__(index)`，`obj[index] = value`则会调用`obj.__setitem__(index, value)`。如果某个类型定义了`__len__()`和`__getitem__()`，则其对象会自动支持for循环以迭代其中的元素。

在下面的程序中，`__getitem__()`和`__setitem__()`要求其下标参数为整型，因此Vector对象不支持切片下标。我们将支持负数下标和下标越界检查单独放在`_check_index()`方法中，该方法的参数是一个

整型指针，可以直接修改传进来的下标变量。如果下标越界，则抛出 `IndexError` 异常。在 Cython 中使用 `p[0]` 访问指针变量 `p` 指向的地址。

```
def __len__(self):
    return self.count

cdef _check_index(self, int *index):
    if index[0] < 0:
        index[0] = self.count + index[0]
    if index[0] < 0 or index[0] > self.count - 1:
        raise IndexError("Vector index out of range")

def __getitem__(self, int index):
    self._check_index(&index)
    return self.data[index]

def __setitem__(self, int index, double value):
    self._check_index(&index)
    self.data[index] = value
```

为了让 `Vector` 对象支持加法运算符，需要定义 `__add__()` 方法：

```
def __add__(self, other):
    cdef Vector new, _self, _other

    if not isinstance(self, Vector): ❶
        self, other = other, self
    _self = <Vector>self ❸
```

```

        if isinstance(other, Vector): ❷
            _other = <Vector>other
            if _self.count != _other.count:
                raise ValueError("Vector size not equal")
            new = Vector(_self.count) ❹
            add_array(_self.data, _other.data, new.data,
                _self.count)

            return new

        new = Vector(_self.count)
        add_number(_self.data, <double>other, new.data,
            _self.count) ❹

        return new

```

对于__add__()、__mul__()这样的二元运算的魔法方法，第一个参数self可能不是当前的对象。当第一个参数对象无法完成运算时，将调用第二个参数对象的魔法方法，而参数的顺序不变。例如，如果调用1 + v，其中v是一个Vector对象，那么整数1的__add__()函数调用将失败，因此调用Vector.__add__()时参数self为1，参数other为v。

程序中，❶首先判断self的类型是否为Vector，如果不是，就交换self和other所表示的对象。❷由于__add__()能处理数值和Vector对象，因此需要根据other对象的类型进行不同的处理。❸由于self和other变量没有类型声明，因此无法通过它们获取保存于C语言结构体中的属性。通过<Vector>将Python对象转换成拥有类型的变量_self和_other，通过这两个变量可以访问count和data属性。❹创建保存运算

结果的Vector对象，并调用add_array()或add_number()进行计算，稍后会介绍这两个函数的代码。

下面是+=操作符对应的魔法方法__iadd__():

```
def __iadd__(self, other):
    cdef Vector _other
    if isinstance(other, Vector):
        _other = <Vector>other
        if self.count != _other.count:
            raise ValueError("Vector size not equal")
        add_array(self.data, _other.data, self.data,
self.count)
    else:
        add_number(self.data, <double>other, self.data,
self.count)
    return self
```

和二元操作符不同，__iadd__()的第一个参数就是当前对象，因此Cython知道它的类型，无须再进行类型转换。

下面用cpdef定义norm()方法以计算矢量的长度，并在__str__()中调用它。当将对象转换成字符串时，将调用__str__()方法。cpdef定义的方法会同时生成Cython和Python调用接口，在__str__()中通过Cython的调用接口运行norm()，而在Python中则通过较慢的接口调用norm()。

```

def __str__(self):
    values = ", ".join(str(self.data[i]) for i in
range(self.count))

    norm = self.norm()

    return "Vector[{}]({})".format(norm, values)

cpdef norm(self):
    cdef double *p
    cdef double s
    cdef int i

    s = 0

    p = self.data

    for i in range(self.count):
        s += p[i] * p[i]

    return s**0.5

```

此外，扩展类型可以在Python中继承，覆盖基类中定义的def和cpdef方法，Python中定义的覆盖方法可以在Cython中正确调用。由于在Cython中调用cpdef方法时，需要检查此方法是否被覆盖，因此其调用速度要略比cdef慢。

最后在代码的头部添加进行计算的add_array()和add_number()两个函数，它们采用cdef定义，只能在Cython代码内部调用。

```

cdef add_array(double *op1, double *op2, double *res, int
count):
    cdef int i
    for i in range(count):

```

```

        res[i] = op1[i] + op2[i]

    cdef add_number(double *op1, double op2, double *res, int
count):
        cdef int i
        for i in range(count):
            res[i] = op1[i] + op2

```

读者可以试着完成其他二元计算函数以及元素存取函数。下面演示Vector对象的使用法：

```

from scipy2.cython.vector import Vector
v1 = Vector(range(5))
v2 = Vector(range(100, 105))
print len(v1)
print v1 + v2
print v1 + 2
print 20 + v2
print v1.norm(), v2.norm()
print [x**2 for x in v1]
5
Vector[232.637056378] (100.0, 102.0, 104.0, 106.0, 108.0)
Vector[9.48683298051] (2.0, 3.0, 4.0, 5.0, 6.0)
Vector[272.818621065] (120.0, 121.0, 122.0, 123.0, 124.0)
5.47722557505 228.100854887
[0.0, 1.0, 4.0, 9.0, 16.0]

```

下面比较Vector对象和NumPy数组的矢量加法的运算速度：

```
v1 = Vector(range(10000))
v2 = Vector(range(10000))
%timeit v1 + v2

a1 = np.arange(10000, dtype=float)
a2 = np.arange(10000, dtype=float)
%timeit a1 + a2

100000 loops, best of 3: 8.04 µs per loop
100000 loops, best of 3: 9.68 µs per loop
```

下面比较Vector对象和NumPy数组的元素存取速度：

```
%timeit v1[100]
%timeit v1[100] = 2.0
%timeit a1[100]
%timeit a1[100] = 2.0

10000000 loops, best of 3: 62.2 ns per loop
10000000 loops, best of 3: 62.1 ns per loop
10000000 loops, best of 3: 122 ns per loop
10000000 loops, best of 3: 108 ns per loop
```

10.5.3 包装ahocorasick库

扩展类型经常用于对C语言函数库进行包装，提供一个面向对象的Python调用接口。本节以包装“多模式匹配算法”的C语言库ahocorasick为例，介绍使用扩展类型包装C语言函数库的方法。

下面演示一下该扩展类型的使用方法。首先使用一组关键字创建 MultiSearch 对象，然后调用其 isin() 方法，在目标字符串中搜索关键字，只要有一个关键字存在于该字符串中，就返回 True，否则返回 False。



scipy2.cython.multisearch 模块对 C 语言函数库 ahocorasick 进行包装。使用该模块可以快速在大量文本中同时搜索多个关键字。

```
from scipy2.cython import MultiSearch

ms = MultiSearch(["abc", "xyz"])
print ms.isin("123abcdef")
print ms.isin("123uvwxyz")
print ms.isin("123456789")

True
True
False
```

search() 方法可用于在目标字符串中搜索关键字所在的位置，它的第二个参数为一个回调函数，每找到一个匹配位置就将该位置和匹配的关键字传递给该回调函数。回调函数返回 0 表示继续搜索，返回 1 表示结束搜索。

```
def process(pos, pattern):  
    print "found {0} at {1}".format(pattern, pos)  
    return 0  
  
ms.search("123abc456xyz789abc", process)  
  
found abc at 3  
found xyz at 9  
found abc at 15
```

还可以使用`iter_search()`方法返回一个迭代器:

```
for pos, pattern in ms.iter_search("123abc456xyz789abc"):  
    print "found {0} at {1}".format(pattern, pos)  
  
found abc at 3  
found xyz at 9  
found abc at 15
```

在开始编写扩展类型之前, 让我们先看看在C语言中如何使用该库:

```
#include <stdio.h>  
  
#include "ahocorasick.h"  
  
/* 搜索关键字列表 */  
AC_ALPHABET_t * allstr[] = {  
    "recent", "from", "college"  
};
```

```

#define PATTERN_NUMBER (sizeof(allstr)/sizeof(AC_ALPHABET_t
*))

/* 搜索文本 */
AC_ALPHABET_t * input_text = {"She recently graduated from
college"};

/** 匹配时的回调函数
int match_handler(AC_MATCH_t * m, void * param)
{
    unsigned int j;

    printf ("%d : %s\n", m->position, m->patterns-
>astring);
    /* 返回0继续搜索, 返回1停止搜索 */
    return 0;
}

int main (int argc, char ** argv)
{
    unsigned int i;

    AC_AUTOMATA_t * acap;
    AC_PATTERN_t tmp_patt;
    AC_TEXT_t tmp_text;

    /** 创建AC_AUTOMATA_t结构体, 并传递回调函数

```

```
    acap = ac_automata_init();

    /*** 添加关键字
    for (i=0; i<PATTERN_NUMBER; i++)
    {
        tmp_patt.astring = allstr[i];
        tmp_patt.rep.number = i+1; // optional
        tmp_patt.length = strlen(tmp_patt.astring);
        ac_automata_add (acap, &tmp_patt);
    }

    /*** 结束添加关键字
    ac_automata_finalize (acap);

    /*** 设置待搜索字符串
    tmp_text.astring = input_text;
    tmp_text.length = strlen(tmp_text.astring);

    /*** 搜索
    ac_automata_search (acap, &tmp_text, 0, match_handler,
NULL);

    /*** 释放内存
    ac_automata_release (acap);
    return 0;
}
```

由上面的程序可以看出，整个函数库都是围绕AC_AUTOMATA_t结构体进行处理的。这是C语言封装数据的一种常用方式。在Cython中使用扩展类型对这种函数库进行包装时，通常会创建一个指向此结构体的指针属性，并在__cinit__()和__dealloc__()中分配和释放此结构体。然后定义一些def方法，调用C语言函数库提供的各个API函数以实现封装。

下面我们分段介绍如何将C语言的函数库使用扩展类型进行包装。

```
cdef extern from "ahocorasick.h": ❶

    ctypedef int (*AC_MATCH_CALLBACK_f)(AC_MATCH_t *, void
*) ❷

    ctypedef enum AC_STATUS_t: ❸

        ACERR_SUCCESS = 0

        ACERR_DUPLICATE_PATTERN

        ACERR_LONG_PATTERN

        ACERR_ZERO_PATTERN

        ACERR_AUTOMATA_CLOSED

    ctypedef struct AC_MATCH_t: ❹

        AC_PATTERN_t * patterns

        long position

        unsigned int match_num

    ctypedef struct AC_AUTOMATA_t:

        AC_MATCH_t match
```

```

    typedef struct AC_PATTERN_t:
        char * astring
        unsigned int length

    typedef struct AC_TEXT_t:
        char * astring
        unsigned int length

    ⑤

    AC_AUTOMATA_t * ac_automata_init()
        AC_STATUS_t ac_automata_add(AC_AUTOMATA_t * this,
AC_PATTERN_t * pattern)

    void ac_automata_finalize(AC_AUTOMATA_t * this)
        int ac_automata_search(AC_AUTOMATA_t * this, AC_TEXT_t
* text, int keep,
        AC_MATCH_CALLBACK_f callback, void * param)
        void ac_automata_settext (AC_AUTOMATA_t * this,
AC_TEXT_t * text, int keep)
        AC_MATCH_t * ac_automata_findnext (AC_AUTOMATA_t *
this)

    void ac_automata_release(AC_AUTOMATA_t * this)

```

❶首先需要通过`cdef extern from ...`告诉Cython：编译之后的C语言程序需要包含`ahocorasick.h`头文件。由于Cython不会自动解析C语言的头文件，因此还需要将其中用到的类型、常量和函数原型都用Cython的语法声明一遍。

❷定义函数指针类型MATCH_CALLBACK_f，它是指向回调函数的指针类型。其第一个参数为指向保存匹配数据的结构体的指针，第二个参数是可以指向任何额外数据的指针。C语言中通常用这种void *类型的指针传递用户自定义的数据。

❸❹定义枚举类型和结构体类型，只需要定义在Cython程序中用到的枚举成员和结构体的字段即可。如果在Cython中不访问某结构体的任何字段，可以使用pass关键字代替字段的定义。

❺定义Cython程序中将要调用的函数原型。

将上面这一大段程序编译成C语言程序之后，只有#include "ahocorasick.h"一句，而其余的类型声明则告诉Cython如何编译对这些类型进行操作的语句。例如结构体AC_PATTERN_t中的length字段被声明为unsigned int类型，因此在必要的时候Cython会调用Python/C API以在Python的整数对象和unsigned int类型之间进行转换。

接下来是MultiSearch扩展类型的定义：

```
cdef class MultiSearch:

    cdef AC_AUTOMATA_t * _auto ❶

    cdef bint found

    cdef object callback

    cdef object exc_info

    def __cinit__(self, keywords):
        self._auto = ac_automata_init()
```

```

        if self._auto is NULL:
            raise MemoryError
        self.add(keywords) ❷

def __dealloc__(self):
    if self._auto is not NULL:
        ac_automata_release(self._auto)

cdef add(self, keywords):
    cdef AC_PATTERN_t pattern
    cdef bytes keyword
    cdef AC_STATUS_t err

    for keyword in keywords: ❸
        pattern.cstring = <char *>keyword
        pattern.length = len(keyword)
        err = ac_automata_add(self._auto, &pattern)
        if err != ACERR_SUCCESS:
            raise ValueError("Error Code:%d" % err)

    ac_automata_finalize(self._auto)

```

❶ `_auto` 属性是一个指向 `AC_AUTOMATA_t` 结构体的指针。在 `__cinit__()` 中调用 `ac_automata_init()` 为其分配内存，而在 `__dealloc__()` 中调用 `ac_automata_release()` 以释放内存。

❷ `AC_AUTOMATA_t` 结构体分配成功之后，调用 `cdef` 函数 `add()` 将所有关键字添加进该结构体中。

③在add() 内部对keywords参数进行迭代，将其每个元素都当作bytes类型处理，通过<char *>将其转换成C语言的字符指针类型，和其长度一起使用AC_PATTERN_t 结构体打包之后传递给ac_automata_add()。由于在该函数返回之后，ahocorasick内部的函数不会再使用字符指针指向的内容，因此这种做法是安全的。如果在后续的函数调用中需要使用字符指针指向的内容，则需要对keywords中的每个字符串对象进行引用，保证它们不会被提前垃圾回收。

接下来是isin()方法的定义：

```
def isin(self, bytes text, bint keep=False):  
    cdef AC_TEXT_t temp_text ❶  
    temp_text.cstring = <char *>text  
    temp_text.length = len(text)  
    self.found = False ❷  
    ac_automata_search(self._auto, &temp_text, keep,  
isin_callback, <void *>self) ❸  
    return self.found
```

❶将目标字符串用AC_TEXT_t 结构体打包之后，❸传递给ac_automata_search() 进行搜索。在搜索之前，❷设置found属性为False。将isin_callback() 函数的地址传递给ac_automata_search() 作为搜索的回调函数。其最后一个参数为传递给回调函数的用户数据，这里通过它将MultiSearch对象的地址传递给回调函数。这样在回调函数中就可以访问MultiSearch对象的属性了。

下面是isin_callback() 回调函数的定义，注意由于该函数在C语言库内部被调用，只能使用cdef定义：

```

cdef int isin_callback(AC_MATCH_t * match, void * param):
    cdef MultiSearch ms = <MultiSearch> param ❶
    ms.found = True ❷
    return 1 ❸

```

isin_callback() 的第一个参数是描述匹配信息的结构体指针，第二个参数为指向MultiSearch对象的指针。❶首先将void *类型的指针转换为MultiSearch对象，然后就可以通过ms访问MultiSearch扩展类中定义的属性和各种方法了。❷设置MultiSearch对象的found属性为True，表示找到一个匹配位置。❸由于isin() 只需要找到一个匹配位置即可，因此函数返回1，表示不需要继续搜索了。

下面定义search()，它的第一个参数为搜索的目标字符串，第二个参数是Python的可调用对象，每找到一个匹配位置就调用该对象进行处理：



为了介绍C语言回调函数和Python回调函数的用法，这里使用 ac_automata_search()，实际上使用后面介绍的 ac_automata_findnext() 可以更方便地编写 isin() 和 search() 函数。

```

def search(self, bytes text, callback, bint
keep=False):
    cdef AC_TEXT_t temp_text
    temp_text.astring = <char *>text

```

```

        temp_text.length = len(text)

        self.found = False

        self.callback = callback ❶

        self.exc_info = None

        ac_automata_search(self._auto, &temp_text, keep,
search_callback, <void *>self) ❷

        if self.exc_info is not None:

            raise self.exc_info[1], None, self.exc_info[2]
❸

```

❷ac_automata_search()的回调函数为search_callback()。为了在其中调用Python的可调用对象，❶将callback参数传递给self.callback。❸由于C语言的函数无法向上传递Python函数所抛出的异常信息，因此还需要通过exc_info属性传递Python回调函数中可能抛出的异常。

下面是search_callback()回调函数的定义：

```

cdef int search_callback(AC_MATCH_t * match, void * param):

    cdef MultiSearch ms = <MultiSearch> param

    cdef bytes pattern = match.patterns.astring

    cdef int res = 1

    try:

        res = ms.callback(match.position - len(pattern),
pattern) ❶

    except Exception as ex:

        import sys

```

```
ms.exc_info = sys.exc_info() ❷  
return res
```

将其第二个参数转换为MultiSearch对象之后，❶调用callback指向的Python回调函数，并捕捉可能抛出的异常。❷将异常及回溯信息保存在属性exc_info中。在search()的最后检查属性exc_info，若被设置，则抛出其中的异常对象。注意这里使用sys.exc_info()获取异常信息，而不是直接保存捕获的异常，这样才能保证异常的回溯信息能正确指示出错的位置。

ahocorasick 库中还提供了ac_automata_settext()和ac_automata_findnext()，使用这两个函数可以编写如下生成器函数iter_search()：

```
def iter_search(self, bytes text, bint keep=False):  
    cdef AC_TEXT_t temp_text  
    cdef AC_MATCH_t * match  
    cdef bytes matched_pattern  
    temp_text.cstring = <char *>text  
    temp_text.length = len(text)  
    ac_automata_settext(self._auto, &temp_text, keep)  
    while True:  
        match = ac_automata_findnext(self._auto)  
        if match == NULL:  
            break  
        matched_pattern = <bytes>match.patterns.cstring  
        yield match.position - len(matched_pattern),  
        matched_pattern
```

10.6 Cython技巧集



与本节内容对应的Notebook为：`10-cython/cython-600-tips.ipynb`。

相信读者通过前面章节的介绍，已经掌握了使用Cython提高Python程序运算速度的一些基本方法。作为本章的最后一节，让我们看看Cython的一些高级使用技巧。

10.6.1 创建ufunc函数

NumPy的ufunc函数是一种能对数组的每个元素进行操作的函数，而NumPy的C-API提供了通过C语言创建ufunc函数的方法，请感兴趣的读者访问下面的网址来阅读相关的教程：



<http://docs.scipy.org/doc/numpy-dev/user/c-info.ufunc-tutorial.html>

使用NumPy的C-API编写ufunc函数的教程。

由于Cython最终被翻译成C语言程序，因此我们可以使用Cython程序调用NumPy的C-API来创建ufunc函数。创建ufunc函数需要三个全局变量：

- **functions**：保存对一维数组进行循环计算的函数指针，如果ufunc函数支持处理多种dtype数组，则每种dtype对应一个函数指针。
- **signatures**：表示ufunc函数的参数和返回值的字符数组。
- **data**：空指针数组，其中的指针指向传递给functions中对应函数的额外数据。

下面的程序创建一个计算Logistic函数的ufunc函数logistic()：

```
%%cython
from libc.math cimport exp
    from numpy cimport (PyUFuncGenericFunction, npy_intp,
import_ufunc,
                                                                    NPY_DOUBLE, PyUFunc_None,
PyUFunc_FromFuncAndData) ❶

import_ufunc() ❷

    cdef void double_logistic(char **args, npy_intp
*dimensions,
                                                                    npy_intp* steps, void* data):
    cdef: #用缩进可以定义多行cdef变量
        npy_intp i
        npy_intp n = dimensions[0]
```

```

        char *in_ptr = args[0]
        char *out_ptr = args[1]
        npy_intp in_step = steps[0]
        npy_intp out_step = steps[1]

        double x, y

    for i in range(n):
        x = (<double *>in_ptr)[0]
        y = 1.0 / (1.0 + exp(-x))
        (<double *>out_ptr)[0] = y

        in_ptr += in_step
        out_ptr += out_step

cdef:
    PyUFuncGenericFunction *functions = [&double_logistic]
3
    char *signatures = [NPY_DOUBLE, NPY_DOUBLE] 4
    void **data = [NULL] 5

    logistic1 = PyUFunc_FromFuncAndData(functions, data,
signatures, 6
        1, #ntypes
        1, #nin
        1, #nout
        PyUFunc_None, #identity

```

```
        "logistic", #name
        "a sigmoid function:  $y = 1 / (1 + \exp(-x))$ ", #doc string
        0) # unused
```

❶ Cython包含了NumPy的C-API的声明文件，首先通过`cimport`从其中载入将要用到的函数和类型声明。❷在调用NumPy的任何ufunc相关的C-API函数之前，需要首先调用`import_ufunc()`，否则程序会异常终止。

我们所创建的`logistic()`函数只对元素类型为`double`的数组进行计算，❸因此`functions`中只保存对一维`double`数组进行计算的`double_logistic()`函数的地址。

`PyUFuncGenericFunction`为`double_logistic()`的函数指针类型，参数的含义如下：

- `args`: 保存所有输入输出数组的地址，`args[0]`为输入数组的地址，`args[1]`是输出数组的地址。
- `dimensions`: 保存循环的次数。
- `steps`: 保存各个数组的元素之间的间隔，`steps[0]`为输入数组的元素间隔，`steps[1]`为输出数组的元素间隔。
- `data`: 传递给该函数的额外参数，在C语言中，为了表示任意类型的额外参数，通常使用`void *`类型的指针。

由上面的参数可知，`PyUFuncGenericFunction`类型的函数可以处理各种元素类型的数组，而数组元素无须连续存储，可以处理使用切

片获得的数组视图，并且输入数组的个数和输出数组的个数是任意的。

④signatures中保存表示ufunc函数的输入和输出数组的元素类型，本例中输入和输出都是双精度浮点数，因此值为[NPY_DOUBLE, NPY_DOUBLE]。注意这里使用的是Cython中初始化数组的语法。它相当于：

```
cdef char signatures[2]
signatures[0] = NPY_DOUBLE
signatures[1] = NPY_DOUBLE
```

⑤double_logistic()无须额外的参数，因此data被初始化为一个指向NULL的空指针。

⑥最后调用PyUFunc_FromFuncAndData()创建ufunc函数对象。它的头三个参数为functions、data、signatures，接下来的参数为：

- ntypes：该ufunc函数支持的数据类型的个数，即functions数组的长度。由于本例中只支持双精度浮点数，因此值为1。
- nin：ufunc函数的输入数组的个数。
- nout：ufunc函数的输出数组的个数。
- identity：可选值为PyUFunc_One、PyUFunc_Zero、PyUFunc_None。该参数指定ufunc函数的reduce()方法的参数为空数组时的返回值。由于本例中的ufunc函数为单输入函数，因此reduce()方法无效。
- name和doc_string：分别指定ufunc函数名和帮助文档。

如果希望`logistic()`能够处理单精度浮点数，可以添加进行单精度浮点数运算的`float_logistic()`函数。只需将`double_logistic()`中的所有`double`都替换为`float`即可。

然后如下定义`functions`、`signatures`、`data`参数：

```
cdef:
    PyUFuncGenericFunction *functions = [&double_logistic,
&float_logistic]
    char *signatures = [NPY_DOUBLE, NPY_DOUBLE, NPY_FLOAT,
NPY_FLOAT]
    void **data = [NULL, NULL]
```

最后将`PyUFunc_FromFuncAndData()`的`ntypes`参数设置为2。

下面测试`logistic1()`函数。虽然只定义了一个`double`类型的计算函数，但是它仍然可以处理各种类型的数组，甚至列表，这是因为在`ufunc`函数内部会将参数转换成`double`数组，然后再传递给`double_logistic()`进行计算。如果参数是多维数组，`ufunc`函数会对每个轴进行循环，多次调用`double_logistic()`实现对整个数组的计算。

```
logistic1([-1, 0, 1])
array([ 0.26894142,  0.5          ,  0.73105858])
```

上面的`double_logistic()`虽然不难编写，但是对每个数学函数都进行类似的包装却是令人厌烦的重复劳动。幸好可以利用NumPy提供的一些辅助函数将单个数值的运算函数转换成`ufunc`函数。在下面的例子中，❶`scalar_logistic()`是一个输入和输出都为`double`的单数值运算

函数。❸我们将该函数通过额外参数data传递给NumPy提供的PyUFunc_d_d()函数。

❷PyUFunc_d_d()对作为输入和输出的两个一维double数组进行循环，将输入数组中的每个元素传给data指向的函数进行计算，并将结果写入输出数组中。

```
%%cython
from libc.math cimport exp
from numpy cimport (PyUFuncGenericFunction, import_ufunc,
PyUFunc_d_d,
NPY_DOUBLE, PyUFunc_None,
PyUFunc_FromFuncAndData)

import_ufunc()

cdef double scalar_logistic(double x): ❶
    return 1.0 / (1.0 + exp(-x))

cdef:
    PyUFuncGenericFunction *functions = [PyUFunc_d_d] ❷
    char *signatures = [NPY_DOUBLE, NPY_DOUBLE]
    void **data = [&scalar_logistic] ❸

    logistic2 = PyUFunc_FromFuncAndData(functions, data,
signatures,
```

```
1, 1, 1, PyUFunc_None,  
"logistic",  
"a sigmoid function:  $y = 1 / (1 + \exp(-x))$ ",  
0)
```

logistic2()与logistic1()的差别仅仅是多了一次C语言的函数调用。下面测试该调用所带来的损耗，从结果可以看出二者的差距并不大，而logistic2()的实现更加简洁、更容易维护：

```
x = np.linspace(-6, 6, 10000)  
%timeit logistic1(x)  
%timeit logistic2(x)  
  
10000 loops, best of 3: 201  $\mu$ s per loop  
1000 loops, best of 3: 208  $\mu$ s per loop
```

实际上，已经有人将NumPy的C-API中与PyUFunc_d_d()类似的函数包装在Cython的include文件中。include文件通过include关键字载入，它和C语言的#include预处理命令类似，将指定文件的内容插入目标文件中。在numpy_ufuncs.pxi中使用了前面介绍的方法来创建ufunc函数，请感兴趣的读者自行阅读源代码。

在numpy_ufuncs.pxi中提供了与functions、signatures、data类似的三个数组，它们可以容纳100个PyUFuncGenericFunction函数。在下面的例子中，通过register_ufunc_d()和register_ufunc_dd()分别将一元函数scalar_logistic()和二元函数scalar_peaks()转换成ufunc函数：

```

%%cython
include "numpy_ufuncs.pxi"
from libc.math cimport exp

cdef double scalar_logistic(double x):
    return 1.0 / (1.0 + exp(-x))

cdef double scalar_peaks(double x, double y):
    return x * exp(-x*x - y*y)

logistic3 = register_ufunc_d(scalar_logistic,
                             "logistic", "logistic
function", PyUFunc_None)

peaks = register_ufunc_dd(scalar_peaks,
                           "peaks", "peaks function",
PyUFunc_None)

```

下面通过ogrid创建两个可广播成二维网格的数组X和Y，然后调用peaks()计算网格上所有点对应的函数值：

```

Y, X = np.ogrid[-2:2:100j, -2:2:100j]
pl.pcolormesh(X, Y, peaks(X, Y))

```

10.6.2 快速调用DLL中的函数

通过Python的标准库ctypes可以很方便地调用动态链接库中的函数。但是由于ctypes在调用实际的函数之前需要进行许多预处理工作，因此函数的调用效率并不高。如果需要在循环中大量调用，这种预处理带来的损耗会极大地影响程序的执行速度。本节介绍如何通过ctypes找到动态链接库中的函数的地址，然后将地址传递给Cython的函数进行循环调用，提高函数的调用效率。

首先用C语言编写函数peaks()，为了确认是否能正确获取该函数的地址，我们通过get_addr()返回peaks()的地址：

```
%%file peaks.c
#include <math.h>

double peaks(double x, double y)
{
    return x * exp(-x*x - y*y);
}

unsigned int get_addr()
{
    return (unsigned int)(void *)peaks;
}
```

接下来调用gcc将peaks.c编译成peaks.dll：

```
!gcc -Ofast -shared -o peaks.dll peaks.c
```

然后通过ctypes载入peaks.dll，并为其中的peaks()函数声明参数类型和返回值类型，最后调用该函数：

```
import ctypes
lib = ctypes.CDLL("peaks.dll")
lib.peaks.argtypes = [ctypes.c_double, ctypes.c_double]
lib.peaks.restype = ctypes.c_double
lib.peaks(1.0, 2.0)
0.006737946999085465
```

`lib.peaks`是`ctypes`对C语言的函数地址进行包装之后的对象，为了获取其中的C语言函数地址，可以将该对象通过`cast()`转换成空指针类型，然后通过空指针对象的`value`属性获得该指针的内容，即C语言函数的地址。下面的程序获得C语言函数的地址，并与`get_addr()`的返回值进行比较：

```
addr = ctypes.cast(lib.peaks, ctypes.c_void_p).value
addr == lib.get_addr()
True
```

下面用Cython编写`vectorize_2d()`函数，其`func`参数是`ctypes`的函数指针对象，`x`和`y`是两个二维的连续存储的数组。

❶首先通过`ctypedef`关键字声明二元双精度浮点数函数指针类型`Function`。❷在`vectorize_2d()`中首先通过`cast()`获得函数的地址，然后❸将该地址转换成`Function`类型的函数指针，接下来就可以在双重循环中通过该函数指针快速调用其指向的C语言函数了。

```
%%cython
import cython
import numpy as np
```

```

from ctypes import cast, c_void_p

ctypedef double(*Function)(double x, double y) ❶

@cython.wraparound(False)
@cython.boundscheck(False)
def vectorize_2d(func, double[:, ::1] x, double[:, ::1] y):
    cdef double[:, ::1] res = np.zeros_like(x.base)
    cdef size_t addr = cast(func, c_void_p).value ❷
    cdef Function func_ptr = <Function><void *>addr ❸
    cdef int i, j

    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            res[i, j] = func_ptr(x[i, j], y[i, j])

    return res.base

```

下面通过 `vectorize_2d()` 调用 `lib.peaks()`，并与通过 `vectorize()` 创建的ufunc函数的结果比较：

```

Y, X = np.mgrid[-2:2:200j, -2:2:200j]
vectorize_peaks = np.vectorize(lib.peaks, otypes=['f8'])
np.allclose(vectorize_peaks(X, Y), vectorize_2d(lib.peaks,
X, Y))

True

```


`vectorize_2d()` 只能对两个连续存储的二维数组进行循环，如果希望像ufunc函数一样对任意数组进行计算，可以使用扩展类型对 `numpy_ufuncs.pxi` 中的函数进行简单包装。

① `UFunc_dd` 扩展类只有一个 `ufunc` 属性，用来保存创建的ufunc函数对象。② 在其 `__cinit__()` 方法中创建一个调用其参数 `func` 的ufunc函数对象。使用 `set_func()` 方法可以修改ufunc函数的额外参数。③ 将 `ufunc` 属性转换为C语言的ufunc结构体，该结构体的定义可通过 `from numpy cimport ufunc` 载入，该载入操作已经在 `numpy_ufuncs.pxi` 定义。④ 然后将传入的函数地址写入ufunc结构体的 `data` 字段所指向的数组中。由于 `register_ufunc_dd()` 创建的ufunc函数包含两个 `PyUFuncGenericFunction` 函数指针，因此需要同时修改这两个函数的额外参数。

```
%%cython
include "numpy_ufuncs.pxi"
from ctypes import cast, c_void_p

cdef class UFunc_dd:
    cdef public object ufunc ①

    def __cinit__(self, func): ②
        cdef size_t addr = cast(func, c_void_p).value
        self.ufunc = register_ufunc_dd(<double (*)(<double,
double)>addr,
                                     "ufunc", "variable
ufunc", PyUFunc_None)
```

```

def set_func(self, func):
    cdef ufunc f = <ufunc> self.ufunc ❸
    cdef size_t addr = cast(func, c_void_p).value
    f.data[0] = <void *>addr ❹
    f.data[1] = <void *>addr

```

下面先让所创建的ufunc函数使用lib.peaks()进行计算，然后通过 set_func() 将其指向的运算函数修改为 ctypes.cdll.msvcrt.atan2，即C语言的math.h中定义的atan2()函数。使用这种方法，可以将任何动态链接库中的类型为 double(double, double)的函数通过set_func()传递给ufunc_dd对象，使其与NumPy内置的ufunc函数一样能对数组进行广播运算。

```

ufunc_dd = UFunc_dd(lib.peaks)
Y, X = np.ogrid[-2:2:200j, -2:2:200j]
assert np.allclose(vectorize_peaks(X, Y), ufunc_dd.ufunc(X,
Y))

ufunc_dd.set_func(ctypes.cdll.msvcrt.atan2)
assert np.allclose(np.arctan2(X, Y), ufunc_dd.ufunc(X, Y))

```

10.6.3 调用BLAS函数

BLAS是一套基础线性代数程序集的API标准，SciPy的许多高速线性代数运算函数内部都会调用用Fortran语言编写的BLAS函数。虽然这些Fortran函数的运算效率很高，但Python的调用接口会造成一定的效

率损耗，在大量循环中调用时这种损耗不能忽略不计。可以使用Cython循环调用这些函数，从而彻底摆脱Python调用接口的束缚。

1. 包装saxpy()函数

BLAS中的API函数可以通过`scipy.linalg.blas`模块访问，下面演示调用其中的`saxpy()`：

```
from scipy.linalg import blas
import numpy as np
x = np.array([1, 2, 3], np.float32)
y = np.array([1, 3, 5], np.float32)
blas.saxpy(x, y, a=0.5)
array([ 1.5,  4. ,  6.5], dtype=float32)
```

`saxpy`是一个<fortran object>，其代码可以在`fortranobject.h`和`fortranobject.c`中找到。这两个文件可以在NumPy的安装目录下找到，位于`numpy\f2py\src`文件夹下。

```
blas.saxpy
<fortran object>
```

其`_cpointer`属性为包装fortran函数地址的PyCObject对象。要获得PyCObject包装的指针，可以调用Python的C-API中的`PyCObject_AsVoidPtr`：

```
void* PyCObject_AsVoidPtr(PyObject* self)
```

下面的程序通过 ctypes 调用 Python 的 C-API 函数，首先通过 `py_object()` 将 Python 对象转换成指向 `PyObject` 结构体的指针，然后调用 `PyCObject_AsVoidPtr()` 获得该 `PyCObject` 对象所包装的地址，`saxpy_addr` 就是 BLAS 扩展库中 Fortran 函数 `saxpy()` 的地址：

```
import ctypes

saxpy_addr = ctypes.pythonapi.PyCObject_AsVoidPtr(
    ctypes.py_object(blas.saxpy._cpointer))

saxpy_addr
196931082
```

获得函数的地址之后，还需要知道函数的调用原型。通过下面的网址查看 `saxpy()` 的帮助文档，可知其调用参数如下：

```
subroutine saxpy(
    integer    N,
    real       SA,
    real, dimension(*)    SX,
    integer    INCX,
    real, dimension(*)    SY,
    integer    INCY
)
```



http://www.netlib.org/lapack/explore-html/d8/daf/saxpy_8f.html

BLAS库中 `saxpy()` 的函数原型。

注意Fortran语言采用传址调用，即调用函数时传递的参数和函数中接收的参数是相同的内存地址。因此其C语言的函数原型为：

```
void saxpy(int *N, float *SA, float *SX, int *INCX, float *SY, int *INCY);
```

在Cython程序中声明指向saxpy()的函数指针类型saxpy_ptr，并通过前面介绍的方法获得Fortran函数的地址，并转换为函数指针_saxpy。然后定义调用_saxpy的blas_saxpy()和直接在Cython中循环的cython_saxpy()：

```
%%cython
import cython
from cpython cimport PyObject_AsVoidPtr
from scipy.linalg import blas

ctypedef void (*saxpy_ptr) (const int *N, const float
*alpha,
    const float *X, const int *incX, float *Y, const int
*incY) nogil

cdef saxpy_ptr _saxpy=
<saxpy_ptr>PyObject_AsVoidPtr(blas.saxpy._cpointer)

def blas_saxpy(float[:] y, float a, float[:] x):
    cdef int n = y.shape[0]
    cdef int inc_x = x.strides[0] / sizeof(float)
```

```

    cdef int inc_y = y.strides[0] / sizeof(float)
    _saxpy(&n, &a, &x[0], &inc_x, &y[0], &inc_y)

@cython.wraparound(False)
@cython.boundscheck(False)
def cython_saxpy(float[:] y, float a, float[:] x):
    cdef int i
    for i in range(y.shape[0]):
        y[i] += a * x[i]

```

下面比较二者的运行速度，blas_saxpy() 要比Cython的循环快三倍左右：

```

a = np.arange(100000, dtype=np.float32)
b = np.zeros_like(a)
%timeit blas_saxpy(b, 0.2, a)
%timeit cython_saxpy(b, 0.2, a)

10000 loops, best of 3: 28.6 µs per loop
10000 loops, best of 3: 98.3 µs per loop

```

2. dgemm() 高速矩阵乘积



http://www.netlib.org/lapack/explore-html/d7/d2b/dgemm_8f.html

DGEMM的说明文档。

BLAS中的DGEMM()实现如下矩阵乘积运算，当参数alpha为1、beta为0时，结果C为矩阵A和B的乘积：

$$C = \alpha * \text{op}(A) * \text{op}(B) + \beta * C$$

其中的op()可以对矩阵进行转置。由于Fortran数组与C语言数组的轴的顺序相反，为了对两个C语言的数组表示的矩阵进行乘积运算，需要设置op()为转置。而无论是否转置，运算结果C都是Fortran格式的数组。

Fortran函数的dgemm()的参数如下：

```
subroutine dgemm (  
    character    TRANSA,  
    character    TRANSB,  
    integer      M,  
    integer      N,  
    integer      K,  
    double precision    ALPHA,  
    double precision, dimension(lda,*)  A,  
    integer      LDA,  
    double precision, dimension(ldb,*)  B,  
    integer      LDB,  
    double precision    BETA,  
    double precision, dimension(ldc,*)  C,
```

```
integer      LDC
)
```

下面为其C语言的调用函数原型：

```
void dgemm( char *ta, char *tb,
            int *m, int *n, int *k,
            double *alpha,
            double *a, int *lda,
            double *b, int *ldb,
            double *beta,
            double *c, int *ldc)
```

在下面的Cython函数dgemm(A,B,index)中，A和B为两个C语言格式的三维数组，形状分别为(La, M, K)和(Lb, K, N)。index是一个形状为(Lc, 2)的整型数组。该函数对index中的每对整数j、k计算 $C[i] = A[j] * B[k]$ ，其中i为该对整数的下标。因此函数的返回值C是形状为(Lc, N, M)的三维数组。可以将C看作Lc个Fortran格式的二维数组。

由于存取内存视图的元素时不涉及任何与Python有关的操作，而每对矩阵的乘积运算相对独立，因此可以对这部分进行并行运算。Cython使用OpenMP实现并行运算，因此在编译时需要设置编译和连接参数-fopenmp。

❶载入并行运算的prange()函数，该函数会被编译成使用并行运算的循环。❷设置其nogil参数为True，表示在并行运算时释放Python的全局锁。


```

%%cython -c-Ofast -c-fopenmp --link-args=-fopenmp

from cython.parallel import prange ❶
import cython
import numpy as np
from cpython cimport PyCObject_AsVoidPtr
from scipy.linalg import blas

ctypedef void (*dgemm_ptr) (char *ta, char *tb,
                           int *m, int *n, int *k,
                           double *alpha,
                           double *a, int *lda,
                           double *b, int *ldb,
                           double *beta,
                           double *c, int *ldc) nogil

cdef          dgemm_ptr          _dgemm=
<dgemm_ptr>PyCObject_AsVoidPtr(blas.dgemm._cpointer)

@cython.wraparound(False)
@cython.boundscheck(False)
def dgemm(double[:, :, :] A, double[:, :, :] B, int[:, ::1]
index):
    cdef int m, n, k, i, length, idx_a, idx_b
    cdef double[:, :, :] C
    cdef char ta, tb
    cdef double alpha = 1.0

```

```

cdef double beta = 0.0

length = index.shape[0]
m, k, n = A.shape[1], A.shape[2], B.shape[2]

C = np.zeros((length, n, m))

ta = "T"
tb = ta

for i in prange(length, nogil=True): ❷
    idx_a = index[i, 0]
    idx_b = index[i, 1]
    _dgemm(&ta, &tb, &m, &n, &k, &alpha,
&A[idx_a, 0, 0], &k,
&B[idx_b, 0, 0], &n,
&beta,
&C[i, 0, 0], &m)

return C.base

```

NumPy中新增加的gufunc函数能把单个矩阵的运算通过广播运用到整个数组之上。虽然numpy.linalg中提供了许多gufunc函数，但是缺少矩阵乘积的gufunc函数。类似的功能可以利用上面的dgemm()来实现。下面的matrix_multiply(a, b)对两个任意维数的数组的最后两个轴进行矩阵乘积运算，而对其他轴进行广播运算。例如，若a的形状为(12, 1, 10, 100, 30)，b的形状为(1, 15, 1, 30, 50)，最后两轴

对应的矩阵乘积之后的形状为(100, 50)，而其他轴广播之后的形状为(12, 15, 10)，因此结果数组的形状为(12, 15, 10, 100, 50)。一共进行了 $12 \times 15 \times 10$ 次矩阵乘积运算。

该程序的实现思路如下，请读者自行研究，细节部分就不详细叙述了：

- 对a中的每个矩阵编号，并把编号的形状修改为a的广播部分的形状，得到idx_a。
- 对b进行同样的运算，得到idx_b。
- 使用broadcast_arrays() 计算idx_a和idx_b广播之后的数组。
- 将上述两个数组平坦化之后排成两列，就得到了dgemm()函数的index参数。
- 将a和b的形状都修改为三维数组并传递给dgemm()函数。

```
def matrix_multiply(a, b):
    if a.ndim <= 2 and b.ndim <= 2:
        return np.dot(a, b)

    a = np.ascontiguousarray(a).astype(np.float,
copy=False)

    b = np.ascontiguousarray(b).astype(np.float,
copy=False)

    if a.ndim == 2:
        a = a[None, :, :]

    if b.ndim == 2:
        b = b[None, :, :]
```

```

shape_a = a.shape[:-2]
shape_b = b.shape[:-2]
len_a = np.prod(shape_a)
len_b = np.prod(shape_b)

idx_a = np.arange(len_a,
dtype=np.int32).reshape(shape_a)
idx_b = np.arange(len_b,
dtype=np.int32).reshape(shape_b)

idx_a, idx_b = np.broadcast_arrays(idx_a, idx_b)

index = np.column_stack((idx_a.ravel(), idx_b.ravel()))
bshape = idx_a.shape

if a.ndim > 3:
    a = a.reshape(-1, a.shape[-2], a.shape[-1])
if b.ndim > 3:
    b = b.reshape(-1, b.shape[-2], b.shape[-1])

if a.shape[-1] != b.shape[-2]:
    raise ValueError("can't do matrix multiply because
k isn't the same")

c = dgemm(a, b, index)
c = np.swapaxes(c, -2, -1)

```

```
c.shape = bshape + c.shape[-2:]  
return c
```

NumPy的umath_tests模块中提供了一个测试用的计算矩阵乘积的gufunc函数。下面与该函数比较，可以看出运算结果是一致的：

```
import numpy.core.umath_tests as umath  
a = np.random.rand(12, 1, 10, 100, 30)  
b = np.random.rand(1, 15, 1, 30, 50)  
np.allclose(matrix_multiply(a, b), umath.matrix_multiply(a,  
b))  
  
True
```

下面是二者的运算速度比较，如果读者查看CPU的使用率，就会发现运行matrix_multiply()时，所有CPU都会用于计算矩阵乘积。由于采用了并行运算与高速的BLAS函数，因此二者的运算速度相差近6倍。

```
%timeit matrix_multiply(a, b)  
%timeit umath.matrix_multiply(a, b)  
10 loops, best of 3: 47.8 ms per loop  
1 loops, best of 3: 313 ms per loop
```


第11章 实例

作为本书的最后一章，让我们综合前面章节介绍的各个扩展库，编写一些有趣的实际程序。

11.1 使用泊松混合合成图像



与本节内容对应的Notebook为：`11-examples/examples-100-possion.ipynb`。

本节通过一个图像处理的实例程序复习NumPy数组、OpenCV以及SciPy中的稀疏矩阵的用法。泊松混合是一种图像合成算法。它将源图像中指定区域内的纹理信息复制到目标图像的对应区域内。效果如图11-1所示，从左侧开始依次为：源图像、混合区域、目标图像以及泊松混合的结果。

40	7	18	27	19	18	18							72	78	78	71	69	87	88	72	78	78	71	69	87	88
41	24	24	18	16	11	21							66	72	85	79	78	80	85	66	72	85	79	78	80	85
8	13	23	29	19	13	29							72	77	?	?	?	85	85	72	77	?	?	?	85	85
21	32	19	24	17	13	17							85	91	?	?	?	93	83	85	91	?	?	?	93	83
28	24	18	27	19	24	21							89	90	?	?	?	86	77	89	90	?	?	?	86	77
19	24	22	13	22	21	13							86	76	57	65	96	83	86	86	76	57	65	96	83	86
13	17	27	19	21	21	24							83	70	61	74	98	87	101	83	70	61	74	98	87	101

图11-1 泊松混合示意图

11.1.1 泊松混合算法

纹理信息通过拉普拉斯算子计算，可以直接通过OpenCV中的Laplacian()来计算。对于 3×3 的情况，Laplacian()实际上就是使用如下卷积核与图像进行卷积运算：

$$\begin{array}{ccc} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{array}$$

即图像中每个像素值乘以-4后加上其上下左右4个像素的值。

所谓泊松混合，就是指对于指定的区域，让目标图像的Laplacian运算结果与源图像的运算结果相同。图11-1显示了泊松混合运算过程。从左侧开始分别为：

- 源图像，其中指定的区域采用浅灰色表示。
- 对源图像进行拉普拉斯运算的结果，这里只显示指定区域内的值。
- 目标图像，其中用“?”表示未知值。这些未知值要保证对于指定的区域内，目标图像的拉普拉斯运算结果与源图像相同。
- 对目标图像中的未知值进行编号，以便生成方程组。

如果用未知数 x_0 到 x_8 表示图11-1(左3)中的9个未知的像素点，那么它们的值需要满足如下方程组：

$$\begin{aligned} -4 \cdot x_0 + 85 + 77 + x_1 + x_3 &= -7 \\ -4 \cdot x_1 + 79 + x_0 + x_2 + x_4 &= -32 \\ \dots & \\ -4 \cdot x_8 + 86 + 96 + x_5 + x_7 &= 14 \end{aligned}$$

每个方程的右边的值就是每个未知像素对应的源图像的拉普拉斯运算值，而方程左边则是通过拉普拉斯卷积核展开的公式。每个未知像素的上下左右4个相邻的像素可能是已知的值或未知的像素。

因此计算泊松混合最终是求解这样一个方程组。我们知道 `numpy.linalg.solve()` 可以用来解线性方程组，但是由于未知数的个数就是区域中像素的个数，因此通常泊松混合所涉及的未知数是非常多的。例如，对于一个 100×100 的区域进行混合，需要创建一个 10000×10000 的系数矩阵A。

显然需要采用 SciPy 的稀疏矩阵运算相关函数：`scipy.sparse.linalg.spsolve()`。剩下的问题就是如何创建 `spsolve()` 所需的两个系数矩阵A和b。

从上面的例子可以看出，所有的方程都是一种形式，但是根据邻接像素是已知像素还是未知像素，实际的方程是不同的。这样我们需要对每个未知像素的邻接像素进行判断，从而增加了创建矩阵A的难度。既然使用稀疏矩阵求解方程组，因此不在乎再多一些未知数。如图11-1(右)所示，我们将指定的区域使用膨胀运算扩展一圈像素。图中浅灰色是原始的混合区域，深灰色为浅灰色区域膨胀后增加的区域。我们把灰色区域内的所有像素都当作未知数，其中浅灰色像素对应的方程由拉普拉斯运算决定：

$$\begin{aligned} -4 \times x_4 + x_0 + x_3 + x_5 + x_9 &= -7 \\ -4 \times x_5 + x_1 + x_4 + x_6 + x_{10} &= -32 \\ \dots \end{aligned}$$

而深灰色像素对应的方程就是对应的目标图像中的值：

```
x0 = 85
x1 = 79
x2 = 78
...
```

只需要对上述两种方程编写程序即可创建稀疏矩阵A。

11.1.2 编写代码

我们要编写的函数的参数如下：

```
poisson_blending(src, dst, src_mask, offset_x, offset_y,
mix=False)
```

其中src和dst分别为源图像和目标图像，它们的形状不一定相同。src_mask是宽和高与src相同的二维数组，其中不为零的元素指定混合的区域。offset_x和offset_y为目标图像中指定区域和源图像区域之间的坐标差值，通过这两个参数可以调整目标图像中混合区域的位置。mix参数指定混合方式，稍后再进行详细解释。

首先需要解决的问题是计算源图像中src_mask指定区域通过拉普拉斯算子运算之后的值：

```
import cv2

offset_x, offset_y = -36, 42

src = cv2.imread("vinci_src.png", 1)
```

```

dst = cv2.imread("vinci_target.png", 1)
mask = cv2.imread("vinci_mask.png", 0)
src_mask = (mask > 128).astype(np.uint8)

src_y, src_x = np.where(src_mask) ❶
    src_laplacian = cv2.Laplacian(src, cv2.CV_16S, ksize=1)
[src_y, src_x, :] ❷

```

❶由于指定区域不一定为矩形，我们用numpy.where()获得区域中各像素的下标，这样得到的src_laplacian数组是一个二维数组，它的第0轴的长度为区域内像素的个数，第1轴的长度为3，分别与三个颜色通道对应。

❷由于Laplacian()计算的结果中存在负数，因此这里通过它的第二个参数ddepth指定结果数组的元素类型，这里指定它为CV_16S，表示结果为一个16位的符号整数数组。接下来计算目标图像中的区域信息：

```

dst_mask = np.zeros(dst.shape[:2], np.uint8)
dst_x, dst_y = src_x + offset_x, src_y + offset_y
dst_mask[dst_y, dst_x] = 1 ❶

    kernel = np.array([[0,1,0],[1,1,1],[0,1,0]],
dtype=np.uint8)

dst_mask2 = cv2.dilate(dst_mask, kernel=kernel) ❷

dst_y2, dst_x2 = np.where(dst_mask2) ❸
dst_ye, dst_xe = np.where(dst_mask2 - dst_mask) ❹

```

❶由于遮罩图像在目标图像和源图像中存在位置偏移，因此先创建目标图像的一个遮罩数组dst_mask，并将偏移之后的下标中对应的元素设置为1。❷将目标遮罩数组进行膨胀处理得到dst_mask2。❸计算dst_mask2中1对应的坐标(dst_x2, dst_y2)，❹计算膨胀处理新增加的像素的坐标(dst_xe, dst_ye)，即目标区域的边缘像素的坐标。

对照图11-1(右)可知，(dst_x2, dst_y2)为所有灰色区域的像素坐标，(dst_x, dst_y)为浅灰色区域的像素坐标，(dst_xe, dst_ye)为深灰色区域的像素坐标。

```
variable_count = len(dst_x2)
variable_index = np.arange(variable_count) ❶

variables = np.zeros(dst.shape[:2], np.int)
variables[dst_y2, dst_x2] = variable_index

x0 = variables[dst_y , dst_x ] ❷
x1 = variables[dst_y-1, dst_x ]
x2 = variables[dst_y+1, dst_x ]
x3 = variables[dst_y , dst_x-1]
x4 = variables[dst_y , dst_x+1]
x_edge = variables[dst_ye, dst_xe] ❸
```

❶(dst_x2, dst_y2)中的每个像素都与一个未知数对应，我们为这些未知数编上序号variable_index，❷并得到(dst_x, dst_y)中每个未知数及其上下左右4个相邻的未知数的序号：x0、x1、x2、x3、x4，❸而x_edge则是与(dst_xe, dst_ye)中的未知数对应的序号。

接下来计算线性方程组中各个未知数的系数矩阵A:

```
from scipy.sparse import coo_matrix

inner_count = len(x0)
edge_count = len(x_edge)

r = np.r_[x0, x0, x0, x0, x0, x_edge]
c = np.r_[x0, x1, x2, x3, x4, x_edge]
v = np.ones(inner_count*5 + edge_count)
v[:inner_count] = -4

A = coo_matrix((v, (r, c))).tocsc()
```

与 (dst_x, dst_y) 中的每个未知数对应的方程有5个系数，与 (dst_xe, dst_ye) 中的未知数对应的方程有1个系数，因此最终的系数矩阵中有 $\text{inner_count} * 5 + \text{edge_count}$ 个非零值。与 x0 中的每个未知数对应的方程中，x0 的系数为-4，而其上下左右4个未知数的系数为1。r 和 c 保存的是 A 中非零值对应的下标，而 v 中保存的是系数，这些系数中除了下标为 (x0, x0) 的元素为-4 之外，其余的都为1。最后通过 `coo_matrix()` 创建稀疏矩阵，并且转换成求解方程组时使用的 CSC 格式。

```
from scipy.sparse.linalg import spsolve

order = np.argsort(np.r_[variables[dst_y, dst_x],
variables[dst_ye, dst_xe]]) ❶

result = dst.copy()
```

```

    for ch in (0, 1, 2): ❷
        b = np.r_[src_laplacian[:,ch], dst[dst_ye, dst_xe, ch]]
❸

        u = spsolve(A, b[order]) ❹
        u = np.clip(u, 0, 255)
        result[dst_y2, dst_x2, ch] = u ❺

```

❶由于方程组是按照未知数的下标顺序排列的，因此我们计算方程组的b时也需要按照未知数的下标排列。但是为了方便计算b，我们按照(dst_x, dst_y)和(dst_xe, dst_ye)的顺序来计算，因此需要事先计算order用于对常数项b中的值进行排序。❷对三个通道的数组进行循环，❸计算常数项b，其中与(dst_x, dst_y)对应的未知数的方程的常数项从源图像的拉普拉斯算子的输出数组获得，而与(dst_xe, dst_ye)对应的未知数的方程的常数项则为目标图像对应的像素值。❹调用spsolve()对线性方程组进行求解，❺最后将未知数的解写到结果数组中，(dst_x2, dst_y2)中的未知数是按照下标顺序排列的。

图11-2(右)为泊松混合的计算结果。

```

fig, axes = plt.subplots(1, 4, figsize=(10, 4))
ax1, ax2, ax3, ax4 = axes.ravel()
ax1.imshow(src[:, :, :-1])
ax2.imshow(mask, cmap="gray")
ax3.imshow(dst[:, :, :-1])
ax4.imshow(result[:, :, :-1])

for ax in axes.ravel():
    ax.axis("off")

```

```
fig.subplots_adjust(wspace=0.05)
```

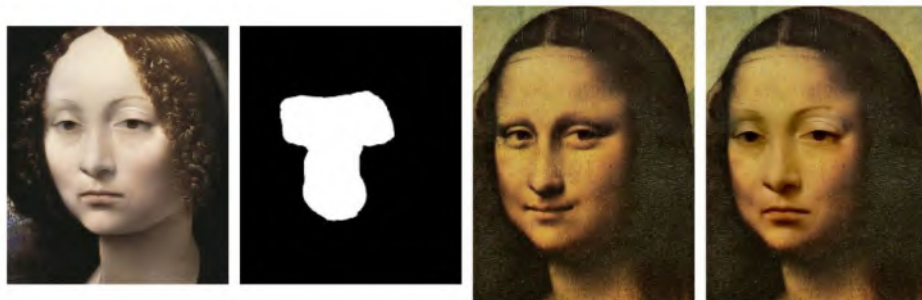


图11-2 使用泊松混合算法将吉内薇拉·班琪肖像中的眼睛和鼻子部分复制到蒙娜丽莎的肖像之上

11.1.3 演示程序

为了方便读者观察泊松混合的效果，本书提供了一个使用 TraitsUI 编写的泊松混合演示程序，界面如图11-3所示。该界面的用法如下：

- 首先选择两幅图像的路径，然后按Load按钮载入图像。
- 使用鼠标左键在图像上绘制混合区域，可以使用鼠标滚轴修改画笔的粗细。按住鼠标右键可以移动混合区域。
- 按Mix按钮进行泊松混合，将左图中区域的内容混合到右图中对应的位置。为了显示混合效果，右图中的区域将自动隐藏，单击可重新显示区域。



scipy2.examples.possion

:

使用 TraitsUI 编写的泊松混合演示程序。该程序使用 `scipy2.matplotlib.freedraw_widget` 中提供的 `ImageMaskDrawer` 在图像上绘制半透明的白色区域。

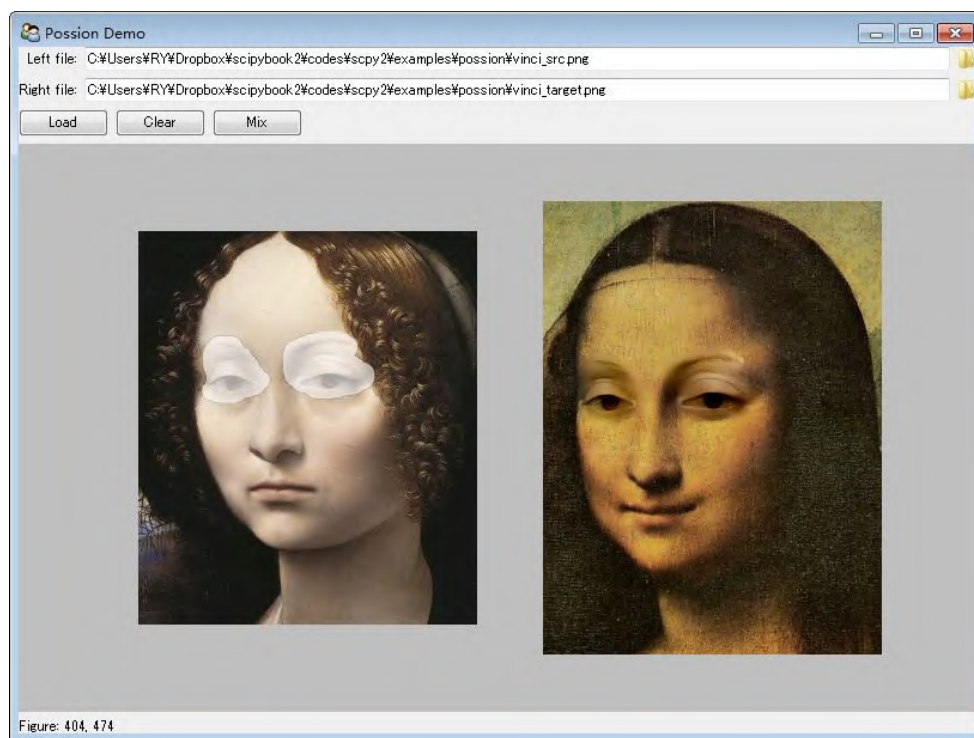


图11-3 泊松混合演示程序的界面截图

11.2 经典力学模拟



与本节内容对应的Notebook为：11-examples/examples-200-physics-simulation.ipynb。

本节以悬链线、最速降线和单摆为例介绍如何使用Scipy中的integrate和optimize库模拟简单的经典力学现象。

11.2.1 悬链线

将绳子的两端固定在同一水平高度，绳子因重力作用而垂下所形成的形状被称为悬链线，它的曲线函数为：

$$y = a \cosh \frac{x}{a}$$

其中a为决定下垂程度的系数，下面的catenary()函数对悬链线方程进行坐标平移，使得它经过(0, 0)和(1, 0)两个点，效果如图11-4所示：

```
def catenary(x, a):  
    return a*np.cosh((x - 0.5)/a) - a*np.cosh((-0.5)/a)  
  
x = np.linspace(0, 1, 100)  
for a in [0.35, 0.5, 0.8]:  
    pl.plot(x, catenary(x, a), label="$a={:g}$".format(a))  
ax = pl.gca()  
ax.set_aspect("equal")  
ax.legend(loc="best")  
pl.margins(0.1)
```

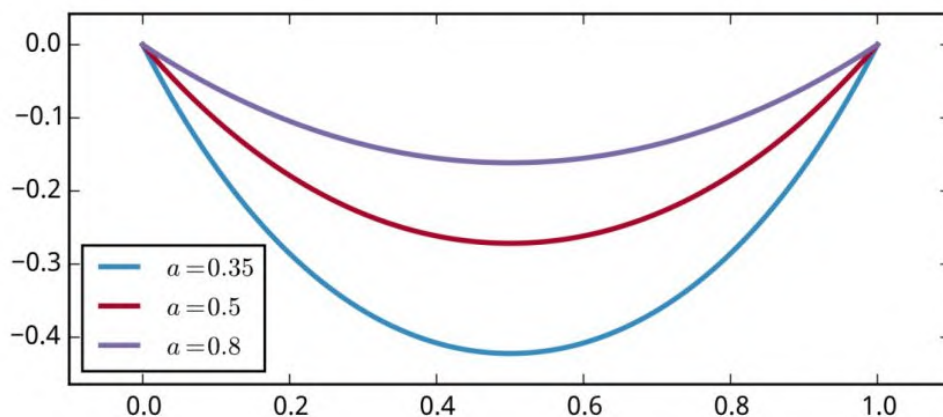


图11-4 各种长度的悬链线

曲线的长度可以使用如下定积分来计算：

$$s = \int_0^1 \sqrt{1 + \left(\frac{dy}{dx}\right)^2} dx$$

下面我们先利用前面的catenary()函数大致计算曲线的长度：

```
y = catenary(x, 0.35)
np.sqrt(np.diff(x)**2 + np.diff(y)**2).sum()
1.37655226488
```

定积分中的积分项可以使用SymPy进行符号运算，并使用lambdify()将最终的表达式转换成函数，然后使用integrate.quad()进行定积分运算：

```
from sympy import symbols, cosh, S, sqrt, lambdify
import sympy
x, a = symbols("x, a")
y = a * cosh((x - S(1)/2) / a)
```

```

s = sqrt(1 + y.diff(x)**2)
fs = lambdify((x, a), s, modules="math")

def catenary_length(a):
    return integrate.quad(lambda x:fs(x, a), 0, 1)[0]

length = catenary_length(0.35)
length
1.3765789965

```

1. 使用运动方程模拟悬链线

为了使用牛顿力学中的运动方程模拟悬链线，我们可以把悬链看成由多个弹簧连接的质点系统，每个质点受到重力以及左右两个弹簧力。当质点运动时，它还会受到大小与速度成正比的阻力。为了使悬链两端的质点保持静止，可以不计算其受力，因此这两个质点的加速度始终为0。每个质点有X和Y方向上的速度与加速度共4个状态，对于由N个质点构成的系统，共有 $4 \times N$ 个状态。

下面的`diff_status(status, t)`计算状态为`status`时的微分，然后使用`odeint()`对该系统进行积分，即可计算该系统在不同时刻的状态。当时间足够长时，由于阻力作用，各个质点最终会处于平衡位置，效果如图11-5所示。

```

N = 31

dump = 0.2 #阻尼系数

```

```

k = 100.0    #弹簧系数
l = length / (N - 1) #弹簧原长度
g = 0.01    #重力加速度

x0 = np.linspace(0, 1, N)
y0 = np.zeros_like(x0)
vx0 = np.zeros_like(x0)
vy0 = np.zeros_like(x0)

def diff_status(status, t):
    x, y, vx, vy = status.reshape(4, -1)
    dvx = np.zeros_like(x)
    dvy = np.zeros_like(x)
    dx = vx
    dy = vy

    s = np.s_[1:-1]

    l1 = np.sqrt((x[s] - x[:-2])**2 + (y[s] - y[:-2])**2)
    l2 = np.sqrt((x[s] - x[2:])**2 + (y[s] - y[2:])**2)
    dl1 = (l1 - l) / l1
    dl2 = (l2 - l) / l2

    dvx[s] = -vx[s] * dump - (x[s] - x[:-2]) * k * dl1 -
(x[s] - x[2:]) * k * dl2
    dvy[s] = -vy[s] * dump - (y[s] - y[:-2]) * k * dl1 -
(y[s] - y[2:]) * k * dl2 + g
    return np.r_[dx, dy, dvx, dvy]

```

```

status0 = np.r_[x0, y0, vx0, vy0]

t = np.linspace(0, 50, 100)
r = integrate.odeint(diff_status, status0, t)
x, y, vx, vy = r[-1].reshape(4, -1)

r, e = optimize.curve_fit(catenary, x, -y, [1])
print "a =", r[0], "length =", catenary_length(r[0])

x2 = np.linspace(0, 1, 100)
pl.plot(x2, catenary(x2, 0.35))
pl.plot(x2, catenary(x2, r))
pl.plot(x, -y, "o")
pl.margins(0.1)
a = 0.336992602016 length = 1.40946777721

```

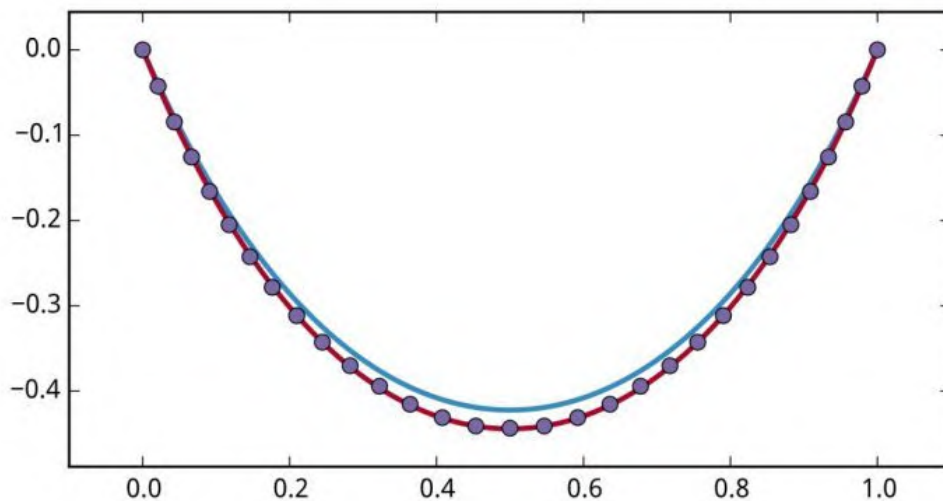


图11-5 使用运动方程模拟悬链线，由于弹簧会被拉伸，因此悬链线略比原始长度长



scpy2.examples.catenary: 使用TraitsUI制作的悬链线的动画演示程序, 可通过界面修改各个参数。

在图11-5中, 圆点表示各个质点的最终位置, 红色曲线为使用悬链线方程对质点位置进行拟合后得到的最佳拟合悬链线, 而蓝色曲线为弹簧保持原长时的悬链线。为了使得最终状态接近原长时的悬链线, 需要尽量大的弹簧系数和尽量小的重力加速度, 这样能保证每根弹簧接近原长。读者可以试着修改前面的系数, 使最终状态尽量接近蓝色曲线。

2. 通过能量最小值计算悬链线

当质点之间为刚性连接时, 弹簧不存储任何弹性势能, 重力使得整个系统的重力势能降为最低, 因此可以通过最小化势能计算各个质点的最终状态。由于悬链线的两端固定, 而质点之间的距离固定, 因此该最小化问题带有许多约束条件。为了尽量减少约束条件, 我们以每个连接杆的角度为变量表示整条悬链线的状态。悬链线的一端固定在 $(0, 0)$ 处, 经过每个连接杆最终到达坐标 $(1, 0)$ 处。因此满足如下两个约束条件, 其中 θ_i 为每根杆的方向, l 为杆的长度。整个系统如图11-6所示。

$$\sum l \cos \theta_i = 1, \sum l \sin \theta_i = 0$$

第 i 个质点的 Y 轴位置为:

$$y_i = \sum_{k=0}^i l \sin \theta_k$$

而势能P可以用下式表示：

$$P = \sum y_i$$

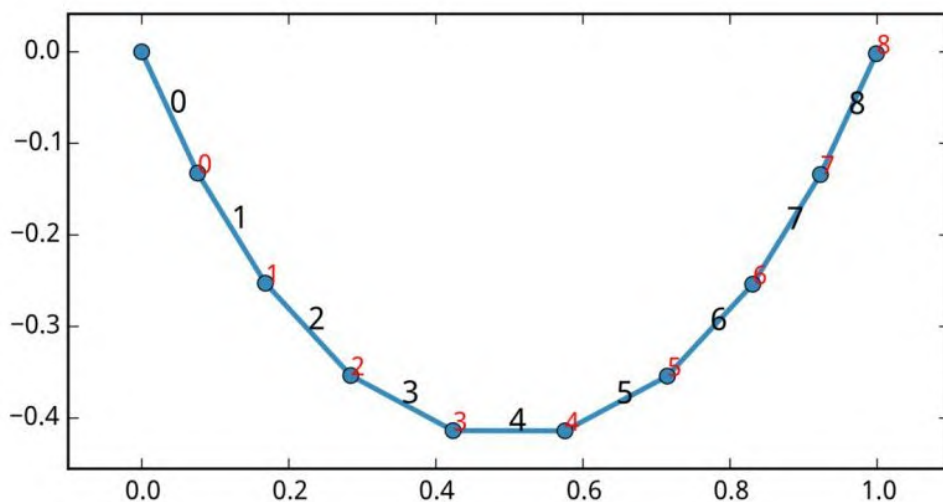


图11-6 把悬链线分为多个质点并用无质量的连接杆相连

因此最小化的问题就是找到一组 θ_i ，它们满足两个约束条件，并且使得P最小， θ_i 的取值范围为 $-\frac{\pi}{2} < \theta_i < \frac{\pi}{2}$ ($-\pi)/2 < \theta_i < \pi/2$ 。这种带等式约束条件的最小化问题可以使用 `scipy.optimize.fmin_slsqp()` 进行求解。

在下面的程序中，`g1()` 计算最右端点的横坐标需要满足的条件，`g2()` 为纵坐标需要满足的条件，这两个函数返回0时，表示满足约束条件。`P(theta)` 计算状态为 `theta` 时的势能。❶为了提高优化的计算速度，我们让初始值满足两个约束条件，如图11-7中的叉点所示。❷`fmin_slsqp()` 的 `eqcons` 参数是计算等式约束条件的函数列表。

3

```
Optimization terminated successfully.      (Exit mode 0)
      Current function value: -7.76529946378
      Iterations: 9
      Function evaluations: 288
      Gradient evaluations: 9
```

可以看到只迭代了9次就找到了最优解，下面根据 θ_i 计算出每个质点的位置，如图11-7所示。图中蓝色曲线为悬链线方程所得的理论值。

```
x_init = np.r_[0, np.cumsum(l * np.cos(theta_init))]
y_init = np.r_[0, np.cumsum(l * np.sin(theta_init))]

x = np.r_[0, np.cumsum(l * np.cos(theta))]
y = np.r_[0, np.cumsum(l * np.sin(theta))]

x2 = np.linspace(0, 1, 100)
pl.plot(x2, catenary(x2, 0.35))
pl.plot(x, y, "o")
pl.plot(x_init, y_init, "x")
pl.margins(0.1)
```

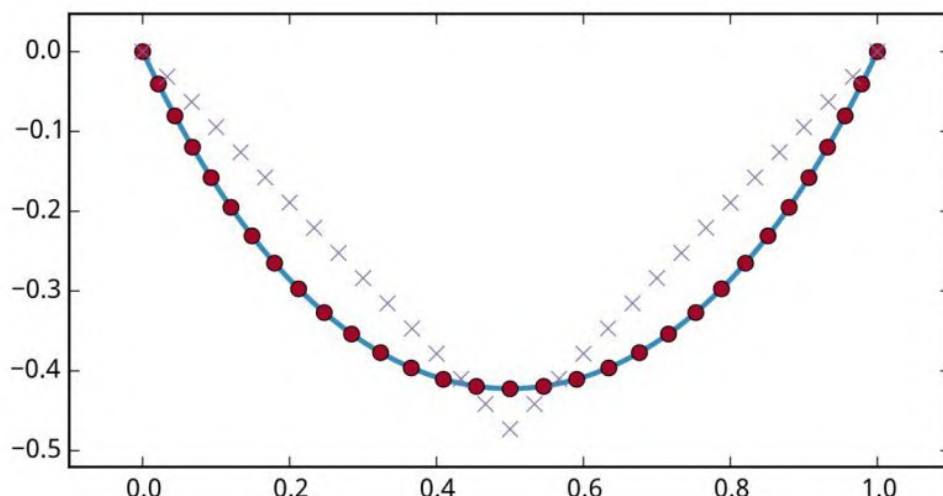


图11-7 使用fmin_slsqp()计算能量最低的状态，叉点表示最优化的初始状态

11.2.2 最速降线

所谓最速降线问题，是指在两点之间建立一条无摩擦的轨道，使得小球从高点到低点所需的时间最短。考虑两点高度相同的极端情况，显然这条曲线不是直线。根据维基百科的相关介绍，下降高度为D的最速降线满足如下方程：

$$dx = \sqrt{\frac{y}{D-y}} dy$$

由公式可知，y的取值范围为0到D之间。下面先用数值定积分计算D为1时，最速降线终点的X轴坐标：

```
x, _ = integrate.quad(lambda y:np.sqrt(y/(1.0-y)), 0, 1)
print x
1.57079632679
```

可以看出曲线终点和起点之间X轴的差为 $\pi/2$ 。

1. 使用odeint() 计算最速降线

使用odeint() 对下面的微分方程进行积分即可得到最速降线的曲线：

$$\frac{dy}{dx} = \sqrt{\frac{D-y}{y}}$$

下面是计算最速降线的程序，效果如图11-8所示。当 $y=0$ 时，曲线的切线为垂直方向， dy/dx 为无穷大。❶限制 y 的值必须大于极小值 eps 并小于 D ，这样才能保证积分能正常进行。

❷使用微分方程计算的曲线只是左半边的曲线，完整的曲线相对于 $x=D\pi/2$ 对称。为了odeint()能计算完整的曲线，需要根据 x 的值判断 $\frac{dy}{dx}$ 的符号。

```
def brachistochrone_curve(D, N=1000):  
    eps = 1e-8  
    def f(y, x):  
        y = min(max(eps, y), D) ❶  
        flag = -1 if x >= D * np.pi / 2 else 1 ❷  
        return flag * ((D - y) / y) ** 0.5  
  
    x0 = np.linspace(0, D * np.pi, N)
```

```
y = integrate.odeint(f, 0, x0)
return x0, y.ravel()

x, y = brachistochrone_curve(2.0)
pl.plot(x, -y)
```

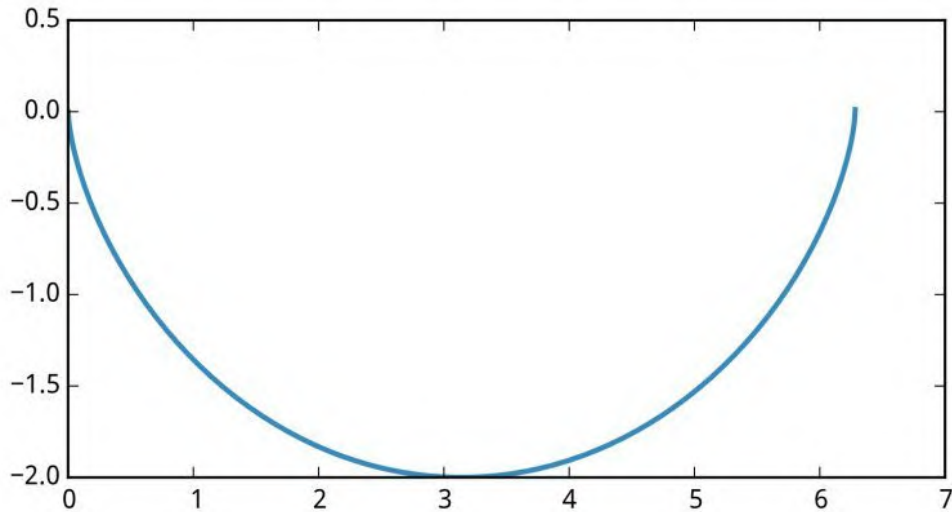


图11-8 使用odeint()计算最速降线

2. 使用优化算法计算最速降线

可以使用优化算法近似计算最速降线。首先将X轴从0到target等分为N-1份，每个点对应的Y轴坐标y就是要优化的自变量。优化的目标是计算小球经过该曲线所需的时间。根据能量守恒定理，可以计算出小球到达每个坐标点时的速度为 $v = \sqrt{2gy}$ 。

小球经过每条线段的速度按照两个端点速度的平均值计算，线段的长度根据两个端点的距离计算，由此可以得出小球经过每条线段的

时间。优化的目标就是所有这些时间之和最小，即小球通过整条曲线的时间最短。

下面是使用优化算法计算最速降线的程序，计算结果如图11-9所示。由图可知最优化的结果和积分运算的结果相同。

```
N = 100.0
target = 10.0
x = np.linspace(0, target, N)
tmp = np.linspace(0, -1, N // 2)
y0 = np.r_[tmp, tmp[::-1]]
g = 9.8

def total_time(y):
    s = np.hypot(np.diff(x), np.diff(y)) ❶
    v = np.sqrt(2 * g * np.abs(y)) ❷
    avg_v = np.maximum((v[1:] + v[:-1])*0.5, 1e-10) ❸
    t = s / avg_v
    return t.sum()

def fix_two_ends(y):
    return y[[0, -1]]

y_opt = optimize.fmin_slsqp(total_time, y0, eqcons=
[fix_two_ends]) ❹

pl.plot(x, y0, "k--", label=u"初始值")
pl.plot(x, y_opt, label=u"优化结果")
```

```

x2, y2 = brachistochrone_curve(target / np.pi)
pl.plot(x2, -y2, label=u"最速降线")
pl.legend(loc="best")

Optimization terminated successfully.      (Exit mode 0)

Current function value: 2.53634462725

Iterations: 72

Function evaluations: 7370

Gradient evaluations: 72

```

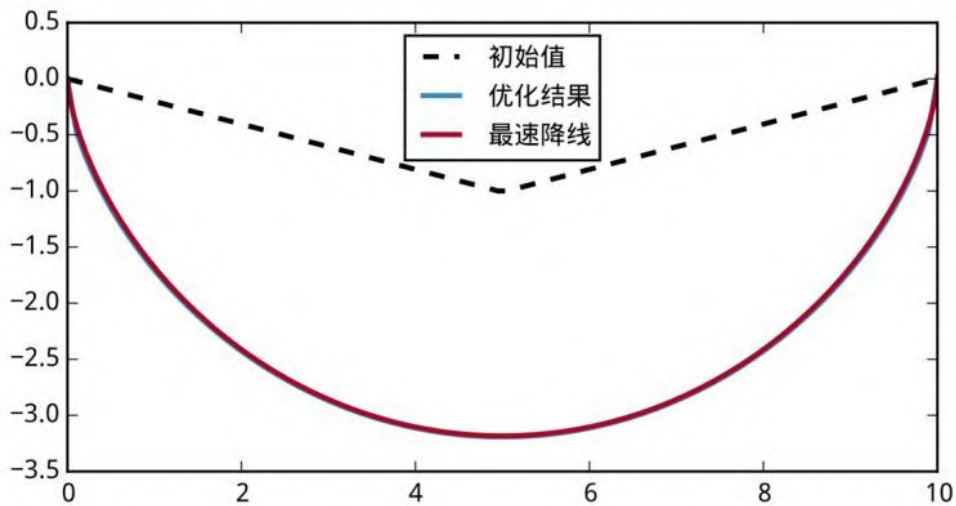


图11-9 使用优化算法计算最速降线

❶调用`hypot()` 计算每条线段的长度，❷使用能量守恒公式计算小球到达每点的速度，❸计算每个线段的平均速度，为了防止平均速度为0导致无法计算时间，这里使用`maximum()`将速度的下限设置为 10^{-10} 。

❹由于需要保证曲线两个端点的Y轴坐标为0，因此我们使用`fmin_slsqp()` 优化函数，并用`fix_two_end()` 保证两个端点的Y轴坐标始终为0。

11.2.3 单摆模拟

如图11-10所示，由一根不可伸长、质量不计的绳子，上端固定，下端系一质点，这样的装置叫作单摆。

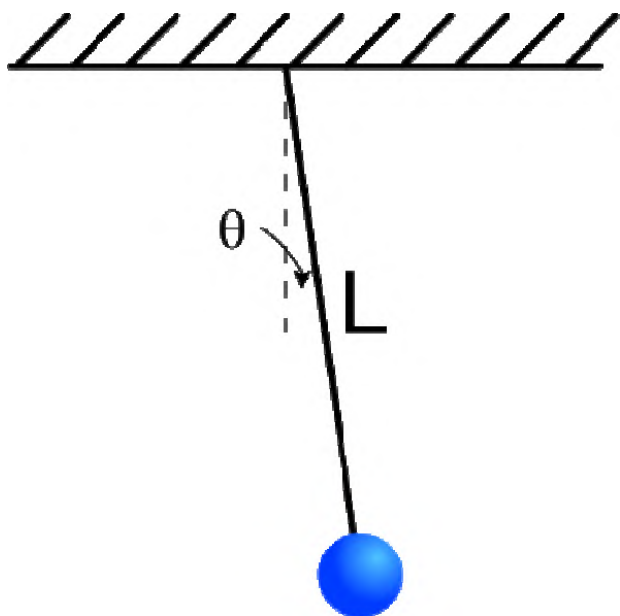


图11-10 单摆装置示意图

根据牛顿力学定律，可以列出如下微分方程：

$$\frac{d^2\theta}{dt^2} + \frac{g}{\ell} \sin\theta = 0$$

其中 θ 为单摆的摆角， l 为单摆的长度， g 为重力加速度。此微分方程的符号解无法直接求出，因此只能调用 `odeint()` 对其求数值解。

`odeint()` 要求每个微分方程只包含一阶导数，因此我们需要对上面的微分方程做如下变形：

$$\frac{d\theta(t)}{dt} = v(t), \quad \frac{dv(t)}{dt} = -\frac{g}{\ell} \sin\theta(t)$$

下面是利用odeint()计算单摆轨迹的程序，摆角和时间的关系如图11-11所示：

```
from math import sin

g = 9.8

def pendulum_equations(w, t, l):
    th, v = w
    dth = v
    dv = - g / l * sin(th)
    return dth, dv

t = np.arange(0, 10, 0.01)
track = integrate.odeint(pendulum_equations, (1.0, 0), t,
args=(1.0,))

pl.plot(t, track[:, 0])
pl.xlabel(u"时间 (秒)")
pl.ylabel(u"振动角度 (弧度)")
```

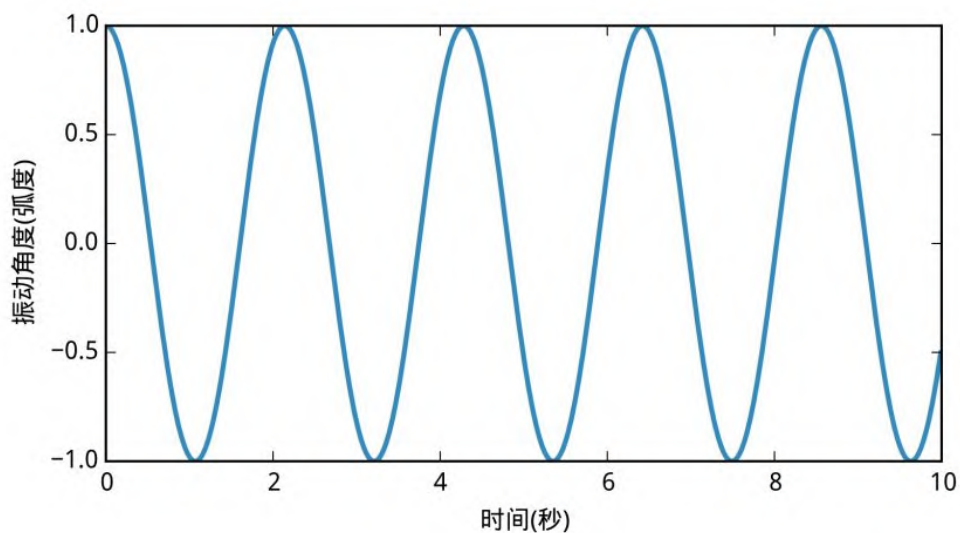



图11-11 初始角度为1弧度的单摆摆动角度和时间的关系

1. 小角度时的摆动周期

高中物理课介绍过当最大摆动角度很小时，单摆的摆动周期可以使用如下公式计算：

$$T_0 = 2\pi \sqrt{\frac{\ell}{g}}$$

这是因为当 $\theta \ll 1$ 时， $\sin \theta \approx \theta$ ，这样微分方程就变成了：

$$\frac{d^2\theta}{dt^2} + \frac{g}{\ell}\theta = 0$$

此微分方程的解是一个简谐振动方程，很容易计算其摆动周期。下面我们用SymPy对这个微分方程进行符号求解：

```

from sympy import symbols, Function, dsolve

t, g, l = symbols("t,g,l", positive=True) # 分别表示时间、重力
加速度和长度

y = Function("y") # 摆角函数用y(t)表示
dsolve(y(t).diff(t,2) + g/l*y(t), y(t))

```

$$y(t) = C_1 \sin\left(\frac{\sqrt{g}t}{\sqrt{l}}\right) + C_2 \cos\left(\frac{\sqrt{g}t}{\sqrt{l}}\right)$$

可以看到简谐振动方程的解是由两个频率相同的三角函数构成的，周期为 $2\pi\sqrt{\frac{l}{g}}$ 。

2. 大角度时的摆动周期

但是当初始摆角增大时，上述近似处理会带来无法忽视的误差。下面让我们看看如何用数值计算的方法求出单摆在任意初始摆角时的摆动周期。

```

g = 9.8

def pendulum_th(t, l, th0):
    track = integrate.odeint(pendulum_equations, (th0, 0),
[0, t], args=(l,))
    return track[-1, 0]

def pendulum_period(l, th0):
    t0 = 2*np.pi*(l / g)**0.5 / 4

```

```
t = fsolve( pendulum_th, t0, args = (l, th0) )  
return t*4
```

要计算摆动周期，只需要计算从最大摆角到0摆角所需的时间，摆动周期是此时间的4倍。为了计算出这个时间值，首先需要定义函数 `pendulum_th()` 来计算长度为 `l`、初始角度为 `th0` 的单摆在时刻 `t` 的摆角：

```
def pendulum_th(t, l, th0):  
    track = integrate.odeint(pendulum_equations, (th0, 0),  
[0, t], args=(l,))  
    return track[-1, 0]
```

此函数仍然使用 `odeint()` 进行微分方程组求解，只是我们只需要计算时刻 `t` 的摆角，因此传递给 `odeint()` 的时间序列为 `[0, t]`。`odeint()` 内部会对时间进行细分，保证最终的解是正确的。

接下来只需要找到第一个使 `pendulum_th()` 的结果为0的时间即可。这相当于对 `pendulum_th()` 求解，可以使用 `scipy.optimize.fsolve()` 对这种非线性方程进行求解：

```
from scipy.optimize import fsolve  
  
def pendulum_period(l, th0):  
    t0 = 2*np.pi*(l / g)**0.5 / 4  
    t = fsolve(pendulum_th, t0, args = (l, th0))  
    return t * 4
```

`fsolve()` 求解时需要一个初始值尽量接近真实的解，用小角度单摆的周期的1/4作为这个初始值是一个不错的选择。下面利用 `pendulum_period()` 计算出初始摆动角度从0到90度的摆动周期：

```
ths = np.arange(0, np.pi/2.0, 0.01)
periods = [pendulum_period(1, th) for th in ths]
```

为了验证`fsolve()` 求解摆动周期的正确性，下面的公式是从维基百科中找到的摆动周期的精确解：

$$T = 4 \sqrt{\frac{\ell}{g}} K\left(\sin \frac{\theta_0}{2}\right)$$

其中的函数K为第一类完全椭圆积分函数，定义如下：

$$K(k) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1 - k^2 \sin^2 \theta}}$$

可以用`scipy.special.ellipk()` 计算此函数的值：

```
from scipy.special import ellipk
periods2 = 4 * (1.0 / g)**0.5 * ellipk(np.sin(th / 2) **2
)
```

图11-12比较两种计算方法，可以看到结果是完全一致的：

```
ths = np.arange(0, np.pi/2.0, 0.01)
periods = [pendulum_period(1, th) for th in ths]
periods2 = 4 * (1.0/g)**0.5 * ellipk(np.sin(th/2)**2) # 计算
单摆周期的精确值

pl.plot(th, periods, label = u"fsolve 计算的单摆周期",
```

```
linewidth=4.0)

    pl.plot(ths, periods2, "r", label = u"单摆周期精确值",
linewidth=2.0)

    pl.legend(loc='upper left')

    pl.xlabel(u"初始摆角 (弧度)")

    pl.ylabel(u"摆动周期 (秒)")
```

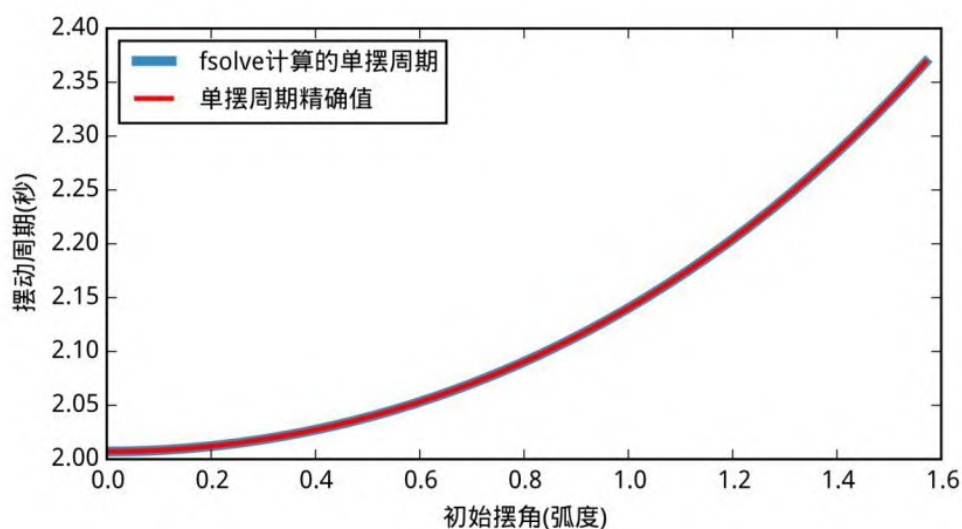


图11-12 单摆的摆动周期和初始角度的关系

11.3 推荐算法



与本节内容对应的Notebook为：`11-examples/examples-300-movielens.ipynb`。

推荐算法是指利用用户的一些行为，通过一些数学算法，推测出用户可能喜欢的东西。本节介绍如何通过MovieLens提供的用户对电影

的评分数据进行一些推荐算法方面的研究。

11.3.1 读入数据

movielens.data文件中是以制表符分隔的评分数据，其第一列为用户ID、第二列为电影ID、第三列为用户对电影的5分制评分值。下面用pandas.read_table()读入该数据，并指定列名与列的数据类型。由于通过推荐算法计算的评分值不是整数，因此这里将评分值的数值类型设置为浮点数。原数据中用户ID和电影ID都是从1开始的，为了方便后续的稀疏矩阵表达，将它们都减去1。

```
columns = ['user_id', 'movie_id', 'rating']
dtypes = [np.int32, np.int32, np.float]
ratings = pd.read_table('data/movielens.data',
                        names=columns,
                        usecols=[0, 1, 2],
                        dtype=dict(zip(columns, dtypes)))

ratings["user_id"] -= 1
ratings["movie_id"] -= 1
```

下面将评分次数少于10的用户和电影的评分删除，这有助于推荐算法将运算集中在有足够数据可以分析的用户和电影之上：

```
u, v, r = ratings.user_id.values, ratings.movie_id.values,
ratings.rating.values

#删除评分数少于10的用户和电影
```

```

u_count = pd.value_counts(u)
v_count = pd.value_counts(v)

mask = (u_count >= 10)[u].values & (v_count >= 10)
[v].values

u, v, r = u[mask], v[mask], r[mask]

```

为了评价推荐算法的性能，需要将评分数据分为训练用数据和测试用数据两部分。这样才能防止推荐算法对训练用数据过度学习造成实际的性能下降。下面的train_test_split()完成这个任务，其test_size参数为测试用数据占全部数据的比例。这里通过产生一个0到1之间的随机数数组对数据进行分组，得到6个一维数组：u_train、v_train、r_train、u_test、v_test、r_test、它们分别为训练用的用户ID、训练用的电影ID、训练用的电影评分、测试用的用户ID、测试用的电影ID和测试用的电影评分。

```

def train_test_split(arrays, test_size=0.1):
    np.random.seed(0)
    mask_test = np.random.rand(len(arrays[0])) < test_size
    mask_train = ~mask_test

    arrays_train = [arr[mask_train] for arr in arrays]
    arrays_test = [arr[mask_test] for arr in arrays]

    return arrays_train + arrays_test

u_train, v_train, r_train, u_test, v_test, r_test =
train_test_split([u, v, r])

```

```
nu = np.max(u) + 1
nv = np.max(v) + 1
nr = len(u_train)
```

下面检测学习用的数据包含所有的电影和用户，如果下面的程序抛出异常，则建议修改seed()的系数：

```
assert np.all(np.unique(u_train) == np.unique(u))
assert np.all(np.unique(v_train) == np.unique(v))
```

11.3.2 推荐性能评价标准

在讨论具体的推荐算法之前，还需要明确如何评价推荐性能。下面是两个常用的性能评价公式：

$$\text{RMSE} = \sqrt{\frac{\sum (y_t - y_p)^2}{N}}$$
$$r^2 = 1 - \frac{\sum (y_t - y_p)^2}{\sum (y_t - \mu(y_t))^2}$$

其中RMSE的值越接近0性能越好，而 r^2 的值通常在0到1之间，越接近1越好。

```
def rmse_score(y_true, y_pred):
    d = y_true - y_pred
```



```

        return np.mean(d**2)**0.5

def r2_score(y_true, y_pred):
    d = y_true - y_pred
    return 1 - (d**2).sum() / ((y_true -
y_true.mean())**2).sum()

```

下面用这两个评价标准对每个电影的平均评分值的预测性能进行评价。通过Pandas的groupby对评分值按照电影ID进行分组，并计算每组的评分值。测试用数据中每部电影的预测评分值为r_pred，将之与测试数据中的实际评分值进行比较，计算RMSE和 r^2 ：

```

movies_mean = pd.Series(r_train).groupby(v_train).mean()
r_pred = movies_mean[v_test]

rmse_avg, r2_avg = rmse_score(r_test, r_pred),
r2_score(r_test, r_pred)

rmse_avg          r2_avg
-----
1.0122857327818662  0.17722524926979077

```

通过计算每个电影的平均评分值，可以为所有用户推荐整体最受欢迎的电影，但是无法根据每位用户的喜好推荐他最可能喜欢的电影。为了实现为不同的用户推荐不同的电影，需要对电影评分进行矩阵分解。

11.3.3 矩阵分解

准备好测试数据以及评价标准之后，下面正式开始实现推荐算法。该算法应该能从训练数据中找到更多的规律，使得其对于测试数据的推荐性能超过上面以均值为预测值的性能。

可以把用户*i*对电影*j*的评分 r_{ij} 按照下面的公式分解成4部分的和：

$$r_{ij} = \mu + u_i + v_j + \sum_{k=1}^K u_{ik} v_{jk}$$

其中 μ 是一个标量，它是所有评分的平均值。 u 是一个矢量，其长度为 N_u ， u_i 为用户*i*的评分系数。 v 是一个矢量，其长度为 N_v ， v_j 为电影*j*的评分系数。这里 N_u 表示用户数， N_v 表示电影数。

U 是一个与用户对应的矩阵，其大小为 $N_u \times K$ 。 V 是一个与电影对应的矩阵，其大小为 $N_v \times K$ 。这里*K*的长度可任意指定。计算*U*的第*i*行与*V*的第*j*行上对应元素的乘积和，就得到了用户*i*对电影*j*的评分系数。可以这样理解*U*和*V*：每个电影都用*K*个属性进行描述(*V*)，而每个用户都对这*K*个属性有不同的喜好权值(*U*)。这两组值的乘积和就是某个用户对某部电影的喜好程度。

下面先计算 u_i 和 v_j ，我们希望找到一组解最符合如下方程组：

$$r_{ij} = \mu + u_i + v_j$$

可以看出这是一个最小二乘法的问题： $\mathbf{u}$ 和 $\mathbf{v}$ 中有 $N_u + N_v$ 个未知数，它们需要满足 N_r 个方程，这里用 N_r 表示评分数。最小二乘法中的矩阵 $\mathbf{A}$ 的大小为 $N_r \times (N_u + N_v)$ ，这是一个非常大的矩阵，而其中绝大部分的元素都为0，实际上 $\mathbf{A}$ 的每行中只有两个不为零的元素，它们的值都为1。因此需要使用稀疏矩阵表示矩阵 $\mathbf{A}$ ：

```
from scipy import sparse
from scipy.sparse import linalg

r_avg = r_train.mean()

row_idx = np.r_[np.arange(nr), np.arange(nr)]
col_idx = np.r_[u_train, v_train + nu]
values = np.ones_like(row_idx)

A = sparse.coo_matrix((values, (row_idx, col_idx)), shape=(nr, nu+nv))
```

矩阵 $\mathbf{A}$ 中每行有两个1，其余的值都为0。我们采用 `coo_matrix()` 创建该稀疏矩阵，`row_idx` 为每个非零元素所在的行，`col_idx` 为非零元素所在的列，`values` 是这些非零元素的值。它们都是长度为 $2N_r$ 的一维数组。

下面使用 `scipy.sparse.linalg` 中的 `lsqr()` 进行最小二乘法求解。得到一组解 $\mathbf{x}$ ，其中前 N_u 个元素为 $\mathbf{u}$ ，后 N_v 个元素为 $\mathbf{v}$ 。

```
x = linalg.lsqr(A, r_train - r_avg)[0]

ub = x[:nu]

vb = x[nu:]
```

下面按照公式 $r_{ij} = \mu + u_i + v_j$ 计算测试数据的预测评分，并与实际评分进行比较：

```
r_pred = r_avg + ub[u_test] + vb[v_test]

rmse, r2 = rmse_score(r_test, r_pred), r2_score(r_test,
r_pred)

                rmse                r2
-----
0.93056259728693425  0.30471011860212072
```

接下来我们将注意力集中到计算U 和V 的矩阵分解算法上。矩阵分解的目标是尽量接近下面的r_train2：

```
r_train2 = r_train - (r_avg + ub[u_train] + vb[v_train])
r_test2  = r_test  - r_pred

#以下程序从该array元组获取数据

arrays = u_train, v_train, r_train2, u_test, v_test,
r_test2
```

11.3.4 使用最小二乘法实现矩阵分解

将 r_{train2} 分解为两个矩阵 U 和 V 的乘积也是一个最小化的问题。矩阵 U 和 V 中共有 $N_u \cdot K + N_v \cdot K$ 个未知数，而最小化的目标方程有 N_r 个。下面是与第 i 个用户对第 j 个电影的评分值对应的方程：

$$U_{i0} \cdot V_{j0} + U_{i1} \cdot V_{j1} + \dots = r_{ij}$$

由于每个方程都包含 U 和 V 中未知数的乘积，因此它不是一个线性方程组，无法使用前面的最小二乘法函数 $\text{lsqr}()$ 。对于这种方程组的最优化问题，可以假设其中的一部分未知数已知，从而将其转换成线性方程组。

如果 V 已知，那么 U 为未知数，个数为 $N_u \cdot K$ 。可以使用最小二乘法对这些未知数进行求解。这时最小二乘法中的矩阵 A 的大小为 $N_r \times N_u \cdot K$ 。同样若 U 已知，则 V 为未知数，个数为 $N_v \cdot K$ ，这时最小二乘法的矩阵 A 的大小为 $N_r \times N_v \cdot K$ 。

具体的运算步骤如下：

- (1) 通过随机数产生 U 和 V 。
- (2) 假设 V 为已知数，使用最小二乘法对 U 进行求解。使用新的解更新 U 。
- (3) 假设 U 为已知数，使用最小二乘法对 V 进行求解。使用新的解更新 V 。
- (4) 使用 U 和 V 对测试数据进行推荐性能评价。

(5) 转到步骤(2)重复执行，直到推荐性能的评价不再提升。

下面的uv_decompose()实现上述运算步骤，arrays参数为包括训练数据和测试数据的6个数组。其他参数均用于控制训练的参数。❶通过随机数产生U和V，为了保证这两个矩阵相乘后得到的结果大小基本一致，需要以 $\frac{1}{\sqrt{K}}$ 为缩放因子。

❷使用coo_matrix()创建当U为未知数、V为已知数时的系数矩阵 A^v ，并将其转换成csr_matrix()格式的稀疏矩阵。假设电影评分中的第r行对应用户为i、电影为j，则 A^v 中的第r行中不为零的元素为从第j·K列到第j·K+K-1列：

$$A_{r,j \cdot K \dots j \cdot K + K - 1}^v = V_j$$

而当U为已知数时的系数矩阵 A^u 的第r行中不为零的元素为从第i·K列到第i·K+K-1列：

$$A_{r,i \cdot K \dots i \cdot K + K - 1}^u = U_i$$

❸调用lsmr()进行最小二乘计算，并将计算结果写入U。由于V的值只是假设已知，我们并不希望使用它作为系数找到精确的最小二乘解，因此通过系数maxiter设置计算最小二乘解时的迭代次数，系数damp控制解的大小。当它不为零时被称为正则化最小二乘解。

❹由于 A^u 是一个CSR格式的稀疏矩阵，其中非零元素的值按照行的顺序保存在data属性中。它正好与数组U平坦化之后的每个元素相对应，因此通过 $A^u.data$ 更新稀疏矩阵 A^u 中的数据。这里通过参数mu控

制Au中元素变化的快慢。 μ 越大则Au越接近U的值。接下来使用新的 A^u 计算 V ，并更新 A^v 。

⑤使用U 和V 计算测试数据的预测评分，并计算与实际评分数据的RMSE评价值。将最佳的RMSE评价值保存在best_rmse中，而与之对应的U 和V 则保存在best U和best V中。

⑥由于对大型稀疏矩阵进行最小二乘运算会消耗大量内存，这里调用`gc.collect()`强制进行垃圾回收，释放内存空间。

```
from scipy import sparse

from scipy.sparse import linalg

import gc


def uv_decompose(arrays, loop_count, k, maxiter, mu, damp):
    u_train, v_train, r_train, u_test, v_test, r_test = \
arrays

    U = np.random.rand(nu, k) * 0.1 / k**0.5 ❶
    V = np.random.rand(nv, k) * 0.1 / k**0.5

    idxv_col = (u_train[:, None]*k + np.arange(k)).ravel()
    idx_row = np.repeat(np.arange(nr), k)

    Av = sparse.coo_matrix((V[v_train].ravel(), (idx_row,
idxv_col))),

    shape=(nr, nu*k)).tocsr() ❷
```

```

        idxu_col = (v_train[:, None]*k + np.arange(k)).ravel()

        Au = sparse.coo_matrix((U[u_train].ravel(), (idx_row,
idxu_col)),

                                shape=(nr, nv*k)).tocsr()

        best_U, best_V = None, None
        best_rmse = 100.0
        rmse_list = []

        for i in range(loop_count):
            U.ravel()[:] = linalg.lstsq(Av, r_train,
maxiter=maxiter, damp=damp)[0] ❸

            Au.data[:] = Au.data[:]*(1-mu) +
U[u_train].ravel()*mu ❹

            V.ravel()[:] = linalg.lstsq(Au, r_train,
maxiter=maxiter, damp=damp)[0]

            Av.data[:] = Av.data[:]*(1-mu) +
V[v_train].ravel()*mu

            r_pred = U.dot(V.T)[u_test, v_test] ❺
            rmse = rmse_score(r_test, r_pred)
            rmse_list.append(rmse)
            if rmse < best_rmse:
                best_rmse = rmse
                best_U, best_V = U.copy(), V.copy()
        gc.collect() ❻

```



```
return best_U, best_V, best_rmse, rmse_list
```

程序虽然不复杂，但是maxiter、k、mu和damp等参数均会对结果有影响，因此找到最佳组合是十分耗时的工作。下面比较了damp为3.5和3.0时的RMSE和迭代次数的关系。结果如图11-13所示，由于damp的目的是防止过度学习，因此3.5比3.0的收敛速度慢，但RMSE能收敛到更小的值。

```
U1, V1, best_rmse1, rmses1 = uv_decompose(arrays,
                                           loop_count=20, maxiter=6, k=30,
mu=0.4, damp=3.5)
U2, V2, best_rmse2, rmses2 = uv_decompose(arrays,
                                           loop_count=20, maxiter=6, k=30,
mu=0.4, damp=3.0)
print best_rmse1, best_rmse2
0.901107139807 0.904096209664
pl.plot(np.arange(1, len(rmses1)+1), rmses1,
label="damp=3.5")
pl.plot(np.arange(1, len(rmses2)+1), rmses2,
label="damp=3.0")
pl.legend(loc="best")
pl.xlabel(u"迭代次数")
pl.ylabel("RMSE")
```

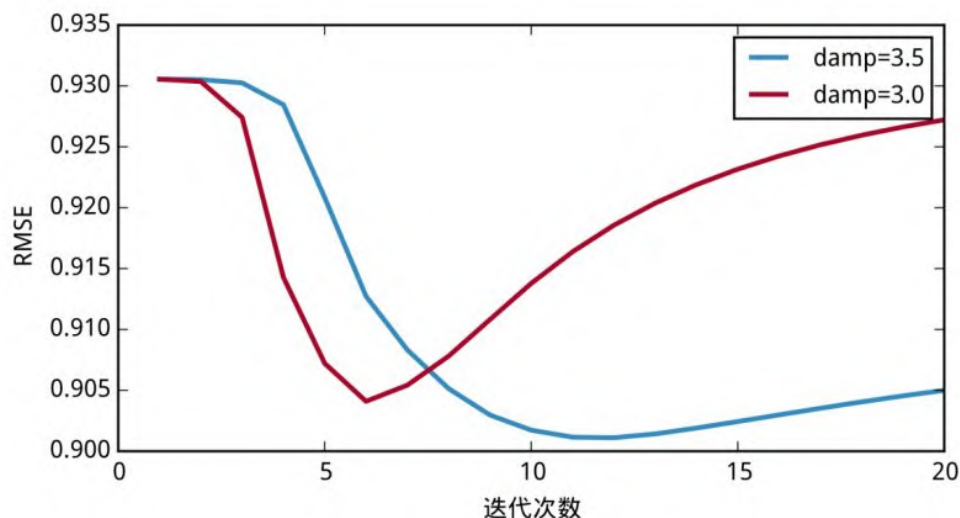


图11-13 damp系数对RMSE的影响

11.3.5 使用Cython迭代实现矩阵分解

还可以使用随机梯度下降法对矩阵进行分解。下面是具体的计算公式。和前面的方法相同，首先用随机数初始化用户矩阵 U 和电影矩阵 V 。随机挑选一个评分 r_{ij} ，它是用户 i 对电影 j 的评分。计算预测评分值 $\hat{r}_{ij}$ ：

$$\hat{r}_{ij} = \sum_{k=1}^K u_{ik} \cdot v_{jk}$$

然后计算评分误差 e ：

$$e = r_{ij} - \hat{r}_{ij}$$

使用误差 e 通过下面两个公式同时更新 U_{ik} 和 V_{jk} ，其中 $k=1\cdots K$ 。 η 为学习系数， β 为防止过度学习的系数。值得注意的是这两个公式是同时更新的，因此第一个公式计算的结果 U_{ik} 并不代入到第二个公式中运算。

$$U_{ik} = U_{ik} + \eta \cdot (e \cdot V_{jk} - \beta \cdot U_{ik})$$
$$V_{jk} = V_{jk} + \eta \cdot (e \cdot U_{ik} - \beta \cdot V_{jk})$$

由于需要大量的循环运算，因此我们使用Cython编写迭代程序。`uv_update()`的参数分别为用户编号、电影编号、评分、用户矩阵 U 、电影矩阵 V 、学习系数和防止过度学习的系数。

```
%%cython
#cython: boundscheck=False
#cython: wraparound=False
import numpy as np

cdef double dot(double[:, ::1] x, double[:, ::1] y, int i,
int j):
    cdef int k
    cdef double s = 0
    for k in range(x.shape[1]):
        s += x[i, k] * y[j, k]
    return s

def uv_update(int[:, ::1] userid, int[:, ::1] movieid,
double[:, ::1] rating,
```

```

        double[:, ::1] uservalue, double[:, ::1]
movievalue,

        double eta, double beta):
    cdef int j, k
    cdef int ratecount = rating.shape[0]
    cdef int uid, mid
    cdef double rvalue, pvalue, error
    cdef double tmp
    cdef int nk = uservalue.shape[1]

    for j in range(ratecount):
        uid = userid[j]
        mid = movieid[j]
        rvalue = rating[j]
        pvalue = dot(uservalue, movievalue, uid, mid)
        error = rvalue - pvalue
        for k in range(nk):
            tmp = uservalue[uid, k]
            uservalue[uid, k] += eta * (error *
movievalue[mid, k]
                                                    - beta *
uservalue[uid, k])
            movievalue[mid, k] += eta * (error * tmp - beta
* movievalue[mid, k])

```

下面的uv_decompose2() 循环iter_count次调用uv_update() 以更新U 和V 。在每次调用之前，通过shuffle() 打乱用户编号、电影编

号、评分这三个数组的顺序，实现评分的随机选择。

```
def uv_decompose2(arrays, k, eta, beta, iter_count):
    u_train, v_train, r_train, u_test, v_test, r_test =
arrays

    U = np.random.rand(nu, k) * 0.1 / k**0.5
    V = np.random.rand(nv, k) * 0.1 / k**0.5

    best_U, best_V = None, None
    best_rmse = 100.0
    rmses = []
    idx = np.arange(nr)

    for i in range(iter_count):
        np.random.shuffle(idx)

        uv_update(u_train[idx], v_train[idx],
r_train2[idx], U, V, eta, beta)

        t = U.dot(V.T)
        r_pred2 = t[u_test, v_test]
        rmse = rmse_score(r_test, r_pred2)
        rmses.append(rmse)

        if best_rmse > rmse:
            best_rmse = rmse
            best_U, best_V = U.copy(), V.copy()

    return best_U, best_V, best_rmse, rmses
```

下面绘制 $k=30$ 、 $\eta=0.008$ 、 $\beta=0.08$ 的收敛曲线，结果如图11-14所示：

```
np.random.seed(2)
U3, V3, best_rmse3, rmses3 = uv_decompose2(arrays, 30,
0.008, 0.08, 100)
pl.plot(rmses3)
idx = np.argmin(rmses3)
pl.axvline(idx, lw=1, ls="--")
pl.ylabel("RMSE")
pl.xlabel(u"迭代次数")
pl.text(idx, best_rmse3 + 0.002, "%g" % best_rmse3)
```

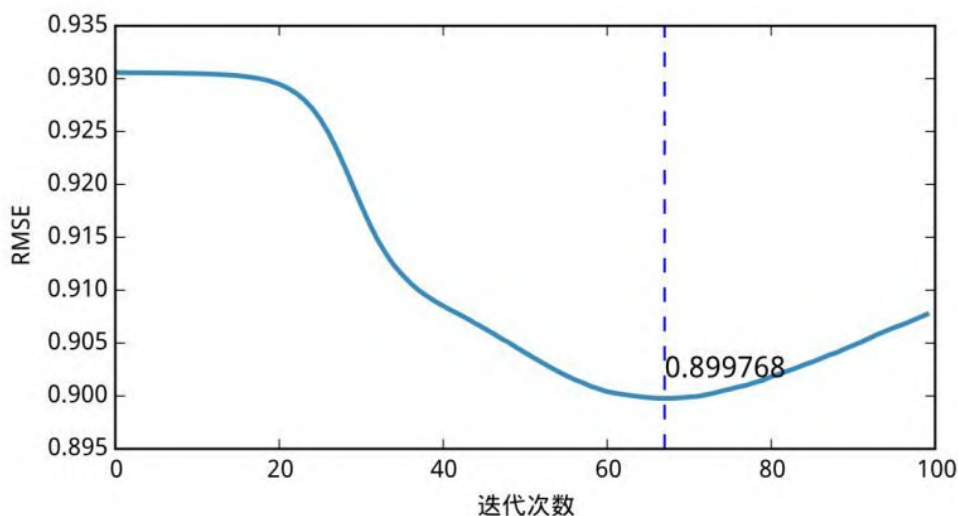


图11-14 随机梯度下降法的收敛曲线

将UV 分解得到的评分预测加上前面的 r_pred ，就得到了最终的评分预测 r_pred3 。下面使用Pandas的`boxplot()`绘制每个评分等级对应的预测评分的箱形图，结果如图11-15所示。图中横坐标为实际评分的5个等级，每个盒子表示每个等级对应的预测评分的分布情况。

```

r_pred3 = U3.dot(V3.T)[u_test, v_test] + r_pred
s = pd.DataFrame({"r":r_test, "$\hat{r}$":r_pred3})
s.boxplot(column="$\hat{r}$", by="r", figsize=(12, 6))

```

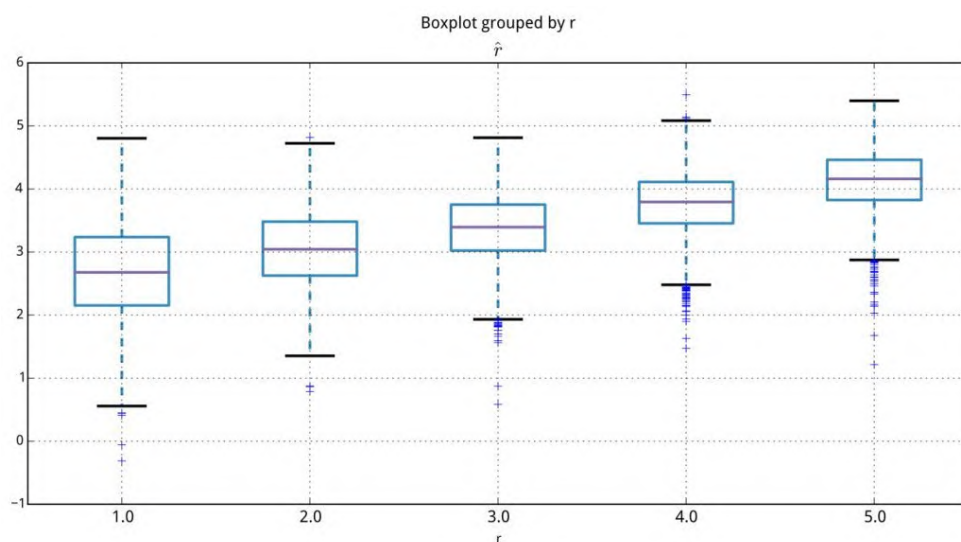


图11-15 以实际评分对预测评分分组，绘制每组的分布情况

11.4 频域信号处理



与本节内容对应的Notebook为：11-examples/examples-400-fft.ipynb。

FFT(快速傅立叶变换)能将时域信号转换为频域信号。转换为频域信号之后，可以很方便地分析出信号的频率成分，在频域上进行处理，最终还可以将处理完毕的频域信号通过IFFT(逆变换)转换为时域信号，实现许多在时域无法完成的信号处理算法。本章将通过许多实例，简单地介绍有关频域信号处理的一些基础知识。

11.4.1 FFT知识复习

FFT变换是针对一个数组的运算，数组的长度N通常是2的整数次幂，例如64、128、256等。数值可以是实数或复数，通常的时域信号都是实数，因此下面都以实数为例。可以把这一组实数想象成对某个连续信号按照一定取样周期进行取样而得，如果有N个实数的数组进行FFT变换，将得到一个有N个复数的数组，它的元素有如下规律：

- 下标为0和N/2的两个复数的虚数部分为0。
- 下标为i和N-i的两个复数共轭，也就是其虚数部分数值相同、符号相反。

下面的例子演示了这一规律，先以rand()随机产生有8个元素的数组x，然后用fft()对其运算之后，观察其结果为8个复数，可以看出结果满足上面两条规律：

```
x = np.random.rand(8)
xf = np.fft.fft(x)

print x
print xf

[ 0.361  0.419  0.499  0.558  0.031  0.705  0.419  0.314]
[ 3.307+0.j      -0.044-0.051j -0.526-0.252j  0.706+0.111j -0.686+0.j      0.706-0.111j
 -0.526+0.252j -0.044+0.051j]
```

FFT变换的结果可以通过IFFT变换还原为原来的值：

```
np.fft.ifft(xf)

array([ 0.361 +0.000e+00j, 0.419 -1.032e-17j, 0.499 -1.388e-17j, 0.558 -1.076e-16j,
        0.031 +0.000e+00j, 0.705 +1.146e-16j, 0.419 +1.388e-17j, 0.314 +3.379e-18j])
```

注意ifft()的运算结果实际上和数组x相同，由于浮点数的运算误差，出现了一些非常小的虚数部分，可以调用np.real()获取其中的实

数部分。

FFT和IFFT变换并没有增加或减少数据的个数：数组x中有8个实数数值，而数组xf中其实也只有8个有效的数值。由于复数共轭和虚数部分为0等规律，真正有用的信息保存在下标从0到N/2的N/2+1个复数中，又由于下标为0和N/2的值的虚数部分为0，因此只有N个有效的实数值。

下面看看FFT变换所得到的复数的含义：

- 下标为0的实数表示时域信号中的直流成分。
- 下标为i的复数 $a+bj$ 表示时域信号中周期为 N/i 个取样值的正弦波和余弦波的成分，其中a表示余弦波形的成分，b表示正弦波形的成分。

让我们通过几个例子验证上述规律，下面对一个直流信号进行FFT变换：

```
x = np.ones(8)

np.fft.fft(x)/len(x) # 为了计算各个成分的能量，需要将FFT的结果除以
FFT的长度

array([ 1.+0.j,  0.+0.j,  0.+0.j,  0.+0.j,  0.+0.j,
        0.+0.j,  0.+0.j,  0.+0.j])
```

所谓直流信号，就是其值不随时间变化，因此我们创建一个值全为1的数组x，它的FFT结果除了下标为0的数值不为0以外，其余的都为0。这表示时域信号是直流的，并且其能量为1。

下面我们产生一个周期为8个取样的正弦波，然后观察其FFT结果：

```
x = np.arange(0, 2*np.pi, 2*np.pi/8)
y = np.sin(x)
tmp = np.fft.fft(y)/len(y)
print np.array_str(tmp, suppress_small=True)
[ 0.+0.j -0.-0.5j  0.-0.j   0.-0.j   0.+0.j   0.-0.j
 0.+0.j   0.+0.5j]
```

为了便于观察结果，这里用`array_str()`将数组转换字符串，并设置`suppress_small`参数为`True`，将一些很小的数值显示为0。现在观察正弦波的FFT的计算结果：下标为1的复数的虚数部分为-0.5，而我们产生的正弦波的振幅为1，它们之间的关系是 $-0.5 * (-2) = 1$ 。接下来看余弦信号的FFT结果：

```
tmp = np.fft.fft(np.cos(x))/len(x)
print np.array_str(tmp, suppress_small=True)
[-0.0+0.j  0.5-0.j  0.0+0.j  0.0+0.j  0.0+0.j -0.0+0.j
 0.0+0.j  0.5-0.j]
```

只有下标为1的复数的实数部分为0.5，和余弦波振幅之间的关系是 $0.5 * 2 = 1$ 。再看两个例子：

```
tmp = np.fft.fft(2*np.sin(2*x))/len(x)
print np.array_str(tmp, suppress_small=True)
tmp = np.fft.fft(0.8*np.cos(2*x))/len(x)
print np.array_str(tmp, suppress_small=True)
[ 0.+0.j  0.+0.j -0.-1.j  0.-0.j  0.+0.j  0.+0.j -0.+1.j  0.-0.j]
[-0.0+0.j -0.0+0.j  0.4-0.j  0.0-0.j  0.0+0.j  0.0-0.j  0.4+0.j -0.0+0.j]
```

上面产生的是周期为4个取样点的正弦和余弦信号，其FFT的有效成分在下标为2的复数中，其中正弦波的振幅为2，其频域虚数部分的值为-1；余弦波的振幅为0.8，频域中对应的值为0.4。

如果将两个同频率的正弦波和余弦波通过不同的系数进行叠加，就可以得到同样频率的各种相位的余弦波。因此我们可以这样来理解频域中的复数：

- 复数的模(绝对值)的两倍为对应频率的余弦波的振幅。
- 复数的辐角表示对应频率的余弦波的相位。

最后再看一个例子：

```
x = np.arange(0, 2*np.pi, 2*np.pi/128)
y = 0.3*np.cos(x) + 0.5*np.cos(2*x+np.pi/4) +
0.8*np.cos(3*x-np.pi/3)
yf = np.fft.fft(y)/len(y)
print np.array_str(yf[:4], suppress_small=True)
print np.abs(yf[1]), np.rad2deg(np.angle(yf[1])) # 周期为128
取样点的余弦波的振幅和相位
print np.abs(yf[2]), np.rad2deg(np.angle(yf[2])) # 周期为64
取样点的余弦波的振幅和相位
# 周期为42.667取样点的余弦波的振幅和相位
print np.abs(yf[3]), np.rad2deg(np.angle(yf[3]))
[ 0.000+0.j      0.150+0.j      0.177+0.177j  0.200-0.346j]
0.15 2.48480834489e-15
0.25 45.0
0.4 -60.0
```

这里 `np.angle()` 计算复数的辐角，得到的是弧度，通过 `np.rad2deg()` 将弧度变换为角度值。在这个例子中产生了三个频率、振幅和相位各不相同的余弦波：

- 周期为128个取样点的余弦波的相位为0，振幅为0.3。
- 周期为64个取样点的余弦波的相位为45度 ($\pi/4$)，振幅为0.5。
- 周期为42.66(128/3.0)个取样点的余弦波的相位为-60($-\pi/3$)度，振幅为0.8。

对照 `yf[1]`、`yf[2]`、`yf[3]` 的复数振幅和辐角，读者应该对FFT结果中的每个数值都有很清晰的认识。

FFT的运算效率由FFT长度N的质因子决定，N能被分解得越小，运算速度越快。例如当N为素数时，FFT的运算效率达到最低。下面的程序比较4096点FFT和4093点FFT运算的时间，由于4096是2的整数次幂，而4093是一个素数，因此它们的运算时间相差非常大：

```
x1 = np.random.random(4096)
x2 = np.random.random(4093)

%timeit np.fft.fft(x1)
%timeit np.fft.fft(x2)

10000 loops, best of 3: 183 µs per loop
10 loops, best of 3: 69.6 ms per loop
```

11.4.2 合成时域信号

在上节的演示中，通过`ifft()`可以将频域信号转换回时域信号，这种转换是精确的。下面的程序完成类似的频域信号转时域信号的计算。不过可以由用户选择一部分频域信号转换为时域信号，这样转换的结果和原始的时域信号会有误差，使用的频率信息越多，此误差越小。通过此程序可以直观地观察到多个余弦波的叠加是如何逐步逼近任意时域信号的，图11-16显示了使用FFT计算的三角波频谱。

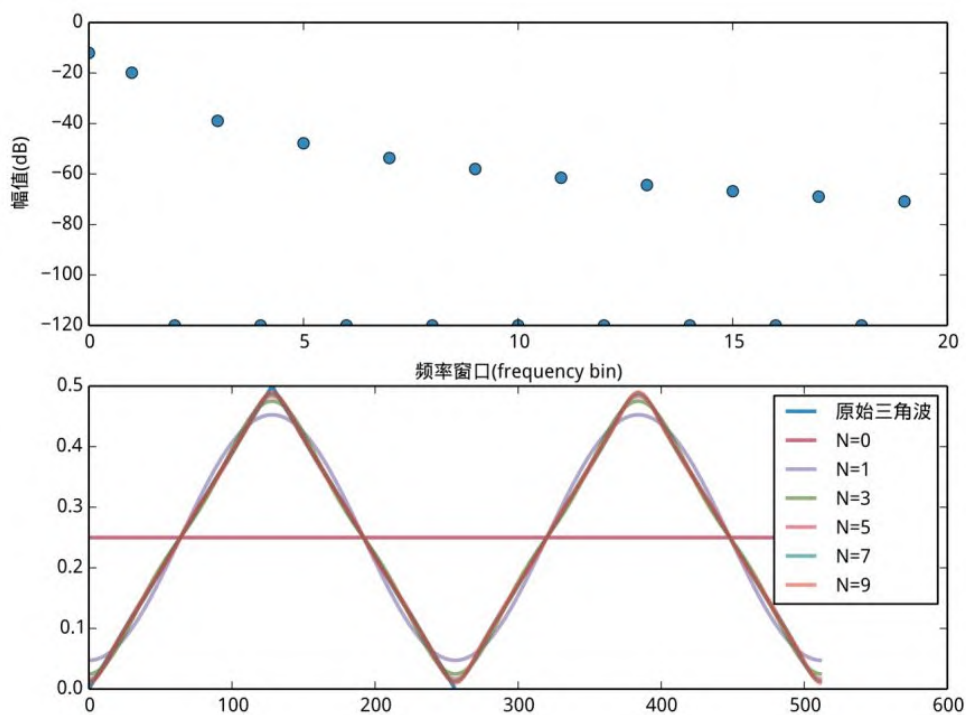


图11-16 三角波的频谱(上)，使用频谱中的部分频率重建的三角波(下)

```
def triangle_wave(size): ❶
    x = np.arange(0, 1, 1.0/size)
    y = np.where(x<0.5, x, 0)
    y = np.where(x>=0.5, 1-x, y)
    return x, y

# 取FFT计算结果bins中的前n项进行合成，返回合成结果，计算loops个周期
```

的波形

```
def fft_combine(bins, n, loops=1): ❷
    length = len(bins) * loops
    data = np.zeros(length)
    index = loops * np.arange(0, length, 1.0) / length * (2
* np.pi)
    for k, p in enumerate(bins[:n]):
        if k != 0: p *= 2 # 除去直流成分之外，其余的系数都*2
        data += np.real(p) * np.cos(k*index) # 余弦成分的系数
为实数部分
        data -= np.imag(p) * np.sin(k*index) # 正弦成分的系数
为负的虚数部分
    return index, data

fft_size = 256

# 计算三角波及其FFT
x, y = triangle_wave(fft_size)
fy = np.fft.fft(y) / fft_size

# 绘制三角波的FFT的前20项的振幅，由于不含下标为偶数的值均为0，因此取
# log之后无穷小，无法绘图，用np.clip函数设置数组值的上下限，保证绘图
正确
fig, axes = pl.subplots(2, 1, figsize=(8, 6))
    axes[0].plot(np.clip(20*np.log10(np.abs(fy[:20])), -120,
120), "o")
    axes[0].set_xlabel(u"频率窗口(frequency bin)")
```

```

axes[0].set_ylabel(u"幅值(dB)")

# 绘制原始的三角波和用正弦波逐级合成的结果，使用的取样点为x轴坐标
axes[1].plot(y, label=u"原始三角波", linewidth=2)

for i in [0,1,3,5,7,9]:
    index, data = fft_combine(fy, i+1, 2) # 计算两个周期的合成波形

    axes[1].plot(data, label = "N=%s" % i, alpha=0.6)
axes[1].legend(loc="best")

```

①triangle_wave() 产生一个周期的三角波形，这里使用 np.where() 计算分段函数。triangle() 返回两个数组，分别表示X轴和Y轴的值。后面的计算和绘图不使用X轴坐标，而是直接用取样序号作为X轴坐标。②fft_combine() 使用FFT结果bins中的前n个数据重新合成时域信号，loops参数指定计算的周期数。通过这个例子可知，合成时域信号时，使用的频率越多，波形越接近原始的三角波。

接下来再看看合成方波信号。由于方波的波形中存在跳变，因此用有限个正弦波合成的方波在跳变处出现抖动现象，如图11-17所示，用正弦波合成的方波的收敛速度比三角波慢得多。计算方波的波形可以采用下面的square_wave()：

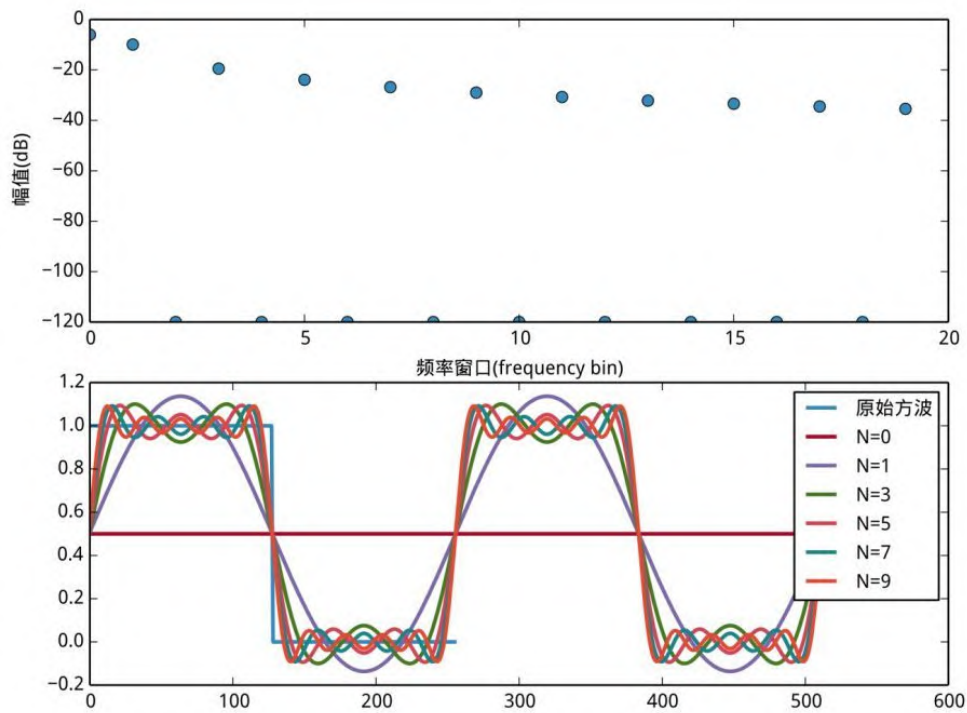


图11-17 方波的频谱，合成方波在跳变处出现抖动

```
def square_wave(size):
    x = np.arange(0, 1, 1.0/size)
    y = np.where(x<0.5, 1.0, 0)
    return x, y

x, y = square_wave(fft_size)
fy = np.fft.fft(y) / fft_size

fig, axes = pl.subplots(2, 1, figsize=(8, 6))
    axes[0].plot(np.clip(20*np.log10(np.abs(fy[:20])), -120,
120), "o")

axes[0].set_xlabel(u"频率窗口(frequency bin)")
axes[0].set_ylabel(u"幅值(dB)")
```



```

axes[1].plot(y, label=u"原始方波", linewidth=2)

for i in [0,1,3,5,7,9]:
    index, data = fft_combine(fy, i+1, 2) # 计算两个周期的合成波形

    axes[1].plot(data, label = "N=%s" % i)

axes[1].legend(loc="best")

```

本书提供了三角波和方波的FFT演示程序，使用它们可以交互式地观察各种三角波和方波的频谱及其正弦合成的近似波形。制作界面是一件很费工夫的事情，幸好有TraitsUI库的帮忙，200多行代码就可以制作出如图11-18所示的效果。

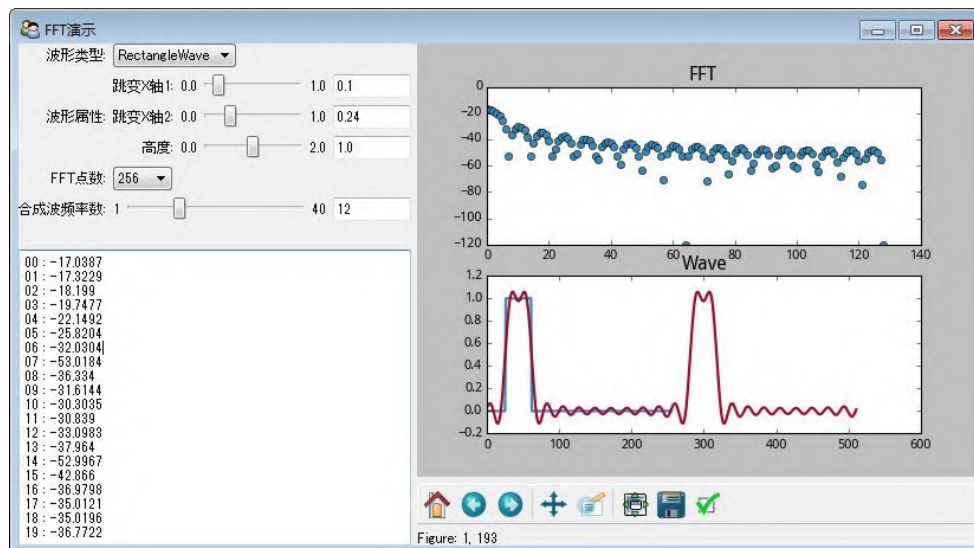


图11-18 波形频谱观察器界面



scipy2.examples.fft_demo: 使用该程序可以交互式地观察各种三角波和方波的频谱及其正弦合成的近似波形。

程序中已经给出了详细的注释，相信读者能够读懂并掌握这类程序的写法。

11.4.3 观察信号的频谱

将时域信号通过FFT转换为频域信号之后，将其各个频率分量的幅值绘制成图，可以很直观地观察信号的频谱。下面的程序能完成这一任务：

```
sampling_rate, fft_size = 8000, 512 ❶
t = np.arange(0, 1.0, 1.0/sampling_rate) ❷
x = np.sin(2*np.pi*156.25*t) + 2*np.sin(2*np.pi*234.375*t)
❸

def show_fft(x):
    xs = x[:fft_size]
    xf = np.fft.rfft(xs)/fft_size ❹
    freqs = np.linspace(0, sampling_rate/2, fft_size/2+1) ❺
    xfp = 20*np.log10(np.clip(np.abs(xf), 1e-20, 1e100)) ❻
    pl.figure(figsize=(8,4))
    pl.subplot(211)
    pl.plot(t[:fft_size], xs)
    pl.xlabel(u"时间(秒)")
    pl.subplot(212)
    pl.plot(freqs, xfp)
    pl.xlabel(u"频率(Hz)")
```

```
pl.subplots_adjust(hspace=0.4)

show_fft(x)
```

图11-19为程序的输出，可以看到频谱中除了两个峰值之外，其余的频率成分都接近于0。如果放大频谱中的两个峰值，可以看到其值分别为：

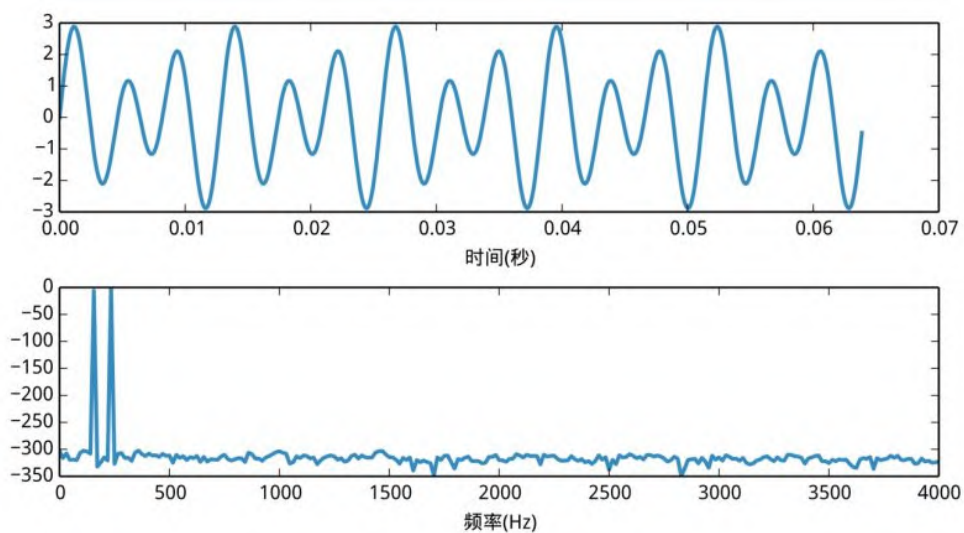


图11-19 156.25Hz和234.375Hz的波形(上)和频谱(下)

```
print xfp[[10, 15]]

[ -6.021e+00  -9.643e-16]
```

即156.25Hz的幅值大小为-6dB，而234.375Hz的幅值大小为0dB。下面详细介绍程序的各个部分：

❶首先定义了两个常数`sampling_rate`和`fft_size`，它们分别表示数字信号的取样频率和FFT的长度。❷然后调用`arange()`产生1秒钟的取样时间，`t`中的每个数值直接表示取样点的时间，因此其间隔为取样周期 $1/\text{sampling_rate}$ 。

③用取样时间数组 t 可以很方便地计算出波形数据，这里计算的是两个正弦波的叠加，一个频率是156.25Hz，另一个是234.375Hz。为什么选择这两个奇怪的频率呢？因为这两个频率的正弦波在512个取样点中正好有整数个周期。只有整数个周期的波形的FFT结果能精确地反映其频率。

假设取样频率为 f_s ，取波形中的 N 个数据进行FFT变换。当这 N 点数据包含整数个周期的波形时，FFT所计算的结果是精确的。因此能精确计算的波形的周期是： $n \cdot f_s / N$ 。对于8kHz的取样频率、512点的FFT来说， $8000/512.0=15.625\text{Hz}$ ，即只能精确表示15.625Hz的整数倍频率。156.25Hz和234.375Hz正好是其10倍和15倍。

④这里使用`rfft()`对从波形数据 x 中截取`fft_size`个取样点进行FFT计算，所得到的结果不包括共轭部分。根据FFT计算公式，为了正确显示波形能量，还需要将结果除以FFT的长度`fft_size`。

对于长度为 N 的FFT运算，`rfft()`返回 $N/2+1$ 个复数，分别表示从0到 $f_s/2$ 的各点频率的成分。⑤因此可以通过`linspace()`计算出`rfft()`的返回值中每个数值对应的真正频率。也可以使用NumPy提供的`fftfreq()`函数，它的第一个参数为FFT的长度，第二个参数为信号的取样周期。它返回与FFT结果对应的`fft_size`个频率，前半部分频率大于等于0，后半部分频率为负值。其中频率为负值的部分也可以将其频率理解为该负值加上取样频率。此外，`rfftfreq()`计算与`rfft()`结果对应的频率。

```

freqs = np.fft.fftfreq(fft_size, 1.0/sampling_rate)
for i in [0, 1, fft_size//2-1, fft_size//2, fft_size//2+1, fft_size-2, fft_size-1]:
    print i, "\t", freqs[i]
0    0.0
1    15.625
255    3984.375
256    -4000.0
257    -3984.375
510    -31.25
511    -15.625

```

⑥最后计算每个频率分量的幅值，并将其转换为以dB度量的值。为了防止0幅值造成`log10()`无法计算，调用`np.clip()`对`xf`的幅值进行上下限处理。

下面看看不能在`fft_size`个取样中形成整数个周期的波形的频谱：

```

x = np.sin(2*np.pi*200*t) + 2*np.sin(2*np.pi*300*t)
show_fft(x)

```

得到的结果如图11-20所示。这次得到的频谱不再是两个完美的峰值，而是两个峰值频率周围的频率都有能量。这显然和两个正弦波的叠加波形的频谱有区别。本来应该属于200Hz和300Hz的能量分散到了周围的频率中，这种现象被称为频谱泄漏。出现频谱泄漏的原因在于`fft_size`个取样点无法放下整数个200Hz和300Hz的波形。

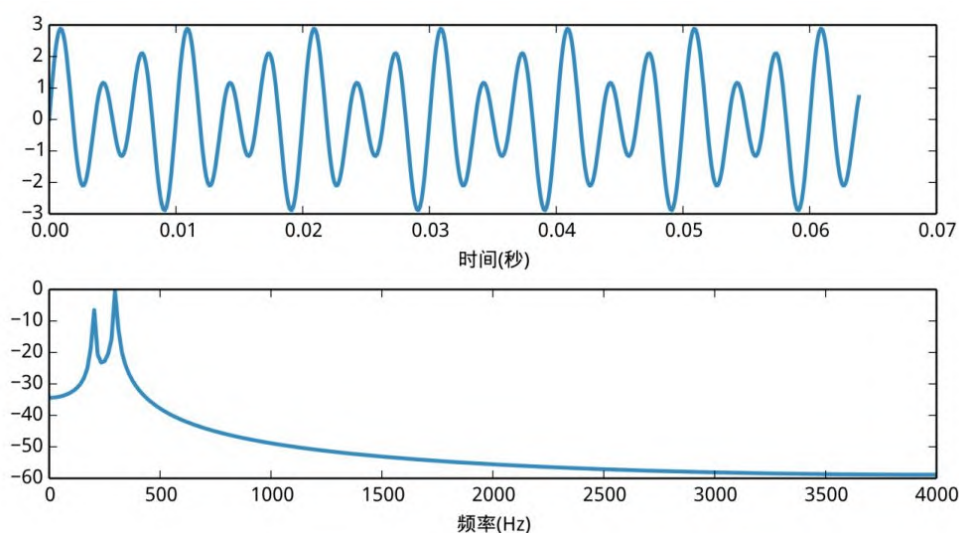


图11-20 非完整周期(200Hz和300Hz)的正弦波经过FFT变换之后出现频谱泄漏

我们只能在有限的时间段中对信号进行测量，无法知道测量范围之外的信号。因此只能对测量范围之外的信号进行假设。而傅立叶变换的假设很简单：测量范围之外的信号是所测量到的信号的重复。

现在考虑512点FFT，从信号中取出的512个数据就是FFT的测量范围，它计算的是这512个数据一直重复的波形的频谱。显然，如果512个数据包含整数个周期，那么得到的结果就是原始信号的频谱；而如果不是整数周期，得到的频谱就是如图11-21所示的波形的频谱。由于波形的前后不是连续的，存在跳变，而跳变处有着非常广泛的频谱，因此FFT结果中出现频谱泄漏。

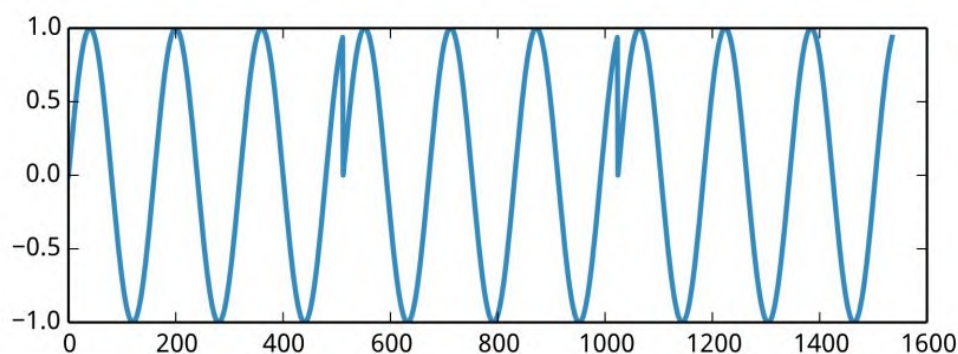


图11-21 50Hz正弦波的512点FFT所计算频谱的实际波形

```
pl.figure(figsize=(6, 2))  
t = np.arange(0, 1.0, 1.0/8000)  
x = np.sin(2*np.pi*50*t)[:512]  
pl.plot(np.hstack([x, x, x]))
```

1. 窗函数

为了减少FFT所截取的数据段前后的跳变，可以把数据与一个窗函数相乘，使得其前后数据能平滑过渡。例如，常用的Hann窗函数的定义如下：

$$w(n) = 0.5 \left(1 - \cos\left(\frac{2\pi n}{N-1}\right) \right)$$

其中N为窗函数的点数，图11-22是一个512点Hann窗的曲线：

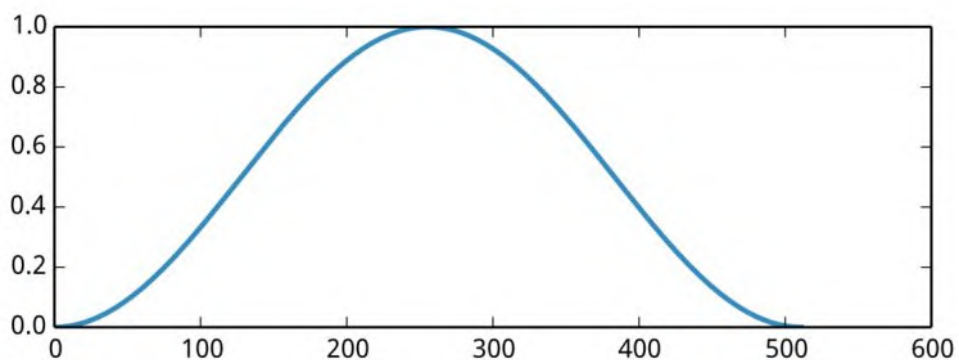


图11-22 Hann窗函数

```
from scipy import signal
pl.figure(figsize=(6, 2))
pl.plot(signal.hann(512))
```

窗函数都在`scipy.signal`库中定义，它们的第一个参数为点数`N`。可以看出Hann窗函数是完全对称的，也就是说第0点和第511点的值完全相同，都为0。如果将这样的窗函数与信号数据相乘，结果中会出现前后两个连续的0，这样会对FFT变换之后的信号频谱有一定的影响。

为了解决连续0值的问题，`hann`函数提供了`sym`参数，如果其值为0，则产生一个`N+1`点的Hann窗函数，并舍去最末端的数值，这样得到的窗函数就适合于周期信号了：

```
print signal.hann(8)
print signal.hann(8, sym=0)
```

[0.	0.188	0.611	0.95	0.95	0.611	0.188	0.]
[0.	0.146	0.5	0.854	1.	0.854	0.5	0.146]

50Hz正弦波与窗函数乘积之后的周期重复波形如图11-23所示：

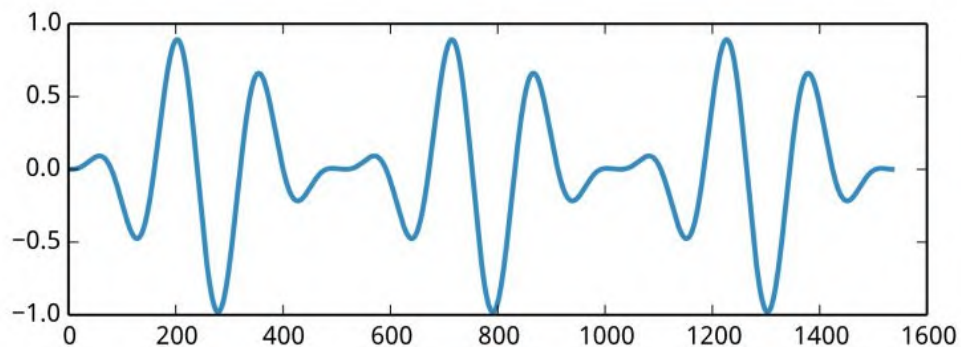


图11-23 加Hann窗的50Hz正弦波的512点FFT所计算的实际波形

```
pl.figure(figsize=(6, 2))
t = np.arange(0, 1.0, 1.0/8000)
```



```
x = np.sin(2*np.pi*50*t)[:512] * signal.hann(512, sym=0)
pl.plot(np.hstack([x, x, x]))
```

回到前面的例子，将200Hz和300Hz的叠加波形与Hann窗相乘之后再计算其频谱，得到如图11-24所示的频谱图。

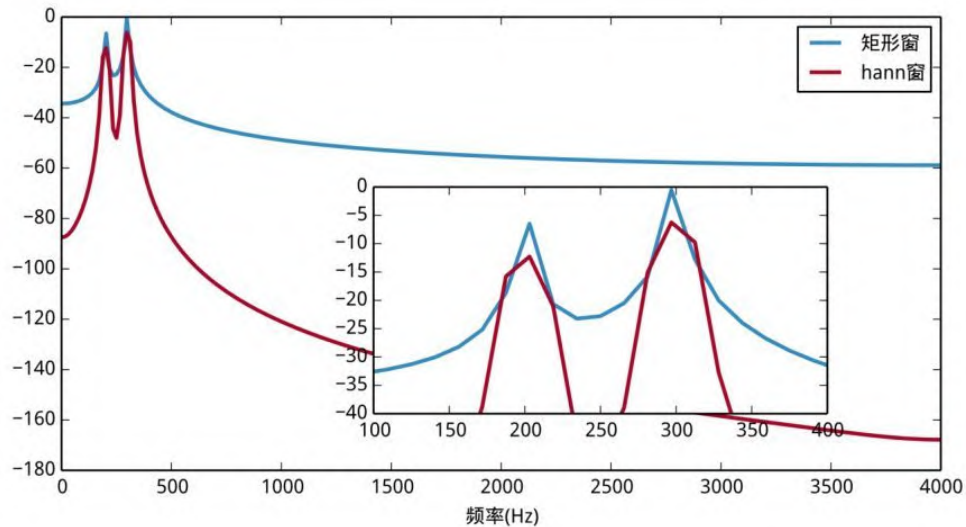


图11-24 加Hann窗前后的频谱，Hann窗能降低频谱泄漏

```
t = np.arange(0, 1.0, 1.0/sampling_rate)
x = np.sin(2*np.pi*200*t) + 2*np.sin(2*np.pi*300*t)

xs = x[:fft_size]
ys = xs * signal.hann(fft_size, sym=0)

xf = np.fft.rfft(xs)/fft_size
yf = np.fft.rfft(ys)/fft_size
freqs = np.linspace(0, sampling_rate/2, fft_size/2+1)
xfp = 20*np.log10(np.clip(np.abs(xf), 1e-20, 1e100))
yfp = 20*np.log10(np.clip(np.abs(yf), 1e-20, 1e100))
```

```
pl.figure(figsize=(8,4))
pl.plot(freqs, xfp, label=u"矩形窗")
pl.plot(freqs, yfp, label=u"hann窗")
pl.legend()
pl.xlabel(u"频率 (Hz) ")

a = pl.axes([.4, .2, .4, .4])
a.plot(freqs, xfp, label=u"矩形窗")
a.plot(freqs, yfp, label=u"hann窗")
a.set_xlim(100, 400)
a.set_ylim(-40, 0)
```

可以看到与Hann窗乘积之后的信号的频谱能量更加集中于200Hz和300Hz，但是其能量有所降低。这是因为Hann窗本身有一定的能量衰减：

```
np.mean(signal.hann(512, sym=0))
0.5
```

如果需要严格保持信号的能量，还需要在与Hann窗相乘之后再把信号扩大一倍。

2. 频谱平均

对于频谱特性不随时间变化的信号，例如引擎、压缩机等机器噪声，可以对其进行长时间的采样，然后分段进行FFT计算，最后对每个

频率分量的幅值求平均值，就可以准确地测量信号的频谱。下面的程序完成这一计算：

```
def average_fft(x, fft_size):  
    n = len(x) // fft_size * fft_size  
    tmp = x[:n].reshape(-1, fft_size) ❶  
    tmp *= signal.hann(fft_size, sym=0) ❷  
    xf = np.abs(np.fft.rfft(tmp)/fft_size) ❸  
    avgf = np.mean(xf, axis=0)  
    return 20*np.log10(avgf)
```

`average_fft(x, fft_size)`对数组`x`进行`fft_size`点FFT运算，并返回以dB为度量的平均幅值。由于`x`的长度可能不是`fft_size`的整数倍，❶因此首先将其缩短为`fft_size`的整数倍，然后用`reshape()`将其转换成一个二维数组`tmp`。`tmp`的第1轴的长度为`fft_size`。

❷将`tmp`数组的第1轴上的数据和Hann窗函数相乘。❸调用`rfft()`对`tmp`数组中的每行数据进行FFT计算，并求其幅值。❹最后，用`mean()`对`xf`沿着第0轴进行平均，这样就得到了每个频率分量的平均幅值。

图11-25是利用`average_fft()`计算随机数序列频谱的例子。

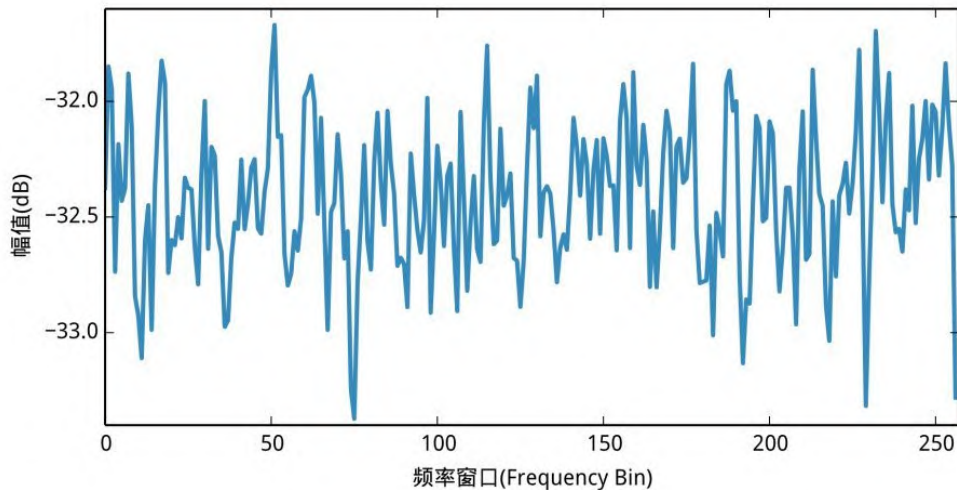


图11-25 白色噪声的频谱接近水平直线(注意Y轴的范围)

```
x = np.random.randn(100000)
xf = average_fft(x, 512)
pl.figure(figsize=(7,3.5))
pl.plot(xf)
pl.xlabel(u"频率窗口(Frequency Bin)")
pl.ylabel(u"幅值(dB)")
pl.xlim([0,257])
pl.subplots_adjust(bottom=0.15)
```

可以看到随机噪声的频谱接近一条水平直线，也就是说每个频率窗口的能量都相同，这种噪声被称作白噪声。如果让白噪声通过一个IIR低通滤波器，绘制其输出信号的平均频谱，就能够观察到IIR滤波器的频率响应特性。下面的程序利用*iirdesign()*设计一个8kHz取样的1kHz的Chebyshev I型低通滤波器，*iirdesign()*需要用正规化的频率(取值范围为0~1)，然后调用*filtfilt()*对白色噪声信号进行低通滤波。如果用*average_fft()*计算滤波器输出信号的平均频谱，将得到如图11-26所示的频谱图。

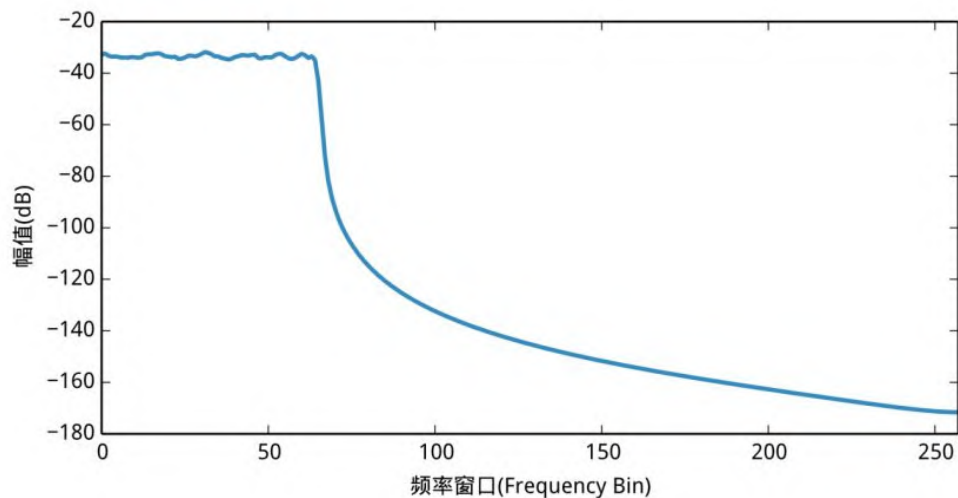


图11-26 经过低通滤波器的白噪声的频谱

```
b, a = signal.iirdesign(1000/4000.0, 1100/4000.0, 1, 40, 0,
"cheby1")
x = np.random.randn(100000)
y = signal.filtfilt(b, a, x)
yf = average_fft(y, 512)
pl.figure(figsize=(7, 3.5))
pl.plot(yf)
pl.xlabel(u"频率窗口(Frequency Bin)")
pl.ylabel(u"幅值(dB) ")
pl.xlim(0, 257)
pl.subplots_adjust(bottom=0.15)
```

3. 谱图

虽然使用FFT能够观察信号的频域特性，但却完全丧失了信号在时间轴上的信息。因此前面所介绍的观察信号频谱的方法只适合于频率

特性不随时间变化的情况。当信号频率随时间变化时，为了既能观察信号频率又能观察其随时间的变化，可以使用短时距傅里叶变换 (STFT)。

STFT算法所得到的结果被称为谱图(Spectrogram)。谱图的横轴表示时间而纵轴表示频率，谱图上每点的值表示信号在此点的能量。STFT算法其实很简单：对信号分段进行FFT处理，每一次处理的结果都是谱图中的一列。每段信号的长度越短，时间轴上的精度越高，而频率轴上的精度就越低，反过来也是一样。时间轴和频率轴上的分辨率是一对不可调和的矛盾，根据傅立叶变换的不确定原理，我们不能指望同时获得频率和时间的高分辨率。

下面是绘制频率扫描波的谱图的程序，效果如图11-27所示。通过此图可以很直观地观察到信号的频率随着时间而逐渐变高，并且是呈指数增长的。

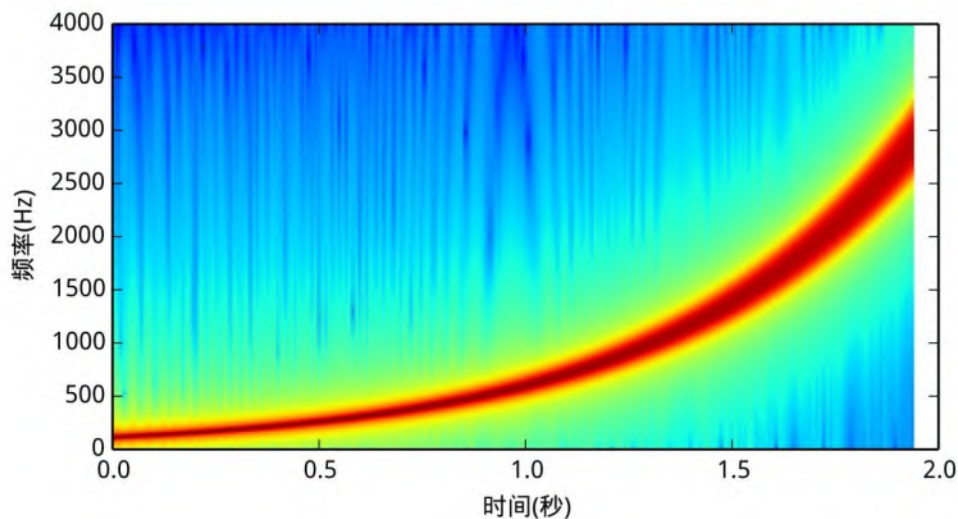


图11-27 频率扫描波的谱图

```
sampling_rate = 8000.0
fft_size = 1024
step = fft_size/16
time = 2

t = np.arange(0, time, 1/sampling_rate)
sweep = signal.chirp(t, f0=100, t1 = time,
f1=0.8*sampling_rate/2, method="logarithmic")

pl.specgram(sweep, fft_size, sampling_rate, noverlap =
1024-step)
pl.xlabel(u"时间(秒)")
pl.ylabel(u"频率(Hz)")
```

这里使用matplotlib提供的绘制谱图的函数specgram()，其第一个参数是表示信号的数组，第二个参数是FFT的长度，第三个参数是信号的取样频率。noverlap参数是连续两块数据之间重叠部分的长度，该参数越接近FFT的长度，FFT运算的次数越多，时间轴上的精度也越大。specgram()还有许多其他的关键字参数，请读者阅读其函数文档以了解详细用法。

本书提供了一个用PyAudio、TraitsUI等制作的谱图观察的程序。它实时地从声卡读入声音数据，并绘制出声音信号的时间波形、频谱以及谱图。由于本程序对计算机的配置要求较高，请读者在较快的机器上运行本程序。另外，为了计算效率，程序中没有使用重叠处理。图11-28是程序的界面截图。

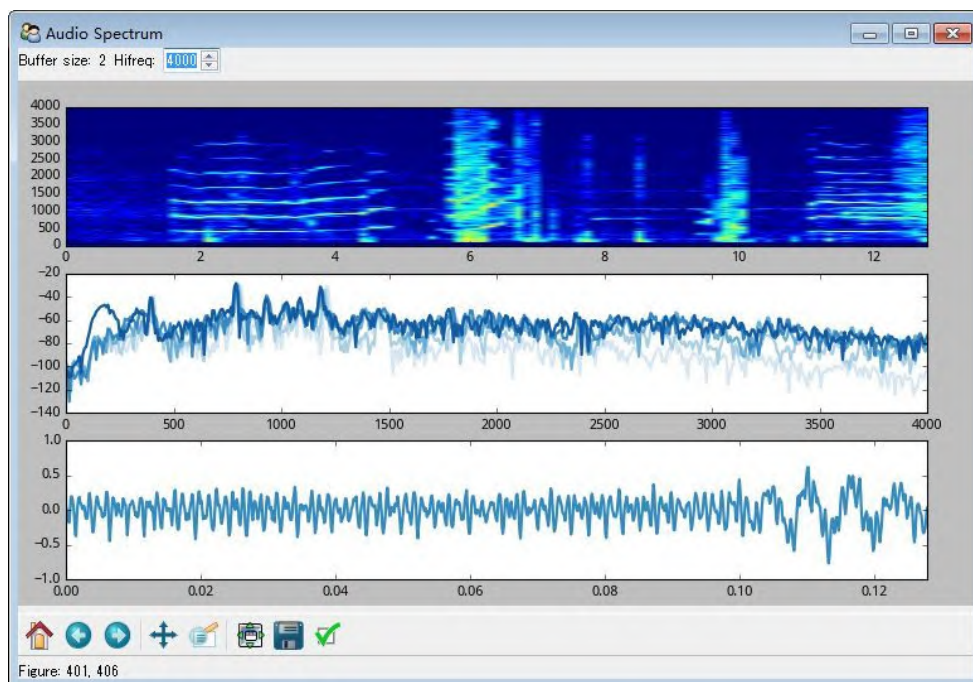


图11-28 使用TraitsUI制作的实时观察声音信号谱图的界面



`scipy2.examples.spectrogram_realtime`: 实时观察声音信号谱图的演示程序，使用TraitsUI、PyAudio等库来实现。

4. 精确测量信号频率

FFT的频率分辨率可以通过“取样频率/FFT长度”计算，若仅根据频谱峰值的位置测量信号频率，则为了精确测量只能提高FFT的长度。本小节介绍一种使用FFT结果中的相位信息，在不增加FFT长度的情况下精确测量频率的方法。

下面首先创建一个包含三个频率44Hz、150Hz和330Hz，以及白色噪声的测试信号x。其取样频率为8000Hz，长度为2400点。如果使用频谱峰值测量频率，则频率的最高分辨率为 $8000/2400=3.33\text{Hz}$ 。

```
def make_wave(amp, freq, phase, tend, rate):
    period = 1.0 / rate
    t = np.arange(0, tend, period)
    x = np.zeros_like(t)
    for a, f, p in zip(amp, freq, phase):
        x += a * np.sin(2*np.pi*f*t + p)
    return t, x

RATE = 8000

t, x = make_wave([1, 2, 0.5], [44, 150, 330], [1, 1.4, 1.8], 0.3, RATE)

x += np.random.randn(len(x))
```

下面用1024点FFT计算频谱峰值对应的频率。峰值需要满足两个条件：❶可以使用`signal.argrelmax()`计算局域最大值。其`order`参数指定比较窗口的大小，3表示峰值需要比其左右相邻的3个值都大。❷峰值需要大于平均值的3倍，这样可以剔除由白色噪声产生的微小局域峰值。

```
FFT_SIZE = 1024

spect1 = np.fft.rfft(x[:FFT_SIZE] * np.hanning(FFT_SIZE))
freqs = np.fft.fftfreq(FFT_SIZE, 1.0/RATE)

bin_width = freqs[1] - freqs[0]
```

```

amp_spect1 = np.abs(spect1)
loc, = signal.argrelmax(amp_spect1, order=3) ❶
mask = amp_spect1[loc] > amp_spect1.mean() * 3 ❷
loc = loc[mask]
peak_freqs = freqs[loc]
print "bin width:", bin_width
print "Peak Frequencies:", peak_freqs
bin width: 7.8125
Peak Frequencies: [ 46.875 148.438 328.125]

```

可以看出峰值的频率与实际的频率相比有近3Hz的误差。利用相位信息可以减小检测频率的误差。为了利用相位信息，我们延时256个取样之后再次计算信号的频谱。256个取样值相当于延时了 $dt = 256 / 8000 = 0.032$ 秒。计算两次FFT结果中频谱峰值对应的相位，并计算相位差值 $phase_delta$ 。如果在 Δt 时间之内，相位变化了 $\Delta \theta$ ，则频率可以根据如下公式计算：

$$f_n = \frac{\Delta \theta + 2n\pi}{2\pi\Delta t}, n = 0 \dots \infty$$

由于旋转一圈之后，相位完全相同，因此在 Δt 时间内，相位变化 $\Delta \theta$ 的频率有无数个，它是一个等差数列。只需要找出该等差数列中与根据频谱峰值获得的频率最接近的那个频率即可。下面首先计算峰值处前后两次FFT的相位差：

```

COUNT = FFT_SIZE//4
dt = COUNT / 8000.0

```

```

spect2 = np.fft.rfft(x[COUNT:COUNT+FFT_SIZE] *
np.hanning(FFT_SIZE))

phase1 = np.angle(spect1[loc])
phase2 = np.angle(spect2[loc])

phase_delta = phase2 - phase1
print phase_delta
[ 2.595 -1.29 -2.899]

```

然后利用相位差计算信号中各个峰值的频率。❶为了减少运算次数，首先利用频谱峰值中的最高频率计算n的最大可能取值max_n。❷利用广播运算，计算各个相位差值所对应的可能的频率possible_freqs。❸找到可能频率中与峰值频率最接近的那个作为信号的频率。

可以看到结果非常接近信号的真实频率，由于存在白噪声，相位差也存在较小的误差，因此最终计算的信号频率也存在误差。

```

max_n = (peak_freqs.max() + 3*bin_width) * dt ❶
n = np.arange(max_n)

possible_freqs = (phase_delta + 2*np.pi*n[:, None]) / (2 *
np.pi * dt) ❷

idx = np.argmin(np.abs(peak_freqs - possible_freqs),
axis=0) ❸

```

```
        peak_freqs2      =    possible_freqs[idx,
np.arange(len(peak_freqs))]

    print "Peak Frequencies:", peak_freqs2

    Peak Frequencies: [  44.155  149.833  329.33 ]
```

11.4.4 卷积运算

信号 x 经过系统 h 之后的输出 y 是 x 和 h 的卷积，虽然卷积的计算方法很简单，但是当 x 和 h 都很长的时候，卷积计算是非常耗费时间的。因此对于响应时间很长的系统 h ，需要找到比直接计算卷积更快的方法。

信号系统理论中有这样一个规律：时域的卷积等于频域的乘积，因此要计算时域的卷积，可以将时域信号转换为频域信号，进行乘积运算之后再将结果转换为时域信号，实现快速卷积。

1. 快速卷积

由于FFT运算可以高效地将时域信号转换为频域信号，其运算的复杂度为 $O(N \log N)$ ，因此三次FFT运算加一次乘积运算的总复杂度仍然为 $O(N \log N)$ 级别，而卷积运算的复杂度为 $O(N^2)$ ，显然通过FFT计算卷积要比直接计算快得多。这里假设需要卷积的两个信号的长度都为 N 。

但是有一个问题：FFT运算假设其所计算的信号为周期信号，因此通过上述方法计算出的结果实际上是两个信号的循环卷积，而不是线

性卷积。为了用FFT计算线性卷积，需要对信号进行补零扩展，使得其长度长于线性卷积结果的长度。

例如，如果要计算数组a和b的卷积，a和b的长度都为128，那么它们的卷积结果的长度为 $\text{len}(a) + \text{len}(b) - 1 = 255$ 。为了让FFT能够计算其线性卷积，需要将a和b的长度都扩展到256。下面的程序演示了这个计算过程。

❶找到大于n的最小的2的整数次幂。❷fft()的第二个参数为FFT的长度，当输入数据的长度不够时，自动对其进行补零。❸最后对两个频域信号的乘积调用ifft()，得到时域信号的卷积。其结果比实际的卷积结果多一个数，这个多出来的数值应该接近于0，请读者自行验证。

```
def fft_convolve(a,b):
    n = len(a) + len(b) - 1
    N = 2**(int(np.log2(n)) + 1) ❶
    A = np.fft.fft(a, N) ❷
    B = np.fft.fft(b, N)
    return np.fft.ifft(A * B)[:n] ❸

a = np.random.rand(128)
b = np.random.rand(128)
c = np.convolve(a,b)
np.allclose(c, fft_convolve(a, b))
True
```

由于直接计算卷积和使用FFT的快速卷积的复杂度级别不同，因此当卷积数据很长时，可以观察到明显的速度差别。下面的程序比较两种卷积算法的运算时间：

```
a=np.random.rand(10000)
b=np.random.rand(10000)
print np.allclose(np.convolve(a, b), fft_convolve(a, b))

%timeit np.convolve(a, b)
%timeit fft_convolve(a, b)

True

10 loops, best of 3: 36.5 ms per loop
100 loops, best of 3: 6.43 ms per loop
```

显然计算两个很长的数组的卷积时，FFT快速卷积要比直接卷积快很多。但是对于较短的数组，直接卷积运算还是更快一些。图11-29显示了直接卷积和快速卷积的平均计算时间和长度之间的关系。其中Y轴显示的是每个数据的平均计算时间，因此直接卷积对应的曲线是线性的： $O(N)$ 。由图可知对于长度大于1024的卷积，快速卷积显示出明显的优势。

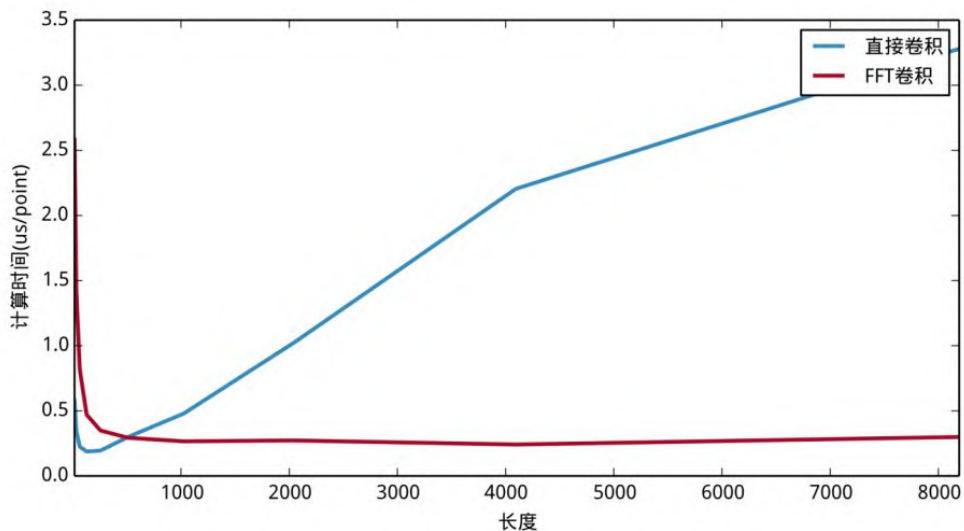


图11-29 比较直接卷积和FFT卷积的运算速度

由于FFT卷积很常用，因此`scipy.signal`中提供了`fftconvolve()`。此函数采用FFT运算，并可计算多维数组的卷积。下面是在Notebook中测试直接卷积与快速卷积的速度的程序：

```
results = []
for n in xrange(4, 14):
    N = 2**n
    a = np.random.rand(N)
    b = np.random.rand(N)
    tr1 = %timeit -r 1 -o -q np.convolve(a, b)
    tr2 = %timeit -r 1 -o -q fft_convolve(a, b)
    t1 = tr1.best * 1e6 / N
    t2 = tr2.best * 1e6 / N
    results.append((N, t1, t2))
results = np.array(results)

pl.figure(figsize=(8,4))
```

```
pl.plot(results[:, 0], results[:, 1], label=u"直接卷积")
pl.plot(results[:, 0], results[:, 2], label=u"FFT卷积")
pl.legend()
pl.ylabel(u"计算时间(us/point)")
pl.xlabel(u"长度")
pl.xlim(min(n_list), max(n_list))
```

2. 卷积的分段运算

现在考虑输入信号 x 和系统响应 h 的卷积运算。通常输入信号是非常长的，例如要对某段录音进行滤波处理，假设取样频率为8kHz，录音长度为1分钟，则数据的长度为480000。而如果需要对麦克风的输入信号进行连续的滤波处理，那么输入信号的长度可以看作无限长的。系统响应 h 的长度通常都是固定的，例如它可能是某个房间的脉冲响应，或是某种FIR滤波器的系数。

根据前面的介绍，为了有效地利用FFT计算卷积，我们希望它的两个输入长度相当，于是就需要对输入信号进行分段处理。卷积的分段运算被称作overlap-add运算，其计算方法如图11-30所示：

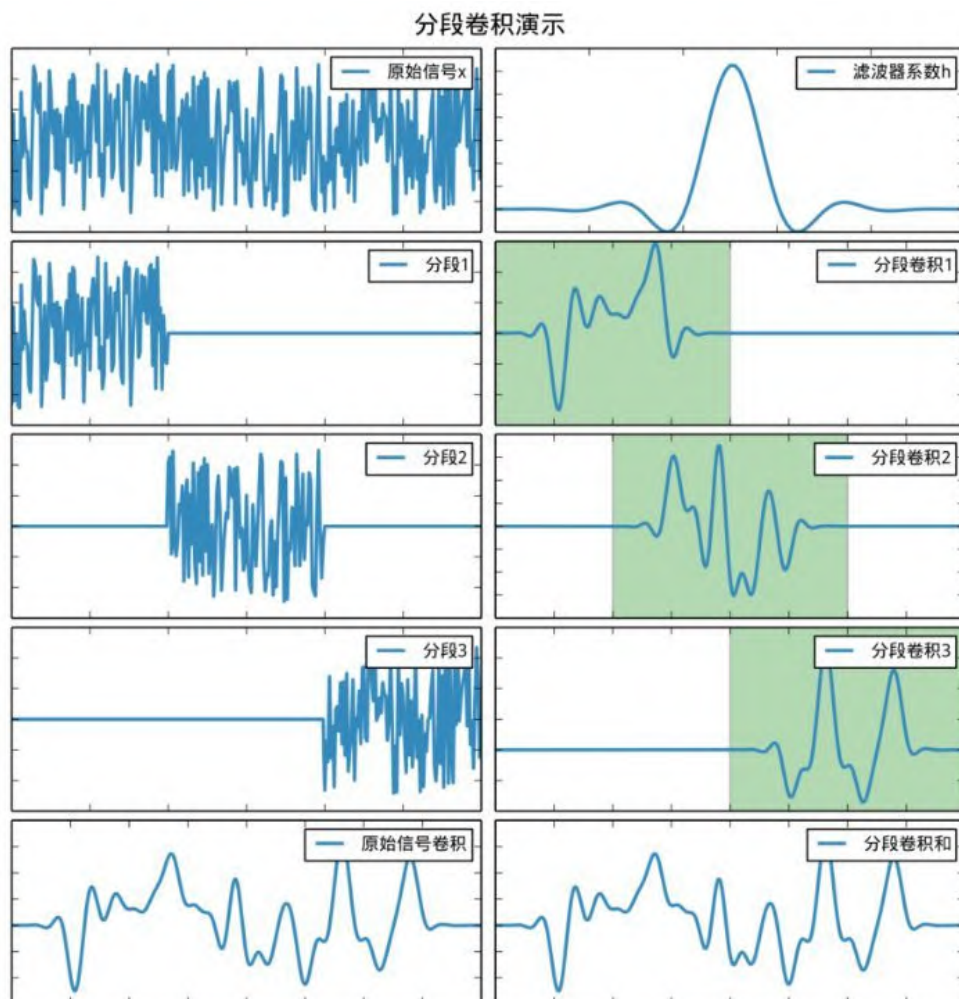


图11-30 使用overlap-add法进行分段卷积的过程演示

图中原始信号 x 的长度为300，将它分为三段，分别与滤波器系数 h 进行卷积计算。 h 的长度为101，因此每段输出200个数据，图中用绿色标出每段输出的200个数据。这3段数据按照时间顺序求和，得到的结果和直接卷积的结果是相同的。

输入信号 x 和滤波器 h 的分段卷积运算可以按照如下步骤进行，假设 h 的长度为 M ：

- (1) 建立一个缓存，其大小为 $N+M - 1$ ，初始值为0。

(2) 每次从x中读取N个数据，和h进行卷积，得到 $N+M - 1$ 个数据。将这些数据与缓存中的数据进行求和，并将结果保存进缓存中，最后输出缓存的前N个数据。

(3) 将缓存中的数据向左移动N个元素，也就是让缓存中的第N个元素成为第0个元素，左移完成之后将缓存中后面的N个元素全部设置为0。

(4) 跳转到(2)重复运行。

下面是实现这一算法的演示程序：

```
x = np.random.rand(1000)
h = np.random.rand(101)
y = np.convolve(x, h)

N = 50 # 分段大小
M = len(h) # 滤波器的长度

output = []

#缓存初始化为0
buffer = np.zeros(M+N-1,dtype=np.float64)

for i in xrange(len(x)/N):
    #从输入信号中读取N个数据
    xslice = x[i*N:(i+1)*N]
    #计算卷积
```

图

```
yslice = np.convolve(xslice, h)
#将卷积的结果加入到缓存中
buffer += yslice
#输出缓存中的前N个数据，注意使用copy，否则输出的是buffer的一个视图

output.append( buffer[:N].copy() ) ❶

#缓存中的数据左移N个元素
buffer[0:-N] = buffer[N:]

#后面的补0
buffer[-N:] = 0

#将输出的数据组合为数组
y2 = np.hstack(output)
#计算和直接卷积的结果之间的误差
print "error:", np.max(np.abs( y2 - y[:len(x)] ) )
error: 7.1054273576e-15
```

❶注意需要输出缓存前N个数据的拷贝，否则输出的是数组的视图。当此后缓存buffer更新时，视图中的数据会一起更新。程序输出直接卷积和overlap-add分段卷积的最大误差。将FFT快速卷积和overlap-add相结合，可以制作出一些快速的实时数据滤波算法。但是由于FFT卷积对于两个长度相当的数组最为有效，因此在分段时也会有所限制。例如，如果滤波器的长度为2048，那么理想的分段长度也为2048；如果将分段长度设置得过低，反而会增加运算量。因此在实时性要求很强的系统中，只能采用直接卷积。

11.5 布尔可满足性问题求解器



与本节内容对应的Notebook为：`11-examples/examples-500-picosat.ipynb`。

可满足性用来解决给定的布尔方程式，寻找一组变量赋值，使问题成为可满足。布尔可满足性问题(以下简称SAT)属于决定性问题，是第一个被证明NP完全问题。它是计算机科学中许多领域的重要问题，包括计算机科学基础理论、算法、人工智能、硬件设计等。本章介绍如何用Cython包装C语言的SAT问题求解器PicoSAT，并使用该扩展模块解决数独游戏和扫雷游戏。



<http://fmv.jku.at/picosat/>

PicoSAT的下载地址。

让我们先通过一个例子解释什么是布尔方程式，以及如何将其转换为SAT求解器所需的合取范式(CNF)，并调用`cycosat`对其求解。

一道逻辑推理题

有4名嫌疑犯，他们做了如下供述：

- 甲：不是我作的案。
- 乙：丁就是罪犯。
- 丙：乙是罪犯。
- 丁：乙有意诬陷我。

已知4人当中只有一人说了真话。请推理出罪犯是谁？

我们用A、B、C和D这4个布尔变量分别表示甲、乙、丙、丁4位嫌疑人是否是罪犯。表达式A表示甲是罪犯， $\sim A$ 表示甲不是罪犯。由此可以得出表11-1所示的4个布尔表达式：

表11-1 布尔表达式

嫌疑犯	供述	布尔表达式
甲	不是我作的案	$S1=\sim A$
乙	丁就是罪犯	$S2=D$
丙	乙是罪犯	$S3=B$
丁	乙有意诬陷我	$S4=\sim D$

由于4个人中只有一人说了真话，因此有如下4种可能，即列举出只有一个表达式为真的所有组合，其中&表示与运算，|表示或运算。

$$\begin{aligned}
& S1 \ \& \ \sim S2 \ \& \ \sim S3 \ \& \ \sim S4 \ | \\
& \sim S1 \ \& \ S2 \ \& \ \sim S3 \ \& \ \sim S4 \ | \\
& \sim S1 \ \& \ \sim S2 \ \& \ S3 \ \& \ \sim S4 \ | \\
& \sim S1 \ \& \ \sim S2 \ \& \ \sim S3 \ \& \ S4
\end{aligned}$$

在上面的表达式中，每行的表达式是一个与运算，而行之间是或运算。这种逻辑公式被称为析取范式(DNF)。然而SAT求解器只能对CNF表达式求解。一个满足CNF的布尔表达式由多个子表达式的与运算构成，而每个子表达式则由逻辑变量的或运算构成。因此需要使用另外的方法来表示S1、S2、S3、S4中只有一个为真：

$$\begin{aligned}
& \sim (S1 \ \& \ S2) \ \& \\
& \sim (S1 \ \& \ S3) \ \& \\
& \sim (S1 \ \& \ S4) \ \& \\
& \sim (S2 \ \& \ S3) \ \& \\
& \sim (S2 \ \& \ S4) \ \& \\
& \sim (S3 \ \& \ S4) \ \& \\
& \sim (\sim S1 \ \& \ \sim S2 \ \& \ \sim S3 \ \& \ \sim S4)
\end{aligned}$$

在上面的表达式中，列举S1、S2、S3、S4中所有的两两组合，任何一个组合都不可能同时为真，最后一个子句 $\sim(\sim S1 \ \& \ \sim S2 \ \& \ \sim S3 \ \& \ \sim S4)$ 表示4个供述不可能都为假。使用布尔公式： $\sim(A \ \& \ B) = \sim A \ | \ \sim B$ ，可得到下面左侧的CNF公式。把S1、S2、S3、S4转换成A、B、C、D之后得到右侧的表示该逻辑问题的CNF公式：

$\sim S1, \ \sim S2$	$A, \ \sim D$
$\sim S1, \ \sim S3$	$A, \ \sim B$
$\sim S1, \ \sim S4$	$A, \ D$

$\sim S2, \sim S3$	$\sim D, \sim B$
$\sim S2, \sim S4$	$\sim D, D$
$\sim S3, \sim S4$	$\sim B, D$
$S1, S2, S3, S4$	$\sim A, D, B, \sim D$

将DNF转换为CNF是一件比较麻烦的工作，可以借助SymPy的布尔代数模块自动转换。❶首先定义A到D共4个符号，❷并使用位操作符创建DNF表达式。❸然后调用to_cnf()将DNF表达式转换为CNF表达式。得到的结果比上面的手工转换结果更加简洁：

```
from sympy import symbols
from sympy.logic.boolalg import to_cnf

A, B, C, D = symbols("A:D") ❶
S1 = ~A
S2 = D
S3 = B
S4 = ~D
dnf = ((S1 & ~S2 & ~S3 & ~S4) |      ❷
       (~S1 & S2 & ~S3 & ~S4) |
       (~S1 & ~S2 & S3 & ~S4) |
       (~S1 & ~S2 & ~S3 & S4))

cnf = to_cnf(dnf) ❸
%sympy_latex cnf
```

$$A \wedge \neg B \wedge (A \vee D) \wedge (A \vee \neg B) \wedge (A \vee \neg D) \wedge (D \vee \neg B) \wedge (D \vee \neg D) \wedge (\neg B \vee \neg D)$$

使用`satisfiable()`可以对逻辑表达式进行推导，下面的结果显示逻辑变量A为真，即甲是罪犯：

```
from sympy.logic.inference import satisfiable
satisfiable(cnf)
{B: False, A: True, D: False}
```

下面使用本章将要介绍的`cycosat`扩展库对这个逻辑问题进行求解。CNF公式可以使用一个嵌套列表表示，列表中的每个整数与一个布尔变量对应，负数表示逻辑非。例如1与A对应，1表示A，而-1表示 $\sim A$ 。

CoSAT类是用Cython编写的扩展类，它对C语言编写的PicoSAT进行包装。❶调用`add_clauses()`将嵌套列表表示的CNF公式添加进求解器。可以多次调用`add_clauses()`逐步添加更多的表达式。❷调用`solve()`进行求解得到一个解列表，列表中的每个元素与一个布尔变量对应，1表示该布尔变量为真，-1表示假。由结果可知甲是罪犯。

```
from scpy2.cycosat import CoSAT

sat = CoSAT()
problem = [[1, -4], [1, -2], [1, 4], [-4, -2],
           [-4, 4], [-2, 4], [-1, 4, 2, -4]]

sat.add_clauses(problem) ❶
print sat.solve() ❷
[1, -1, -1, -1]
```


11.5.1 用Cython包装PicoSAT

PicoSAT采用C语言编写，其源代码只有两个文件：picosat.c和picosat.h。我们编写的Cython扩展库将对其中的如下函数进行包装：

```
#define PICOSAT_UNKNOWN          0
#define PICOSAT_SATISFIABLE      10
#define PICOSAT_UNSATISFIABLE    20

typedef struct PicoSAT PicoSAT;

PicoSAT * picosat_init (void);
void picosat_reset (PicoSAT *);
int picosat_add (PicoSAT *, int lit);
int picosat_add_lits (PicoSAT *, int * lits);
int picosat_sat (PicoSAT *, int decision_limit);
int picosat_variables (PicoSAT *);
int picosat_deref (PicoSAT *, int lit);
void picosat_assume (PicoSAT *, int lit);
```

PicoSAT求解器的所有状态都保存在PicoSAT结构体中，调用picosat_init()将返回一个指向新分配的结构体的指针，而picosat_res()则释放该指针所指向的结构体。

调用picosat_add()和picosat_add_lits()函数往结构体中添加逻辑子句。调用一次picosat_add()只能添加一条子句中的一个变量，而

`picosat_add_lits()` 则可添加整条子句，用0表示子句结束。例如添加子句1、-4，可以如下调用：

```
picosat_add(sat, 1);  
picosat_add(sat, -4);  
picosat_add(sat, 0);
```

或者：

```
int clause[3] = {1, -4, 0};  
picosat_add_lits(sat, clause);
```

调用`picosat_sat()`进行求解，返回`PICOSAT_SATISFIABLE`表示求解成功。调用`picosat_variables()`得到逻辑变量的个数，`picosat_deref()`得到第`lit`个逻辑变量的解，它返回1表示该逻辑变量取值为True，-1表示取值为False，0表示不确定。

下面在Cython中对`picosat.h`文件中定义的常量宏、结构体以及函数进行声明。由于不需要在Cython中存取PicoSAT结构体的字段内容，因此无须对其各个字段进行声明。

```
cdef extern from "picosat.h":  
    ctypedef enum:  
        PICOSAT_UNKNOWN  
        PICOSAT_SATISFIABLE  
        PICOSAT_UNSATISFIABLE  
  
    ctypedef struct PicoSAT:  
        pass
```

```

PicoSAT * picosat_init ()
void picosat_reset (PicoSAT *)
int picosat_add (PicoSAT *, int lit)
int picosat_add_lits(PicoSAT *, int * lits)
int picosat_sat (PicoSAT *, int decision_limit)
int picosat_variables (PicoSAT *)
int picosat_deref (PicoSAT *, int lit)
void picosat_assume (PicoSAT *, int lit)

```

我们使用下面的扩展类CoSAT对PicoSAT结构体进行包装。❶为了在Cython程序中快速重建PicoSAT结构体，这里采用clauses属性缓存将被添加进PicoSAT结构体中的所有子句。它是Python标准库array中的数组对象。由于array对象同时具有C语言数组的连续存储数值元素的功能，以及列表的动态扩容功能，因此很适合作为子句的缓存使用。在clauses中，每条子句都以0开始，以0结束，因此在初始化时为其添加一个元素0。

❷buf_pos属性保存clauses中已经添加进PocoSAT结构体的子句的最终位置，-1表示需要创建新的PicoSAT结构体。

```

from cpython cimport array

cdef class CoSAT:

    cdef PicoSAT * sat
    cdef public array.array clauses ❶
    cdef int buf_pos ❷

```

```

def __cinit__(self):
    self.buf_pos = -1
    self.clauses = array.array("i", [0])

def __dealloc__(self):
    self.close_sat()

cdef close_sat(self):
    if self.sat is not NULL:
        picosat_reset(self.sat)
        self.sat = NULL

```

在每次调用`picosat_sat()`求解之前，都需要将缓存中位于`buf_pos`之后的子句添加进PicoSAT结构体，这个工作由`build_clauses()`完成。^❶在Cython中可以通过`array`对象的`array.data.as_ints`属性获得指向数组对象的数据缓存区，从而通过C语言的指针对数组中的数据进行快速操作。

^❷而当`buf_pos`为-1时，需要重新创建PicoSAT结构体，这个工作由`build_sat()`完成。

```

cdef build_clauses(self):
    cdef int * p
    cdef int i
    cdef int count = len(self.clauses)
    if count - 1 == self.buf_pos:
        return

```

```

        p = self.clauses.data.as_ints ❶
        for i in range(self.buf_pos, count - 1):
            if p[i] == 0:
                picosat_add_lits(self.sat, p+i+1)
        self.buf_pos = count - 1

cdef build_sat(self): ❷
    if self.buf_pos == -1:
        self.close_sat()
        self.sat = picosat_init()
        if self.sat is NULL:
            raise MemoryError()
        self.buf_pos = 0
    self.build_clauses()

```

接下来是往缓存中添加单个子句的`add_clause()`和添加多个子句的`add_clauses()`，它们都是通过调用内部方法`_add_clause()`来实现的。

```

cdef _add_clause(self, clause):
    self.clauses.extend(clause)
    self.clauses.append(0)

def add_clause(self, clause):
    self._add_clause(clause)

def add_clauses(self, clauses):

```

```
for clause in clauses:
    self._add_clause(clause)
```

get_solution() 返回保存当前解的列表，❶首先调用picosat_variables() 获得布尔变量的个数，❷然后调用picosat_deref() 获得第i个变量的解。注意在PicoSAT中，布尔变量的序号从1开始，因此返回列表中下标为0的元素表示的是序号为1的布尔变量的解。

在solve() 中❸首先调用build_sat() 方法更新结构体，❹然后调用picosat_sat() 求解，若返回值为PICOSAT_SATISFIABLE，则返回get_solution() 得到的解。

```
def get_solution(self):
    cdef list solution = []
    cdef int i, v
    cdef int max_index

    max_index = picosat_variables(self.sat) ❶
    for i in range(max_index):
        v = picosat_deref(self.sat, i+1) ❷
        solution.append(v)
    return solution

def solve(self, limit=-1):
    cdef int res
    self.build_sat() ❸
    res = picosat_sat(self.sat, limit) ❹
```

```

        if res == PICOSAT_SATISFIABLE:
            return self.get_solution()

        elif res == PICOSAT_UNSATISFIABLE:
            return "PICOSAT_UNSATISFIABLE"

        elif res == PICOSAT_UNKNOWN:
            return "PICOSAT_UNKNOWN"

```

上面的solve()方法只能获得一个解，下面的iter_solve()返回一个能遍历所有解的迭代器。❶在每次获得解solution之后，添加一条否定该解的子句。例如，如果解为[-1, 1, 1, -1]，其含义为 $\sim B1 \ \& \ B2 \ \& \ B3 \ \& \ \sim B4$ ，其中 B_i 为第 i 个布尔变量。对该子句取反得到 $B1 \ \mid \ \sim B2 \ \mid \ \sim B3 \ \mid \ B4$ ，转换为PicoSAT的子句为：1, -2, -3, 4。❷由于iter_solve()修改了结构体的内容，无法将其还原到求解之前的状态，因此调用iter_reset()将buf_pos属性设置为-1，这样下次求解时，将重新创建新的PicoSAT结构体。

```

def iter_solve(self):
    cdef int res, i
    cdef list solution

    self.build_sat()

    while True:
        res = picosat_sat(self.sat, -1)

        if res == PICOSAT_SATISFIABLE:
            solution = self.get_solution()
            yield solution
            for i in range(len(solution)):
                picosat_add(self.sat, -solution[i] *

```

```

(i+1)) ❶

        picosat_add(self.sat, 0)
    else:
        break
    self.iter_reset()

def iter_reset(self): ❷
    self.buf_pos = -1

```

下面让我们看看如何使用CoSAT类解决两个较困难的逻辑题。

11.5.2 数独游戏

数独是一种数字填充游戏，玩家需要把从1到9的数字填进每一格，保证每行、每列和每个宫(3×3的方块)都有1至9所有数字。如图11-31所示，黑色数字为游戏设计者提供的部分数字，它们使该谜题只有一个答案，灰色数字显示该游戏的解答。

3	6	2	7	9	4	1	8	5
4	1	7	5	3	8	2	6	9
5	9	8	6	2	1	4	3	7
8	7	9	4	1	6	5	2	3
2	4	3	9	7	5	6	1	8
1	5	6	3	8	2	7	9	4
7	2	4	8	6	3	9	5	1
6	8	5	1	4	9	3	7	2
9	3	1	2	5	7	8	4	6

图11-31 数独游戏示例

由于SAT是一个布尔求解器，其中的每个变量只有两个候选值：False或True。为了表示数独的表格，我们需要一个三维数组bools，其第0轴对应数独表格的行，第1轴对应列，第2轴对应每个单元的候选数字。例如，如果在最终的解中，bools[4, 1, 3]对应的布尔变量为真，则表示数独表格中的第5行、第2列的值为4。注意这里数独游戏中的行、列以及数字都从1开始，而程序中所有的数组下标都从0开始。

下面的bools数组中保存用于SAT求解的布尔变量的序号，注意SAT中的布尔变量的序号从1开始：

```
bools = np.arange(1, 9 * 9 * 9 + 1).reshape(9, 9, 9)
```

根据数独的约束条件：

1) 每个单元只能填写一个数字，即bools中第2轴上的每组布尔变量只有一个为真。

2) 每行中不能有重复的数字，第1轴上的每组布尔变量只有一个为真。例如，bools[0, :, 2]为第一行中的每个数字为3的布尔变量，由于每行中有且只有一个数字为3，因此bools[0, :, 2]中只有一个为True。

3) 每列中不能有重复的数字：第0轴上的每组布尔变量只有一个为真。

4) 每块中不能有重复的数字：这个条件稍微复杂，需要对bools中变量的位置进行一些调整，稍后再做分析。

用一句话说明数独的约束条件就是：对布尔变量进行不同的分组，保证每个分组中只有一个布尔变量为真。下面我们看看如何用SAT表示一组布尔变量中只有一个为真这个条件。

SAT采用合取范式，其中每条表达式中的布尔变量采用或运算连接，而表达式之间采用与运算连接。我们需要使用这种逻辑表达式表示“一组布尔变量中只有一个为真”。例如，对于布尔变量A、B、C，下面两条表达式的与运算能表示A、B、C中只有一个为真：

1) $A \mid B \mid C$ ：表示A、B、C中至少有一个为真。

2) $\sim(A \& B) \& \sim(A \& C) \& \sim(B \& C)$: 任意两个布尔变量都不同时为真。

第二个条件可以重写为 $(\sim A \mid \sim B) \& (\sim A \mid \sim C) \& (\sim B \mid \sim C)$, 对应三条SAT的表达式, 因此最终的或与表达式为:

$$(A \mid B \mid C) \& (\sim A \mid \sim B) \& (\sim A \mid \sim C) \& (\sim B \mid \sim C)$$

根据上面的表达式, 我们需要从9个元素中取两个元素的所有组合, 这种组合运算可以使用 `itertools.combinations` 来实现。下面的 `get_conditions()` 返回使得 `bools` 中最终轴上的所有布尔数组只有一个为真的SAT表达式。

❶ 首先将从 N 个数中选取两个数的所有组合转换成一个二维数组 `index`, 其形状为 $(N*(N-1)/2, 2)$ 。❷ 创建第一个条件对应的逻辑表达式, 即最终轴上每组逻辑变量的序号。❸ 创建第二个条件对应的逻辑表达式, 即最终轴上每两个逻辑变量序号的负值。

```
from itertools import combinations

def get_conditions(bools):
    conditions = []
    n = bools.shape[-1]
    index = np.array(list(combinations(range(n), 2))) ❶
    # 以最后一轴为组
    # 第一个条件: 每组只能有一个为真
    conditions.extend(bools.reshape(-1, n).tolist()) ❷
    # 第二个条件: 每组中没有两个同时为真
```

```

        conditions.extend((-bools[..., index].reshape(-1,
2)).tolist()) ❸

    return conditions

```

下面是对于1、2、3和4、5、6这两组逻辑变量的运算结果：

```

print get_conditions(np.array([[1, 2, 3], [4, 5, 6]]))

[[1, 2, 3], [4, 5, 6], [-1, -2], [-1, -3], [-2, -3], [-4,
-5], [-4, -6], [-5, -6]]

```

下面使用`get_conditions()`计算数独的前三个约束条件，由于它只针对最终轴进行运算，因此对于“每行的数字不能重复”和“每列的数字不能重复”这两个条件，需要将条件对应的轴调换为最终轴。

```

c1 = get_conditions(bools)  # 每个单元格只能取1-9中的一个数字
c2 = get_conditions(np.swapaxes(bools, 1, 2))  # 每行的数字不能重复
c3 = get_conditions(np.swapaxes(bools, 0, 2))  # 每列的数字不能重复

```

对于约束条件4)，需要将每块的布尔变量调换为最终轴，这需要交替使用`reshape()`和`swapaxes()`。最后将`c1`、`c2`、`c3`、`c4`连接成一个列表`conditions`，就得到了数独游戏所有约束条件的SAT表达式。

```

tmp = np.swapaxes(bools.reshape(3, 3, 3, 3, 9), 1,
2).reshape(9, 9, 9)

c4 = get_conditions(np.swapaxes(tmp, 1, 2))  # 每块的数字不能重复

```

```
conditions = c1 + c2 + c3 + c4
```

最后使用CoSAT对数独游戏求解。❶solve()方法返回的是一个列表，首先将其还原成形状为(9, 9, 9)的数组，❷然后找到该数组中最终轴上1对应的下标，而实际应该填写的数字为该下标加1。

请读者仔细观察程序的输出，分析其是否满足数独游戏的约束条件。

```
def format_solution(solution):  
    solution = np.array(solution).reshape(9, 9, 9) ❶  
    return (np.where(solution == 1)[2] + 1).reshape(9, 9)  
❷  
  
sat = CoSAT()  
sat.add_clauses(conditions)  
solution = sat.solve()  
format_solution(solution)  
array([[9, 8, 7, 6, 5, 4, 3, 2, 1],  
       [6, 5, 4, 3, 1, 2, 9, 8, 7],  
       [3, 2, 1, 9, 8, 7, 6, 5, 4],  
       [8, 9, 6, 7, 4, 5, 2, 1, 3],  
       [7, 4, 5, 2, 3, 1, 8, 9, 6],  
       [2, 1, 3, 8, 9, 6, 7, 4, 5],  
       [5, 7, 9, 4, 6, 8, 1, 3, 2],  
       [4, 6, 8, 1, 2, 3, 5, 7, 9],  
       [1, 3, 2, 5, 7, 9, 4, 6, 8]])
```

下面用CoSAT和conditions解决一个实际的数独游戏。游戏的初始状态由sudoku_str指定，其中0表示需要填写数字的空白格。❶对于初始状态中的每个非0数字，都创建一个与其对应的布尔变量为真的布尔表达式，得到conditions2。❷该游戏的解同时满足conditions和conditions2条件。

```
sudoku_str = ""
00000185
007030000
000021400
800000020
003905600
050000004
004860000
000040300
931000000""

sudoku = np.array([[int(x) for x in line]
                    for line in sudoku_str.strip().split()])
r, c = np.where(sudoku != 0)
v = sudoku[r, c] - 1

conditions2 = [[x] for x in bools[r, c, v]] ❶
print "conditions2:"
conditions2
sat = CoSAT()
sat.add_clauses(conditions + conditions2) ❷
solution = sat.solve()
format_solution(solution)

conditions2:
[[55], [71], [77], [106], [120], [200],
 [208], [220], [251], [308], [345], [360],
 [374], [384], [419], [481], [508], [521],
 [528], [607], [624], [657], [660], [667]]
array([[3, 6, 2, 7, 9, 4, 1, 8, 5],
       [4, 1, 7, 5, 3, 8, 2, 6, 9],
       [5, 9, 8, 6, 2, 1, 4, 3, 7],
       [8, 7, 9, 4, 1, 6, 5, 2, 3],
       [2, 4, 3, 9, 7, 5, 6, 1, 8],
       [1, 5, 6, 3, 8, 2, 7, 9, 4],
       [7, 2, 4, 8, 6, 3, 9, 5, 1],
       [6, 8, 5, 1, 4, 9, 3, 7, 2],
       [9, 3, 1, 2, 5, 7, 8, 4, 6]])
```

我们注意到conditions2中的每条或表示式只包含一个布尔变量，对于这种简单的表达式，可以使用void picosat_assume(PicoSAT *, int lit)添加假设条件。其中lit为布尔变量的序号，正数表示假设该布尔变量为真，负数表示假。下面是CoSAT类的assume_solve()方法，它在调用solve()方法求解之前添加assumes指定的所有假设条件。下面是使用assume_solve()求解的程序：

```
def assume_solve(self, assumes):
    self.build_sat()
    for assume in assumes:
        picosat_assume(self.sat, assume)
    return self.solve()

sat = CoSAT()
sat.add_clauses(conditions)
solution = sat.assume_solve(bools[r, c, v].tolist())
format_solution(solution)
array([[3, 6, 2, 7, 9, 4, 1, 8, 5],
       [4, 1, 7, 5, 3, 8, 2, 6, 9],
       [5, 9, 8, 6, 2, 1, 4, 3, 7],
       [8, 7, 9, 4, 1, 6, 5, 2, 3],
       [2, 4, 3, 9, 7, 5, 6, 1, 8],
       [1, 5, 6, 3, 8, 2, 7, 9, 4],
       [7, 2, 4, 8, 6, 3, 9, 5, 1],
       [6, 8, 5, 1, 4, 9, 3, 7, 2],
       [9, 3, 1, 2, 5, 7, 8, 4, 6]])
```

使用`assume_solve()`的一个优点是在求解完成之后所有的假设条件将被清空，因此可以重复使用同一个`sat`对象对多个数独游戏求解。



`scipy2.examples.sudoku_solver`: 采用`matplotlib`制作的数独游戏求解器。

本书提供了一个交互式数独游戏求解器程序，可以使用如下键盘按键来操作：

- 使用方向键改变当前单元格的位置。
- 使用数字键1-9填写当前单元格，0清除当前单元格，用户输入的数字采用黑色显示，空白单元格的解采用浅灰色显示。
- 使用`Ctrl+E`清除所有单元格的内容。

11.5.3 扫雷游戏

扫雷游戏是Windows操作系统中最著名的附带游戏，玩家点开方块后如果没有地雷，则会有一个数字显现其上，这个数字为邻近的8个方块中的地雷数，玩家运用逻辑推断哪些方块有或没有地雷。

扫雷游戏也可以使用SAT进行求解。可以用一个逻辑变量表示每个方块是否有地雷，而每个已经打开的方块中的数字则对其周围地雷的分布提供了约束条件，可以使用SAT所需的或与逻辑表达式表示这些约束条件。与前面介绍的逻辑游戏不同的是，自动扫雷的目标不是找到

一组可能的解，而是要找到绝对不是雷的方块，从而进一步打开它们。

1. 识别雷区中的数字

下面让我们先使用两幅扫雷游戏的界面截图找出所有数字及其坐标，然后讨论如何使用SAT找到所有可以打开的方块的坐标。

mine_init.png为扫雷初始状态时的界面截图，mine01.png为打开了一些方格之后的截图，mine_numbers.png为已打开的方格的所有可能图像。

下面还定义了一些与捕捉的扫雷图像相关的数据：(X0, Y0)为雷区左上角像素的坐标，SIZE为每个正方形方格的边长，COLS为雷区中方格的列数，ROWS为行数。

- ❶通过np.s_可以定义一个切片元组以方便提取雷区部分的图像。
- ❷mine_numbers.png是由8幅 12×12 的方格图像在垂直方向上合并而成的，因此其形状为(8*12, 12, 3)。

```
import cv2

X0, Y0, SIZE, COLS, ROWS = 30, 30, 18, 30, 16

SHAPE = ROWS, SIZE, COLS, SIZE, -1

mine_area = np.s_[Y0:Y0 + SIZE * ROWS, X0:X0 + SIZE * COLS,
:] ❶
```

```
img_init = cv2.imread("mine_init.png")[mine_area]
img_mine = cv2.imread("mine01.png")[mine_area]
img_numbers = cv2.imread("mine_numbers.png") ②
img_numbers.shape
(96, 12, 3)
```

下面找出

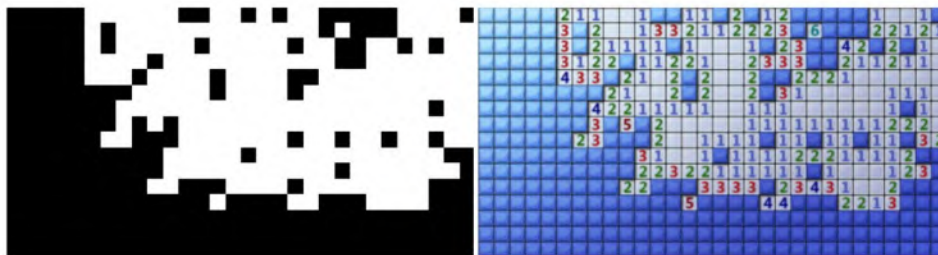


图11-32 计算已打开方块的位置



可以通过`pl.hist()`绘制mask_mean数组的直方图，找到最佳阈值。

图11-33 识别扫雷界面中的数字

```
from scipy.spatial import distance

img_mine2 = np.swapaxes(img_mine.reshape(SHAPE), 1, 2)

blocks = img_mine2[block_mask][:, 3:-3, 3:-3, :].copy()
blocks = blocks.reshape(blocks.shape[0], -1)

img_numbers.shape = 8, -1
numbers = np.argmin(distance.cdist(blocks, img_numbers),
axis=1)

rows, cols = np.where(block_mask)
```

下面用本书提供的draw_grid()函数绘制识别出的数字：

```
from scipy2.matplotlib import draw_grid

table = np.full((ROWS, COLS), u" ", dtype="unicode")
table[rows, cols] = numbers.astype(unicode)
draw_grid(table, fontsize=12)
```

2. 用SAT扫雷

根据方块周围的相邻方块数，可以分为如下三类：

- 角方块：位于4个角的方块，它们只有3个与之相邻的方块。
- 边方块：位于4边上的方块，它们有5个相邻的方块。

- 一般方块：其他的方块均有8个相邻的方块。

每个方块中的数字对其相邻方块中地雷的可能组合提供了约束条件，如果能用或与表达式表示所有这些约束条件，就可以用SAT对扫雷问题进行求解。下面以8个相邻方块为例，介绍如何用合取范式表示这8个方块中有三个地雷。

我们用序号为1到8的逻辑变量表示某个方块周围的8个方块中是否有雷。可以用下面的逻辑表示8个逻辑变量中有3个为真：

- 任意取4个变量，这4个变量不可能同时为真。例如，如果选择A、B、C、D这4个变量，则 $\sim(A \ \& \ B \ \& \ C \ \& \ D)$ 表示它们不全为真，展开之后得到 $\sim A \mid \sim B \mid \sim C \mid \sim D$ 。
- 任意取6个变量，至少有一个为真。如果用A、B、C、D、E、F表示这6个变量，则 $A \mid B \mid C \mid D \mid E \mid F$ 表示它们中至少有一个为真。

上面所有条件之间都用与运算连接，而每个子句都是用或运算表示，因此整个表达式为合取范式。

下面的程序中用`combinations()`穷举所有4个变量和6个变量的组合，并添加相应的逻辑表达式。可以看到解中包括3个1，表示有三个逻辑变量为真，即有三个地雷。如果调用`sat.iter_solve()`，可穷举所有三个地雷的组合。

```
variables = range(1, 9)

from itertools import combinations

clauses = []
```

```

for vs in combinations(variables, 4):
    clauses.append([-x for x in vs])

for vs in combinations(variables, 6):
    clauses.append(vs)

sat = CoSAT()
sat.add_clauses(clauses)
sat.solve()
[-1, -1, -1, -1, -1, 1, 1, 1]

```

在生成用于SAT求解的逻辑表达式之前，我们先为每个方格设置一个逻辑变量的编号。编号从左上角的方格为1开始，右下角的方格的编号为COLS * ROWS。根据下面的算式可以将编号转换为方格所在的行和列：

- 行号 = (编号- 1) // COLS
- 列号 = (编号- 1) % COLS

二维数组 variables 中保存逻辑变量的编号，而 variable_neighbors字典保存与每个方格编号相邻的方格的编号。

```

from collections import defaultdict

variable_neighbors = defaultdict(list)

directs = [(-1, -1), (-1, 0), (-1, 1), (0, -1),
           (0, 1), (1, -1), (1, 0), (1, 1)]

```

```

variables = np.arange(1, COLS * ROWS + 1).reshape(ROWS,
COLS)

for (i, j), v in np.ndenumerate(variables):
    for di, dj in directs:
        i2 = i + di
        j2 = j + dj
        if 0 <= i2 < ROWS and 0 <= j2 < COLS:
            variable_neighbors[v].append(variables[i2, j2])

```

下面查看与编号为50的方格相邻的方格的编号：

```

variable_neighbors[50]
[19, 20, 21, 49, 51, 79, 80, 81]

```

用variable_neighbors字典以及前面介绍的生成逻辑表达式的方法，很容易编写如下get_clauses(var_id, num)函数。它返回编号为var_id的方块中的数字为num时的逻辑表达式。

```

def get_clauses(var_id, num):
    clauses = []
    neighbors = variable_neighbors[var_id]
    neg_neighbors = [-x for x in neighbors]
    clauses.extend(combinations(neg_neighbors, num + 1))
    clauses.extend(combinations(neighbors, len(neighbors) -
num + 1))

```

```
clauses.append([-var_id])

return clauses
```

扫雷游戏与前面介绍的逻辑问题不同，它并不是要找到一组可能的解，而是要找到绝对是雷以及绝对不是雷的方块。下面是CoSAT中实现这一功能的`get_failed_assumes()`方法。它对每个逻辑变量进行循环并调用`picosat_assume()`，假设该逻辑变量为假或为真，并分别调用`picosat_sat()`计算实施假设之后是否有解。如果无解，则表示该逻辑变量不可能为假或真。使用`picosat_assume()`的好处是每次调用`picosat_sat()`之后假设将自动被重置。

`get_failed_assumes()`返回所有失败的假设，因此返回值为负数表示该逻辑变量必须为真，而正数表示必须为假。

```
def get_failed_assumes(self):
    cdef int max_index
    cdef int ret1, ret0
    cdef list assumes = []
    self.build_sat()
    max_index = picosat_variables(self.sat)
    for i in range(1, max_index+1):
        picosat_assume(self.sat, i)
        ret1 = picosat_sat(self.sat, -1)
        picosat_assume(self.sat, -i)
        ret0 = picosat_sat(self.sat, -1)
        if ret0 == PICOSAT_UNSATISFIABLE:
            assumes.append(-i)
        if ret1 == PICOSAT_UNSATISFIABLE:
```



```

        assumes.append(i)

    return assumes

```

最后对所有已打开的方格计算逻辑表达式，并添加进SAT求解器，然后利用 `get_failed_assumes()` 找到所有失败的假设 `failed_assumes`。对于其中未打开的方格，正数表示不是雷，用“★”表示；负数表示是雷，用“●”表示。

				●	2	1	1	0	0	1	●	●	1	1	●	2	●	1	2	●								1	0	0	1	●
				★	3	●	2	0	0	1	3	3	2	1	1	2	2	2	3	●	6	●					2	2	1	2	1	
				★	3	●	2	1	1	1	1	●	1	0	0	0	1	●	2	3	●	●	4	2	●	2	●	1	0			
				●	3	1	2	2	●	1	1	2	2	1	0	0	2	3	3	3	●	●	2	1	1	2	1	1	0			
				●	4	3	3	●	2	1	0	2	●	2	0	0	2	●	●	2	2	2	1	0	0	0	0	0	0			
				★	●	●	●	2	1	0	0	2	●	2	0	0	2	●	3	1	0	0	0	0	0	1	1	1	0			
								4	2	2	1	1	1	1	1	0	0	1	1	1	0	0	0	0	0	1	●	1	0			
								3	●	5	●	2	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	0		
								2	3	●	●	●	2	0	0	1	1	1	1	●	1	1	●	1	1	●	1	1	3	2		
												3	1	0	0	1	●	1	1	1	1	2	2	2	1	1	1	2	●	●		
												2	2	3	2	2	1	1	1	1	1	●	1	0	0	0	1	2	3	★		
												2	2	●	●	●	3	3	3	3	●	2	3	4	3	1	0	0	2	●	★	
																5	●	●	●	●	4	4	●	●	●	2	2	1	3	●		
																									★	●	★	●	★	★		

图11-34 使用SAT求解器推断方格中是否有地雷

```

sat = CoSAT()

for var_id, num in zip(variables[rows, cols], numbers):
    sat.add_clauses(get_clauses(var_id, num))

failed_assumes = sat.get_failed_assumes()

for v in failed_assumes:
    av = abs(v)

    col = (av - 1) % COLS
    row = (av - 1) // COLS

```

```
if table[row, col] == u" ":
    if v > 0:
        table[row, col] = u"★"
    else:
        table[row, col] = u"●"
draw_grid(table, fontsize=12)
```

3. 自动扫雷

本书提供了一个自动扫雷实例程序，它能够自动启动扫雷游戏，并自动操纵鼠标进行扫雷。



scpy2.examples.automine: 在Windows 7系统下自动扫雷，需将扫雷游戏的难度设置为高级(99个雷)，并且关闭“显示动画”、“播放声音”以及“显示提示”等选项。

当程序无法确定可以打开的方块时，将随机点开任意的方块。为了实现更高的成功率，读者可以尝试在剩余非常少的雷时，穷举所有的可能解，并计算每个方块是雷的概率，选择点击概率最小的那个方块。

除了本节介绍的cycosat扩展库之外，程序中还使用了如下扩展库：

- `scipy2.utils.autoit`: 使用`ctypes`标准库对AutoIt提供的DLL进行包装，提供了自动扫雷所需的基本Windows自动化功能。
- `win32gui`: 获取游戏界面所在的位置。
- `pillow`: 读取PNG图像文件以及捕捉屏幕中的游戏界面。

11.6 分形



与本节内容对应的Notebook为: `11-examples/examples-600-fractal.ipynb`。

自然界中的很多事物，例如树木、云彩、山脉、闪电、雪花以及海岸线等都呈现出传统几何学难以描述的形状。这些形状都有如下特性：

- 有着十分精细的不规则的结构。
- 整体与局部相似，例如一根树杈的形状和一棵树很像。

分形几何学就是用来研究这样一类几何形状的科学，借助计算机的高速计算和图像显示，我们可以更深入、更直观地研究分形几何。作为本书的最后一节，我们看看如何使用Python绘制一些经典的分形图形。

11.6.1 Mandelbrot集合

Mandelbrot (曼德布洛特) 集合是在复平面上构成分形图案的点的集合。它可以用下面的复二次多项式定义：

$$f_c(z) = z^2 + c$$

其中复数函数 $f_c(z)$ 的自变量为 z 。而 c 是一个复数参数，对于每一个 c ，从 $z=0$ 开始对函数 $f_c(z)$ 进行迭代。序列 $(0, f_c(0), f_c(f_c(0)), \dots)$ 的值或者延伸到无限大，或者只停留在有限半径的圆盘内。Mandelbrot集合就是使以上序列不发散的所有参数 c 的集合。

从数学上来讲，Mandelbrot集合是一个复数的集合。一个给定的复数 c 或者属于Mandelbrot集合，或者不是。但是用程序绘制Mandelbrot集合时不能进行无限次迭代，最简单的方法是使用逃逸时间(迭代次数)进行绘制，具体算法如下：

- 判断每次调用函数 $f_c(z)$ 得到的结果是否在半径 R 之内，即复数的模是否小于 R 。
- 记录下迭代结果的模值大于 R 时的迭代次数，也称之为逃逸时间。
- 迭代最多进行 N 次。
- 不同迭代次数的点使用不同的颜色绘制。

1. 纯Python实现

下面是完整的绘制Mandelbrot集合的程序，它所绘制的图案如图11-35所示。

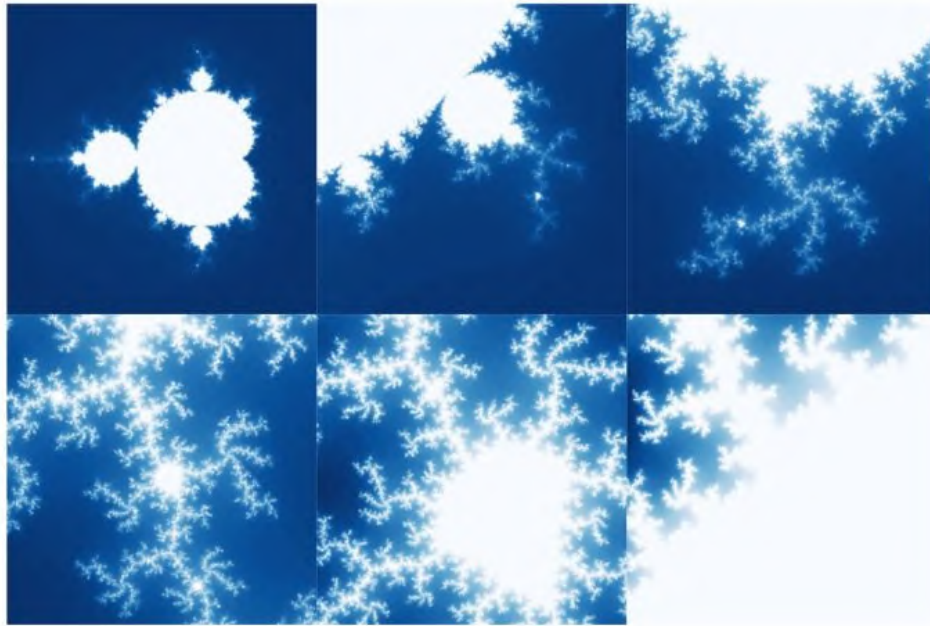


图11-35 Mandelbrot集合，以5倍的倍率放大点(0.273, 0.595)附近

```
from matplotlib import cm

def iter_point(c): ❶
    z = c
    for i in xrange(1, 100): # 最多迭代100次
        if abs(z) > 2: break # 半径大于2则认为逃逸
        z = z * z + c
    return i # 返回迭代次数

def mandelbrot(cx, cy, d, n=200):
    x0, x1, y0, y1 = cx-d, cx+d, cy-d, cy+d
    y, x = np.ogrid[y0:y1:n*1j, x0:x1:n*1j]
    c = x + y*1j ❸

    return np.frompyfunc(iter_point, 1, 1)
```

```

(c).astype(np.float) ❹

def draw_mandelbrot(cx, cy, d, n=200): ❷
    """
    绘制点(cx, cy)附近正负d范围内的Mandelbrot
    """
    pl.imshow(mandelbrot(cx, cy, d, n), cmap=cm.Blues_r)
❸

    pl.gca().set_axis_off()

x, y = 0.27322626, 0.595153338

pl.figure(figsize=(9, 6))
pl.subplot(231)
draw_mandelbrot(-0.5, 0, 1.5)
for i in range(2,7):
    pl.subplot(230+i)
    draw_mandelbrot(x, y, 0.2**(i-1))
pl.subplots_adjust(0, 0, 1, 1, 0.0, 0)

```

❶函数`iter_point()`计算点`c`的逃逸时间，逃逸半径`R`为2.0，最大迭代次数为100。❷`draw_mandelbrot()`绘制以点`(cx, cy)`为中心，边长为`2*d`的正方形区域内的Mandelbrot集合的图案。

❸计算指定范围内的参数`c`，它是一个二维的复数数组，形状为`(200, 200)`。这里用`ogrid`对象快速产生实部和虚部网格`x`和`y`，然后通过广播运算得到数组`c`。

④接下来通过`frompyfunc()`将`iter_point()`转换为ufunc函数，这样它可以自动对`c`中的每个元素调用`iter_point()`进行运算。由于结果数组的元素类型为`object`，还需要调用`astype()`将其元素类型转换为浮点数类型。⑤最后调用`imshow()`将结果数组绘制成图，通过关键字参数`cmap`指定颜色映射表。

2. 用Cython提速

使用Python绘制Mandelbrot集合，最大的问题就是运算速度太慢：

```
%timeit mandelbrot(-0.5, 0, 1.5)
1 loops, best of 3: 398 ms per loop
```

而由于`iter_point()`函数中存在迭代，无法将其转换成NumPy的数组运算。下面我们用Cython重新编写`iter_point()`。

首先为Python的`iter_point()`中的各个变量——`c`、`z`、`i`添加类型声明，然后调用`%timeit`查看运行速度，发现速度提高得不是很明显。用`%%cython -a`查看编译之后的C语言源代码，你会发现`abs(z)`调用Python的`abs()`函数，该函数限制了运行速度。

```
%%cython
def iter_point(complex c):
    cdef complex z = c
    cdef int i
    for i in range(1, 100):
```

```
        if z.real*z.real + z.imag*z.imag > 4: break ❶  
        z = z * z + c  
    return i
```

❶将计算复数绝对值的代码修改为实数部分的平方与虚数部分的平方之和后，运行速度提高了近40倍：

```
%timeit mandelbrot(-0.5, 0, 1.5)  
100 loops, best of 3: 8.89 ms per loop
```

3. 连续的逃逸时间

修改逃逸半径R和最大迭代次数N，可以绘制出不同效果的Mandelbrot集合图案。但是前述方法计算出的逃逸时间是大于逃逸半径时的迭代次数，因此输出图像最多只有N种不同的颜色值，有很强的梯度感。为了在不同的梯度之间进行渐变处理，可是使用下面的公式计算逃逸时间：

$$n - \log_2 \log_2 |z_n|$$

z_n 是迭代n次之后的结果，通过在逃逸时间的计算中引入迭代结果的模值，结果将不再是整数，而是平滑渐变的。下面是计算此逃逸时间的程序：

```
%%cython  
from libc.math cimport log2
```



```

def iter_point(complex c):
    cdef complex z = c
    cdef int i
    cdef double r2, mu
    for i in range(1, 20):
        r2 = z.real*z.real + z.imag*z.imag
        if r2 > 100: break
        z = z * z + c
    if r2 > 4.0:
        mu = i - log2(0.5 * log2(r2))
    else:
        mu = i
    return mu

```

如果逃逸半径设置得很小，例如2.0，那么有可能结果不够平滑，这时可以在迭代循环之后添加几次迭代，这能保证z的模值足够大。例如：

```

z = z * z + c
z = z * z + c
i += 2

```

图11-36是逃逸半径为10、最大迭代次数为20的结果。



图11-36 平滑处理后的Mandelbrot集合：逃逸半径=10，最大迭代次数=20

```
pl.figure(figsize=(8, 8))  
draw_mandelbrot(-0.5, 0, 1.5, n=600)
```

4. Mandelbrot演示程序

为了实时计算Mandelbrot集合的图像，我们需要更快的运算速度，可以将所有的循环都使用Cython编写。本书提供了用matplotlib制作的实时绘制Mandelbrot集合的演示程序，界面截图如图11-37所示。

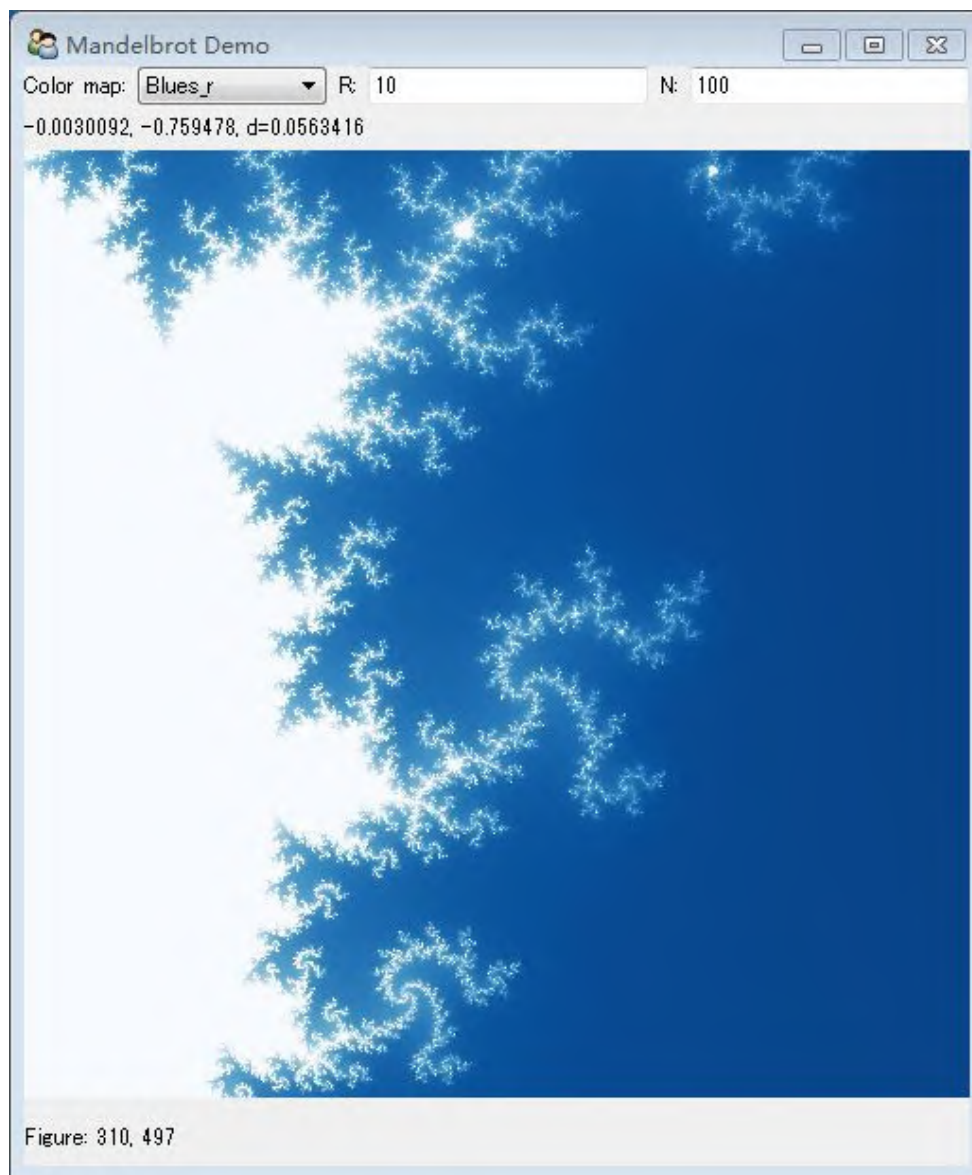


图11-37 实时绘制Mandelbrot集合的演示程序



`scipy2.examples.fractal.mandelbrot_demo` : 使用 TraitsUI 和 matplotlib 实时绘制 Mandelbrot 图像，按住鼠标左键进行平移，使用鼠标滚轴进行缩放。

下面是计算Mandelbrot集合图像的Cython函数。该函数所在模块的完整路径为scipy2.examples.fractal.fastfractal。参数cx和cy是复平面上的计算范围的中心点，参数d是中心点到计算边界的实轴上的长度，参数out是保存计算结果的二维数组。如果不指定out，则需通过h和w参数指定返回数组的大小。参数n为最大迭代次数，R为逃逸半径。

```
from libc.math cimport log2
import numpy as np
import cython

cdef double iter_point(complex c, int n, double R):
    cdef complex z = c
    cdef int i
    cdef double r2, mu
    cdef double R2 = R*R
    for i in range(1, n):
        r2 = z.real*z.real + z.imag*z.imag
        if r2 > R2: break
        z = z * z + c
    if r2 > 4.0:
        mu = i - log2(0.5 * log2(r2))
    else:
        mu = i
    return mu

def mandelbrot(double cx, double cy, double d, int h=0, int
```

```

w=0,
                                double[:, ::1] out=None, int n=20, double
R=10.0):
    cdef double x0, x1, y0, y1, dx, dy
    cdef double[:, ::1] r
    cdef int i, j
    cdef complex z
    x0, x1, y0, y1 = cx - d, cx + d, cy - d, cy + d
    if out is not None:
        r = out
    else:
        r = np.zeros((h, w))
    h, w = r.shape[0], r.shape[1]
    dx = (x1 - x0) / (w - 1)
    dy = (y1 - y0) / (h - 1)
    for i in range(h):
        for j in range(w):
            z.imag = y0 + i * dy
            z.real = x0 + j * dx
            r[i, j] = iter_point(z, n, R)
    return r.base

```

11.6.2 迭代函数系统

迭代函数系统是一种创建分形图案的简单算法，它所创建的分形图永远是绝对自相似的。下面以绘制某种蕨类植物叶子的图案为例，

介绍迭代函数系统算法以及如何用Python实现。

有下面4组线性函数用于对二维平面上的坐标进行线性变换：

1.

$$x(n+1) = 0$$

$$y(n+1) = 0.16 * y(n)$$

2.

$$x(n+1) = 0.2 * x(n) - 0.26 * y(n)$$

$$y(n+1) = 0.23 * x(n) + 0.22 * y(n) + 1.6$$

3.

$$x(n+1) = -0.15 * x(n) + 0.28 * y(n)$$

$$y(n+1) = 0.26 * x(n) + 0.24 * y(n) + 0.44$$

4.

$$x(n+1) = 0.85 * x(n) + 0.04 * y(n)$$

$$y(n+1) = -0.04 * x(n) + 0.85 * y(n) + 1.6$$

所谓迭代函数系统，是指将函数的输出再次当作输入进行迭代计算，因此上面的公式都是通过坐标 $x(n)$ ， $y(n)$ 计算变换后的坐标 $x(n+1)$ ， $y(n+1)$ 。问题是有4个迭代函数，迭代时选择哪个函数进行计算呢？我们为每个函数指定一个概率值，它们依次为1%、7%、7%和85%。通过每个函数的概率随机选择一个函数进行迭代。在上面的例子中，第4个函数被选择进行迭代的概率最高。

最后我们从坐标原点(0, 0)开始迭代，绘制每次迭代后得到的坐标点，就得到了迭代函数系统的分形图案。下面的程序演示了这一计算过程：

```

%config InlineBackend.figure_format = 'png'
eq1 = np.array([[0,0,0],[0,0.16,0]])
p1 = 0.01

eq2 = np.array([[0.2,-0.26,0],[0.23,0.22,1.6]])
p2 = 0.07

eq3 = np.array([[-0.15, 0.28, 0],[0.26,0.24,0.44]])
p3 = 0.07

eq4 = np.array([[0.85, 0.04, 0],[-0.04, 0.85, 1.6]])
p4 = 0.85

```

```
def ifs(p, eq, init, n):
```

```
    """
```

进行函数迭代

p: 每个函数的选择概率列表

eq: 迭代函数列表

init: 迭代初始点

n: 迭代次数

返回值: 每次迭代所得的x坐标数组、y坐标数组, 计算所用的函数下标

```
    """
```

```
    # 迭代向量的初始化
```

```
    pos = np.ones(3, dtype=np.float) ❶
```

```
    pos[:2] = init
```

```

# 通过函数概率，计算函数的选择序列

p = np.cumsum(p)
rands = np.random.rand(n)
select = np.searchsorted(p, rands) ❷

# 结果的初始化

result = np.zeros((n,2), dtype=np.float)
c = np.zeros(n, dtype=np.float)

for i in xrange(n):
    eqidx = select[i] # 所选的函数下标
    tmp = np.dot(eq[eqidx], pos) # 进行迭代
    pos[:2] = tmp # 更新迭代向量

# 保存结果

result[i] = tmp
c[i] = eqidx

return result[:,0], result[:, 1], c

x, y, c = ifs([p1,p2,p3,p4], [eq1,eq2,eq3,eq4], [0,0],
100000)

fig, axes = pl.subplots(1, 2, figsize=(6, 5))
axes[0].scatter(x, y, s=1, c="g", marker="s", linewidths=0)
❸
axes[1].scatter(x, y, s=1, c=c, marker="s", linewidths=0)

```


4

```

for ax in axes:
    ax.set_aspect("equal")
    ax.set_ylim(0, 10.5)
    ax.axis("off")

pl.subplots_adjust(left=0, right=1, bottom=0, top=1, wspace=0, hspace=0)

```

`ifs()` 是进行函数迭代的主函数，❶我们希望通过矩阵乘法计算迭代方程的输出，因此需要将乘法向量扩充为三维：这样每次和迭代函数系数进行矩阵乘积运算的向量就变成了 $x(n)$, $y(n)$, 1.0。

❷为了减少计算时间，不在迭代循环中计算随机数选择迭代方程，而是事先通过每个函数的概率，计算出函数选择数组 `select`。注意这里使用 `cumsum()` 先将概率累加，然后产生一组0到1之间的随机数，通过判断随机数所在的概率区间选择不同的方程下标。

❸最后调用 `scatter()` 将得到的坐标绘制成散列图，其中每个关键字参数的含义如下：

- `s`: 每个散列点的大小，因为要绘制10万个点，为了提高绘图速度，我们选择点的大小为1个像素。
- `c`: 点的颜色，这里选择绿色。
- `marker`: 点的形状，“s”表示正方形，方形的绘制是最快的。
- `linewidths`: 点的边框宽度，0表示没有边框。

④此外，参数c还可以传入一个数组，作为每个点的颜色值。我们将迭代用的函数下标传入，这样可以直观地看出哪个点是哪个函数迭代产生的。

图11-38是程序的输出，观察右图的4种颜色的部分可以发现：概率为1%的函数1所计算的是叶杆部分(深蓝色)，概率为7%的两个函数计算的是左右两片子叶，而概率为85%的函数计算的是整片子叶的迭代，即最下面的三种颜色的点通过此函数的迭代产生上面所有的深红色的点。可以看出整片叶子呈现出完美的自相似特性，任意取其中的一片子叶，将其旋转放大之后都和整片叶子相同。



图11-38 函数迭代系统所绘制的蕨类植物的叶子

1. 2D仿射变换

上面所介绍的4个变换方程的一般形式如下：

$$x(n+1) = A * x(n) + B * y(n) + C$$

$$y(n+1) = D * x(n) + E * y(n) + F$$

这种变换被称为2D仿射变换，它是从2D坐标到其他2D坐标的线性映射，保留直线性和平行性。即原来是一条直线上的点，变换之后仍然在一条直线上，原来是平行的直线，变换之后仍然是平行的。这种变换可以看作是由一系列平移、缩放、翻转和旋转变换构成的。

可以使用平面上的两个三角形直观地表示仿射变换。因为仿射变换公式中有6个未知数——A、B、C、D、E、F，而每两个点之间的变换是两个方程，因此一共需要3组点来决定6个变换方程，正好是两个三角形，如图11-39所示：

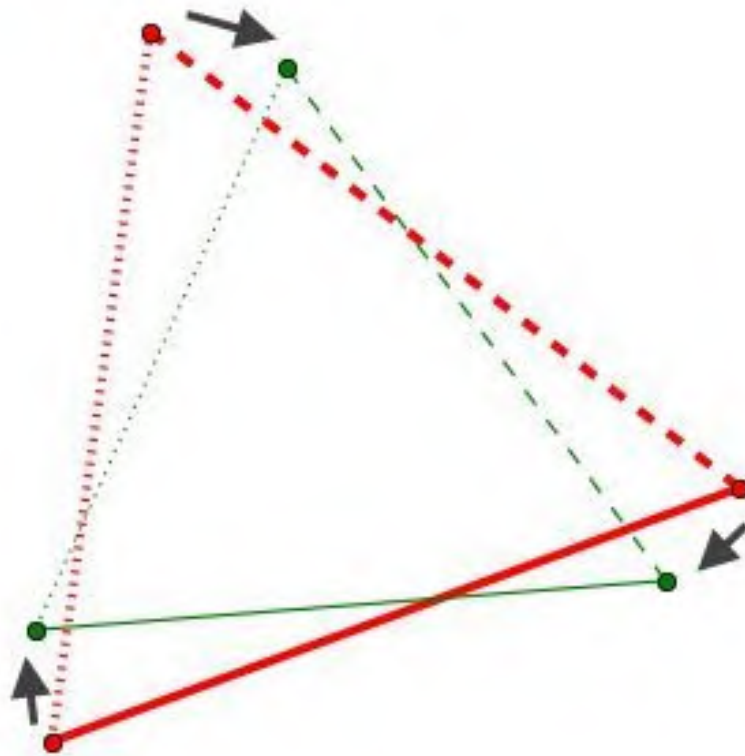


图11-39 两个三角形决定一个2D仿射变换的6个参数

从红色三角形的每个顶点变换到绿色三角形的对应顶点，正好能够决定仿射变换中的6个参数。这样我们可使用 $N+1$ 个三角形，决定 N 个仿射变换，其中的每个变换的参数都是由第0个三角形和其他的三角形决定的。第0个三角形被称为基础三角形，其余的三角形被称为变换三角形。

为了绘制迭代函数系统的图像，还需要给每个仿射变换方程指定迭代概率。此参数也可以使用三角形直观地表达出来：迭代概率和变换三角形的面积成正比，即迭代概率为变换三角形的面积除以所有变换三角形的面积之和。

如图11-40所示，前面介绍的蕨类植物的分形图案的迭代方程由5个三角形决定，可以很直观地看出紫色的小三角形决定了叶子的茎；

而两个蓝色的三角形决定了左右两片子叶；绿色的三角形将茎和两片子叶往上复制，形成整片叶子。

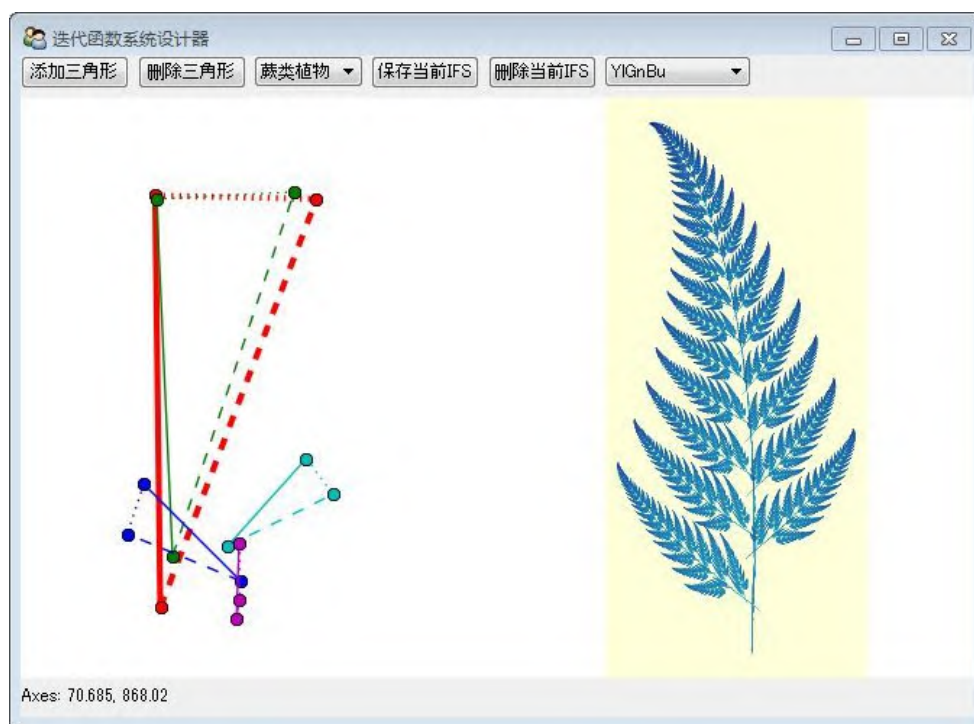


图11-40 5个三角形的仿射方程绘制蕨类植物的叶子

2. 迭代函数系统设计器

按照上节所介绍的三角形法，我们可以编写一个迭代函数系统的设计工具。用户通过程序界面绘制或修改一组三角形，程序计算这组三角形所对应的迭代方程组的系数，并实时地绘制迭代图案。图11-40是本书提供的设计迭代函数分形系统的程序界面截图。



scipy2.examples.fractal.ifs_demo: 迭代函数分形系统的演示程序，通过修改左侧三角形的顶点实时地计算坐标变换矩阵，并在右侧显示迭代结果。

下面简要地介绍该演示程序中用到的一些函数和类。首先通过两个三角形求解仿射方程的系数，相当于求六元线性方程组的解，这个计算通过solve_eq()完成，它先计算出线性方程组的矩阵a和b，然后调用NumPy的linalg.solve()对线性方程组 $a \cdot x = b$ 求解：

```
def solve_eq(triangle1, triangle2):  
    """  
    解方程，从triangle1变换到triangle2的变换系数  
    triangle1、triangle2是二维数组：  
    x0,y0  
    x1,y1  
    x2,y2  
    """  
    x0, y0 = triangle1[0]  
    x1, y1 = triangle1[1]  
    x2, y2 = triangle1[2]  
  
    a = np.zeros((6, 6), dtype=np.float)  
    b = triangle2.reshape(-1)  
    a[0, 0:3] = x0, y0, 1  
    a[1, 3:6] = x0, y0, 1
```

```

a[2, 0:3] = x1, y1, 1
a[3, 3:6] = x1, y1, 1
a[4, 0:3] = x2, y2, 1
a[5, 3:6] = x2, y2, 1

x = np.linalg.solve(a, b)
x.shape = (2, 3)
return x

```

每个仿射方程的迭代概率与对应三角形的面积成正比，三角形的面积通过`triangle_area()`计算，它使用NumPy的`cross()`计算三角形的两个边的矢量的叉积：

```

def triangle_area(triangle):
    """
    计算三角形的面积
    """
    A, B, C = triangle
    AB = A - B
    AC = A - C
    return np.abs(np.cross(AB, AC)) / 2.0

```

绘图界面采用matplotlib绘图库，由于绘制大量散列点会导致界面刷新速度变慢，因此在本演示程序中对迭代生成的坐标点进行二维直方图统计，并使用`imshow()`绘制统计结果。为了提高程序运行速度，方程迭代以及二维直方图统计均在Cython编写的IFS扩展类中实现。下面通过一个例子说明这些函数的用法。IFS扩展类的源代码可以在`fastfractal.pyx`中找到。

❶在下面的triangles中保存着3个三角形的顶点坐标，其中第一个三角形为基础三角形，后两个三角形为变换三角形。❷调用triangle_area()计算每个变换三角形的面积，并用面积和归一化得到每个三角形的迭代概率p。❸调用solve_eq()得到从基础三角形到变换三角形的仿射变换矩阵，并将所有反射变换矩阵按照第0轴连接成一个形状为(4, 3)的数组eqs。数组中的每两行表示一个迭代方程的系数。

❹创建IFS()对象，其前两个参数分别为三角形的迭代概率和迭代系数，第3个参数为每次调用update()方法的迭代次数，size参数为二维直方图统计结果数组的长轴的长度。❺每次调用update()方法都迭代指定的次数，并返回更新后的直方图统计结果。counts是一个形状为(600, 477)的整数数组。❻为了更清晰地显示统计结果，这里使用对数正规化对象LogNorm对counts中的值进行正规化。由于0的对数值为负无穷，因此counts中保存的值实际上是直方图统计值加1。程序的输出如图11-41所示。

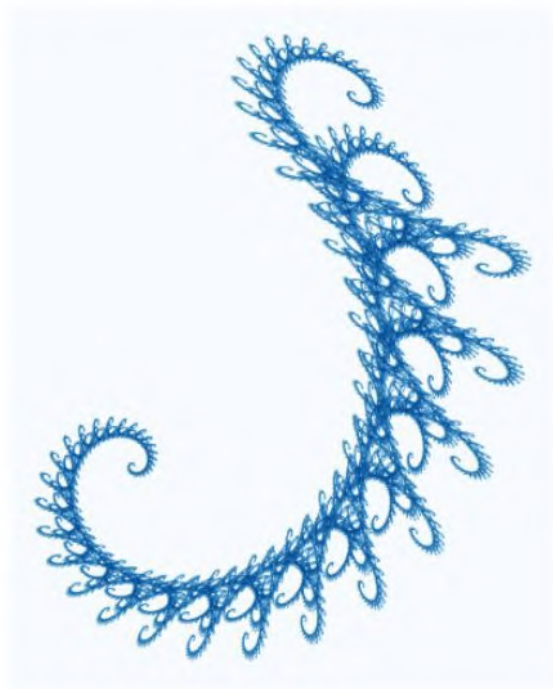


图11-41 使用IFS类绘制迭代函数系统

```
from scipy2.examples.fractal.ifs_demo import solve_eq,
triangle_area

from scipy2.examples.fractal.fastfractal import IFS

triangles = np.array([
    [-1.945392491467576, -5.331010452961673],
    [6.109215017064848, -0.8710801393728236],
    [-1.1945392491467572, 5.400696864111497],
    [-2.5597269624573373, -4.21602787456446],
    [5.426621160409557, -2.125435540069687],
    [0.5119453924914676, 4.912891986062718],
    [3.5836177474402735, 8.397212543554005],
    [4.0614334470989775, 5.121951219512194],
    [8.56655290102389, 4.7038327526132395]])

base_triangle = triangles[:3]
triangle1 = triangles[3:6]
triangle2 = triangles[6:]

area1 = triangle_area(triangle1)
area2 = triangle_area(triangle2)
total_area = area1 + area2
p = [area1 / total_area, area2 / total_area]

eq1 = solve_eq(base_triangle, triangle1)
```

```

eq2 = solve_eq(base_triangle, triangle2)
eqs = np.vstack([eq1, eq2])

ifs = IFS(p, eqs, 2000000, size=600) ❹
counts = ifs.update() ❺
print "shape of counts:", counts.shape
from matplotlib.colors import LogNorm
fig, ax = plt.subplots(figsize=(5, 8))
        plt.imshow(counts, cmap="Blues", norm=LogNorm(),
origin="lower") ❻
        ax.axis("off")
        shape of counts: (600, 477)

```

11.6.3 L-System分形

前面所绘制的分形图案都是使用数学函数的迭代产生的，而L-System分形则采用符号的递归迭代产生。首先定义如下几个有含义的符号：

- F：向前走固定单位
- +：正方向旋转固定角度
- -：负方向旋转固定角度

使用这三个符号很容易描述图11-42中左上方由4条线段构成的图案：

F+F--F+F

如果将此符号串中的所有F都替换为F+F--F+F，就能得到如下新字符串：

F+F--F+F+F+F--F+F--F+F--F+F+F+F--F+F

如此替换迭代下去，并根据字串进行绘图(符号+和-分别正负旋转60度)，可得到如图11-42右下方的分形图案：

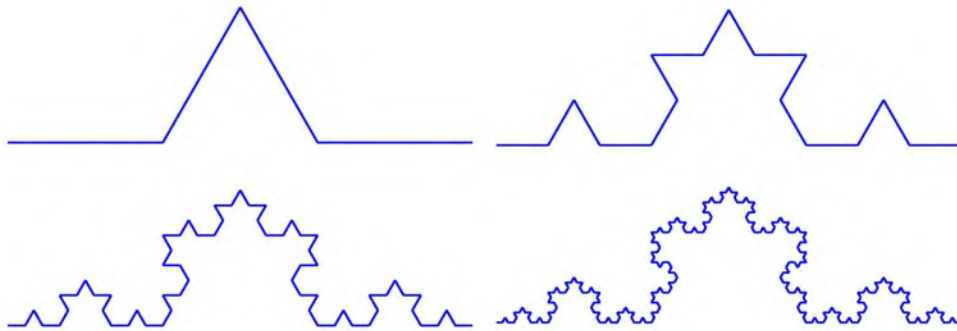


图11-42 使用F+F - F+F迭代的分形图案

除了F、+、-之外我们再定义如下几个符号：

- f：与F的含义相同，向前走固定单位，为了定义不同的迭代公式
- [：将当前的位置入堆栈
-]：从堆栈中读取坐标，修改当前位置
- S：初始迭代符号

所有的符号(包括上面未定义的)都可以用来定义迭代，通过引入两个方括号符号，可以描述分岔的图案。例如下面的符号迭代能够绘制出一棵植物：

S -> X

X -> F- [[X]+X]+F [+FX]-X

F -> FF

下面用一个字典定义所有的迭代公式和其他的一些绘图信息：

```
rules = [  
    {  
        "F": "F+F--F+F", "S": "F",  
        "direct": 180,  
        "angle": 60,  
        "iter": 5,  
        "title": "Koch"  
    },  
    {  
        "X": "X+YF+", "Y": "-FX-Y", "S": "FX",  
        "direct": 0,  
        "angle": 90,  
        "iter": 13,  
        "title": "Dragon"  
    },  
    {  
        "f": "F-f-F", "F": "f+F+f", "S": "f",  
        "direct": 0,  
        "angle": 60,  
        "iter": 7,  
        "title": "Triangle"
```

```

    },
    {
        "X": "F- [ [X]+X]+F[+FX]-X", "F": "FF", "S": "X",
        "direct": -45,
        "angle": 25,
        "iter": 6,
        "title": "Plant"
    }
    ,
    {
        "S": "X", "X": "-YF+XFX+FY-", "Y": "+XF-YFY-FX+",
        "direct": 0,
        "angle": 90,
        "iter": 6,
        "title": "Hilbert"
    },
    {
        "S": "L--F--L--F", "L": "+R-F-R+", "R": "-L+F+L-",
        "direct": 0,
        "angle": 45,
        "iter": 10,
        "title": "Sierpinski"
    },
]

```

其中：

- direct: 绘图的初始角度，通过指定不同的值可以旋转整个图案
- angle: 定义符号+和-旋转时的角度，不同的值能产生完全不同的图案
- iter: 迭代次数

下面的程序将上述字典转换为需要绘制的线段坐标：

```
class L_System(object):  
    def __init__(self, rule):  
        info = rule['S']  
        for i in range(rule['iter']):  
            ninfo = []  
            for c in info:  
                if c in rule:  
                    ninfo.append(rule[c])  
                else:  
                    ninfo.append(c)  
            info = "".join(ninfo)  
        self.rule = rule  
        self.info = info  
  
    def get_lines(self):  
        from math import sin, cos, pi  
        d = self.rule['direct']  
        a = self.rule['angle']  
        p = (0.0, 0.0)
```

```

l = 1.0
lines = []
stack = []
for c in self.info:
    if c in "Ff":
        r = d * pi / 180
        t = p[0] + l*cos(r), p[1] + l*sin(r)
        lines.append((p[0], p[1]), (t[0], t[1]))
        p = t
    elif c == "+":
        d += a
    elif c == "-":
        d -= a
    elif c == "[":
        stack.append((p,d))
    elif c == "]":
        p, d = stack[-1]
        del stack[-1]
return lines

```

下面的draw()完成迭代计算和绘图工作：

```

def draw(ax, rule, iter=None):
    from matplotlib import collections
    if iter!=None:
        rule["iter"] = iter
    lines = L_System(rule).get_lines() ❶

```

```

        linecollections = collections.LineCollection(lines,
lw=0.7, color="black") ❷

    ax.add_collection(linecollections, autolim=True) ❸

    ax.axis("equal")

    ax.set_axis_off()

    ax.set_xlim(ax.dataLim.xmin, ax.dataLim.xmax)

    ax.invert_yaxis()

```

❶用L_System的get_lines()计算出每个线段的坐标之后，❷创建一个表示所有线段集合的LineCollection对象，❸并调用Axes对象的add_collection()将此线段集合添加进ax.collections列表中。这样能一次添加多条线段，提高显示速度。图11-43是程序所绘制的几种L-System的分形图案。

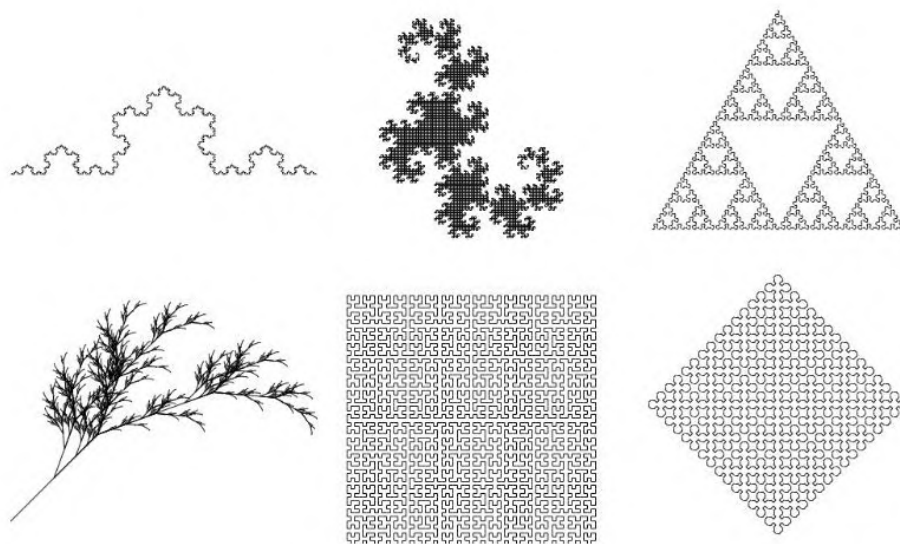


图11-43 几种L-System的迭代图案

```

%config InlineBackend.figure_format = 'png'

fig = plt.figure(figsize=(10, 6))

```



```
fig.patch.set_facecolor("w")

for i in xrange(6):
    ax = fig.add_subplot(231+i)
    draw(ax, rules[i])

fig.subplots_adjust(left=0, right=1, bottom=0, top=1, wspace=0, hspace=0)
```

11.6.4 分形山脉

前面介绍的分形图案都是严格按照指定的规则迭代生成的，然而自然界中的山川、云彩、树木等都不是精确的自相似图形，而是在统计意义上的自相似图形。本节将介绍几种经典的山脉地形的生成算法，以及如何用Python快速实现这些算法。

1. 一维中点移位法

让我们从绘制一条分形曲线开始。使用中点位移算法 (Midpoint Displacement)，能够有效地模拟山脉或海岸线的分形形状，算法如下：

- (1) 首先在X轴上取两个初始点A和B。

(2) 找到A、B两点的中点，并在Y轴方向上进行随机移位，移位后的点为C。

(3) 对于线段AC和BC重复步骤(2)。

每次迭代时，随机移位的最大幅度都成比例地衰减。迭代足够多次之后，将所得到的点连接起来，就得到了一条随机的分形曲线。

下面是实现此算法的源程序，程序所绘制的山脉曲线如图11-44所示。

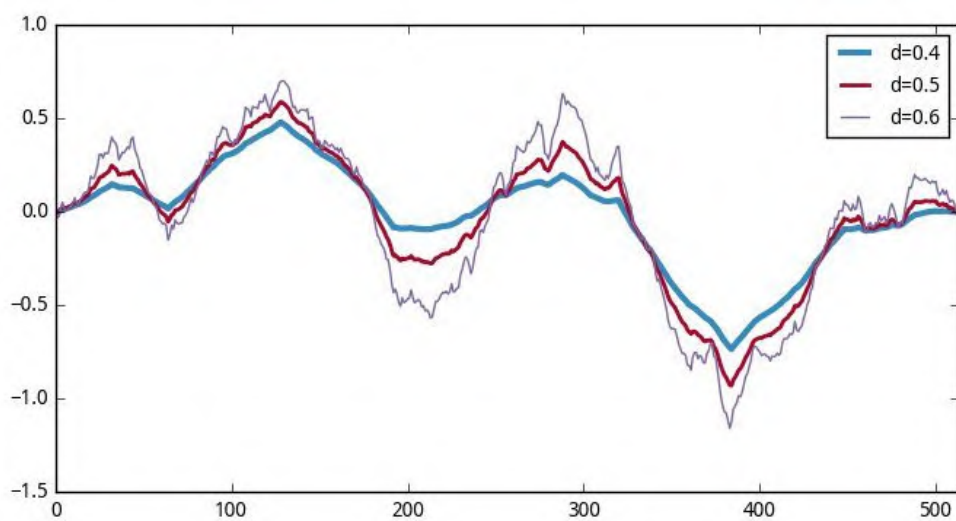


图11-44 一维分形山脉曲线，衰减值越小，最大幅度的衰减越快，曲线越平滑

```
def hill1d(n, d):  
    """  
    绘制山脉曲线，2*n+1为曲线在x轴上的长度，d为衰减系数  
    """  
    a = np.zeros(2*n+1) ❶  
    scale = 1.0  
    for i in xrange(n, 0, -1): ❷
```

```

        s = 2**(i-1) ❸

        s2 = 2*s

        tmp = a[::s2]

        a[s::s2] += (tmp[:-1] + tmp[1:]) * 0.5 ❹

        a[s::s2] += np.random.normal(size=len(tmp)-1,
scale=scale) ❺

        scale *= d ❻

    return a

pl.figure(figsize=(8,4))
for i, d in enumerate([0.4, 0.5, 0.6]):
    np.random.seed(8) ❼
    a = hill1d(9, d)
    pl.plot(a, label="d=%s" % d, linewidth=3-i)
pl.xlim(0, len(a))
pl.legend()

```

程序中，hill1d() 计算一维分形山脉曲线。❶为了运算方便，我们用一维数组a保存曲线上每点的高度，而曲线上每点的X轴坐标则由数组的下标决定。这要求每次计算中点时所得到的X轴坐标必须是整数。显然当数组长度为 $2^n + 1$ 时满足这个要求。

❷由于数组a的长度为 $2^n + 1$ ，因此需要循环n次才能够计算到数组上所有的点。每次循环时，都要对数组中的某些点计算中值。例如对于n=8，即数组长度为257时，循环变量i和数组中需要计算中点的下标如表11-2所示：

表11-2 循环变量i与对应的数组的下标

i	需要计算中点的数组下标
8	128
7	64、192
6	32、96、160、224
5	16、48、80、112、144、176、208、240

③数组中每次要计算的中点的下标是一个等差数列，起始下标为 $s=2^{i-1}$ ，间隔为 $s=2^i$ 。④而每个中点都由其左右下标相差s的两个数值计算。⑤给每个中点一定的随机位移。这里使用normal()产生一个正态分布的随机数组，其期望值为0，标准偏差为scale。这样产生的山脉曲线才能既有山峰也有山谷。⑥最后将下一次迭代的标准偏差乘上系数d，因此d越小，标准偏差的衰减越快。

接下来绘制d为0.4、0.5、0.6时的山脉曲线。⑦这里为了对不同的衰减系数所产生的曲线进行比较，需要保证每次都使用相同的随机数计算曲线，因此使用seed()指定生成随机数的种子。在需要真正随机产生曲线时，请将此句注释掉。

2. 二维中点移位法

中点移位法很容易扩展到二维，可以用它计算山脉曲面，算法如图11-45所示。左图中，从白色圆点的值(值为0、2、4和8的4个点)计算5个用灰色圆点表示的中值点。边上4个中值点的计算和一维的情况相同，而正中间的中值则是4个角上的值的平均值。

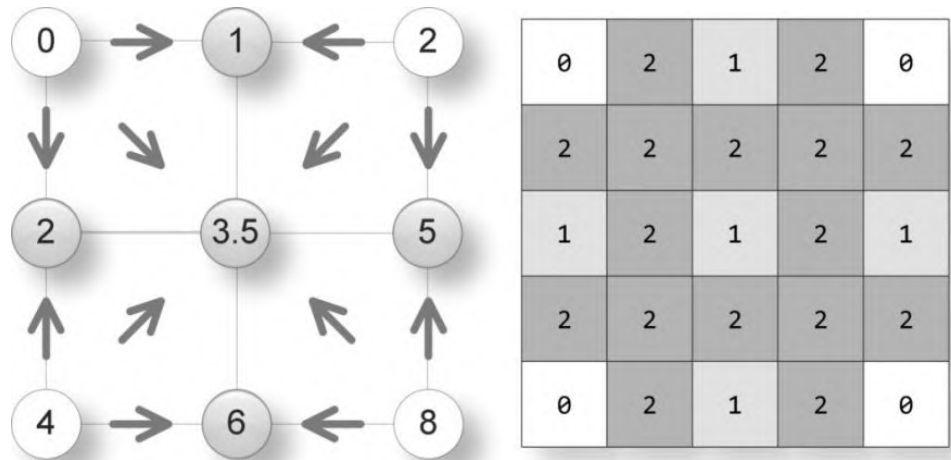


图11-45 二维中点移位法示意图

右图以计算 5×5 的方格为例，演示了每步迭代时所计算的点。方格中的数字表示计算此点的值的迭代次数。初期情况下4个角上的点已知，标记它们的迭代次数为0。根据左图的中值计算方式，计算出标记为1的5个方格的值。然后对于由迭代0和迭代1的点组成的4个方块，再次进行中值计算，计算出所有标记为2的16个方格的值。

完整的计算程序如下，程序的显示效果如图11-46所示。

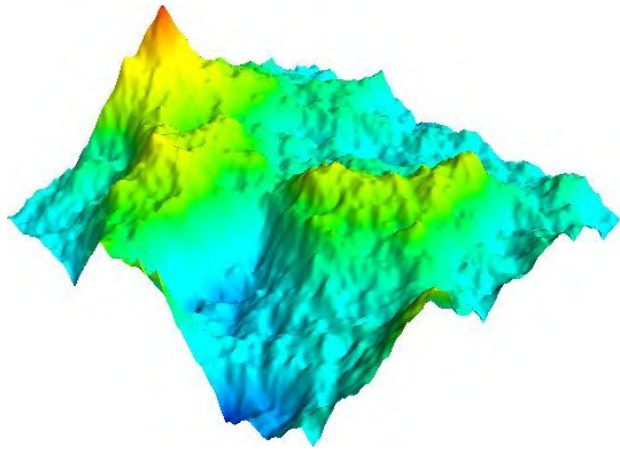


图11-46 二维中点移位法计算山脉曲面

```
def hill2d(n, d):
    """
    绘制山脉曲面，曲面是一个  $(2*n + 1) * (2*n + 1)$  的图像，d为衰减系数
    """
    from numpy.random import normal
    size = 2*n + 1
    scale = 1.0
    a = np.zeros((size, size))

    for i in xrange(n, 0, -1):
        s = 2*(i-1)
        s2 = s*2
        tmp = a[::s2, ::s2]
        tmp1 = (tmp[1:, :] + tmp[:, -1])*0.5
        tmp2 = (tmp[:, 1:] + tmp[:, -1])*0.5
        tmp3 = (tmp1[:, 1:] + tmp1[:, -1])*0.5
        a[s::s2, ::s2] = tmp1 + normal(0, scale,
tmp1.shape)
```

```

        a[:,s2, s:s2] = tmp2 + normal(0, scale,
tmp2.shape)

        a[s:s2,s:s2] = tmp3 + normal(0, scale,
tmp3.shape)

    scale *= d

    return a

from scipy2 import vtk_scene_to_array
from mayavi import mlab
from scipy.ndimage.filters import convolve

np.random.seed(42)
a = hill2d(8, 0.5)
a/= np.ptp(a) / (0.5*2**8) ❶
a = convolve(a, np.ones((3,3))/9) ❷

mlab.options.offscreen = True
scene = mlab.figure(size=(800, 600))
scene.scene.background = 1, 1, 1
mlab.surf(a)
img = vtk_scene_to_array(scene.scene)
%array_image img

```

hill2d() 程序的算法和一维的情况类似，这里就不多解释了。❶
 在计算出表示山脉曲面的二维数组a之后，调用np.ptp()得到数组a中的最大值和最小值之间的差，并将其值放大到数组形状的0.5倍，以便

调用 `mlab.surf()` 绘制曲面。❷ 使用 SciPy 的多维数组卷积函数 `convolve()` 对二维数组 `a` 进行平滑处理。

3. 菱形方形算法

每次迭代都是通过正方形四个角上的点的值，计算其边上4点和中心点的值。这种计算方法有很多种，上节介绍的是最简单的一种方法。但是如果读者放大它所生成的曲面，就会发现它上面有一些大大小小的正方形的痕迹。菱形方形算法 (Diamond-square algorithm) 通过两种中值计算方法的交替使用，能够有效地消除这种正方形痕迹，算法如图11-47所示。

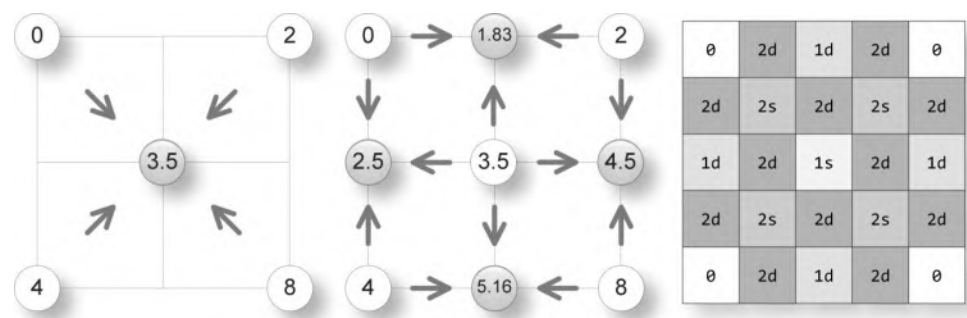


图11-47 菱形方形算法

首先如左图所示，通过正方形的四个角点计算位于其中心的点的平均值。然后如中图所示，通过菱形的四个角点计算位于其中心的点的平均值。图中没有一个完整的菱形，而是4个半边菱形。也可以把正方形平均看作左上、右上、左下和右下四个方向上的点的平均值，而菱形平均看作上下左右四个方向上的点的平均值。右图显示了采用菱形正方形计算 5×5 的数组时的运算顺序。从标记为0的方格开始，以方形平均计算标记为1s的方格值，然后以菱形平均计算标记为1d的4个方

格的值。接下来重复上面的步骤，方形平均计算2s方格，最后菱形平均计算2d方格。

使用菱形方形算法绘制山脉曲面的程序如下。观察图11-47(右)中的标记为2d方格，可以发现菱形平均所对应的方格无法用一个数组表示，因此程序中将它们分为水平和垂直方向上的两个数组分别计算。程序中大量使用数组切片，从而提高程序的运算速度。如果读者觉得较难理解，可以如下修改程序：

(1) 注释掉随机数部分，并在迭代之前为数组的4个角上的元素赋值为不为零的数值。

(2) 使用较小的数组，并且输出每次赋值之后的数组a的值。通过观察数组a的变化可以帮助理解程序和菱形方形算法。

```
def hill2d_ds(n, d):  
    from numpy.random import normal  
    size = 2**n + 1  
    scale = 1.0  
    a = np.zeros((size, size))  
  
    for i in xrange(n, 0, -1):  
        s = 2**(i-1)  
        s2 = 2*s  
  
        # 方形平均  
        t = a[::s2, ::s2]  
        t2 = (t[:-1, :-1] + t[1:, 1:] + t[1:, :-1] +
```

```

t[:-1,1:])/4

    tmp = a[s::s2,s::s2]
    tmp[...] = t2 + normal(0, scale, tmp.shape)

    buf = a[:,s2, :s2]

    # 菱形平均分两步，分别计算水平和垂直方向上的点
    t = a[:,s2,s::s2]
    t[...] = buf[:, :-1] + buf[:, 1:]
    t[:-1] += tmp
    t[1:] += tmp
    t[[0,-1],:] /= 3 # 边上3个值的平均
    t[1:-1,:] /= 4 # 中间的是4个值的平均
    t[...] += np.random.normal(0, scale, t.shape)

    t = a[s::s2,::s2]
    t[...] = buf[:, :-1] + buf[1:,:]
    t[:, :-1] += tmp
    t[:, 1:] += tmp
    t[:, [0,-1]] /= 3
    t[:, 1:-1] /= 4
    t[...] += np.random.normal(0, scale, t.shape)

    scale *= d

return a

```

```
np.random.seed(42)
a = hill2d_ds(8, 0.5)
a/= np.ptp(a) / (0.5*2**8)
a = convolve(a, np.ones((3,3))/9)

mlab.options.offscreen = True
scene = mlab.figure(size=(800, 600))
scene.scene.background = 1, 1, 1
mlab.surf(a)
img = vtk_scene_to_array(scene.scene)
%array_image img
```

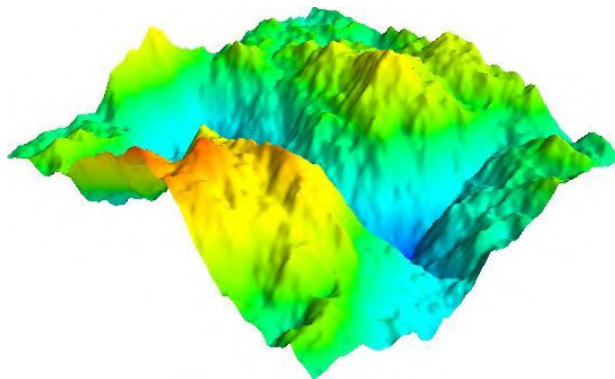


图11-48 使用菱形方形算法计算山脉曲面

附录

CD链接: <http://pan.baidu.com/s/1hs354zy> 密码: jlt9

通过Python建模，深度掌握交易量化投资技术

构建Python股票交易量化平台，实现交易量化技术
详解股票交易主要模型、算法及其实现、应用
量化交易工具提供网盘下载，方便读者学习



Python Quantitative
Transaction Actual Combat

Python量化交易实战

王晓华 著



清华大学出版社



Python

量化交易实战

王晓华 著

清华大学出版社
北京

本书封面贴有清华大学出版社防伪标签，无标签者不得销售
版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

图书在版编目（CIP）数据

Python量化交易实战 / 王晓华著. —北京：清华大学出版社，
2019

ISBN 978-7-302-51763-4

I . ①P… II . ①王… III . ①股票交易—应用软件 IV .
①F830.91

中国版本图书馆CIP数据核字（2018）第271437号

责任编辑：夏毓彦

封面设计：王 翔

责任校对：闫秀华

责任印制：宋 林

出版发行：清华大学出版社

网 址： <http://www.tup.com.cn>, <http://www.wqbook.com>

地 址：北京清华大学学研大厦A座

邮 编：100084

社 总 机：010-62770175

邮 购：010-62786544

投稿与读者服务： 010-62776969, c-service@tup.tsinghua.edu.cn

质量反馈： 010-62772015, zhiliang@tup.tsinghua.edu.cn

印 装 者： 三河市铭诚印务有限公司

经 销： 全国新华书店

开 本： 190mm×260mm

印 张： 17.5

字 数： 448千字

版 次： 2019年2月第1版

印 次： 2019年2月第1次印刷

产品编号： 079806-01



王晓华，计算机专业资深讲师，为研究生和本科生讲授面向对象程序设计、数据结构、Hadoop程序设计等相关课程。主要研究方向为云计算、数据挖掘。曾主持和参与多项国家和省级科研课题，独立完成一项科研成果并获省级成果认定，发表过多篇论文，申请有一项专利。著有《Spark MLlib机器学习实践》《TensorFlow深度学习应用实践》等图书。

内容简介

在目前不断变化、蓬勃发展的中国资本市场，量化投资作为新兴的投资方法，引来越来越多的关注，使用量化投资技术的证券从业人员也越来越多。

本书分为11章，内容包括Python环境的搭建、Python数据相关类库的使用、掘金量化终端的使用、Talib金融库的详解、多因子策略的介绍、带技术指标的多因子策略、中证红利指数增强策略、回归分析与TensorFlow、回归模型的经典应用、配对交易的魔力等。

本书可作为量化投资技术初学者、证券从业人员、金融投资人员的自学用书，也可作为金融机构的培训用书，还可作为高等院校相关专业师生的教学参考书。

前言

量化投资是一种新兴的系统化的金融投资方法，它综合利用现代金融、计算机、数学以及其他相关行业的知识和方法（包括行为学、心理学等），把投资理念、科学理论和实际数据量化为客观的数理模型，使用计算机技术完成全部或部分的投资决策。

由于量化投资需要把数据、策略、系统、执行4个方面综合起来形成一个有机的整体，因此想使用量化策略去对金融市场进行分析的投资者，除了需要有基本的计算机编程知识外，还需要掌握对金融市场的分析，研究过基本的投资方法。目前图书市场上关于金融投资方面的图书不少，但多数投资只是浅显地进行讲解，过于注重零碎的知识点和心得体会。本书以实战为宗旨，通过不同方面的阶段案例，让读者全面、深入、透彻地理解量化投资的原理，提高实际开发水平和项目实战能力。

本书是基于作者2017~2018年参与“北京四两资本”与“南京红树林（量化掘金）”私募项目的实战工作的总结。感谢投资人兼基金经理谭云博士给我的很多指导性意见，感谢黄雪雪女士对本书基础工作的大力支持，也感谢夏编辑给予的很多写作意见和巨大的支持和鼓励，最后感谢吴雪女士在本书出版过程中给予的协助。

本书写作特色

1. 详细深入的解说

为了便于读者理解本书内容，提高学习效率，本书从最基本的Python程序设计开始介绍，直到使用专用的程序工具包进行多种金融

投资回测，便于初学者快速入门。

2. 原理与实战结合

为了让有一定量化投资技术基础的读者进一步提升自己，本书部分内容更偏向于原理的讲解，结合书中的实战案例能使初级量化投资者快速提高自己。

3. 项目案例典型，实战性强，有较高的应用价值

本书使用项目实战案例进行解说，这些案例来源于作者所开发的实际项目，具有很高的应用价值和参考性。学习这些案例便于读者把所介绍的技术融会贯通，部分案例稍加修改，便可用于实际项目开发中。

本书内容

第1、2章 程序设计语言的基本介绍

第1、2章是本书的基本内容，包括程序设计语言Python的基本介绍和安装，以及使用专用编辑器（IDE）PyCharm进行程序设计的方法。此外，还介绍了部分统计学习方法，为后续的内容打下基础。

第3～5章 量化掘金的基本使用

第3～5章介绍了量化掘金工具包的安装和专用编辑器的使用，以及专用金融工具分析包Talib（官网名为TA-Lib，遵从中国用户的习惯，本书称为Talib）的使用。

第6、7章 多因子策略介绍和应用

第6、7章是本书首要讲述的重点内容，主要介绍量化投资中最重要的分析方法—多因子策略，详细介绍了基本面多因子以及技术分析多因子策略，以及综合运用多因子策略进行基金组合的设计。

第8～11章 回归分析和应用

第8～11章主要介绍回归分析的使用以及使用其进行量化投资的方法，并且额外介绍了一种单独的投资方法—配对交易。这些都是量化投资最基本的策略方法。

源码及相关资源下载地址

本书配套的示例源码及相关资源下载地址可以通过扫描右边的二维码获取。

如果下载有问题，或者对本书内容有建议和疑问，请联系 booksaga@163.com，邮件主题为“Python量化交易实战”。



本书读者

- 需要全面学习量化投资技术的初学者。

- 希望使用量化投资技术的证券投资经理和从业人员。
- 希望提高金融投资水平的其他从业人员。
- 高校相关专业的师生和金融机构的学员。
- 需要一本量化投资案头查询手册的投资者。

作 者

2018年9月

目 录

[前言](#)

[第1章 走进量化投资](#)

[1.1 量化投资的诞生背景](#)

[1.2 量化投资的特点](#)

[1.3 量化投资的应用](#)

[1.4 量化投资在我国股市的发展前景](#)

[1.5 小结](#)

[第2章 Python的安装与使用](#)

[2.1 Python的基本安装和用法](#)

[2.1.1 Anaconda的下载与安装](#)

[2.1.2 Python编译器PyCharm的安装](#)

[2.1.3 使用Python计算softmax函数](#)

[2.2 Python常用类库中的threading](#)

[2.2.1 threading库的使用](#)

[2.2.2 threading模块中最重要的Thread类](#)

[2.2.3 threading中的Lock类](#)

[2.2.4 threading中的join类](#)

[2.3 小结](#)

[第3章 Python类库的使用——数据处理及可视化展示](#)

[3.1 从小例子起步——NumPy的初步使用](#)

[3.1.1 数据的矩阵化](#)

[3.1.2 数据分析](#)

[3.1.3 基于统计分析的数据处理](#)

[3.2 图形化数据处理——Matplotlib包的使用](#)

[3.2.1 差异的可视化](#)

[3.2.2 坐标图的展示](#)

[3.2.3 大规模数据的可视化](#)

[3.3 常用的统计分析方法——相似度计算](#)

[3.3.1 基于欧几里得距离的相似度计算](#)

[3.3.2 基于余弦角度的相似度计算](#)

[3.3.3 欧几里得相似度与余弦相似度的比较](#)

[3.4 数据的统计学可视化展示](#)

[3.4.1 数据的四分位](#)

[3.4.2 数据的四分位示例](#)

[3.4.3 数据的标准化](#)

[3.4.4 数据的平行化处理](#)

[3.4.5 热点图—属性相关性检测](#)

[3.5 Python实战：某地降雨的关系处理](#)

[3.5.1 不同年份的相同月份统计](#)

[3.5.2 不同月份之间的增减程度比较](#)

[3.5.3 每月的降水量是否相关](#)

[3.6 小结](#)

[第4章 欢迎来到掘金量化](#)

[4.1 基础工作](#)

[4.1.1 安装掘金终端](#)

[4.1.2 获取帮助](#)

[4.2 实战：使用掘金终端进行回测工作](#)

[4.2.1 创建第一个策略](#)

[4.2.2 运行回测](#)

[4.2.3 查看回测结果](#)

[4.2.4 使用PyCharm进行回测](#)

[4.3 小结](#)

[第5章 Talib金融库使用详解](#)

[5.1 Talib金融工具库的介绍](#)

[5.1.1 使用Talib获取3日、7日、15日均线](#)

[5.1.2 EMA的计算](#)

[5.1.3 MACD的计算](#)

[5.1.4 MACD斜率的计算方法](#)

[5.1.5 使用Talib实现国内金融数据指标](#)

[5.2 Talib金融工具库函数](#)

[5.2.1 Talib常用函数介绍](#)

[5.2.2 Talib图像形态识别](#)

[5.3 实战：Talib金融工具回测实战](#)

[5.3.1 根据MACD变化回测2017年盈利情况](#)

[5.3.2 股价的波动范围及未来走势判定](#)

[5.4 两种经典的轨道突破策略](#)

[5.4.1 Dual Thrust策略](#)

[5.4.2 Dynamic Breakout II策略](#)

[5.5 小结](#)

[第6章 多因子策略](#)

[6.1 一个奇怪的问题](#)

[6.1.1 因子是什么](#)

[6.1.2 选取因子](#)

[6.1.3 单因子选股轮动测试](#)

[6.2 因子的量化选择](#)

[6.2.1 基于IC值的多因子计算方法](#)

[6.2.2 基于IC值的多因子计算方法（续）](#)

[6.2.3 因子IC值计算的目标，等权法因子值的合成](#)

[6.3 实战：基于成长因子的模型测试](#)

[6.3.1 模型说明](#)

[6.3.2 使用模型进行回测](#)

[6.4 霍华·罗斯曼的投资模型](#)

[6.4.1 霍华·罗斯曼简介](#)

[6.4.2 霍华·罗斯曼的投资模型](#)

[6.4.3 对霍华·罗斯曼模型的分析](#)

[6.5 小结](#)

[第7章 带技术指标的多因子策略](#)

[7.1 技术面多因子介绍](#)

[7.1.1 101个技术因子](#)

[7.1.2 基于Talib的技术因子重写](#)

[7.1.3 一个基于放量技术因子策略的回测](#)

[7.2 较为复杂的技术因子](#)

[7.2.1 阻力支撑相对强度因子介绍](#)

[7.2.2 改进的RSRS因子与回测数据](#)

[7.2.3 价差偏离度因子介绍](#)

[7.3 简单的技术性因子—波动率因子](#)

[7.3.1 波动率因子介绍](#)

[7.3.2 更多的波动率因子](#)

[7.4 实战：一个回测成功率100%的中长线买卖例子](#)

[7.4.1 技术指标的设计](#)

[7.4.2 回测的设计](#)

[7.5 小结](#)

[第8章 人人都是基金经理——中证红利指数增强策略](#)

[8.1 中证红利指数基金介绍](#)

[8.1.1 红利指数基金的由来](#)

[8.1.2 中证红利简介](#)

[8.2 基于中证红利的指数增强基金策略的构建](#)

[8.2.1 中证红利策略的构建方法](#)

[8.2.2 策略回测与优化](#)

[8.3 小结](#)

[第9章 掘金量化——回归分析基础](#)

[9.1 回归分析基础](#)

[9.1.1 回归法简介](#)

[9.1.2 一元线性回归](#)

[9.1.3 多元线性回归](#)

[9.1.4 回归法的解法——最小二乘法详解](#)

[9.2 回归分析的一些其他计算方法](#)

[9.2.1 梯度下降算法与使用TensorFlow计算线性回归](#)

[9.2.2 线性回归的姐妹——逻辑回归](#)

[9.3 实战：回归分析——短时间开盘价与收盘价之间的关系](#)

[9.3.1 量化策略基本思路与简单实现](#)

[9.3.2 使用掘金量化实现回测](#)

[9.4 买还是卖——逻辑回归帮你做决定](#)

[9.4.1 逻辑回归是一种分类算法](#)

[9.4.2 逻辑回归的TensorFlow实现](#)

[9.4.3 使用TensorFlow的逻辑回归进行回测](#)

[9.5 机器学习策略——支持向量机](#)

[9.5.1 支持向量机的基本概念](#)

[9.5.2 使用支持向量机进行回测](#)

[9.6 小结](#)

[第10章 回归模型的经典应用](#)

[10.1 CAPM模型简介](#)

[10.1.1 CAPM定价模型的提出](#)

[10.1.2 CAPM定价模型的公式与假设](#)

[10.1.3 CAPM中Beta的定义](#)

[10.2 Fama-French三因子模型](#)

[10.2.1 Fama-French模型的基础公式](#)

[10.2.2 Fama-French模型的实现与回测](#)

[10.3 PB-ROE回归模型的使用](#)

[10.3.1 PB-ROE模型介绍](#)

[10.3.2 PB-ROE模型的实现](#)

[10.3.3 基于上证180的股票回测](#)

[10.3.4 使用自定义股票池的PB-ROE回测](#)

[10.4 小结](#)

[第11章 配对交易的魔力](#)

[11.1 配对交易的基本理论](#)

[11.1.1 相关性分析](#)

[11.1.2 均值、方差与协方差](#)

[11.2 协整性的判定与检验](#)

[11.2.1 协整性](#)

[11.2.2 平稳性的检验方法](#)

[11.3 配对交易](#)

[11.3.1 配对交易的算法](#)

[11.3.2 提取股票的相关性](#)

[11.3.3 协整系数的计算方法](#)

[11.4 配对交易的魔力](#)

[11.4.1 前期计算](#)

[11.4.2 协整性判断](#)

[11.4.3 使用量化掘金回测系统对结果进行判定](#)

[11.5 小结](#)

[返回总目录](#)

第1章

走进量化投资

“量化”这个词来自于数字图像处理，指的是把经过抽样得到的瞬时值进行幅度离散，即用一组规定的电平把瞬时抽样值用最接近的电平值表示出来，如图1-1所示。经过抽样的图像只是在空间上被离散成像素（样本）的阵列。而每个样本灰度值还是一个有无穷多个取值的连续变化量，必须将其转化为有限个离散值，赋予不同码字才能真正成为数字图像。这种转化称为量化。

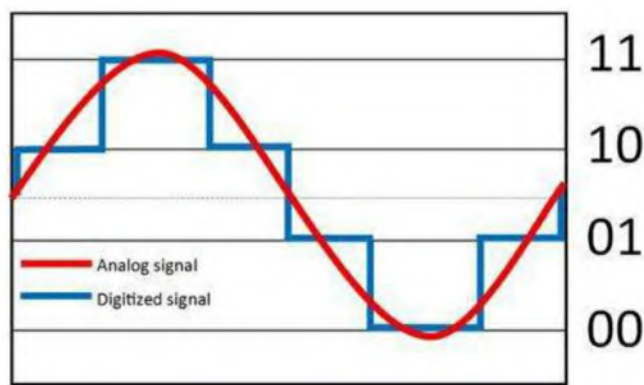


图1-1 量化——数字信号处理

而将量化用于投资的历史已经无法追溯。从“按数据买股票的第一个人”依照自己的判断听取“卖股票的人”介绍想要推销标的的盈利和增长状况时，量化已经产生。投资人依照自己的认知和对数据的掌握做出买卖决定的想法，就是量化的雏形。

1.1 量化投资的诞生背景

接触过资本市场的人，大都听说过基本面投资和价值投资，这方面的天才人物“股神”巴菲特更是几乎家喻户晓，妇孺皆知。他以企业财务报表的分析见长，擅长挖掘企业的内在价值，一旦买入便长期持有，持续获得稳定高额收益，为股东创造了丰厚利润，无人能及。

但并不是人人都能成为巴菲特，那需要长时间的认知积累、敏锐的判断和丰富的经验。20世纪90年代，使用量化工具进行股票和期货买卖的西蒙斯（见图1-2）横空出世，不久就被称为最能赚钱的基金经理人，在20年的时间里，他创造了年均净回报率高达35%的惊人传奇，而他使用的就是量化投资技术。



图1-2 詹姆斯·西蒙斯

但是，没有前期来自象牙塔的现代金融理论，没有其他大师的相关研究，便没有量化投资的兴起，西蒙斯的成功也就无从谈起。

夏普1963年1月提出了“投资组合的简化模型”，一般称为“单一指数模型”。1964年，夏普又发展出了资本资产定价模型（CAPM），这是他最重要的突破，这个模型不仅可以作为预测风险和预期回报的

工具，还可以衡量投资组合的绩效，以及衍生出在指数型基金、企业财务和企业投资、市场行为和资产评价等多领域的应用和理论创新。

1976年，罗斯在CAPM的基础上提出了“套利定价理论”（APT），提供一个方法评估影响股价变化的多种经济因素。布莱克和斯科尔斯提出了“期权定价理论”。莫顿则发明了“跨期的资本资产定价模型”。

有趣的是，不少人最初并非经济学家，如巴契里耶和布莱克原先是数学家，夏普则从事医学，奥斯伯恩为天文学家，沃金与坎德尔是统计学家，而特雷诺则是数学家兼物理学家。他们转行都是因为被金融市场研究深深吸引，沉迷于其中的无穷魅力。

量化投资不会出现在个人投资者为主的时代。个人投资者既缺乏闲暇的时间，也普遍无此技术能力。同时随着退休基金和共同基金资产的大幅增加，财富基金逐渐取代个人投资者成为市场上的主要投资者，并委托专业机构进行投资管理和操作。管理大规模资产需要新的运作方式和金融创新技术，同时专业的投资管理人也有能力和精力专注地研究、运用这些技术。

没有发达的电脑技术，量化投资将成为无源之水，无本之木。在电脑革命发生前，根本无法根据上述模型进行运算。1961年，获得1990年诺贝尔奖的夏普曾说，当时即使是用IBM最好的商用电脑，解出含有100只证券的问题也需要33分钟。当今，面对数不胜数的证券产品以及庞大的成交量，缺少了先进电脑的运算速度和容量，许多复杂的证券定价甚至不可能完成。

1973～1974年，美国债券市场和股票市场全面崩盘，明星基金经理人烟消云散，财富缩水堪比30年代大萧条时期。当时，颇有先见的投资顾问兼作家彼得·伯恩斯坦认为，必须采用更好的方法管理投资组合，因此他创办了《投资组合》杂志，一出刊便获得成功。此后，

随着80年代以来各类证券和期权类产品的丰富和交易量的大增，量化投资光彩炫目，但它同时也具有魔鬼般的力量。

1987年10月大股灾，黑色星期一，当天股市和期货成交量高达令人吃惊的410亿美元，价值瞬间缩水6000亿美元。很多股份直接通过电脑而不是经由交易所交易。一些采用投资组合保险策略的公司，在电脑模式的驱使下，不问价格，机械地卖出股票。很多交易员清楚这些投资组合会有大单卖出，宁愿走在前面争相出逃，加剧了恐慌。针对整个投资组合而非单个证券，机械式的交易、电脑的自动操作使得这种量化投资出现助跌之效，大量的空单在瞬间涌出，将市场彻底砸垮。

虽然麻烦不断，但量化投资依然必要且有效。要知道，在金融危机发生前，量化基金的表现连续8年超过其他投资方式。当然，挫折也会带来量化投资技术的更新和完善，比如在模型中设定新的变量，尤其是加入以往并未包含的宏观经济参数。时过境迁，从2017年开始，量化基金再次表现优异。虽然量化投资能否就此再度复兴仍属未知，一旦趋势形成就会不可逆转，量化投资依然拥有光明的未来。

1.2 量化投资的特点

量化投资并非只可远观而不可亵玩，在本质上它和传统投资是相同的，二者都是基于市场非有效或弱有效的理论基础之上，而投资经理可以通过对个股估值、成长等基本面的分析研究建立战胜市场，产生超额收益的组合。不同的是，传统投资管理较依赖对上市公司的调研，以及基金经理个人的经验及主观的判断，而定量化投资管理则是“定性思想的量化应用”，更加强调数据。

量化投资有以下几个优点：

1. 纪律性

所有的决策都是依据模型做出的。量化投资中一般有三个模型：

- 大类资产配置模型，根据大类资产配置决定股票和债券的投资比例。
- 行业模型，按照行业配置模型确定超配或低配的行业。
- 股票模型，依靠股票模型挑选股票。

纪律性首先表现在依靠模型和相信模型，每一天决策之前，首先要运行模型，根据模型的运行结果进行决策，而不是凭感觉。

可能有读者会提出疑问，模型出错怎么办？不可否认，模型可能出错，就像医生可能误诊一样。但是，在大概率下，医生是不会出错的，同样地，模型在大概率下也是不会出错的，所以，还是需要相信模型。

纪律性的好处很多，可以克服人性的弱点，如贪婪、恐惧、侥幸心理，也可以克服认知偏差，行为金融理论在这方面有许多论述。纪律性的另一个好处是可跟踪。量化投资作为一种定性思想的理性应用，客观地在组合中体现这样的组合思想。一个好的投资方法应该是一个“透明的盒子”。

这样做的好处是每一个买卖决策都是有理有据的，特别是有数据支持的。当打开模型时，模型会显示当时选择的这只股票与其他的股票相比在成长面上、估值上、动量上、技术指标上的得分情况，这个评价是非常全面的，只有汇总得分比其他得分高才有说服力。

2. 系统性

系统性具体表现为“三多”：

- 首先表现在多层次，包括在大类资产配置、行业选择、精选个股三个层次上都有对应的取舍模型。
- 其次是多角度，定量投资的核心投资思想包括宏观周期、市场结构、估值、成长、盈利质量、分析师盈利预测、市场情绪等多个角度。
- 再者就是多数据，就是海量数据的处理。

人脑处理信息的能力是有限的，当一个资本市场只有100只股票时，对定性投资基金经理是有优势的，他可以深刻分析这100家公司。但在一个很大的资本市场，比如有成千上万只股票的时候，强大的定量投资的信息处理能力能反映它的优势，捕捉更多的投资机会，拓展更大的投资机会。

3. 套利思想

定量投资正是在找估值洼地，通过全面、系统性的扫描捕捉错误定价、错误估值带来的机会。定性投资经理大部分时间在琢磨哪一个企业是伟大的企业，哪个股票是可以翻倍的股票；与定性投资经理不同，定量基金经理大部分精力花在分析哪里是估值洼地，哪一个品种被低估了，买入低估的，卖出高估的。

4. 概率取胜

表现为两个方面：

- 一是定量投资不断地从历史中挖掘有望在未来重复的规律并且加以利用。
- 二是依靠一组股票取胜，而不是一个或几个股票取胜。

1.3 量化投资的应用

量化投资主要应用于以下几个方面。

1. 量化择时

股市的可预测性问题与有效市场假说密切相关。如果有效市场理论或有效市场假说成立，股票价格充分反映了所有相关的信息，价格变化服从随机游走，股票价格的预测就会毫无意义。

众多的研究发现，股市的指数收益中存在经典线性相关之外的非线性相关，貌似随机、杂乱，但在其复杂表面的背后却隐藏着确定性的机制，因此存在可预测成分。

2. 股指期货

股指期货套利是指利用股指期货市场存在的不合理价格，同时参与股指期货与股票现货市场交易，或者同时进行不同期限、不同（但相近）类别股票指数合约交易，以赚取差价的行为。股指期货套利主要分为期现套利和跨期套利两种。股指期货套利的研究主要包括现货构建、套利定价、保证金管理、冲击成本、成分股调整等内容。

3. 商品期货

商品期货套利盈利的逻辑原理基于以下几个方面：

- 相关商品在不同地点、不同时间，对应都有一个合理的价格差价。
- 由于价格的波动性，价格差价经常出现不合理。
- 不合理必然要回到合理。
- 不合理回到合理的这部分价格区间就是盈利区间。

4. 统计套利

有别于无风险套利，统计套利是利用证券价格的历史统计规律进行套利，是一种风险套利，其风险在于这种历史统计规律在未来一段时间内是否继续存在。

统计套利在方法上可以分为两类：

- 一类是利用股票的收益率序列建模，目标是在组合的Beta值等于零的前提下实现Alpha收益，我们称之为Beta中性策略。
- 另一类是利用股票价格序列的协整关系建模，我们称之为协整策略。

5. 期权套利

期权套利交易是指同时买进卖出同一相关期货但有不同的敲定价格，或不同到期月份的看涨或看跌期权合约，希望在日后对冲交易部位或履约时获利的交易。期权套利的交易策略和方式多种多样，是多种相关期权交易的组合，具体包括水平套利、垂直套利、转换套利、反向转换套利、跨式套利、蝶式套利、飞鹰式套利等。

6. 算法交易

算法交易又被称为自动交易、黑盒交易或者机器交易，它指的是通过计算机程序来发出交易指令。在交易中，程序可以决定的范围包括交易时间的选择、交易的价格，甚至可以包括最后需要成交的证券数量。根据各个算法交易中算法的主动程度不同，可以把不同算法交易分为被动型算法交易、主动型算法交易、综合型算法交易三大类。

7. 资产配置

资产配置是指资产类别选择，投资组合中各类资产的适当配置以及对这些混合资产进行实时管理。量化投资管理将传统投资组合理论与量化分析技术结合，极大地丰富了资产配置的内涵，形成了现代资产配置理论的基本框架。

它突破了传统积极型投资和指数型投资的局限，将投资方法建立在对各种资产类股票公开数据的统计分析上，通过比较不同资产类的统计特征建立数学模型，进而确定组合资产的配置目标和分配比例。

1.4 量化投资在我国股市的发展前景

首先，相较于海外成熟市场，我国股票市场的发展历史较短，投资者队伍参差不齐，投资理念还不够成熟，留给主动投资发掘市场非有效性，产生阿尔法的潜力和空间也更大。

总体来说，国内的量化投资整体还处于起步阶段，不像国外那样成熟。但好在国内的资本市场没有完全放开，而且期指、期权等对冲手段也不够成熟，很多品种还是T+1交易，即便国外对冲基金进来也需要修改策略来适应国内的资本市场，所以国内的量化投资者们还是有很多投资机会的，还有一定的时间可以充分吸收国外已有的知识和模型，从而做出更好的量化投资模型。

1.5 小结

本章讲述了量化投资的诞生背景、特点、应用范围，并简单介绍了量化投资在我国股市的发展前景。读者掌握这些内容能对量化投资的概念和方法有个基本的印象。

第2章

Python的安装与使用

“人生苦短，我用Python”。

这是Python在自身宣传和推广中使用的口号，做投资也是这样的，简单朴素的投资策略和思想永远是最好的。

对于相关研究人员，最直接、最简洁的需求就是将自己的想法从纸面进化到可以运行的计算机代码，在这个过程中，所需花费的精力越少越好。

Python完全可以满足这个需求，在计算机代码的编写和实现过程中，Python简洁的语言设计本身可以帮助用户避开没必要的陷阱，减少变量声明，随用随写，无须对内存进行释放，这些都可以极大地帮助用户使用Python编写出需要的程序。

其次，Python的社区开发成熟，有非常多的第三方类库可以使用。在本章中还会介绍NumPy、PIL以及threading三个主要的类库，这些开源的算法类库在后面的程序编写过程中会起到极大的作用。

最后，相对于其他语言，Python有较高的运行效率，而且得益于Python开发人员的不懈努力，Python友好的接口库甚至可以加速程序的运行效率，而无须去了解底层的运行机制。

“人生苦短，何不用Python”。Python让其使用者专注于逻辑和算法本身而无须纠结一些技术细节。Python作为深度学习以及TensorFlow框架主要的编程语言，更需要读者去掌握与学习。

2.1 Python的基本安装和用法

Python是深度学习的首选开发语言，但是对于安装来说，很多第三方提供了集成大量科学计算类库的Python标准安装包，而最常用的是Anaconda。

Anaconda里面集成了很多关于Python科学计算的第三方库，主要是安装方便，而Python是一个脚本语言，如果不使用Anaconda，那么第三方库的安装会较为困难，各个库之间的依赖性就很难连接得很好。因此，在这里推荐使用集合了大量第三方类库的安装程序Anaconda来替代Python的安装。

2.1.1 Anaconda的下载与安装

1. 下载和安装

Anaconda的下载地址是<https://www.continuum.io/downloads/>，页面如图2-1所示。

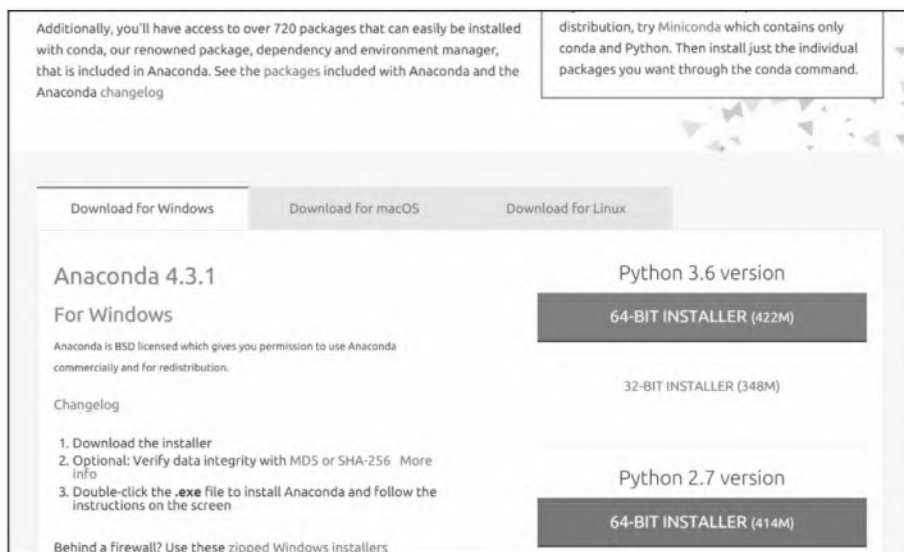


图2-1 Anaconda下载页面

本书使用的是Anaconda 4.3.1版本，里面集成了Python 3.6。读者可以根据自己的操作系统进行下载。

这里作者选择的是Windows版本，单击“运行”即可安装，与普通软件一样。安装完成以后，出现程序面板，目录如图2-2所示。

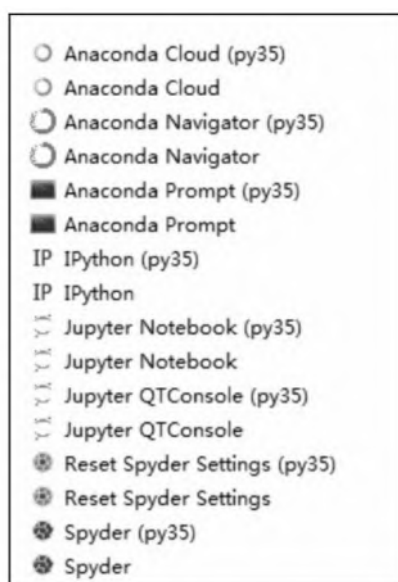


图2-2 Anaconda安装目录

2. 打开控制台

之后依次单击：开始→所有程序→Anaconda→Anaconda Prompt，打开窗口的效果如图2-3所示。这些步骤和打开CMD控制台类似，输入命令就可以控制和配置Python。在Anaconda中最常用的是conda命令，这个命令可以执行一些基本操作。

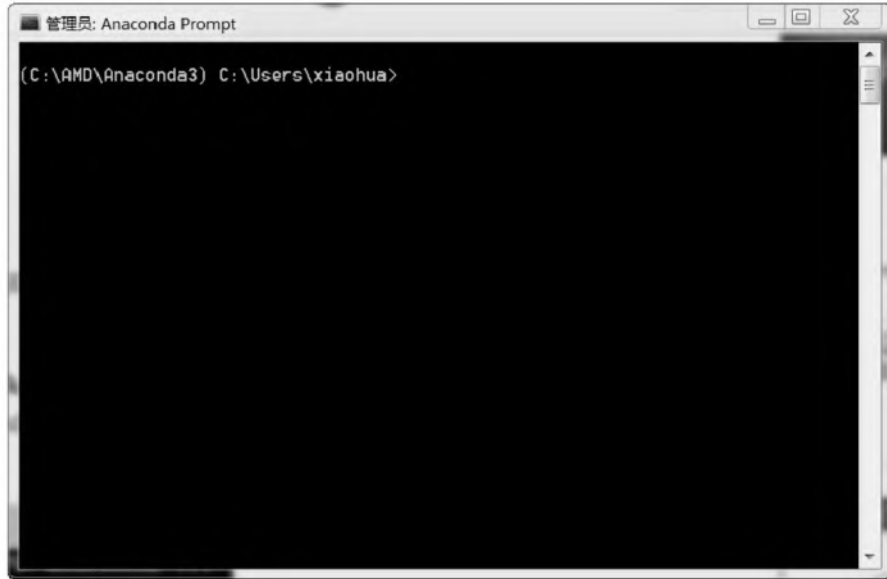


图2-3 Anaconda Prompt控制台

3. 验证Python

之后在控制台中输入python，打印出版版本号以及控制符号，并在控制符号下输入代码：

```
print("hello Python")
```

结果如图2-4所示。

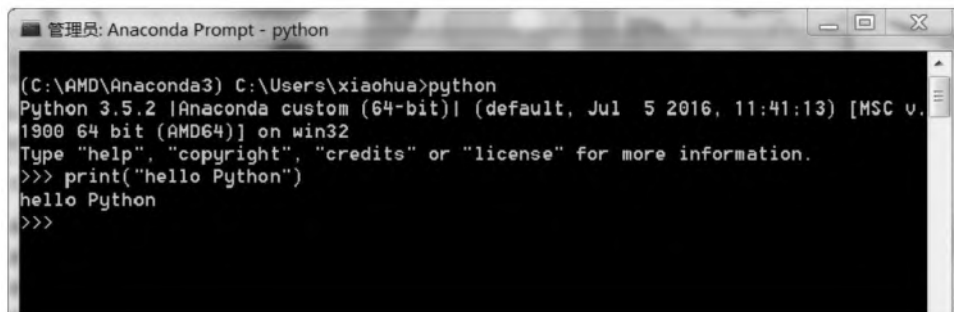


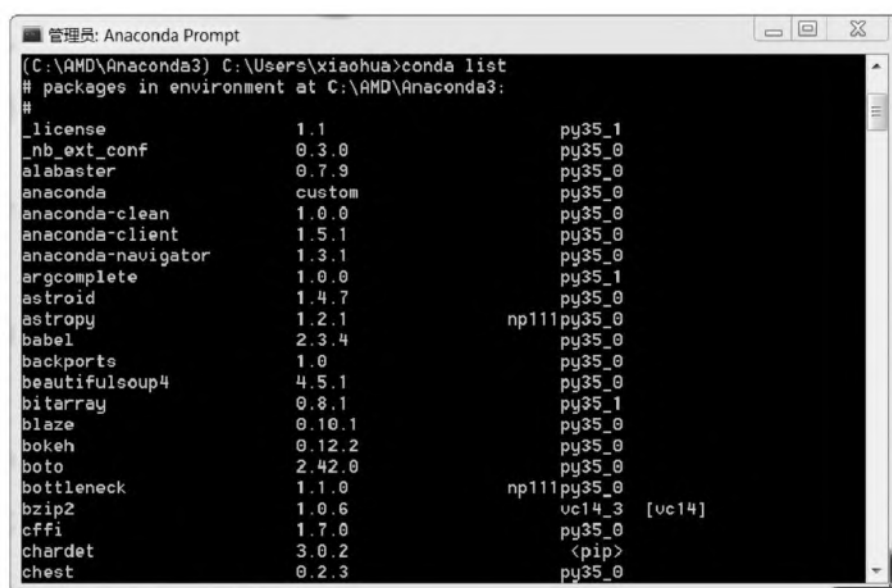
图2-4 验证Anaconda Python安装成功

4. 使用conda命令

作者建议读者使用Anaconda的好处在于，它能够方便地帮助读者安装和使用大量第三方类库，查看已安装的第三方类库的命令是：

```
conda list
```

在Anaconda Prompt控制台中输入exit()或者重新打开Anaconda Prompt控制台后直接输入conda list命令，结果如图2-5所示。



```
(C:\AMD\Anaconda3) C:\Users\xiaohua>conda list
# packages in environment at C:\AMD\Anaconda3:
#
_license                1.1                                py35_1
_nb_ext_conf             0.3.0                             py35_0
alabaster                0.7.9                             py35_0
anaconda                 custom                             py35_0
anaconda-clean           1.0.0                             py35_0
anaconda-client          1.5.1                             py35_0
anaconda-navigator       1.3.1                             py35_0
argcomplete              1.0.0                             py35_1
astroid                  1.4.7                             py35_0
astropy                  1.2.1                             np111py35_0
babel                    2.3.4                             py35_0
backports                1.0                                py35_0
beautifulsoup4           4.5.1                             py35_0
bitarray                 0.8.1                             py35_1
blaze                    0.10.1                            py35_0
bokeh                    0.12.2                            py35_0
boto                     2.42.0                            py35_0
bottleneck               1.1.0                             np111py35_0
bzip2                    1.0.6                             vc14_3 [vc14]
cffi                     1.7.0                             py35_0
chardet                  3.0.2                             <pip>
chest                    0.2.3                             py35_0
```

图2-5 列出已安装的第三方类库

Anaconda中使用conda进行操作的方法还有很多，其中最重要的是安装第三方类库，命令如下：

```
conda install name
```

这里的name是需要安装的第三方类库名，例如当需要安装NumPy包（这个包已经安装过）时，输入相应的命令：

```
conda install numpy
```

使用Anaconda的一个特别的好处是，所安装的包的依赖类库可以自动安装，如图2-6所示，这样可以大大减轻使用者在安装和使用某个

特定类库的情况下造成的依赖类库缺失的困难，使得后续工作顺利进行。

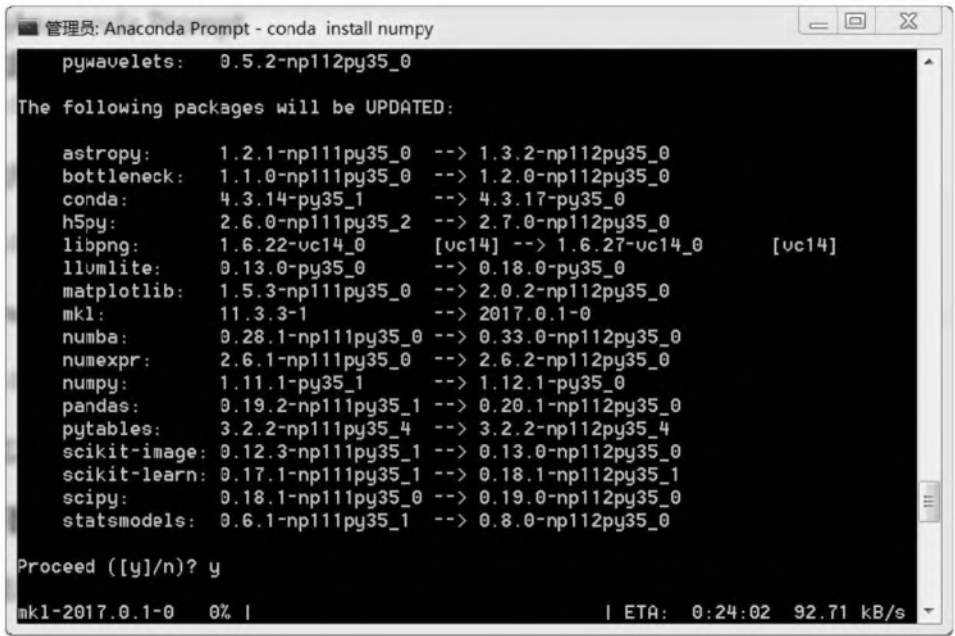


图2-6 自动获取或更新依赖类库

2.1.2 Python编译器PyCharm的安装

和其他语言类似，Python程序的编写可以使用Windows自带的控制台进行。但是这种方式对于较为复杂的程序工程来说，容易混淆相互之间的层级和交互文件，因此在编写程序工程时，作者建议使用专用的Python编译器PyCharm。

1. PyCharm的下载和安装

PyCharm的下载地址为<http://www.jetbrains.com/pycharm/>。

进入Download页面后可以选择不同的版本，有收费的专业版和免费的社区版，如图2-7所示。这里选择免费的社区版即可。



图2-7 PyCharm的免费版

双击运行后进入安装界面，直接单击Next按钮采用默认安装即可，如图2-8所示。



图2-8 PyCharm的安装文件

需要注意的是，在安装PyCharm的过程中需要对安装的位数进行选择，这里建议读者选择与所安装Python相同位数的文件，如图2-9所示。

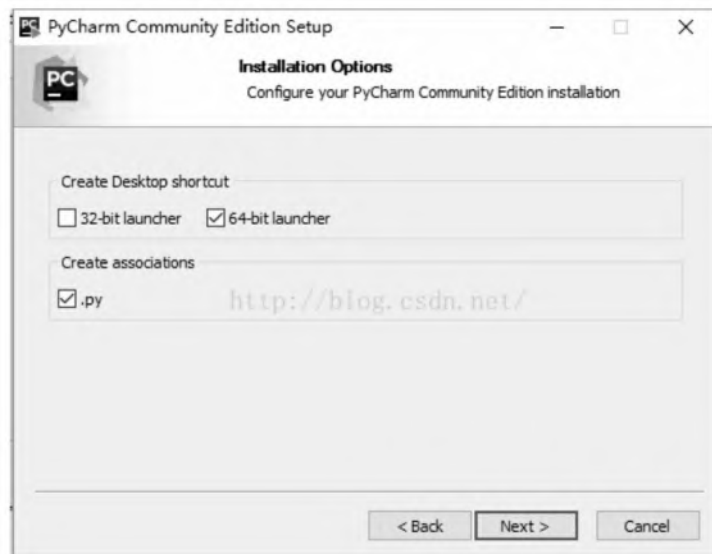


图2-9 PyCharm的位数选择

安装完成后出现Finish按钮，单击后安装完成，如图2-10所示。



图2-10 PyCharm安装完成

2. 使用PyCharm创建程序

单击桌面上新生成的  图标进入PyCharm程序界面，首先是第一次启动的定位，如图2-11所示。

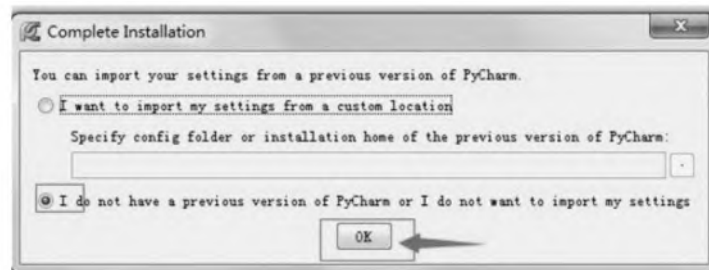


图2-11 PyCharm启动定位

这里是对程序存储的定位，一般建议选择第二个，由PyCharm自动指定即可。单击OK按钮，之后单击弹出的Accept按钮，接受相应的协议，进入界面配置选项，如图2-12所示。

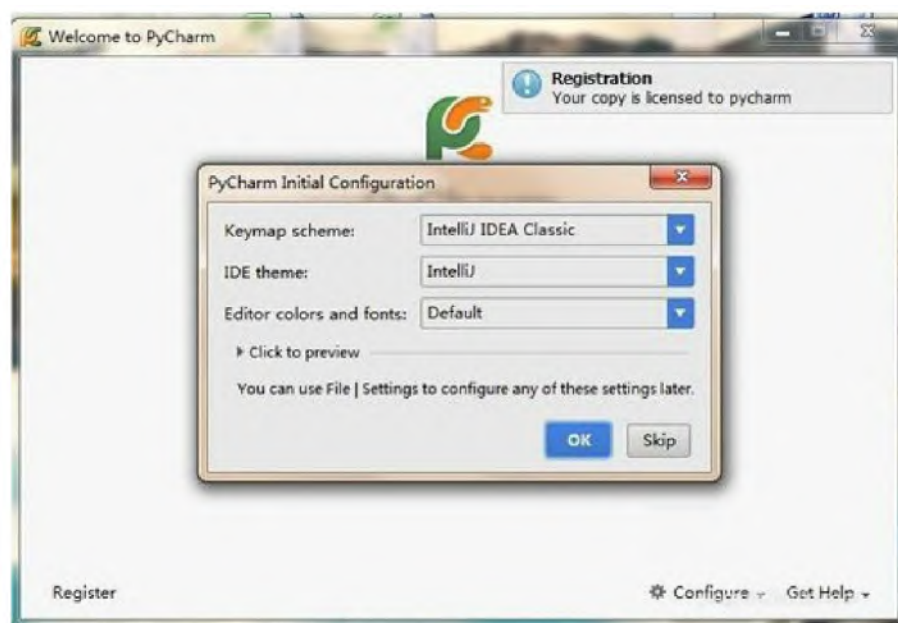


图2-12 PyCharm界面配置

在配置区域可以选择自己的使用风格，对PyCharm的界面进行配置，如果对其不熟悉，直接单击OK按钮保持默认配置即可。

最后就是创建一个新的工程，如图2-13所示。



图2-13 PyCharm工程创建界面

在这里，建议读者新建一个PyCharm的工程文件，如图2-14所示。

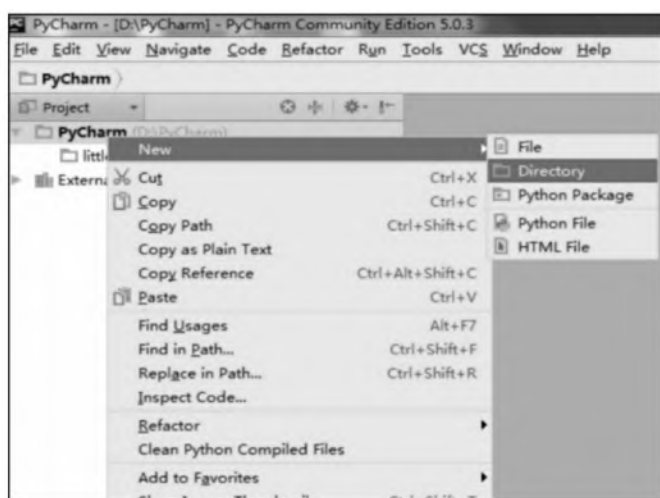


图2-14 PyCharm新建文件界面

之后右击新建的工程名“PyCharm”，在弹出的菜单中单击new→Python File，新建一个helloworld.py文件，内容如图2-15所示。



图2-15 PyCharm工程创建界面

输入代码后，单击run→run…菜单开始运行，或者直接右击helloworld.py后，在弹出的菜单中选择run。如果成功输出“hello world”，那么恭喜你，Python与PyCharm的配置就完成了。

2.1.3 使用Python计算softmax函数

对于Python科学计算来说，最简单的想法就是可以将数学公式直接表达成程序语言，可以说，Python满足了这个想法。本小节将使用Python实现和计算一个深度学习中最为常见的函数——softmax函数。至于这个函数的作用现在不加以说明，这里只是带领读者尝试实现其程序的编写。

softmax计算公式如下：

$$s_i = \frac{e^{v_i}}{\sum_j e^{v_j}}$$

其中 v_i 是长度为 j 的数列 V 中的一个数，带入softmax的结果其实就是先对每一个 v_i 取 e 为底的指数计算变成非负，然后除以所有

项之和进行归一化，之后每个 V_i 就可以解释成观察到的数据 V_i 属于某个类别的概率，或者称作似然（Likelihood）。

提示

softmax用以解决概率计算中概率结果大占绝对优势的问题。例如，函数计算结果中，两个值a和b，且 $a > b$ ，如果简单地以值的大小为单位衡量的话，那么在后续的使用过程中，a永远被选用，而b由于数值较小而不会被选择，但是有时候也需要数值小的b被使用，softmax就可以解决这个问题。

softmax按照概率选择a和b，由于a的概率值大于b，在计算时a经常会被选择，而b由于概率较小，选择的可能性也较小，但是也有概率被选择。

公式softmax的代码如下：

```
import numpy
def softmax(inMatrix):
    m,n = numpy.shape(inMatrix)
    outMatrix = numpy.mat(numpy.zeros((m,n)))
    soft_sum = 0
    for idx in range(0,n):
        outMatrix[0,idx] = math.exp(inMatrix[0,idx])
        soft_sum += outMatrix[0,idx]
    for idx in range(0,n):
        outMatrix[0,idx] = outMatrix[0,idx] / soft_sum
    return outMatrix
```

可以看到，当传入一个数列后，分别计算每个数值所对应的指数函数值，之后将其相加后，计算每个数值在数值和中的概率。

```
a = numpy.array([[1,2,1,2,1,1,3]])
```

结果如下：

```
[[ 0.05943317  0.16155612  0.05943317  0.16155612  0.05943317  0.05943317
  0.43915506]]
```

2.2 Python常用类库中的threading

如果说Python的简单易用奠定了Python发展的基石，那么丰富的第三方类库给予了Python不断前进发展的动力。随着科技的发展，Python应用得更为广泛，更多涉及不同种类的第三方类库被加入Python中。

Python常用类库参见表2-1。

表2-1 Python常用类库

分 类	名 称	库 用 途
科学计算	Matplotlib	用 Python 实现的类 matlab 的第三方库，用以绘制一些高质量的数学二维图形
	SciPy	基于 Python 的 matlab 实现，旨在实现 matlab 的所有功能
	NumPy	基于 Python 的科学计算第三方库，提供了矩阵、线性代数、傅立叶变换等解决方案
GUI	PyGtk	基于 Python 的 GUI 程序开发 GTK+库
	PyQt	用于 Python 的 QT 开发库
	WxPython	Python 下的 GUI 编程框架，与 MFC 的架构相似
	Tkinter	Python 下标准的界面编程包，因此不算是第三方库
其他	BeautifulSoup	基于 Python 的 HTML/XML 解析器，简单易用
	PIL	基于 Python 的图像处理库，功能强大，对图形文件的格式支持广泛
	MySQLdb	用于连接 MySQL 数据库
	cElementTree	高性能 XML 解析库，Py2.5 应该已经包含了该模块，因此不算一个第三方库
	PyGame	基于 Python 的多媒体开发和游戏软件开发模块
	Py2exe	将 Python 脚本转换为 Windows 上可以独立运行的可执行程序
	pefile	Windows PE 文件解析器

表2-1中给出了Python中常用类库的名称和说明。到目前为止，Python中已经有7000多个可以使用的类库供计算机工程人员以及科学研究人员使用。

2.2.1 threading库的使用

对于希望充分利用计算机性能的程序设计者来说，多线程的应用是一个必不可少的技能。多线程类似于使用计算机的一个核心执行多个不同的任务。多线程的好处如下：

- 使用线程可以把需要使用大量时间的计算任务放到后台去处理。
- 减少资源占用，加快程序的运行速度。
- 在传统的输入输出以及网络收发等普通操作上，后台处理可以美化当前界面，增加界面的人性化。

下面将详细介绍Python中操作线程的模块：threading，相对于Python既有的多线程模块thread，threading重写了部分API模块，对thread进行了二次封装，从而大大提高了执行效率。

2.2.2 threading模块中最重要的Thread类

Thread是threading模块中最重要的类之一，可以使用它来创建线程。其具体使用方法是创建一个threading.Thread对象，在它的初始化函数中将需要调用的对象作为初始化参数传入，具体代码如【程序2-1】所示。

【程序2-1】

```
#coding = utf8
import threading,time
count = 0
class MyThread(threading.Thread):
    def __init__(self,threadName):
        super(MyThread,self).__init__(name = threadName)

    def run(self):
        global count
        for i in range(100):
            count = count + 1
            time.sleep(0.3)
            print(self.getName() , count)

for i in range(2):
    MyThread("MyThreadName:" + str(i)).start()
```

在作者定义的MyThread类中，重写了从父对象继承的run方法，run方法中，将一个全局变量逐一增加。在接下来的代码中，创建了5个独立的对象，分别调用其start方法，最后将结果逐一打印。

可以看到在程序中，每个线程被赋予了一个名字，然后设置每隔0.3秒打印输出本线程的计数，即计数加1。而count被人为地设置成全局共享变量，因此在每个线程中都可以自由地对其进行访问。

程序运行结果如图2-16所示。

```
MyThreadName:0 2
MyThreadName:1 2
MyThreadName:1 4
MyThreadName:0 4
MyThreadName:1 6
MyThreadName:0 6
MyThreadName:1 8
MyThreadName:0 8
```

图2-16 程序运行结果

通过上面的结果可以看到，每个线程都起了一个对应的名字，而在运行的时候，线程所计算的计数同时增加。这样可以证明，在程序

运行过程中，两个线程同时对一个数进行操作，并将其结果进行打印。

提示

其中的run方法和start方法并不是threading自带的方法，而是从Python本身的线程处理模块Thread中继承来的。run方法的作用是在线程被启动以后，执行预先写入的程序代码。一般而言，run方法所执行的内容被称为Activity，而start方法是用于启动线程的方法。

2.2.3 threading中的Lock类

虽然线程可以在程序执行的过程中极大地提高程序的执行效率，但是其带来的影响却难以忽略。例如在【程序2-1】中，由于每隔一定时间打印当前的数值，应该逐次打印的数据变成了两个相同的数值被打印出来，因此需要一个能够解决这类问题的方案。

Lock类是threading中用于锁定当前线程的锁定类，顾名思义，其作用是对当前运行中的线程进行锁定，只有被当前线程释放后，后续线程才可以继续操作。

类中主要的代码如下：

```
import threading
lock = threading.Lock()
lock.acquire()
lock.release()
```

acquire方法提供了确定对象被锁定的标志，release在对象被当前线程使用完毕后将当前对象释放。修改后的代码如【程序2-2】所

示。

【程序2-2】

```
#coding = utf8
import threading,time,random

count = 0
class MyThread (threading.Thread):

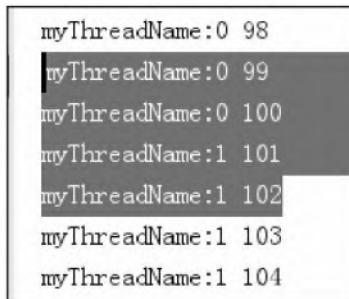
    def __init__(self,lock,threadName):
        super(MyThread,self).__init__(name = threadName)
        self.lock = lock

    def run(self):
        global count
        self.lock.acquire()
        for i in range(100):
            count = count + 1
            time.sleep(0.3)
            print(self.getName() , count)
        self.lock.release()

lock = threading.Lock()
for i in range(2):
    MyThread (lock,"MyThreadName:" + str(i)).start()
```

可以看到Lock被传递给MyThread，并在run方法中人为锁定当前的线程，必须等线程执行完毕后，后续的线程才可以继续执行。程序运行结果如图2-17所示。

可以看到，其中变色的部分，线程2只有等线程1完全结束后，才执行后续的操作。本程序中，Thread1等到Thread0完全结束后，才执行第二个操作。



```
myThreadName:0 98
myThreadName:0 99
myThreadName:0 100
myThreadName:1 101
myThreadName:1 102
myThreadName:1 103
myThreadName:1 104
```

图2-17 程序运行结果

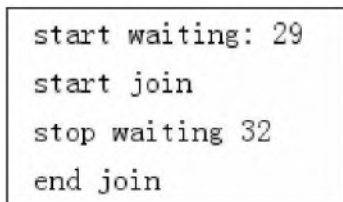
2.2.4 threading中的join类

join类是threading中用于堵塞当前主线程的类，其作用是阻止全部的线程继续运行，直到被调用的线程执行完毕或者超时。具体代码如【程序2-3】所示。

【程序2-3】

```
import threading, time
def doWaiting():
    print('start waiting:', time.strftime('%S'))
    time.sleep(3)
    print('stop waiting', time.strftime('%S'))
    thread1 = threading.Thread(target = doWaiting)
    thread1.start()
    time.sleep(1)                                #确保线程 thread1 已经启动
    print('start join')
    thread1.join()                                #将一直堵塞，直到 thread1 运行结束
    print('end join')
```

程序的运行结果如图2-18所示。



```
start waiting: 29
start join
stop waiting 32
end join
```

图2-18 程序运行结果

其中的time方法设定了当前的时间，当join启动后，堵塞了调用整体进程的主进程，而只有当被堵塞的进程执行完毕后，后续的进程才继续执行。

除此之外，对于线程的使用，Python还有很多其他的方法，例如threading.Event、threading.Condition等，这些都是在程序设计时能够极大地帮助程序设计人员编写合适程序的工具。限于篇幅，这里

不再一一进行介绍，在后续的使用过程中，作者会带领读者了解和掌握更多的相关内容。

2.3 小结

本章介绍了Python的基本安装和编译器的使用。在这里推荐读者使用PyCharm免费版作为Python编辑器，这有助于更好地安排工程文件的配置和程序的编写。

同时，本章中还介绍了最常用的一些类库，这里只是对线程类做了一个详细的介绍，线程类是Python最为重要的一个类库，在后面的代码编写中会频繁遇到。

本章是Python最基础的内容，后面的章节将以Python的使用为主，并且还会介绍更多的Python类库，希望读者能够掌握相关内容。

第3章

Python类库的使用——数据处理及可视化展示

前面对Python的安装做了基本的介绍，并且建议读者使用PyCharm免费版作为使用Python编写程序的编译器。相对于使用控制台或自带的编译器，可以更加直观和明晰化地对所构建的工程做出层次安排。

本章将介绍使用Python对数据进行处理和可视化。对于大多数的Python程序设计，建议读者使用已有的类库解决问题而不是自行编写相应的代码。这是初学者非常易犯的错误，对于Python来说，大多数的类库都是在底层使用效率更高的C语言实现的，并且由经验丰富的程序设计人员编写，因此不建议读者自行设计和完成相应的程序。

本章将着重介绍几个常用的统计学类库，针对量化投资的专用类库将在后面的章节介绍。

“人生苦短，我用Python！编程复杂，请用类库！”

3.1 从小例子起步——NumPy的初步使用

从小例子起步，本节将介绍NumPy的基础使用。

3.1.1 数据的矩阵化

对于数据处理来说，数据是一切的基础。而一切数据又不是单一存在的，其构成往往由很多的特征值决定。表3-1是用以计算回归分析的房屋面积与价格对应表，主要参数为面积、卧室以及地下室的个数等。

表3-1 某地区房屋面积与价格对应表

价格（千）	面积（平方米）	卧室（个）	地下室
200	105	3	无
165	80	2	无
184.5	120	2	无
116	70.8	1	无
270	150	4	有

表3-1是数据的一般表示形式，但是对于数据处理的过程来说，这是不可辨识的数据，因此需要对其进行调整。

常用的数据处理表示形式为数据矩阵，即可以将表3-1表示为一个专门的矩阵，见表3-2。

表3-2 某地区房屋面积与价格计算矩阵

ID	Price	area	bedroom	basement
1	200	105	3	False
2	165	80	2	False
3	184.5	120	2	False
4	116	70.8	1	False
5	270	150	4	True

从表3-2中可以看到，一行代表一个单独的房屋价格和对应的特征属性。第一列是ID，即每行的标签。标签是独一无二的，一般不会有重复现象。第二列是价格，一般被称为矩阵的目标。目标可以是单纯的数字，也可以是布尔变量或者一个特定的表示。表3-2中的标签是房

屋的ID，是一个数字标签。第2、3、4列是属性值，也是标签所对应的特征值，根据此特征值的不同，每行所对应的目标也有所不同。

不同的ID用于表示不同的目标。一般来说，数据处理的最终目的就是使用不同的特征属性对目标进行区分和计算。已有的目标是观察和记录的结果，而数据处理的过程就是创建一个可进行目标识别的模型的过程。

建立模型的过程称为数据处理的训练过程，其速度和正确率主要取决于算法的选择，而算法是目标和属性之间建立某种一一对应关系的过程。这点在前面介绍数据处理过程的时候已经有所介绍。

继续回到表3-2的矩阵中。通过观察可知，矩阵中所包含的属性有两种，分别是数值型变量和布尔型变量。其中第2、3、4列是数值变量，这也是数据处理中最常使用和辨识的类型。而第5列是布尔型变量，用以标识对地下室存在的判定。

这样做的好处在于，数据处理在工作时是根据采用的算法进行建模的，算法的描述只能对数值型变量和布尔型变量进行处理，而对于其他类型变量的处理相对较少。即使后文有针对文字进行处理的数据处理模型，其本质也是将文字转化成矩阵向量进行处理。

当数据处理建模的最终目标是求得一个具体数值时，即目标是一个数字，那么数据处理建模的过程基本上可以被转化为回归问题，差别在于是逻辑回归还是线性回归。

对于目标为布尔型变量时，问题大多数被称为分类问题，而常用的建模方法是决策树方法。一般来说，当分类的目标是两个的时候，问题被转化为二元分类；而分类的结果多于两个的时候，分类称为多元分类。

许多情况下，数据处理建模和算法的设计是由程序设计和研究人员所选择的，而具体采用何种算法和模型也没有一定的要求，回归问

题可以被转化为分类问题，而分类问题往往也可以由建立的回归模型解决。这点没有特定的要求。

3.1.2 数据分析

对于数据来说，在进行数据处理建模之前，需要对数据进行基本的分析和处理。

从图3-1可以看到，对于数据集来说，在进行数据分析之前，需要知道很多东西。首先需要知道一个数据集数据的多少和每个数据所拥有的属性个数，对于程序设计人员和科研人员来说，这些都是简单的事，但是对于数据处理的模型来说，是必不可少的内容。

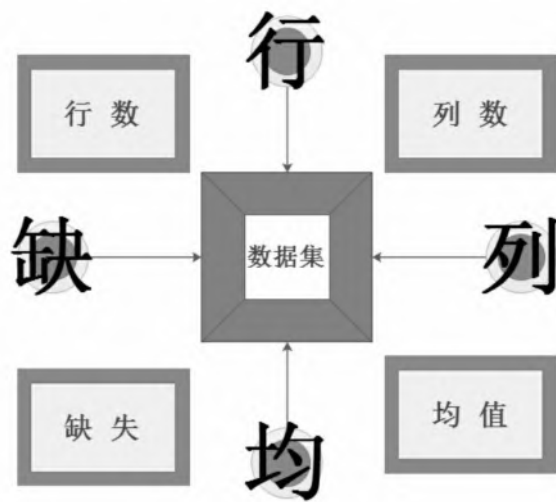


图3-1 数据分析的要求

除此之外，对于数据集来说，缺失值的处理也是一个非常重要的过程。最简单的处理方法是对有缺失值的数据进行整体删除。但是问题在于，数据处理的数据往往来自于现实社会，因此可能数据集中大多数的数据都会有某些特征属性缺失，而解决的办法往往是采用均值或者与目标数据近似的数据特征属性替代。有些情况替代方法可取，

而有些情况下，替代或者采用均值的办法处理缺失值是不可取的，因此要根据具体情况具体处理。

首先从一个小例子开始介绍。以表3-2的矩阵为例，需要建立一个包含数据集的数据矩阵，之后可以利用不同的方法对其进行处理。代码如【程序3-1】所示。

【程序3-1】

```
import numpy as np
data = np.mat([[1,200,105,3,False],[2,165,80,2,False],
               [3,184.5,120,2,False],[4,116,70.8,1,False],[5,270,150,4,True]])
row = 0
for line in data:
    row += 1
print( row )
print( data.size )
```

【程序3-1】第一行引入了Anaconda自带的一个数据矩阵化的包。对于NumPy，读者只需要知道，NumPy系统是Python的一种开源的数值计算扩展。这种工具可用来存储和处理大型矩阵，比Python自身的嵌套列表（nested list structure）结构要高效得多。

第一行代码的意思是引入NumPy，将其重命名为np使用，第二行使用NumPy中的mat()方法建立一个数据矩阵，row是引入的计算行数的变量，使用for循环将data数据读出到line中，而每读一行则row的计数加一。data.size是计算数据集中全部数据的数据量，一般其与行数相除，则为列数。最终打印结果请读者自行测试。

需要说明的是，NumPy将数据转化成一个矩阵的形式进行处理，其中具体的数据可以通过二元的形式读出，如【程序3-2】所示。

【程序3-2】

```
import numpy as np
data = np.mat([[1,200,105,3,False],[2,165,80,2,False],
               [3,184.5,120,2,False],[4,116,70.8,1,False],[5,270,150,4,True]])
print( print( data[0,3])
print( print( data[0,4])
```

最终打印结果如下：

```
3.0
0.0
```

细心的读者可能已经注意到，下标为[0,3]的数据对应的是矩阵中第1行第4列数据，其数值为3，而打印结果为3.0，这个没什么问题。而对于下标为[0,4]的数据，在矩阵中是False的布尔类型，打印结果是0。这点牵涉Python的语言定义，其布尔值都可以近似地表示为0和1。读者需要注意：

```
True = 1.0
False = 0
```

如果需要打印全部的数据集，即调用如下方法：

```
Print( data)
```

将全部的数据以一个数据的形式进行打印，请读者自行测试。

3.1.3 基于统计分析的数据处理

除了最基本的数据记录和提取外，数据处理还需要知道一些基本数据的统计量，例如每一类型数据的均值、方差以及标准差等。当然在本书中，并不需要手动或者使用计算器计算以上数值，NumPy提供了相关方法。程序如下。

【程序3-3】

```
import numpy as np
data = np.mat([[1,200,105,3,False],[2,165,80,2,False],
               [3,184.5,120,2,False],[4,116,70.8,1,False],[5,270,150,4,True]])

coll = []
for row in data:
    coll.append(row[0,1])

print( np.sum(coll))
print( np.mean(coll) )
print( np.std(coll))
print( np.var(coll))
```

首先，coll生成了一个空的数据集，之后采用for循环对数据集进行填充。在【程序3-3】中，第一列数据被填入coll数据集中，这也是一个类型数据的集合，之后依次计算数据集的和、均值、标准差以及方差，这些对于数据处理模型的建立有一定的帮助。

3.2 图形化数据处理——Matplotlib包的使用

对于单纯的数字来说，光从读数据的角度并不能直观反映数字的偏差和集中程度，因此需要采用另一种方法更好地分析数据。对于数据来说，没有什么能够比用图形来解释更为形象和直观了。

3.2.1 差异的可视化

继续回到表3-2的数据，第二列是各个房屋的价格，其价格并不相同，因此直观地查看价格的差异和偏移程度是较为困难的一件事。

研究数值差异和异常的方法是绘制数据的分布程度，相对于合适的直线或曲线，其差异程度如何，以便帮助确定数据的分布。

【程序3-4】

```
import numpy as np
import pylab
import scipy.stats as stats

data = np.mat([[1,200,105,3,False],[2,165,80,2,False],
               [3,184.5,120,2,False],[4,116,70.8,1,False],[5,270,150,4,True]])

coll = []
for row in data:
    coll.append(row[0,1])

stats.probplot(coll,plot=pylab)
pylab.show()
```

结果如图3-2所示。

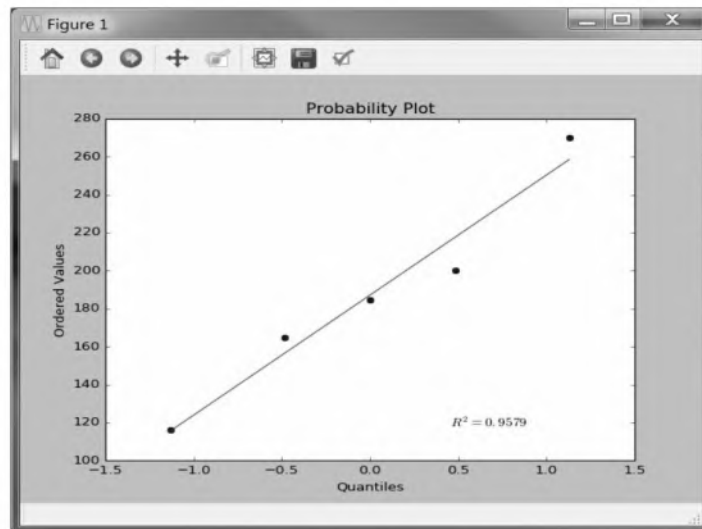


图3-2 房屋价格的偏离展示

【程序3-4】展示了一个对价格的偏离程度的代码实现例子，coll集合是价格的合集，scipy是专门进行数据处理的数据处理包，probplot计算了coll数据集中数据在正态分布下的偏离程度。从图3-2可以看到，价格围绕一条直线上下波动，有一定的偏离，但是偏离情况不太明显。

其中，R为0.9579，指的是数据拟合的相关性，一般0.95以上就可以认为数据拟合程度比较好。

3.2.2 坐标图的展示

通过前面对回归的可视化处理可以看到，可视化能够让数据更加直观地展现出来。同时，可以对数据的误差表现得更为直观。

图3-3展示了一个横向坐标图，用以展示不同类别所占的比重。系列1、2、3分别代表不同的属性，而类别1~6可以看作是6个不同的特例。通过坐标图可以非常直观地看到不同的类别中不同的属性所占的比重。

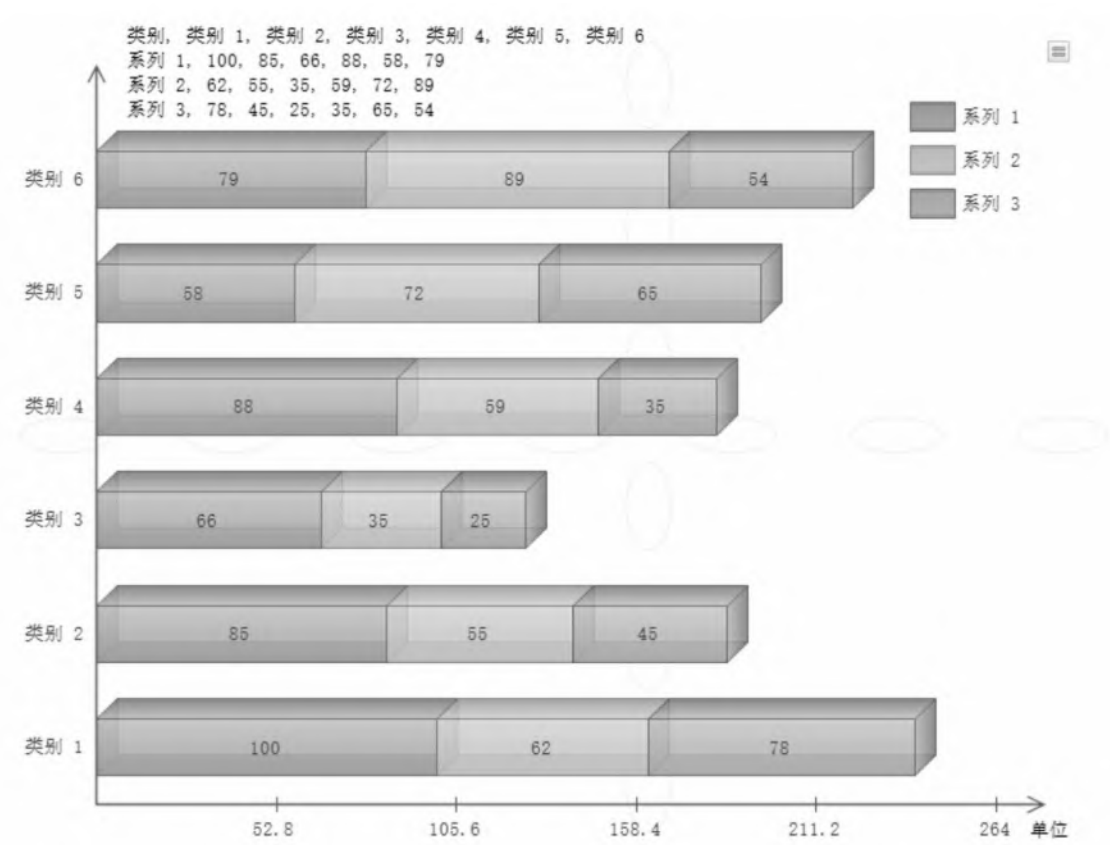


图3-3 横向坐标图

可以看到，一个坐标图能够对数据进行展示，其最基本的要求是可以通过不同的行或者列表现出数据的某些具体值，不同的标签使用不同的颜色和样式用以展示不同的系统关系。【程序3-5】展示了对于不同目标的数据提取不同的行进行显示的代码。

【程序3-5】

```
import pandas as pd
import matplotlib.pyplot as plot
rocksVMines = pd.DataFrame([[1,200,105,3,False],[2,165,80,2,False],
                             [3,184.5,120,2,False],[4,116,70.8,1,False],[5,270,150,4,True]])

dataRow1 = rocksVMines.iloc[1,0:3]
dataRow2 = rocksVMines.iloc[2,0:3]
plot.scatter(dataRow1, dataRow2)
plot.xlabel("Attribute1")
plot.ylabel(("Attribute2"))
plot.show()

dataRow3 = rocksVMines.iloc[3,0:3]
plot.scatter(dataRow2, dataRow3)
plot.xlabel("Attribute2")
plot.ylabel("Attribute3")
plot.show()
```

从图3-4可以看到，通过选定不同目标行中不同的属性，可以对其进行较好的衡量，比较两个行之间的属性关系以及属性之间的相关性。不同的目标，即使属性千差万别，也可以构建相互关系图。

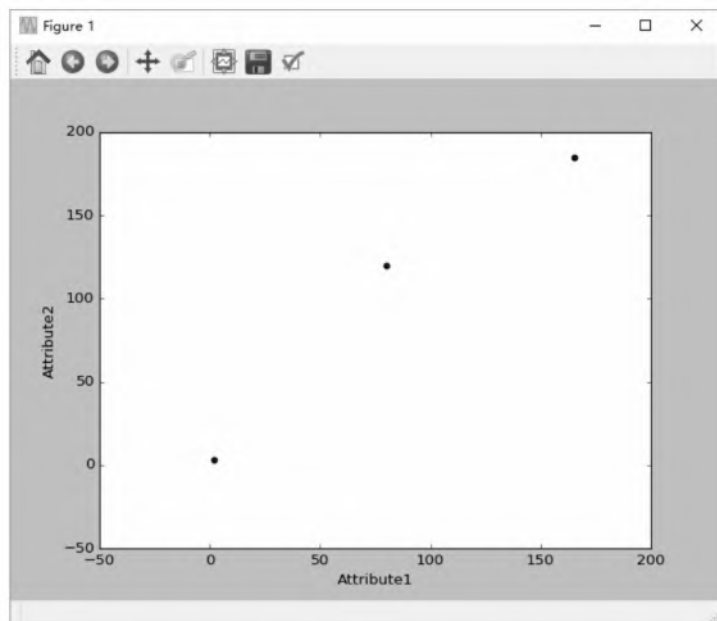


图3-4 不同目标属性之间的关系

顺带说一句，本例中采用的数据较少，随着数据增加，属性之间一般呈现一种正态分布，这点请读者自行验证。

提示

运行【程序3-5】后，会生成两幅不同的图，建议读者通过对比代码认真研究其不同。

3.2.3 大规模数据的可视化

对于大规模数据来说，由于涉及的目标比较多，属性特征值也比较多，对其查看更是一项非常复杂的工作，因此，为了更好地理解和掌握大数据的处理，将其转化成可视性较强的图形是更好的做法。

前面对小数据集进行了图形化查阅，现在对现实中的大规模数据进行处理。

数据来源于真实的信用贷款数据，从50 000个数据记录中随机选取200个数据进行计算，每个数据又有较多的属性值。大多数情况下，数据是以CSV格式进行存储的，pandas包同样提供了相关读取程序。具体代码见【程序3-6】。

【程序3-6】

```
import pandas as pd
import matplotlib.pyplot as plot
filePath = ("c://dataTest.csv")
dataFile = pd.read_csv(filePath,header=None, prefix="V")

dataRow1 = dataFile.iloc[100,1:300]
dataRow2 = dataFile.iloc[101,1:300]
plot.scatter(dataRow1, dataRow2)
plot.xlabel("Attribute1")
plot.ylabel("Attribute2")
plot.show()
```

从【程序3-6】可以看到，首先使用filePath创建了一个文件路径，用以建立数据地址。之后使用pandas自带的read_csv读取CSV格式的文件。dataFile是读取的数据集，之后使用iloc方法获取其中行的属性数据，scatter是做出分散图的方法，对属性进行画图。最终结果如图3-5所示。

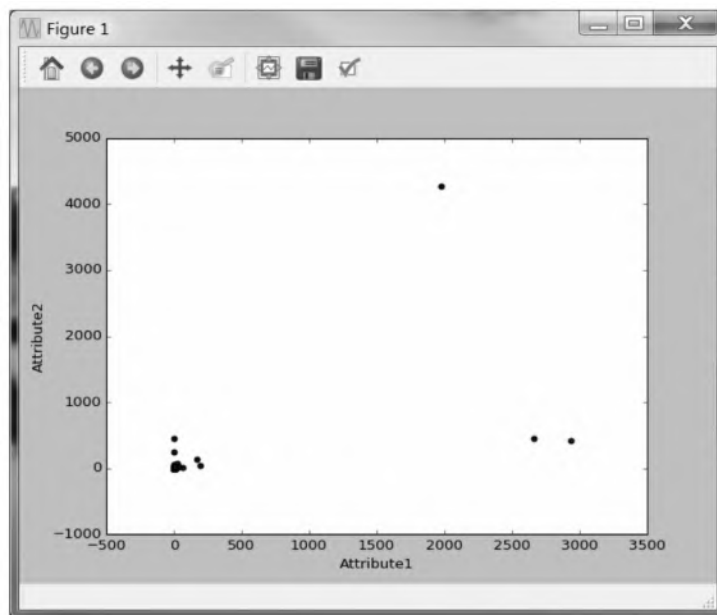


图3-5 大数据集中不同目标属性之间的关系

可以看到，数据在(0, 0)的位置有较大的集合，表明属性在此的偏离程度较小，而几个特定点是偏离程度较大的点。这可以帮助读者对离群值进行分析。

提示

在【程序3-6】出现了两幅图，请读者自行分析。

下面继续对数据集进行分析。【程序3-5】和【程序3-6】让读者看到了对数据的同一行中不同的属性进行处理的方法，如果要对不同目标行的同一种属性进行分析，那么如何做呢？请读者参阅【程序3-7】。

【程序3-7】

```

import pandas as pd
import matplotlib.pyplot as plot
    filePath = ("c://dataTest.csv")
dataFile = pd.read_csv(filePath,header=None, prefix="V")

target = []
for i in range(200):
    if dataFile.iat[i,10] >= 7:
        target.append(1.0)
    else:
        target.append(0.0)

dataRow = dataFile.iloc[0:200,10]
plot.scatter(dataRow, target)
plot.xlabel("Attribute")
plot.ylabel("Target")
plot.show()

```

【程序3-7】中对数据进行处理，提取了200行数据中的第10个属性，并对其进行判定，单纯的判定规则是根据均值对其区分，之后计算判定结果，如图3-6所示。

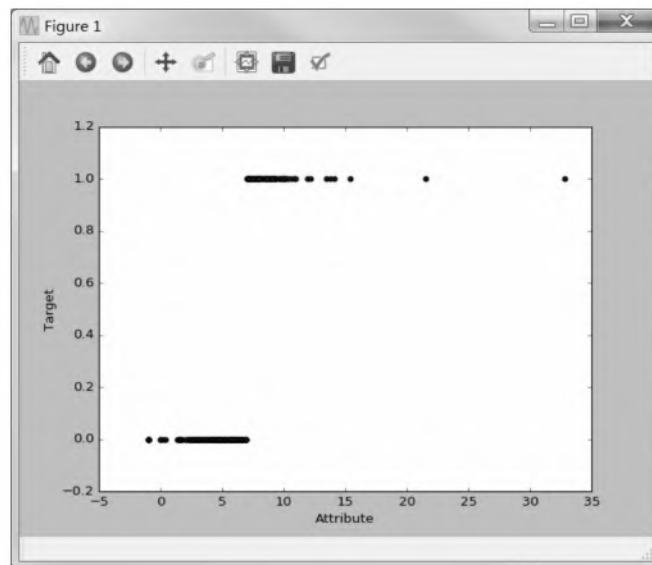


图3-6 大数据集中不同行相同属性之间的关系

通过图3-6可以看到，属性被人为地分成两部分，数据集合的程度也显示了偏离程度。如果下一步需要对属性的离散情况进行反映，那么应该使用【程序3-8】。

【程序3-8】

```
import pandas as pd
import matplotlib.pyplot as plot
    filePath = ("c://dataTest.csv")
dataFile = pd.read_csv(filePath,header=None, prefix="V")

target = []
for i in range(200):
    if dataFile.iat[i,10] >= 7:
        target.append(1.0 + uniform(-0.3, 0.3))
    else:
        target.append(0.0 + uniform(-0.3, 0.3))
dataRow = dataFile.iloc[0:200,10]
plot.scatter(dataRow, target, alpha=0.5, s=100)
plot.xlabel("Attribute")
plot.ylabel("Target")
plot.show()
```

此段程序中，离散的数据被人为地加入了离散变量，具体显示结果请读者自行完成。

提示

读者可以对程序的属性做出诸多的抽取，并尝试使用更多的方法和变量进行处理。

3.3 常用的统计分析方法——相似度计算

我们从3.2节的内容中可以看到，由于不同目标行之间的属性不同，画出的散点图也是千差万别的，而不同的属性对于数据处理来说，需要一个统一的度量进行计算，即需要对其相似度进行计算。

相似度的计算方法很多，这里选用最常用的两种，即欧几里得相似度计算和余弦相似度计算。如果读者对此不感兴趣，可以跳过本节内容继续学习。

3.3.1 基于欧几里得距离的相似度计算

欧几里得距离（Euclidean Distance）是最常用的计算距离的公式，它用来表示三维空间中两个点的真实距离。

欧几里得相似度计算是一种基于用户之间直线距离的计算方式。在相似度计算中，不同的物品或者用户可以将其定义为不同的坐标点，而将特定目标定位为坐标原点。使用欧几里得距离计算两个点之间的绝对距离。欧几里得相似度的计算如【公式3-1】所示。

【公式3-1】

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

从【公式3-1】可以看到，作为计算结果的欧式值显示的是两点之间的直线距离，该值的大小表示两个物品或者用户差异性的大小，即用户的相似性。两个物品或者用户距离越大，可以看到其相似度越小；距离越小则相似度越大。

提示

简而言之，欧几里得的计算数值与最终的相似度计算成反比，欧氏距离越小，两组数据相似度就越大，欧氏距离越大，两组

数据相似度就越小。因此，在实际中往往使用欧几里得距离的倒数作为相似度计算的近似值，即使用 $1/(d+1)$ 作为近似值。

下面来看一个常用的用户—物品推荐评分表的例子，如表3-3所示。

表3-3 用户与物品评分对应表

	物品 1	物品 2	物品 3	物品 4
用户 1	1	1	3	1
用户 2	1	2	3	2
用户 3	2	2	1	1

表3-3是3个用户对物品的打分表，如果需要计算用户1和其他用户之间的相似度，通过欧几里得距离公式可以得出：

$$d_{12} = \sqrt{(1-1)^2 + (1-2)^2 + (3-3)^2 + (1-2)^2} \approx 1.414$$

可以看到，用户1和用户2的相似度为1.414，而用户1和用户3的相似度是：

$$d_{13} = \sqrt{(1-2)^2 + (1-2)^2 + (3-1)^2 + (1-1)^2} \approx 2.449$$

从得到的计算值可以看出， d_{12} 的分值小于 d_{13} 的分值，根据欧氏距离与相似度成反比的法则，可以认为用户2相对于用户3更加近似于用户1。

3.3.2 基于余弦角度的相似度计算

与欧几里得距离相类似，余弦相似度也将特定目标（物品或者用户）作为坐标上的点，但不是坐标原点，与特定的计算目标进行夹角计算，具体如图3-7所示。

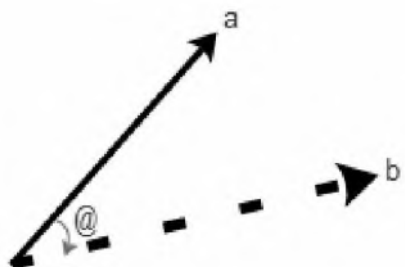


图3-7 余弦相似度示例

从图3-7可以很明显地看出，两条直线分别从坐标原点触发，引出一定的角度。如果两个目标较为相似，那么其线段形成的夹角较小。如果两个用户不相似，那么两条射线形成的夹角较大。因此，在使用余弦度量的相似度计算中，可以用夹角的大小来反映目标之间的相似性。余弦相似度的计算如【公式3-2】所示。

【公式3-2】

$$\cos@ = \frac{\sum (x_i \times y_i)}{\sqrt{\sum x_i^2} \times \sqrt{\sum y_i^2}}$$

从【公式3-2】可以看到，余弦值一般在 $[-1, 1]$ 之间，而这个值的大小同时与余弦夹角的大小成正比。如果用余弦相似度计算表3-3中用户1和用户2之间的相似性，结果如下：

$$d_{12} = \frac{1 \times 1 + 1 \times 2 + 3 \times 3 + 1 \times 2}{\sqrt{1^2 + 1^2 + 3^2 + 1^2} \times \sqrt{1^2 + 2^2 + 3^2 + 2^2}} = \frac{14}{\sqrt{12} \times \sqrt{18}} \approx 0.95$$

而用户1和用户3的相似性如下：

$$d_{13} = \frac{1 \times 2 + 1 \times 2 + 3 \times 1 + 1 \times 1}{\sqrt{1^2 + 1^2 + 3^2 + 1^2} \times \sqrt{2^2 + 2^2 + 1^2 + 1^2}} = \frac{8}{\sqrt{12} \times \sqrt{10}} \approx 0.73$$

从计算结果可得，用户2相对于用户3，与用户1更为相似。

3.3.3 欧几里得相似度与余弦相似度的比较

欧几里得相似度以目标绝对距离作为衡量的标准，而余弦相似度以目标差异的大小作为衡量标准，其表述如图3-8所示。

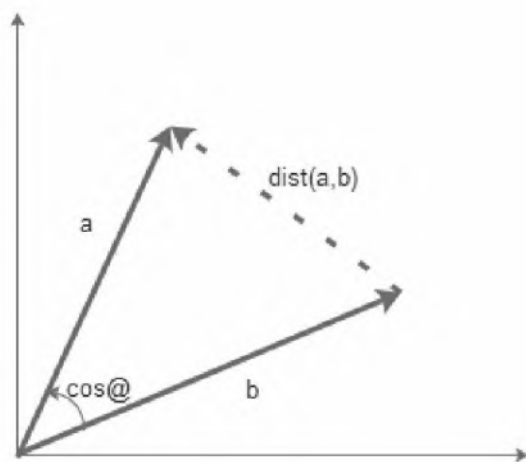


图3-8 欧几里得相似度与余弦相似度

可以看到，欧几里得相似度注重目标之间的差异，与目标在空间中的位置直接相关。而余弦相似度是不同目标在空间中的夹角，更加表现在前进趋势上的差异。

欧几里得相似度和余弦相似度具有不同的计算方法和描述特征。一般来说，欧几里得相似度用以表现不同目标的绝对差异性，从而分析目标之间的相似度与差异情况。而余弦相似度更多的是对目标从方向趋势上区分，对特定坐标数字不敏感。

提示

举例来说，两个目标在不同的两个用户之间的评分分别是 $(1, 1)$ 和 $(5, 5)$ ，这两个评分在表述上是一样的。但是在分析用户相似度时，更多的是使用欧几里得相似度而不是余弦相似度对其进行计算。余弦相似度更好地区分了用户的分离状态。

3.4 数据的统计学可视化展示

在3.3节中，读者对数据，特别是大数据的处理有了基本的认识，通过数据的可视化处理，对数据的基本属性和分布都有了较为直观的理解。本节将对数据进行更多的分析处理，需要用到更为精准和科学的统计学分析方面的知识。

3.4.1 数据的四分位

四分位数 (Quartile) 是统计学中分位数的一种，即把所有数据由小到大排列并分成四等份，处于三个分割点位置的数据就是四分位数。

- 第一四分位数 (Q_1) 又称“下四分位数”，等于该样本中所有数据由小到大排列后第25%的数据。
- 第二四分位数 (Q_2) 又称“中位数”，等于该样本中所有数据由小到大排列后第50%数据。

- 第三四分位数（Q3）又称“上四分位数”，等于该样本中所有数据由小到大排列后第75%的数据。
- 第三四分位数与第一四分位数的差距又称四分位距（Inter Quartile Range, IQR）。

首先确定四分位数的位置，n表示项数的话，四分位数的位置分别为：

- Q1的位置= $(n+1) \times 0.25$
- Q2的位置= $(n+1) \times 0.5$
- Q3的位置= $(n+1) \times 0.75$

通过图形表示如图3-9所示。

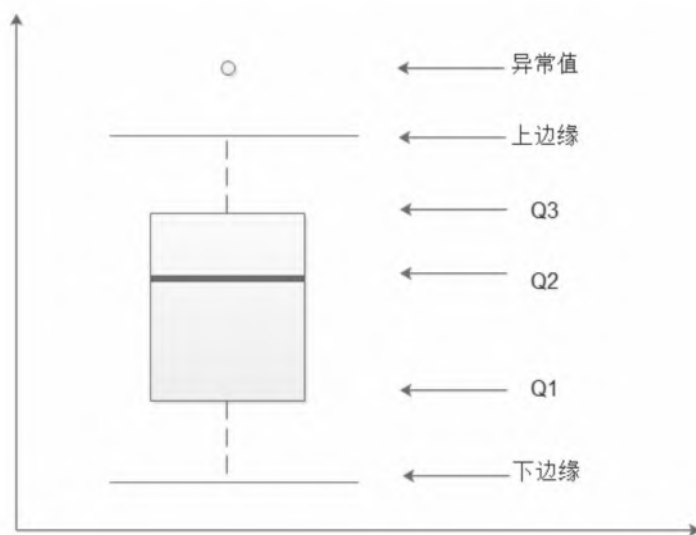


图3-9 四分位的计算

从图3-9可以看到，四分位在图形中根据Q1和Q3的位置绘制了一个箱体结构，即根据一组数据的5个特征绘制的一个箱子和两条线段的图形。这种直观的箱线图反映出一组数据的特征分布，还显示了数据的最小值、中位数和最大值。


```
from pylab import *
import pandas as pd
import matplotlib.pyplot as plot
    filePath = ("c://dataTest.csv")
dataFile = pd.read_csv(filePath,header=None, prefix="V")

print(dataFile.head())
print((dataFile.tail()))

summary = dataFile.describe()
print(summary)

array = dataFile.iloc[:,10:16].values
boxplot(array)
plot.xlabel("Attribute")
plot.ylabel(("Score"))
show()
```

下面来看数据的结果：

	V0	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V1129	\
0	20001	6.15	7.06	5.24	2.61	0.00	4.36	0.00	5.76	3.83	...		7
1	20002	6.53	6.15	9.85	4.03	0.10	1.32	0.69	6.24	7.06	...		6
2	20003	8.22	3.23	1.69	0.41	0.02	2.89	0.13	10.05	8.76	...		1
3	20004	6.79	4.99	1.50	2.85	5.53	1.89	5.41	6.79	6.11	...		3
4	20005	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	...		7

	V1130	V1131	V1132	V1133	V1134	V1135	V1136	V1137	V1138
0	6	1	2	5	7	3	6	8	12
1	7	15	2	6	7	1	8	1	24
2	8	3	1	1	8	8	1	7	6
3	6	20	1	6	8	1	6	5	12
4	8	1	1	8	8	1	8	8	1

[5 rows x 1139 columns]

	V0	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	\
196	20197	3.59	5.63	6.21	5.24	1.88	1.65	4.74	3.73	7.19	...	
197	20198	7.27	5.31	9.35	2.77	0.00	1.37	0.74	5.77	4.64	...	
198	20199	6.18	5.05	6.43	6.05	1.93	2.58	3.75	7.32	4.19	...	
199	20200	6.12	7.45	1.05	1.03	0.16	1.44	0.32	6.49	10.79	...	
200	20201	5.60	6.29	6.11	2.64	0.11	4.08	2.44	7.04	5.60	...	

	V1129	V1130	V1131	V1132	V1133	V1134	V1135	V1136	V1137	V1138
196	6	6	1	1	6	8	9	8	4	28
197	7	1	1	1	1	8	24	7	8	14
198	3	7	1	2	7	7	3	3	7	4
199	7	8	1	2	4	7	6	8	7	12
200	7	7	3	1	7	8	1	2	7	23

[5 rows x 1139 columns]

V0	V1	V2	V3	V4	\
----	----	----	----	----	---

count	201.000000	201.000000	201.000000	201.000000	201.000000	
mean	20101.000000	5.266219	6.447015	6.156020	3.319303	
std	58.167861	2.273933	2.443789	2.967566	3.134570	
min	20001.000000	-1.000000	-1.000000	-1.000000	-1.000000	
25%	20051.000000	4.130000	5.190000	4.660000	1.200000	
50%	20101.000000	5.240000	6.410000	6.000000	2.830000	
75%	20151.000000	6.590000	7.790000	7.640000	4.570000	
max	20201.000000	13.150000	13.960000	16.620000	28.440000	

	V5	V6	V7	V8	V9	...	\
count	201.000000	201.000000	201.000000	201.000000	201.000000	...	...
mean	0.907662	2.680149	2.649254	5.149055	5.532736	...	...
std	1.360489	2.292231	2.912611	2.965096	2.763270	...	...
min	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	...	...
25%	0.020000	1.270000	0.320000	3.260000	3.720000	...	...
50%	0.300000	2.030000	1.870000	4.870000	5.540000	...	...
75%	1.390000	3.710000	4.140000	6.760000	7.400000	...	...
max	8.480000	12.970000	18.850000	15.520000	13.490000	...	...

	V1129	V1130	V1131	V1132	V1133	V1134	\
count	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	
mean	6.054726	6.039801	7.756219	1.353234	4.830846	7.731343	
std	1.934422	2.314824	9.145232	0.836422	2.161306	0.444368	
min	1.000000	1.000000	1.000000	1.000000	1.000000	7.000000	
25%	6.000000	5.000000	1.000000	1.000000	3.000000	7.000000	
50%	7.000000	7.000000	1.000000	1.000000	6.000000	8.000000	
75%	7.000000	8.000000	15.000000	2.000000	7.000000	8.000000	
max	8.000000	8.000000	35.000000	7.000000	8.000000	8.000000	

	V1135	V1136	V1137	V1138
count	201.000000	201.000000	201.000000	201.000000
mean	10.960199	5.631841	5.572139	16.776119
std	9.851315	2.510733	2.517145	8.507916
min	1.000000	1.000000	1.000000	1.000000
25%	3.000000	3.000000	4.000000	11.000000
50%	8.000000	7.000000	7.000000	17.000000
75%	18.000000	8.000000	7.000000	23.000000
max	36.000000	8.000000	8.000000	33.000000

这一部分是打印出的计算后的数据头和尾部，为了节省空间，只选择了前6个和最后尾部的6个数据。第一列是数据的编号，对数据目

标行进行区分，其后是每个不同的目标行的属性。

`dataFile.describe()` 方法是对数据进行统计学估计，`count`、`mean`、`std`、`min`分别求得每列数据的计数、均值、方差以及最小值。最后的几个百分比是求得四分位的数据，具体图形如图3-11所示。

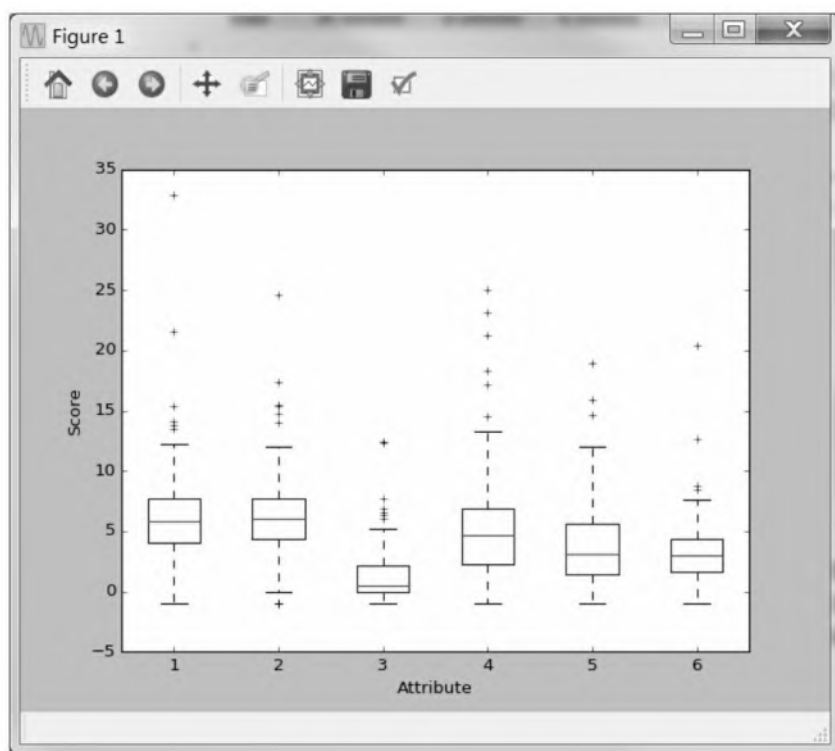


图3-11 小贷数据集的四分位显示

代码中选择了第11~16列的数据作为分析数据集，可以看到，不同的数据列做出的箱体四分位图也是不同的，而部分明显超越四分位位置的数据被称为离群值，一般被视作特异点加以处理。

提示

读者可以选择不同的目标行和属性点进行分析。

从图3-11可以看到，四分位图是一个以更好、更直观的方式来识别数据中异常值的方法，比起数据处理的其他方式，它能够更有效地让分析人员判断离群值。

3.4.3 数据的标准化

继续对数据进行分析，相信读者在进行数据选择的时候，可能会遇到某一列的数值过大或者过小的问题，即数据的显示超出其他数据部分较大时，就会产生数据图形失真的问题，如图3-12所示。

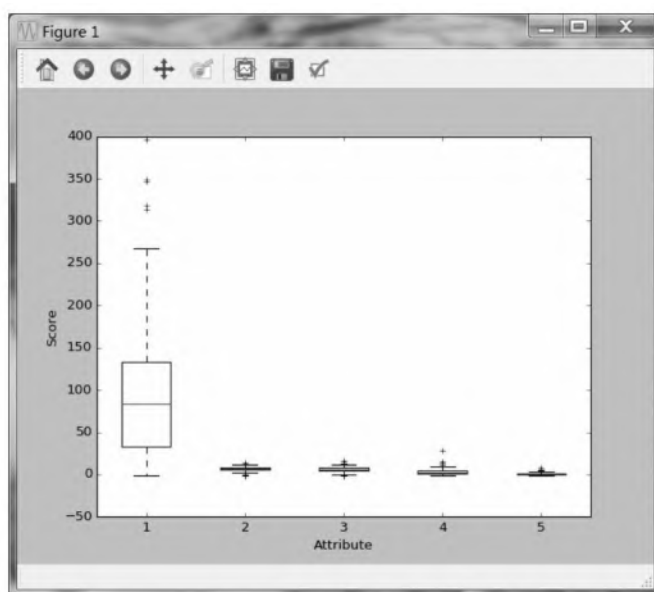


图3-12 数据超预期的四分位图

因此，需要一个能够对数据进行处理，使其具有共同计算均值的方法，这样的方法称为数据的标准化处理。

顾名思义，数据的标准化是将数据根据自身一定比例进行处理，使之落入一个小的特定区间，一般为 $(-1, 1)$ 之间。这样做的目的是去除数据的单位限制，将其转化为无量纲的纯数值，使得不同单位或量

级的指标能够进行比较和加权，其中最常用的就是0-1标准化（0-1 normalization）和Z-score标准化（zero-mean normalization）。

1. 0-1标准化

0-1标准化也叫离差标准化，是对原始数据的线性变换，使结果落到[0, 1]区间，转换函数如下：

$$X = \frac{x - \min}{\max - \min}$$

其中，max为样本数据的最大值，min为样本数据的最小值。这种方法有一个缺陷，就是当有新数据加入时，可能导致max和min的变化，需要重新定义。

2. Z-score标准化

Z-score标准化也叫标准差标准化，经过处理的数据符合标准正态分布，即均值为0，标准差为1，其转化函数为：

$$X = \frac{x - \mu}{\sigma}$$

其中， μ 为所有样本数据的均值， σ 为所有样本数据的标准差。

一般情况下，通过数据的标准化处理后，数据最终落在(-1, 1)之间的概率为99.7%，而在(-1, 1)之外的数据被设置成-1和1，以便处理。

【程序3-10】

```

from pylab import *
import pandas as pd
import matplotlib.pyplot as plot
filePath = ("c://dataTest.csv")
dataFile = pd.read_csv(filePath,header=None, prefix="V")

summary = dataFile.describe()
dataFileNormalized = dataFile.iloc[:,1:6]
for i in range(5):
    mean = summary.iloc[1, i]
    sd = summary.iloc[2, i]

dataFileNormalized.iloc[:,i:(i + 1)] = (dataFileNormalized.iloc[:,i:(i + 1)] -
mean) / sd
array = dataFileNormalized.values
boxplot(array)
plot.xlabel("Attribute")
plot.ylabel(("Score"))
show()

```

从代码中可以看到，数据被处理为标准差标准化的方法，dataFileNormalized被重新计算并定义，大数值被人为限定在(-1,1)之间，请读者自行运行验证。

提示

【程序3-10】中所使用的数据被人为修改，请读者自行修改验证，这里不再进行演示。此外，读者可以对数据进行处理，验证更多的标准化方法。

3.4.4 数据的平行化处理

从3.4.2小节可以看到，对于每种单独的数据属性来说，可以通过数据的四分位法进行处理、查找和寻找离群值，从而对其进行分析处

理。

但是对于属性之间的横向比较，每个目标行属性之间的比较，使用四分位法则较难判断，因此为了描述和表现每一个不同目标行之间数据的差异和不同，需要另一种处理和展示方法。

平行坐标（Parallel Coordinates）是一种常用的可视化方法，用于对高维几何和多元数据进行可视化。

平行坐标为了表示在高维空间的一个点集，在N条平行的线的背景下（一般这N条线都竖直且等距），一个在高维空间的点被表示为一条拐点在N条平行坐标轴的折线，在第K个坐标轴上的位置就表示这个点在第K维的值。

平行坐标是信息可视化的一种重要技术。为了克服传统的笛卡尔直角坐标系容易耗尽空间、难以表达三维以上数据的问题，平行坐标将高维数据的各个变量用一系列相互平行的坐标轴表示，变量值对应轴上的位置。为了反映变化趋势和各个变量间的相互关系，往往将描述不同变量的各个点连接成折线。所以平行坐标图的实质是将欧式空间的一个点 $X_i (x_{i1}, x_{i2}, \dots, x_{im})$ 映射到二维平面上的一条曲线。

平行坐标图可以表示超高维数据。平行坐标的一个显著优点是具有良好的数学基础，其射影几何解释和对偶特性使它很适合用于可视化数据分析。

【程序3-11】

```

from pylab import *
import pandas as pd
import matplotlib.pyplot as plot
filePath = ("c://dataTest.csv")
dataFile = pd.read_csv(filePath,header=None, prefix="V")

summary = dataFile.describe()
minRings = -1
maxRings = 99
nrows = 10
for i in range(nrows):
    dataRow = dataFile.iloc[i,1:10]
    labelColor = (dataFile.iloc[i,10] - minRings) / (maxRings - minRings)
    dataRow.plot(color=plot.cm.RdYlBu(labelColor), alpha=0.5)
plot.xlabel("Attribute")
plot.ylabel("Score")
show()

```

从代码中可以看到，首先计算总体的统计量，之后设置计算的最大值和最小值。本例中人为设置-1为最小值，99为最大值。为了计算简便，选择了前10行作为目标行数进行计算。使用for循环对数据进行训练。

最终图形结果如图3-13所示。

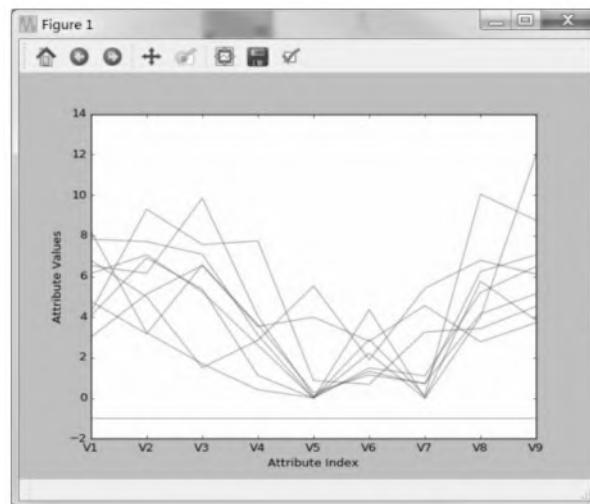


图3-13 属性的图形化展示

从图3-13中可以看到，不同的属性画出了10条不同的曲线，这些曲线根据不同的属性从而画出不同的运行轨迹。

提示

可以选择不同的目标行和不同的属性进行验证，可以观察更多的数据中所展示的结果有何不同。

3.4.5 热点图—属性相关性检测

前面对数据集中数据的属性分别进行了横向和纵向的比较，现在换一种思路，如果对数据属性之间的相关性进行检测的话，该怎么办呢？

热点图是一种判断属性相关性的常用方法，根据不同目标行数据对应的数据相关性进行检测。【程序3-12】展示了对数据相关性进行检测的方法，根据不同数据之间的相关性，做出图形。

【程序3-12】

```
from pylab import *
import pandas as pd
import matplotlib.pyplot as plot
    filePath = ("c://dataTest.csv")
dataFile = pd.read_csv(filePath,header=None, prefix="V")

summary = dataFile.describe()
corMat = DataFrame(dataFile.iloc[1:20,1:20].corr())

plot.pcolor(corMat)
plot.show()
```

最终结果如图3-14所示。

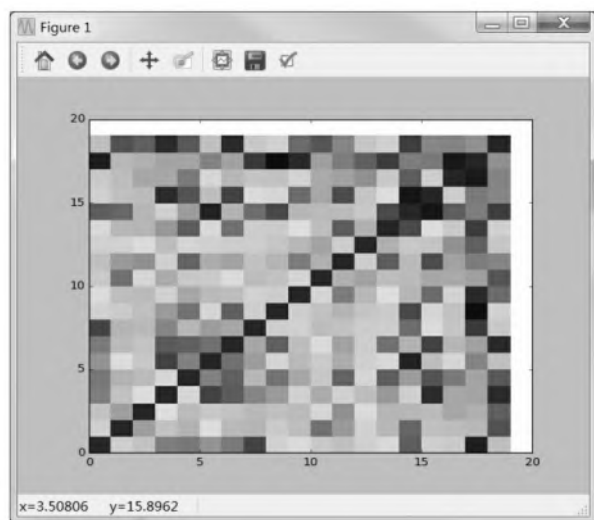


图3-14 属性之间的相关性

不同颜色之间显示了不同的相关性，彩色的深浅显示了相关性的强弱程度。对此读者可以通过打印相关系数来直观地显示数据，相关系数打印方法如下：

```
print(corMat)
```

提示

此处选择了前20行中的前20列数据属性进行计算，可以对其进行更多的验证和显示处理。

3.5 Python实战：某地降雨的关系处理

前面对数据属性间的处理做了大致的介绍，本节将使用这个方法解决一个实际问题。

农业灌溉用水主要来自于天然降水和地下水。随着中原经济区的发展和城镇化水平的提高，城市用水日趋紧张。下面以河南省降水量的变化及分布规律为例进行介绍，为合理调度和利用水资源提供决策。

数据集名为rain.csv，记录了从2000年开始到2011年之间每月的降水量数据，本节将以降水量进行统计计算，找出其规律进行分析。

3.5.1 不同年份的相同月份统计

对于不同年份，每月的降水量也是不同，一般情况下，降水量会随着春夏秋冬的交替呈现不同的状态，一个横向变化的过程。对于不同的年份来说，每月的降水量应该在一个范围内浮动，而不应偏离均值太大。

【程序3-13】

```
from pylab import *
import pandas as pd
import matplotlib.pyplot as plot
filePath = ("c://rain.csv")
dataFile = pd.read_csv(filePath)

summary = dataFile.describe()
print(summary)

array = dataFile.iloc[:,1:13].values
boxplot(array)
plot.xlabel("month")
plot.ylabel(("rain"))
show()
```

打印结果如下。

	0	1	2	3	4
count	12.000000	12.000000	12.000000	12.000000	12.000000
mean	2005.500000	121.083333	67.833333	102.916667	263.416667
std	3.605551	103.021144	72.148626	137.993714	246.690258
min	2000.000000	0.000000	0.000000	0.000000	70.000000
25%	2002.750000	17.750000	9.750000	3.000000	136.250000
50%	2005.500000	125.000000	39.500000	51.500000	155.000000
75%	2008.250000	204.500000	123.250000	150.000000	232.500000
max	2011.000000	295.000000	192.000000	437.000000	833.000000

	5	6	7	8	9
count	12.000000	12.000000	12.000000	12.000000	12.000000
mean	1134.583333	2365.666667	2529.000000	1875.500000	1992.416667
std	618.225240	705.323180	1120.231226	603.135821	670.834414
min	218.000000	766.000000	865.000000	746.000000	621.000000
25%	685.500000	2117.000000	1770.250000	1723.500000	1630.000000
50%	951.500000	2440.500000	2023.500000	1943.500000	1961.000000
75%	1599.000000	2723.750000	3603.000000	2321.750000	2231.750000
max	2134.000000	3375.000000	4163.000000	2508.000000	3097.000000

	10	11	12
count	12.000000	12.000000	12.000000
mean	1219.250000	159.333333	38.333333
std	743.534938	124.611639	34.494620
min	328.000000	0.000000	0.000000
25%	612.250000	64.000000	18.750000
50%	1208.500000	123.000000	25.500000
75%	1672.250000	278.250000	46.250000
max	2561.000000	357.000000	100.000000

从打印结果可以看到，程序对平均每个月份的降水量进行了计算，获得了其偏移值、均值以及均方差的大小。

通过四分位的计算可以获得一个波动范围，具体结果如图3-15所示。

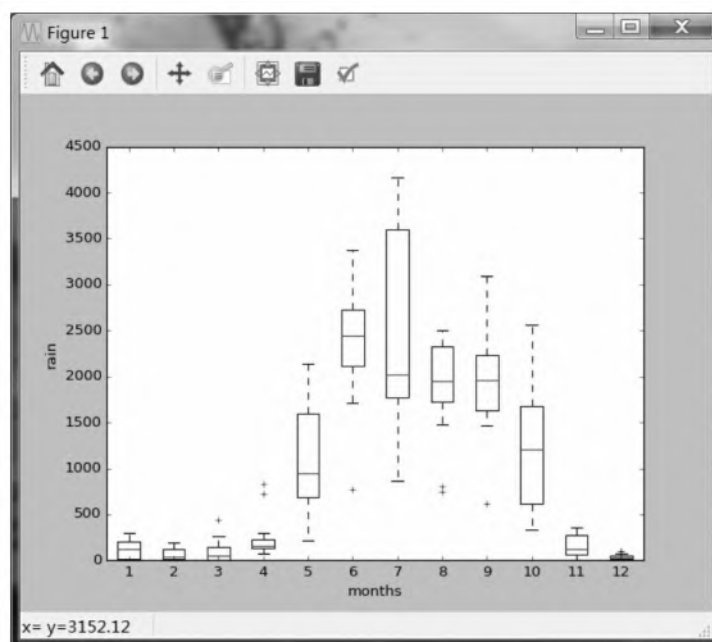


图3-15 降水量的四分位图

从图3-15中可以直观地看到，不同月份之间的降水量有很大的差距，春秋（1~4月）降水量明显较少，5月份开始降水量有个明显增多的过程，而在7月份达到顶峰后回落，11月份和12月份达到最低的降水量。

同时可以看到，有几个月份的降水量有明显的偏移，即离群值出现，这点可能跟年度情况有关，需要继续进行分析。

3.5.2 不同月份之间的增减程度比较

正常情况下，每年降水量都呈现一个平稳的增长或者减少的过程，其下降的坡度（趋势线）应该是一样的。【程序3-14】展示了这种趋势。

【程序3-14】

```

from pylab import *
import pandas as pd
import matplotlib.pyplot as plot
    filePath = ("c://rain.csv")
dataFile = pd.read_csv(filePath)

summary = dataFile.describe()
minRings = -1
maxRings = 99
nrows = 11
for i in range(nrows):
    dataRow = dataFile.iloc[i,1:13]
    labelColor = (dataFile.iloc[i,12] - minRings) / (maxRings - minRings)
    dataRow.plot(color=plot.cm.RdYlBu(labelColor), alpha=0.5)
plot.xlabel("Attribute")
plot.ylabel(("Score"))
show()

```

最终打印结果如图3-16所示。

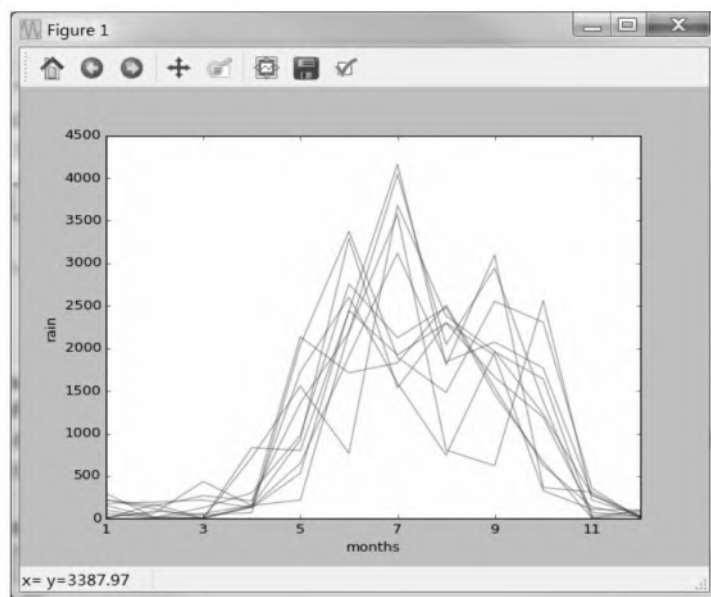


图3-16 降水量的趋势图

从图3-16中可以明显地看到，降雨的月份并不是规律地上涨或下跌，而是呈现一个不规则的浮动状态，增长最快的为6、7月份，而下降最快的为7、8月份，之后有一个明显的回升过程。

3.5.3 每月的降水量是否相关

每月的降水量理论上来说应该是具有相互独立性的，即每月的降水量和其他月份没有关系。但是实际是这样的吗？

【程序3-15】

```
from pylab import *
import pandas as pd
import matplotlib.pyplot as plot
    filePath = ("c:// rain.csv")
dataFile = pd.read_csv(filePath)

summary = dataFile.describe()
corMat = DataFrame(dataFile.iloc[1:20,1:20].corr())

plot.pcolor(corMat)
plot.show()
```

通过计算，最终结果如图3-17所示。

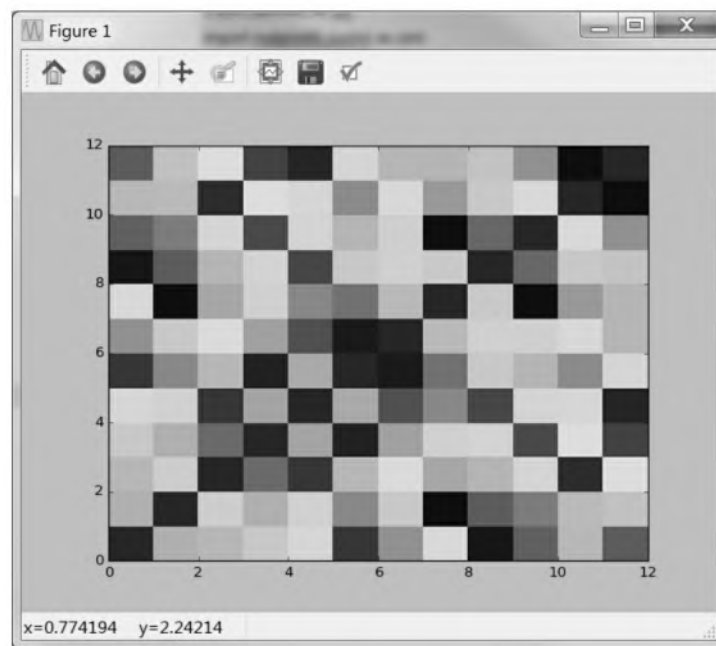


图3-17 月份之间的相关性显示

从图3-17可以看到，颜色分布比较平均，对应的区块相互之间没有太大的相关性，因此可以认为每月的降水量和其他月份没有关系，是独立行为。

3.6 小结

本章从直观的观察开始，逐渐深入介绍和研究了数据集和分析工具，了解了使用Python类库进行数据分析的基本方法。数据分析从最基本的矩阵转换开始，直到对数据集特征值进行分析和处理，通过对本章内容的学习，读者可以为学习数据分析打下基础。

使用相应的类库进行深度学习程序设计是本章的重点，也是希望读者能够掌握的内容。再一次强调，请读者尽量使用Python已有的类库进行程序设计。在数据的可视化展示过程中，通过做出多种数据图形向读者演示了使用不同的类库可以非常直观地进行数据分析，希望本章中提供的不同研究方法和程序设计思路能够帮助读者掌握基本数据集的描述性和统计值之间的关系，这些非常有利于对数据的掌握。

本章是数据处理的基础，内容简单，但是非常重要，希望读者能够使用不同的数据集进行处理并演示更多的值。

第4章

欢迎来到掘金量化

欢迎来到掘金量化！

掘金量化交易平台（软件启动界面见图4-1）涵盖量化交易完整的生命周期，支持多语言策略开发、tick级回测、仿真交易与实盘交易，为客户提供安全、专业、高效的量化IT解决方案。

本章将介绍掘金量化平台的注册与使用。



图4-1 掘金量化

4.1 基础工作

本节将介绍使用掘金量化平台前需要做的各种准备工作，包括从开始注册到获取帮助。

4.1.1 安装掘金终端

在处理测量数据时，经常要研究变量之间的关系。变量之间的关系一般分为两种：

- 完全确定关系，即函数关系。
- 相关关系，即变量之间既存在着密切联系，但又不能由一个或多个变量的值求出另一个变量的值。

1. 下载终端

登录掘金官网（<http://www.myquant.cn>）申请试用，申请通过后下载终端。完成后双击安装应用程序，进入安装向导，根据向导完成安装。

2. 注册用户 / 登录用户

启动终端后，进入用户登录界面（见图4-2），使用手机号为用户名进行注册并登录。



图4-2 登录掘金量化终端

3. 安装对应的SDK

进入终端后，单击左上角的logo，如图4-3所示。



图4-3 单击登录logo

在选择策略编辑语言界面上选择自己熟悉的语言，进行相应SDK的安装，如图4-4所示。



图4-4 安装SDK

之后进入掘金量化终端的首页，如图4-5所示。



图4-5 掘金量化终端首页

读者需要注意的是，在安装SDK之前，需确认已经安装有相应的编程语言环境，比如安装Python语言的SDK前，需要先安装Python 2或者Python 3.6的环境。这里建议安装Python 3.6。

4.1.2 获取帮助

在掘金量化中，获取帮助一般分为两种：从网站直接获取帮助和通过QQ群获取帮助。这里建议读者先尝试自己从网站查找对应的API文档，遇到经过查询无法解决的问题时，再通过QQ群获取帮助。

1. 获取帮助

通过首页右上方的“帮助中心”获取帮助，如图4-6所示。



图4-6 单击“帮助中心”

2. 一些常用页面介绍

(1) 依次单击“帮助中心”→“API文档”→Python，在代码编写过程中，可通过该页面查阅基本API函数和一些重要概念的解释，如图4-7所示。

图4-8 数据文档介绍

除此之外，掘金网站上有很多不同的页面能够提供不同的文档介绍和帮助，希望读者多多使用。对于一些无法解决的问题，还可以通过网站社区上提供的问答来解决。

4.2 实战：使用掘金终端进行回测工作

下面开始使用掘金终端进行回测工作。使用掘金量化相比较其他平台的好处在于，可以使用Python IDE编辑器进行数据处理和编辑，并通过掘金终端获取回测结果。推荐使用IDE对数据进行编辑，而不是使用掘金终端对工作进行回测。

4.2.1 创建第一个策略

下面将带领读者创建第一个策略。

选择主界面中“策略研究”，进入策略研究页面，如图4-9所示。该页面有“新建策略”和“策略研究”两个菜单。



图4-9 策略研究

单击“新建策略”菜单，这里提供了多个不同类型的策略模板，可选择最合适的模板开始编写，如果都不匹配，可单击“空策略”来构建全新的策略，如图4-10所示。



图4-10 新建策略

选择所需要的策略类型，输入策略名之后，单击“确认”按钮，这时会出来一个策略ID，如需在熟悉的IDE中编写代码，则可将此ID复制到所编写的代码中与此终端进行连接，方可进行后续操作，如图4-11所示。



图4-11 策略ID

如果需要再次查看策略ID或者修改策略解析器路径等一系列策略环境配置，可以单击“设置”按钮进行设置，如图4-12所示。



图4-12 查看相关内容

这是使用掘金量化终端创建第一个策略的方法，更多的尝试希望读者自行测试完成。

4.2.2 运行回测

下面进行运行回测。

(1) 单击“策略研究”，选择指定策略进入策略编写界面，如图4-13所示。

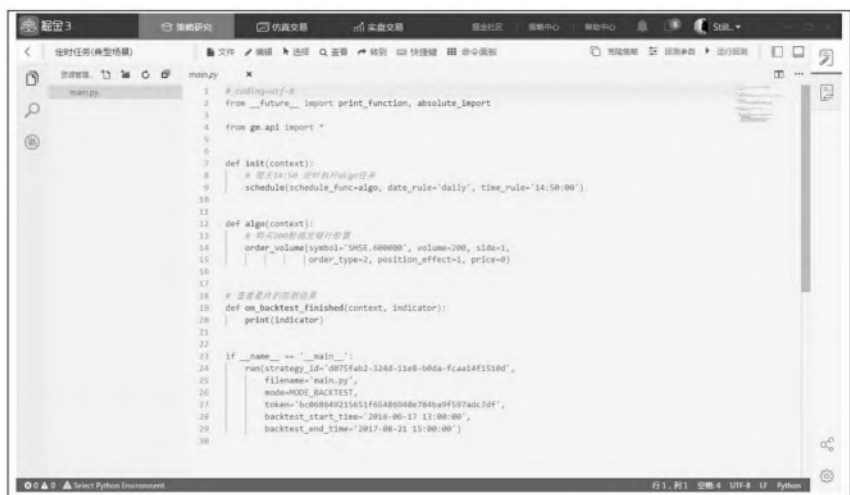


图4-13 策略编写界面

(2) 该页面是一个完整的IDE工作区，可对策略内容进行修改调试以及回测，若策略较复杂，则可以选择右上角的“分页”按钮对策略进行分段编辑，如图4-14所示。



图4-14 IDE回测编辑界面

(3) 策略修改完成之后，单击右上方的“回测参数”按钮可对策略基本数据参数进行调整（见图4-15），随后单击“运行回测”按钮即可开始回测。



图4-15 设置起始参数

4.2.3 查看回测结果

在研究策略页面单击该策略中“回测次数”，如图4-16所示。



图4-16 回测次数

进入对应的回测结果界面，如图4-17所示。



图4-17 进入回测结果页面

该页面展示了此策略回测的所有明细情况，包括收益概览、信号分析、交易明细、每日持仓和输出日志。可单击屏幕右上方的“结果下载”将回测结果保存在本地，如图4-18所示。



图4-18 保存回测结果

至此，查看回测结果介绍完毕，读者可以通过下载掘金IDE和安装Python类库来自行测试，这里主要以了解为主，具体的使用推荐Python专用IDE，例如Pycharm。

4.2.4 使用PyCharm进行回测

前面已经介绍过，PyCharm是一种Python IDE，带有一整套可以帮助用户在使用Python语言开发时提高效率的工具，比如调试、语法高亮、Project管理、代码跳转、智能提示、自动完成、单元测试、版本控制。此外，该IDE提供了一些高级功能，以用于支持Django框架下的专业Web开发。

掘金量化支持并鼓励读者使用PyCharm进行量化的回测工作，在这里通过一个例子演示使用PyCharm进行回测的结果。

1. 创建一个新的Python文件

单击File菜单并选择New Project，在弹出的菜单中选择Pure Python（编译器会默认选择当前已安装的Python版本），如图4-19所示。

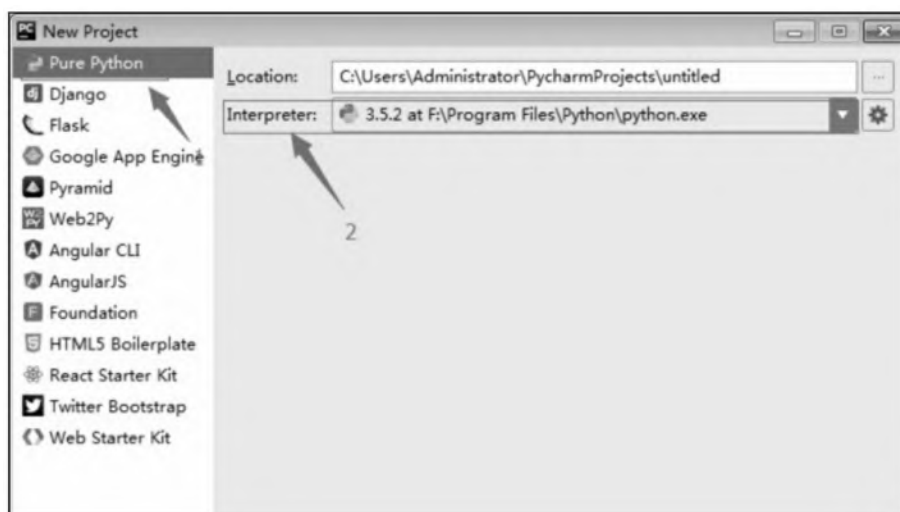


图4-19 新建一个Python工程

在名为untitled的新建项目里，右击它并选择New→File来新建文件，如图4-20所示。

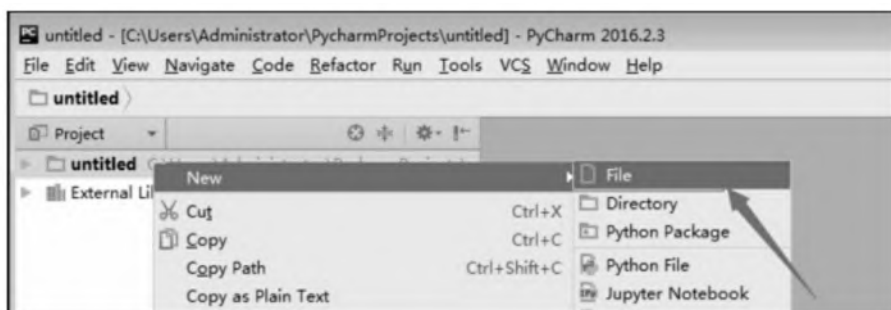


图4-20 创建一个新文件

输入新的文件名，例如HelloQuant，如图4-21所示。

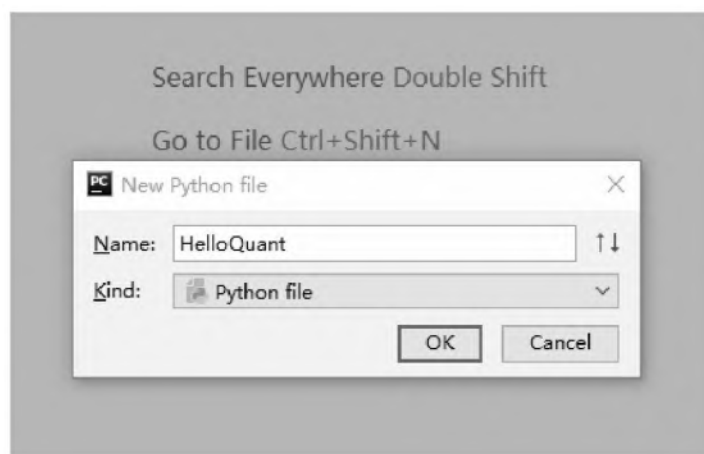


图4-21 设置文件名

2. 从掘金量化终端中复制回测框架

在“策略研究”页面单击“新建策略”菜单，如图4-22所示。



图4-22 新建策略

之后选择一个任务，这里建议选择定时任务，单击“确认”按钮进行创建。之后确定部分变为如图4-23所示的形态。



图4-23 确定已创建的任务

此时单击“策略编辑”按钮对策略进行编辑处理。这里生成的是一个具有完整框架的回测程序，如图4-24所示。

```

main.py x
1  # coding=utf-8
2  from __future__ import print_function, absolute_import
3
4  from gm.api import *
5
6
7  def init(context):
8      # 每天14:50 定时执行algo任务
9      schedule(schedule_func=algo, date_rule='daily', time_rule='14:50:00')
10
11
12  def algo(context):
13      # 购买200股浦发银行股票
14      order_volume(symbol='SHSE.600000', volume=200, side=1,
15                  order_type=2, position_effect=1, price=0)
16
17
18  # 查看最终的回测结果
19  def on_backtest_finished(context, indicator):
20      print(indicator)
21
22
23  if __name__ == '__main__':
24      run(strategy_id='36b05b5f-8fe6-11e8-a628-4cedfb681747',
25          filename='main.py',
26          mode=MODE_BACKTEST,
27          token='e8978d765c4822e5a85fcaa73e044065cf17b58b',
28          backtest_start_time='2016-06-17 13:00:00',
29          backtest_end_time='2017-08-21 15:00:00')
30

```

图4-24 创建策略文件

【程序4-1】

```

# coding=utf-8
from __future__ import print_function, absolute_import

from gm.api import *

def init(context):
    # 每天 14:50 定时执行 algo 任务
    schedule(schedule_func=algo, date_rule='daily', time_rule='14:50:00')

def algo(context):
    # 购买 200 股浦发银行股票
    order_volume(symbol='SHSE.600000', volume=200, side=1,
                  order_type=2, position_effect=1, price=0)
    # 查看最终的回测结果
def on backtest finished(context, indicator):
    print(indicator)
if __name__ == '__main__':
    run(strategy_id='36b05b5f-8fe6-11e8-a628-4cedfb681747',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='e8978d765c4822e5a85fcaa73e044065cf17b58b',
        backtest_start_time='2016-06-17 13:00:00',
        backtest_end_time='2017-08-21 15:00:00')

```

可以将以上代码段复制到PyCharm IDE中直接使用，与正常的掘金IDE一样。

注意

对于不同的文件名，需要将文件名改成PyCharm中Python的名称，即将 `filename='main.py'` 改成 `filename='HelloQuant.py'`，这点请读者切记。

显示的回测结果如图4-25所示。

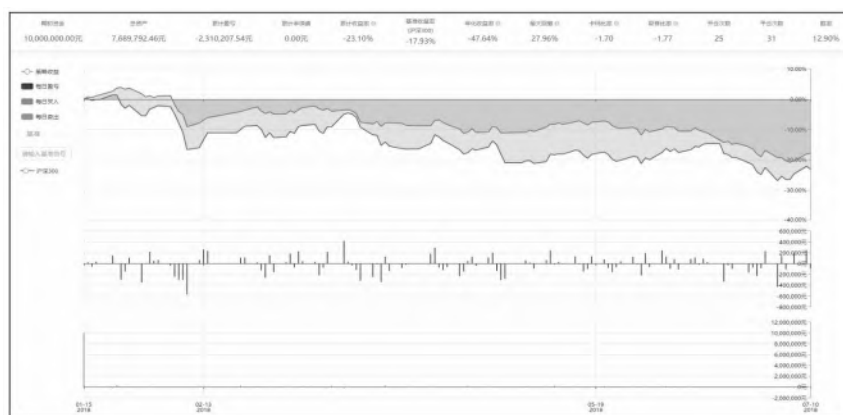


图4-25 显示策略回测结果

4.3 小结

本章主要介绍使用掘金IDE对数据进行回测，首先使用的是掘金专用的客户端，但是这并不是作者建议的方式，建议使用专门的PyCharm IDE编辑器对数据进行编辑。

第5章

Talib金融库使用详解

工欲善其事必先利其器。

前面介绍了Numpy、pandas以及Matplot等多种常用的Python类库的使用。虽然这些工具能帮助读者准确获取所关注的内容，但是由于其不具有专业性，还需要额外让读者自行编写程序进行下一步的解析。

Talib的简称是Technical Analysis Library，主要功能是计算行情数据的技术分析指标，用来开发技术分析策略。官网称其为TA-Lib，为了简便起见，本书使用Talib指代TA-Lib。

5.1 Talib金融工具库的介绍

Talib的中文翻译为技术分析库，是一种以Python为基础的广泛用于量化交易中对金融市场数据进行分析的函数库。其通过内置的函数给使用者提供了多种技术分析函数和多种变量，极大地方便了量化投资中的数据编程工作。下面以股票分析为例进行介绍，其他如期货、期权的交易在程序设计上类似，就不再额外阐述了。

5.1.1 使用Talib获取3日、7日、15日均线

学习一个新的技术，最好的方法是实际进行在实际中学习。

(1) 对于Talib的使用，首先需要获取实时的股票数据，通过使用第4章所提供的API可以很方便地取得所需要的数据，代码如下：

```
from gm.api import *
import numpy as np
import talib
```

第一步是引入所需要的Python库包，这里gm.api是掘金量化通信包；numpy是数学处理包，对于获得的数据进行处理；Talib是本章介绍的包，用于对数据进行下一步处理。

(2) 在前面介绍掘金量化库包的时候说过，与掘金量化数据库进行通信必须使用对应的账户与密码，而下面的代码中，set_token是将所对应的账户与密码进行加密后发送给服务器端。history_n函数用于获取前若干段的时间序列内容。

```
set_token("*****")

data = history_n(symbol="SZSE.399006", frequency="1d", count=100, end_time="2017-12-31", fields="close", fill_missing="last", adjust=ADJUST_PREV, df=True)
```

(3) 对获取的数据进行处理，对于不同的目的来说，所需要的均线值不同，Talib提供了相应的均线计算值，代码如下：

```
close = np.asarray(data["close"].values)
```

(4) 前面已经说过，history_n函数通过设置参数df=True获得一个pandas类型的返回值。提取pandas的返回值，并通过Numpy设置成数组矩阵。

```
ma3 = talib.MA(close, timeperiod = 3)
```

(5) Talib中的MA函数是计算滑动平均值的专用函数，通过设置timeperiod时间周期可以计算相应的滑动平均周期。

【程序5-1】

```

from gm.api import *
import talib
import numpy as np

set_token("9e02621e7fcf850731def2e8a9delfdbd5b21ad6")

data = history_n(symbol="SZSE.399006", frequency="1d", count=100, end_time="2017-12-31", fields="close", fill_missing="last", adjust=ADJUST_PREV, df=True)
close = np.asarray(data["close"].values)
ma3 = talib.MA(close, timeperiod = 3)
print(ma3)

```

最终打印结果如图5-1所示。

[	nan	nan	nan	nan	1756.86557617
1763.07770996	1770.18171387	1782.60927734	1798.55507812	1814.4878418	
1821.63481445	1824.90405273	1821.33994141	1813.80812988	1812.0230957	
1814.74545898	1818.08063965	1824.33354492	1835.06149902	1845.53588867	
1853.69377441	1864.4079834	1877.25314941	1885.45383301	1889.45856934	
1892.01005859	1891.87443848	1888.9895752	1886.76359863	1884.83579102	
1884.52116699	1883.68361816	1884.53093262	1883.23439941	1881.39238281	
1871.15620117	1862.98798828	1855.44274902	1851.83562012	1851.9470459	
1861.19040527	1876.79086914	1886.11245117	1895.16286621	1906.98139648	
1905.55100098	1897.45773926	1890.77932129	1883.30432129	1874.1322998	

图5-1 5日滑动平均值

从打印结果可以看到，前4项数据内容由于滑动平均周期的设置，没有相应的内容，这在显示时并没有什么问题，但是在后续计算时，由于nan并不是一个具体的数值，有很大可能报错。

这里NumPy提供了相应的函数对其进行处理，代码如下：

```
ma3 = np.nan_to_num(ma3)
```

具体结果请读者自行验证。

而对于不同滑动平均值的计算，通过设置不同的timeperiod值进行针对性设置。

可能有读者提出疑问，在介绍history_n函数时，着重介绍了其中的参数设置，其中的参数之一frequency是对函数获取的标的的时间周期进行设置，此时在【程序5-2】中可以通过获取不同的frequency对序列时间进行设置。

【程序5-2】

```
from gm.api import *
import talib
import numpy as np

set_token("9e02621e7fcf850731def2e8a9de1fdbd5b21ad6")

data =
history_n(symbol="SZSE.399006",frequency="3600s",count=100,end_time="2017-12-31",fields="close",fill_missing="last",adjust=ADJUST_PREV,df=True)
close = np.asarray(data["close"].values)
ma5 = talib.MA(close,timeperiod = 5)
ma5 = np.nan_to_num(ma5)
print(ma5)
```

通过不同timeperiod可以获取不同的时间端数据。然而无论获取的数据多少，其都是基于统一的“1d”，也就是以日为时间周期去获取。而有时读者在进行程序设计的时候需要使用不同的时间周期去进行处理，此时则只需要将frequency设置成

```
frequency="3600s"
```

即可。

5.1.2 EMA的计算

EMA的中文意思是指数平均数指标，也叫EXPMA指标，是一种趋向类指标，是以指数式递减加权的移动平均。

EMA的公式如下：

$$EMA = \alpha \times Price_N + (1 - \alpha) \times Price_{(N-1)}$$

其中， α 为平滑指数，一般取作 $2/(N+1)$ 。

【程序5-3】

```
from gm.api import *
import talib
import numpy as np

set_token("9e02621e7fcf850731def2e8a9de1fdbd5b21ad6")
data = history_n(symbol="SZSE.399006", frequency="1d", count=100, end_time="2017-12-31", fields="close", fill_missing="last", adjust=ADJUST_PREV, df=True)
close = np.asarray(data["close"].values)
ema = talib.EMA(close)
ema = np.nan_to_num(ema)
print(ema)
```

同样，EMA的最终计算结果上，前面若干个值为空，通过nan_to_num将其转化成值为0的数。结果如图5-2所示。

[	0.	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.	
	1833.86779378	1837.79904877	1840.54081251	1843.90391785	1845.76623124	
	1847.09902668	1846.87465966	1846.39599273	1846.94739636	1847.44976633	
	1848.7098904	1851.36332552	1855.62747495	1858.59026103	1861.26096716	
	1865.44247514	1866.55497076	1867.22828445	1867.28830283	1866.98655465	

图5-2 滑动平均值计算结果

有读者看到这里会提出疑问，为什么前面会有29个0值（nan）。这里EMA的函数代码如下：

```
def EMA(real, *args, **kwargs): # real signature unknown; NOTE: unreliably
restored from __doc__
    """
    EMA(real[, timeperiod=?])
    Exponential Moving Average (Overlap Studies)
    Inputs:
        real: (any ndarray)
    Parameters:
        timeperiod: 30
    Outputs:
        real
    """
    pass
```

可以看到，这里的时间周期被设置成30，如果有别的需要，例如在MACD或者RSI中计算不同的时间趋势时，则可以通过对时间周期进行设置从而获得不同的结果。请读者自行设置验证。

5.1.3 MACD的计算

MACD称为指数平滑移动平均线，是金融分析指标中常用的一个数据指标，用于对股票趋势的分析。

MACD的意义和双移动平均线基本相同，即由快、慢均线的离散、聚合表征当前的多空状态和股价可能的发展变化趋势，但阅读起来更方便。当MACD从负数转向正数时，是买的信号。当MACD从正数转向负数时，是卖的信号。当MACD以大角度变化时，表示快的移动平均线和慢的移动平均线的差距非常迅速地拉开，代表一个市场大趋势的转变。

同样，Talib中也提供了这个指标的处理函数，代码如下：

```
def MACD(real, *args, **kwargs): # real signature unknown; NOTE: unreliably
restored from __doc__
    """
    MACD(real[, fastperiod=?, slowperiod=?, signalperiod=?])

    Inputs:
        real: (any ndarray)

    Parameters:
        fastperiod: 12
        slowperiod: 26
        signalperiod: 9

    Outputs:
        macd
        macdsignal
        macdhist

    """
    pass
```

这里简单介绍一下MACD函数。可以看到，在函数说明中，MACD函数的默认值分别为12、26、9。这里分别是快速（一般选12日）移动平均值与慢速（一般选26日）移动平均值。

以这两个数值作为测量两者（快速与慢速线）间的“差离值”的依据。所谓“差离值”（DIF），即12日EMA数值减去26日EMA数值。因此，在持续的涨势中，12日EMA在26日EMA之上。其间的正差离值（+DIF）会愈来愈大。反之在跌势中，差离值可能变负（-DIF），也愈来愈大。至于行情开始回转，正或负差离值要缩小到一定的程度，才真正是行情反转的信号。MACD的反转信号界定为“差离值”的9日移动平均值（9日EMA）。

MACD的使用代码如下：

```
macd, signal, hist = talib.MACD(close, fastperiod=12,
slowperiod=26,
signalperiod=9)
```

对于其返回值来说，signal与hist分别为MACD的参数辅助量，现在可以不考虑。完整代码如下：

【程序5-4】

```
from gm.api import *
import talib
import numpy as np
set_token("9e02621e7fcf850731def2e8a9de1fdbd5b21ad6")

data = history_n(symbol="SZSE.399006", frequency="1d", count=100, end_time="2017-12-31", fields="close", fill_missing="last", adjust=ADJUST_PREV, df=True)
close = np.asarray(data["close"].values)

macd, signal, hist = talib.MACD(close, fastperiod=12, slowperiod=26, signalperiod=9)
macd = np.nan_to_num(macd)
print(macd)
```

结果如图5-3所示。

0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	25.7819188	0.
22.91714579	18.59244096	14.65992805	12.64729896	10.90968213		
10.40098779	11.70721449	14.80046144	15.78634621	16.254062		
18.51637928	16.61667157	14.48464484	11.94457842	9.37585263		
8.66070096	9.48108596	10.24979057	11.65833057	12.42620038		
11.72964957	7.84044392	5.80493218	4.38005903	1.20500812		
-2.4608435	-3.92195266	-4.32276903	-4.02322122	-2.30923084		
0.37772329	2.94176082	4.14201739	2.7505036	2.44091842		
-1.36848967	-2.27499697	-3.18178692	-4.14176998	-9.52112149		
-14.59436889	-20.13843504	-21.62265145	-23.05648575	-25.27961185		
-23.9954489	-23.26288752	-25.54896309	-24.99674712	-24.86356266		
-23.26608476	-19.76867039	-18.26258327	-16.28422296	-15.48566071		
-15.51084573	-15.61936569	-14.10181762	-14.03561284	-13.23962222		
-13.29449209	-15.06326968	-16.03692785	-17.70067982	-18.81990271		
-18.87329326						

图5-3 MACD计算结果

可以看到图中的值有正值和负值，根据MACD的设计形式，当MACD的值由负变成正的时候，是买入点。而值由正变成负的时候，就是对应的卖出点。

其次，对于MACD函数中参数的设置，读者也可以直接写成：

```
macd, signal, hist = talib.MACD(close)
```

这样使用的是talib.MACD的默认参数，如果读者有其他的需求，也可以自定义使用不同的时间周期去进行演练。

```
macd, signal, hist = talib.MACD(close, fastperiod=x,  
slowperiod=y  
signalperiod=z)
```

代码中的x、y、z分别为对应的周期值。

有的时候对于数据的展示，更多的是希望以图形的形式将其展示出来，根据生成的macd返回值可以获得一组用于图形展示的数组，而此时通过对应的作图程序可以很方便地将其展示出来，代码如下：

【程序5-5】

```
from gm.api import *  
import talib  
import numpy as np  
import matplotlib.pyplot as plt  
  
set_token("9e02621e7fcf850731def2e8a9de1fdbd5b21ad6")  
  
data = history_n(symbol="SZSE.399006", frequency="1d", count=100, end_time="2017-  
12-31", fields="close", fill_missing="last", adjust=ADJUST_PREV, df=True)  
  
close = np.asarray(data["close"].values)  
macd, signal, hist = talib.MACD(close)  
macd = np.nan_to_num(macd)  
  
plt.plot(macd, "r")  
plt.show()
```

最终显示结果如图5-4所示。

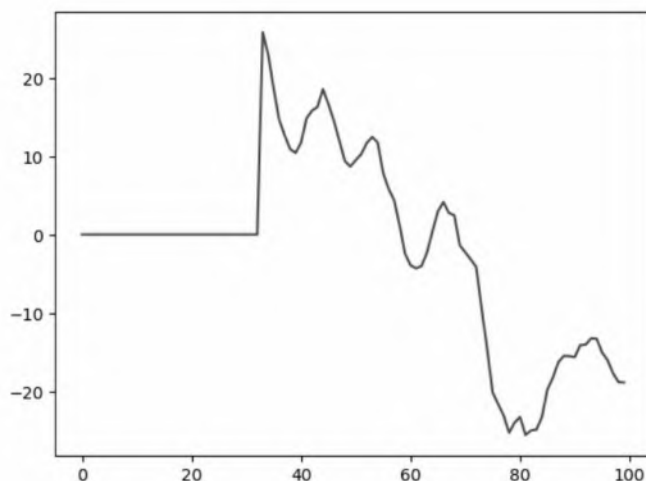


图5-4 MACD计算结果的图形化展示

这里可以看到，在设定的时间周期上，MACD的值在连续降低，负值明显多于正值，可以认为这段时间周期中指数在不停地降低。

5.1.4 MACD斜率的计算方法

MACD的值只能用于反映不同时间周期内的买卖趋势情况。而对于上涨的快慢，MACD并没有提供相应的函数去进行计算，此时读者可以通过其他的既有函数去获取相应的值。

斜率表示一条直线（或曲线的切线）关于（横）坐标轴倾斜程度的量。它通常用直线（或曲线的切线）与（横）坐标轴夹角的正切，或两点的纵坐标之差与横坐标之差的比来表示。

在这里指的是MACD值在单位时间内变化情况的大小。NumPy提供了专门的函数去处理相邻数间的差值大小。

```
def diff(a, n=1, axis=-1):
```

其作用是对传入的一维数组进行差值计算，返回相邻数之间的对应差值。

【程序5-6】

```
from gm.api import *
import talib
import numpy as np
import matplotlib.pyplot as plt

set_token("9e02621e7fcf850731def2e8a9de1fdbd5b21ad6")

data = history_n(symbol="SZSE.002310", frequency="1d", count=100, end_time="2018-12-31", fields="close", fill_missing="last", adjust=ADJUST_PREV, df=True)

close = np.asarray(data["close"].values)
macd, signal, hist = talib.MACD(close)
macd = np.nan_to_num(macd)

macd_gradient = np.diff(macd)
print(macd_gradient)
```

【程序5-6】展示了使用diff函数计算MACD斜率数值的方法，其结果如图5-5所示。

[	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.46324258	0.02475937		
0.02739835	0.01718792	0.01537402	-0.05402847	0.00070214	-0.01386174		
0.00679473	0.04603497	0.04035663	-0.00762896	0.04261175	0.02963671		
0.02391946	0.02156092	0.00590579	-0.03514447	-0.00947015	-0.05386237		

图5-5 MACD斜率的计算结果

图5-5展示了部分内容，这里需要与图5-3对比观察。

5.1.5 使用Talib实现国内金融数据指标

对于国内常用的金融指标，在设计和具体的数据计算上略有不同，这里通过使用Talib实现部分数据指标。

1. SMA_CN的实现

```
def SMA_CN(close, timeperiod) :  
    return reduce(lambda x, y: ((timeperiod - 1) * x + y) / timeperiod, close)
```

2. RSI_CN的实现

```
def RSI_CN(close, timeperiod):  
    diff = map(lambda x, y: x - y, close[1:], close[:-1])  
    diffGt0 = map(lambda x: 0 if x < 0 else x, diff)  
    diffABS = map(lambda x: abs(x), diff)  
    diff = np.array(diff)  
    diffGt0 = np.array(diffGt0)  
    diffABS = np.array(diffABS)  
    diff = np.append(diff[0], diff)  
    diffGt0 = np.append(diffGt0[0], diffGt0)  
    diffABS = np.append(diffABS[0], diffABS)  
    rsi = map(lambda x: SMA_CN(diffGt0[:x], timeperiod) / SMA_CN(diffABS[:x],  
timeperiod) * 100,  
              range(1, len(diffGt0) + 1))  
    return np.array(rsi)
```

3. KDJ_CN的实现

```
def KDJ_CN(high, low, close, fastk_period, slowk_period, fastd_period):  
    kValue, dValue = talib.STOCHF(high, low, close, fastk_period,  
fastd_period=1, fastd_matype=0)  
    kValue = np.array(map(lambda x: SMA_CN(kValue[:x], slowk_period), range(1,  
len(kValue) + 1)))  
    dValue = np.array(map(lambda x: SMA_CN(kValue[:x], fastd_period), range(1,  
len(kValue) + 1)))  
  
    jValue = 3 * kValue - 2 * dValue  
  
    func = lambda arr: np.array([0 if x < 0 else (100 if x > 100 else x) for x  
in arr])  
  
    kValue = func(kValue)  
    dValue = func(dValue)  
    jValue = func(jValue)  
    return kValue, dValue, jValue
```

4. MACD_CN的实现

```
def MACD_CN(close, fastperiod = 12, slowperiod = 26, signalperiod = 9):  
    macdDIFF, macdDEA, macd = talib.MACDEXT(close, fastperiod=fastperiod,  
    fastmatype=1,slowperiod=slowperiod, slowmatype=1, signalperiod=signalperiod,  
    signalmatype=1)  
    macd = macd * 2  
    return macdDIFF, macdDEA, macd
```

5.2 Talib金融工具库函数

5.1节介绍了量化金融中常用的滑动平均、MACD等，除此之外，Talib还提供了一些其他常用的用以量化分析的函数，参见表5-1。

表5-1 Talib常用函数名及简称

简称	函数名
AD	Chaikin A/D Line
ADOSC	Chaikin A/D Oscillator
ADX	Average Directional Movement Index
ADXR	Average Directional Movement Index
APO	Absolute Price Oscillator
AROON	Aroon
AROONOSC	Aroon Oscillator
ATR	Average True Range
AVGPRICE	Average Price
BBANDS	Bollinger Bands
BETA	Beta
BOP	Balance Of Power
CCI	Commodity Channel Index
CDL2CROWS	Two Crows
CDL3BLACKCROWS	Three Black Crows
CDL3INSIDE	Three Inside Up/Down
CDL3LINESTRIKE	Three-Line Strike
CDL3OUTSIDE	Three Outside Up/Down
CDL3STARSINSOUTH	Three Stars In The South
CDL3WHITESOLDIERS	Three Advancing White Soldiers
CDLABANDONEDBABY	Abandoned Baby
CDLADVANCEBLOCK	Advance Block
CDLBELTHOLD	Belt-hold
CDLBREAKAWAY	Breakaway
CDLCLOSINGMARUBOZU	Closing Marubozu

(续表)

简称	函数名
CDLCONCEALBABYSWALL	Concealing Baby Swallow
CDLCOUNTERATTACK	Counterattack
CDLDARKCLOUDCOVER	Dark Cloud Cover
CDLDOJI	Doji
CDLDOJISTAR	Doji Star
CDLDRAGONFLYDOJI	Dragonfly Doji
CDLENGULFING	Engulfing Pattern
CDLEVENINGDOJISTAR	Evening Doji Star
CDLEVENINGSTAR	Evening Star
CDLGAPSIDESIDEWHITE	Up/Down-gap side-by-side white lines
CDLGRAVESTONEDOJI	Gravestone Doji
CDLHAMMER	Hammer
CDLHANGINGMAN	Hanging Man
CDLHARAMI	Harami Pattern
CDLHARAMICROSS	Harami Cross Pattern
CDLHIGHWAVE	High-Wave Candle
CDLHIKKAKE	Hikkake Pattern
CDLHIKKAKEMOD	Modified Hikkake Pattern
CDLHOMINGPIGEON	Homing Pigeon
CDLIDENTICAL3CROWS	Identical Three Crows
CDLINNECK	In-Neck Pattern
CDLINVERTEDHAMMER	Inverted Hammer
CDLKICKING	Kicking
CDLKICKINGBYLENGTH	Kicking - bull/bear determined by the longer marubozu
CDLLADDERBOTTOM	Ladder Bottom
CDLLONGLEGGEDOJI	Long Legged Doji
CDLLONGLINE	Long Line Candle
CDLMARUBOZU	Marubozu
CDLMATCHINGLOW	Matching Low
CDLMATHOLD	Mat Hold
CDLMORNINGDOJISTAR	Morning Doji Star

(续表)

简称	函数名
CDLMORNINGSTAR	Morning Star
CDLONNECK	On-Neck Pattern
CDLPIERCING	Piercing Pattern
CDLRICKSHAWMAN	Rickshaw Man
CDLRISEFALL3METHODS	Rising/Falling Three Methods
CDLSEPARATINGLINES	Separating Lines
CDLSHOOTINGSTAR	Shooting Star
CDLSHORTLINE	Short Line Candle
CDLSPINNINGTOP	Spinning Top
CDLSTALLEDPATTERN	Stalled Pattern
CDLSTICKSANDWICH	Stick Sandwich
CDLTAKURI	Takuri (Dragonfly Doji with very long lower shadow)
CDLTASUKIGAP	Tasuki Gap
CDLTHRUSTING	Thrusting Pattern
CDLTRISTAR	Tristar Pattern
CDLUNIQUE3RIVER	Unique 3 River
CDLUPSIDEGAP2CROWS	Upside Gap Two Crows
CDLXSIDEGAP3METHODS	Upside/Downside Gap Three Methods
CMO	Chande Momentum Oscillator
CORREL	Pearson's Correlation Coefficient (r)
DEMA	Double Exponential Moving Average
DX	Directional Movement Index
EMA	Exponential Moving Average
HT_DCPERIOD	Hilbert Transform - Dominant Cycle Period
HT_DCPHASE	Hilbert Transform - Dominant Cycle Phase
HT_PHASOR	Hilbert Transform - Phasor Components
HT_SINE	Hilbert Transform - SineWave
HT_TRENDLINE	Hilbert Transform - Instantaneous Trendline
HT_TRENDMODE	Hilbert Transform - Trend vs Cycle Mode
KAMA	Kaufman Adaptive Moving Average
LINEARREG	Linear Regression

(续表)

简称	函数名
LINEARREG_ANGLE	Linear Regression Angle
LINEARREG_INTERCEPT	Linear Regression Intercept
LINEARREG_SLOPE	Linear Regression Slope
MA	All Moving Average
MACD	Moving Average Convergence/Divergence
MACDEXT	MACD with controllable MA type
MACDFIX	Moving Average Convergence/Divergence Fix 12/26
MAMA	MESA Adaptive Moving Average
MAX	Highest value over a specified period
MAXINDEX	Index of highest value over a specified period
MEDPRICE	Median Price
MFI	Money Flow Index
MIDPOINT	MidPoint over period
MIDPRICE	Midpoint Price over period
MIN	Lowest value over a specified period
MININDEX	Index of lowest value over a specified period
MINMAX	Lowest and highest values over a specified period
MINMAXINDEX	Indexes of lowest and highest values over a specified period
MINUS_DI	Minus Directional Indicator
MINUS_DM	Minus Directional Movement
MOM	Momentum
NATR	Normalized Average True Range
OBV	On Balance Volume
PLUS_DI	Plus Directional Indicator
PLUS_DM	Plus Directional Movement
PPO	Percentage Price Oscillator
ROC	Rate of change : $((\text{price}/\text{prevPrice})-1)*100$
ROCP	Rate of change Percentage: $(\text{price}-\text{prevPrice})/\text{prevPrice}$
ROCR	Rate of change ratio: $(\text{price}/\text{prevPrice})$
ROCR100	Rate of change ratio 100 scale: $(\text{price}/\text{prevPrice})*100$
RSI	Relative Strength Index

(续表)

简称	函数名
SAR	Parabolic SAR
SAREXT	Parabolic SAR - Extended
SMA	Simple Moving Average
STDDEV	Standard Deviation
STOCH	Stochastic
STOCHF	Stochastic Fast
STOCHRSI	Stochastic Relative Strength Index
SUM	Summation
T3	Triple Exponential Moving Average (T3)
TEMA	Triple Exponential Moving Average
TRANGE	True Range
TRIMA	Triangular Moving Average
TRIX	1-day Rate-Of-Change (ROC) of a Triple Smooth EMA
TSF	Time Series Forecast
TYPPRICE	Typical Price
ULTOSC	Ultimate Oscillator
VAR	Variance
WCLPRICE	Weighted Close Price
WILLR	Williams' %R
WMA	Weighted Moving Average

表5-1中提供了Talib相关函数，接下来会对部分内容进行介绍。

5.2.1 Talib常用函数介绍

1. 平均趋向指数ADX和累积派发线AD

AD（累积派发线）是一种平衡交易量指标，以当日的收盘价位来估算成交量，用于估算一段时间内该证券累积的资金流量。它通常与ADX一起使用，利用多空趋向的变化差异与总和判定平均趋势，ADX

数值不能显示趋势的发展方向。但是如果趋势存在，ADX可以衡量趋势的强度。

ADX与AD的函数如下：

```
def ADX(high, low, close, *args, **kwargs):  
def AD(high, low, close, volume):
```

可以看到，ADX函数在常规的high、low、close参数后面还有更多的参数可以输入，这和5.1节的MACD类似，可以对不同的时间函数进行设定。这里默认的timeperiod为14。

AD为资金趋势函数，向上的A/D表明买方占优势，而向下的A/D表明卖方占优势。AD与价格的背离可视为买卖信号，即底背离考虑买入，顶背离考虑卖出。

【程序5-7】

```

from gm.api import *
import talib
import numpy as np
import matplotlib.pyplot as plt

set_token("9e02621e7fcf850731def2e8a9de1fdbd5b21ad6")

data = history_n(symbol="SZSE.002310", frequency="1d", count=100, end_time="2017-12-31", fields="high,low,close,open,volume", fill_missing="last", adjust=ADJUST_PREV, df=True)

close = np.asarray(data["close"].values)
open = np.asarray(data["open"].values)
high = np.asarray(data["high"].values)
low = np.asarray(data["low"].values)
volume = np.asarray(data["volume"].values).astype(np.double)

adx = talib.ADX(high, low, close)
adx = np.nan_to_num(adx)
adx = adx/np.max(adx)

ad = talib.AD(high, low, close, volume)
ad = np.nan_to_num(ad)
ad = ad/np.max(ad) + 20
plt.plot(ad, "r", adx, "b")
plt.show()

```

【程序5-7】中，首先获取了对应股票的数据集，之后使用NumPy库包将其提取，对提取后的数据进行命名，输入到ADX与AD函数中进行计算。最终显示结果如图5-6所示。

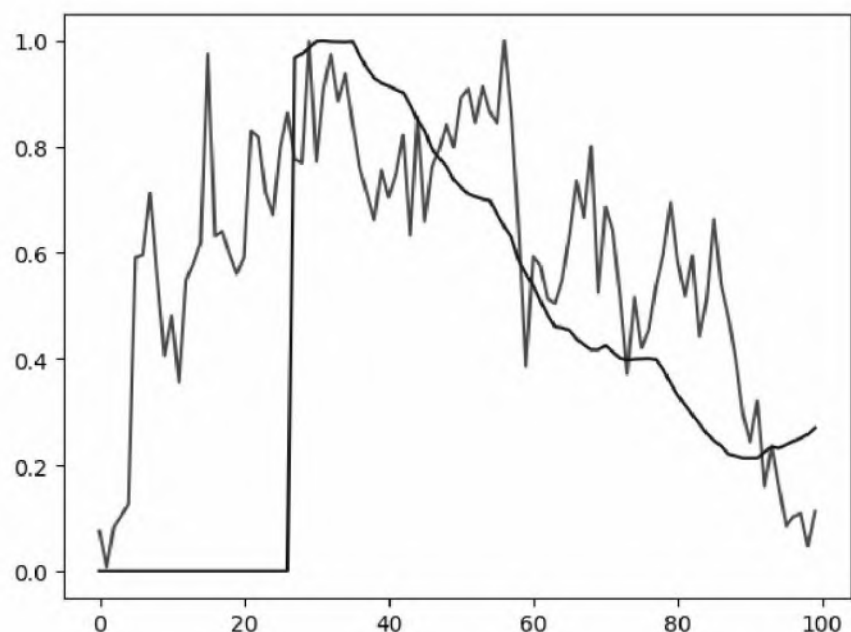


图5-6 ADX与AD曲线结果

由于周期的选择 and 对应标的不同，图像的展示效果也不同，请读者自行演示。

2. ADOSC (Chaikin A/D Oscillator Chaikin) 震荡指标

ADOSC将资金流动情况与价格行为相对比，检测市场中资金流入和流出的情况。其函数如下：

```
def ADOSC(high, low, close, volume, *args, **kwargs):
```

3. ATR (Average True Range, 平均真实波幅)

ATR主要用来衡量价格的波动。因此，这一技术指标并不能直接反映价格走向及其趋势的稳定性，而只是表明价格波动的程度。

```
def ADOSC(high, low, close, volume, *args, **kwargs):
```

4. OBV (On Balance Volume, 能量潮)

OBV通过统计成交量变动的趋势推测股价趋势。它以某日为基期，逐日累计每日上市股票总成交量，若隔日指数或股票上涨，则基期OBV加上本日成交量为本日OBV。若隔日指数或股票下跌，则基期OBV减去本日成交量为本日OBV。函数如下：

```
def OBV(real, volume):
```

以上只是对部分常用的函数进行说明，其他相关函数请读者自行演示。

5.2.2 Talib图像形态识别

除了常用的函数之外，Talib还提供了蜡烛图（K线图）的图像识别方法。

1. 函数名：CDL2CROWS

名称：Two Crows（两只乌鸦）

简介：三日K线模式，第一天长阳，第二天高开收阴，第三天再次高开继续收阴，收盘比前一日收盘价低，预示股价下跌。

函数：def CDL2CROWS(open, high, low, close)

2. 函数名：CDL3BLACKCROWS

名称：Three Black Crows（三只乌鸦）

简介：三日K线模式，连续三根阴线，每日收盘价都下跌且接近最低价，每日开盘价都在上根K线实体内，预示股价下跌。

函数：def CD3BLACKCROWS(open, high, low, close)

3. 函数名：CDL3INSIDE

名称: Three Inside Up/Down (三内部上涨和下跌)

简介: 三日K线模式, 母子信号+长K线, 以三内部上涨为例, K线为阴阳阳, 第三天收盘价高于第一天开盘价, 第二天K线在第一天K线内部, 预示着股价上涨。

函数: `def CDL3INSIDE(open, high, low, close)`

4. 函数名: CDL3LINESTRIKE

名称: Three-Line Strike (三线打击)

简介: 四日K线模式, 前三根阳线, 每日收盘价都比前一日高, 开盘价在前一日实体内, 第四日市场高开, 收盘价低于第一日开盘价, 预示股价下跌。

函数: `def CDL3LINESTRIKE(open, high, low, close)`

5. 函数名: CDL3OUTSIDE

名称: Three Outside Up/Down (三外部上涨和下跌)

简介: 三日K线模式, 与三内部上涨和下跌类似, K线为阴阳阳, 但第一日与第二日的K线形态相反, 以三外部上涨为例, 第一日K线在第二日K线内部, 预示着股价上涨。

函数: `def CDL3OUTSIDE(open, high, low, close)`

6. 函数名: CDL3STARSINSOUTH

名称: Three Stars In The South (南方三星)

简介: 三日K线模式, 与大敌当前相反, 三日K线皆阴, 第一日有长下影线, 第二日与第一日类似, K线整体小于第一日, 第三日无下影

线实体信号，成交价格都在第一日振幅之内，预示下跌趋势反转，股价上升。

函数：def CDL3STARSINSOUTH(open, high, low, close)

7. 函数名：CDL3WHITESOLDIERS

名称：Three Advancing White Soldiers（三个白兵）

简介：三日K线模式，三日K线皆阳，每日收盘价变高且接近最高价，开盘价在前一日实体上半部，预示股价上升。

函数：def CDL3WHITESOLDIERS(open, high, low, close)

8. 函数名：CDLABANDONEDBABY

名称：Abandoned Baby（弃婴）

简介：三日K线模式，第二日价格跳空且收十字星（开盘价与收盘价接近，最高价与最低价相差不大），预示趋势反转，发生在顶部下跌，底部上涨。

函数：def CDLABANDONEDBABY(open, high, low, close, penetration=0)

9. 函数名：CDLADVANCEBLOCK

名称：Advance Block（大敌当前）

简介：三日K线模式，三日都收阳，每日收盘价都比前一日高，开盘价都在前一日实体以内，实体变短，上影线变长。

函数：def CDLADVANCEBLOCK(open, high, low, close)

10. 函数名：CDLBELTHOLD

名称: Belt-Hold (捉腰带线)

简介: 两日K线模式, 下跌趋势中, 第一日阴线, 第二日开盘价为最低价, 阳线, 收盘价接近最高价, 预示价格上涨。

函数: `def CDLBELTHOLD(open, high, low, close)`

11. 函数名: CDLBREAKAWAY

名称: Breakaway (脱离)

简介: 五日K线模式, 以看涨脱离为例, 下跌趋势中, 第一日长阴线, 第二日跳空阴线, 延续趋势开始震荡, 第五日长阳线, 收盘价在第一天收盘价与第二天开盘价之间, 预示价格上涨。

函数: `def CDLBREAKAWAY(open, high, low, close)`

12. 函数名: CDLCLOSINGMARUBOZU

名称: Closing Marubozu (收盘缺影线)

简介: 一日K线模式, 以阳线为例, 最低价低于开盘价, 收盘价等于最高价, 预示着趋势持续。

函数: `def CDLCLOSINGMARUBOZU(open, high, low, close)`

13. 函数名: CDLCONCEALBABYSWALL

名称: Concealing Baby Swallow (藏婴吞没)

简介: 四日K线模式, 下跌趋势中, 前两日阴线无影线, 第三日开盘、收盘价皆低于第二日, 且第三日呈现倒锤头, 第四日开盘价高于前一日最高价, 收盘价低于前一日最低价, 预示着底部反转。

函数: `def CDLCONCEALBABYSWALL(open, high, low, close)`

14. 函数名：CDLCOUNTERATTACK

名称：Counterattack（反击线）

简介：二日K线模式，与分离线类似。

函数：def CDLCOUNTERATTACK(open, high, low, close)

15. 函数名：CDLDARKCLOUDCOVER

名称：Dark Cloud Cover（乌云压顶）

简介：二日K线模式，第一日为长阳，第二日开盘价高于前一日最高价，收盘价处于前一日实体中部以下，预示着股价下跌。

函数：def CDLDARKCLOUDCOVER(open, high, low, close, penetration=0)

16. 函数名：CDLDOJI

名称：Doji（十字）

简介：一日K线模式，开盘价与收盘价基本相同。

函数：def CDLDOJI(open, high, low, close)

17. 函数名：CDLDOJISTAR

名称：Doji Star（十字星）

简介：一日K线模式，开盘价与收盘价基本相同，上下影线不会很长，预示着当前趋势反转。

函数：def CDLDOJISTAR(open, high, low, close)

18. 函数名：CDLDAGONFLYDOJI

名称：Dragonfly Doji（蜻蜓十字/T形十字）

简介：一日K线模式，开盘后价格一路走低，之后收复，收盘价与开盘价相同，预示趋势反转。

函数：`def CDLDRAGONFLYDOJI(open, high, low, close)`

19. 函数名：CDLENGULFING

名称：Engulfing Pattern（吞噬模式）

简介：两日K线模式，分多头吞噬和空头吞噬，以多头吞噬为例，第一日为阴线，第二日为阳线，第一日的开盘价和收盘价在第二日开盘价和收盘价之内，但不能完全相同。

函数：`def CDLENGULFING(open, high, low, close)`

20. 函数名：CDLEVENINGDOJISTAR

名称：Evening Doji Star（十字暮星）

简介：三日K线模式，基本模式为暮星，第二日收盘价和开盘价相同，预示顶部反转。

函数：`def CDLEVENINGDOJISTAR(open, high, low, close, penetration=0)`

21. 函数名：CDLEVENINGSTAR

名称：Evening Star（暮星）

简介：三日K线模式，与晨星相反，上升趋势中，第一日为阳线，第二日价格振幅较小，第三日为阴线，预示顶部反转。

函数：`def CDLEVENINGSTAR(open, high, low, close, penetration=0)`

22. 函数名：CDLGAPSIDESIDEWHITE

名称：Up/Down-gap side-by-side white lines（向上 / 下跳空并列阳线）

简介：二日K线模式，上升趋势向上跳空，下跌趋势向下跳空，第一日与第二日有相同的开盘价，实体长度差不多，则趋势持续。

函数：def CDLGAPSIDESIDEWHITE(open, high, low, close)

23. 函数名：CDLGRAVESTONEDOJI

名称：Gravestone Doji（墓碑十字 / 倒T十字）

简介：一日K线模式，开盘价与收盘价相同，上影线长，无下影线，预示底部反转。

函数：def CDLGRAVESTONEDOJI(open, high, low, close)

24. 函数名：CDLHAMMER

名称：Hammer（锤头）

简介：一日K线模式，实体较短，无上影线，下影线大于实体长度两倍，处于下跌趋势底部，预示反转。

函数：def CDLHAMMER(open, high, low, close)

25. 函数名：CDLHANGINGMAN

名称：Hanging Man（上吊线）

简介：一日K线模式，形状与锤子类似，处于上升趋势的顶部，预示着趋势反转。

函数：def CDLHANGINGMAN(open, high, low, close)

26. 函数名：CDLHARAMI

名称: Harami Pattern (母子线)

简介: 二日K线模式, 分多头母子与空头母子, 两者相反, 以多头母子为例, 在下跌趋势中, 第一日K线长阴, 第二日开盘价和收盘价在第一日价格振幅之内, 为阳线, 预示趋势反转, 股价上升。

函数: `def CDLHARAMI(open, high, low, close)`

27. 函数名: CDLHARAMICROSS

名称: Harami Cross Pattern (十字孕线)

简介: 二日K线模式, 与母子线类似, 若第二日K线是十字线, 便称为十字孕线, 预示着趋势反转。

函数: `def CDLHARAMICROSS(open, high, low, close)`

28. 函数名: CDLHIGHWAVE

名称: High-Wave Candle (风高浪大线)

简介: 三日K线模式, 具有极长的上 / 下影线与短的实体, 预示着趋势反转。

函数: `def CDLHIGHWAVE(open, high, low, close)`

29. 函数名: CDLHIKKAKE

名称: Hikkake Pattern (陷阱)

简介: 三日K线模式, 与母子线类似, 第二日价格在前一日实体范围内, 第三日收盘价高于前两日, 反转失败, 趋势继续。

函数: `def CDLHIKKAKE(open, high, low, close)`

30. 函数名: CDLHIKKAKEMOD

名称: Modified Hikkake Pattern (修正陷阱)

简介: 三日K线模式, 与陷阱类似, 上升趋势中, 第三日跳空高开; 下跌趋势中, 第三日跳空低开, 反转失败, 趋势继续。

函数: `def CDLHIKKAKEMOD(open, high, low, close)`

31. 函数名: CDLHOMINGPIGEON

名称: Homing Pigeon (家鸽)

简介: 二日K线模式, 与母子线类似, 不同的是二日K线颜色相同, 第二日最高价、最低价都在第一日实体之内, 预示着趋势反转。

函数: `def CDLHOMINGPIGEON(open, high, low, close)`

32. 函数名: CDLIDENTICAL3CROWS

名称: Identical Three Crows (三胞胎乌鸦)

简介: 三日K线模式, 上涨趋势中, 三日都为阴线, 长度大致相等, 每日开盘价等于前一日收盘价, 收盘价接近当日最低价, 预示价格下跌。

函数: `def CDLIDENTICAL3CROWS(open, high, low, close)`

33. 函数名: CDLINNECK

名称: In-Neck Pattern (颈内线)

简介: 二日K线模式, 下跌趋势中, 第一日为长阴线, 第二日开盘价较低, 收盘价略高于第一日收盘价, 阳线, 实体较短, 预示着下跌继续。

函数: `def CDLINNECK(open, high, low, close)`

34. 函数名：CDLINVERTEDHAMMER

名称：Inverted Hammer（倒锤头）

简介：一日K线模式，上影线较长，长度为实体的2倍以上，无下影线，在下跌趋势底部，预示着趋势反转。

函数：def CDLINVERTEDHAMMER(open, high, low, close)

35. 函数名：CDLKICKING

名称：Kicking（反冲形态）

简介：二日K线模式，与分离线类似，两日K线为秃线，颜色相反，存在跳空缺口。

函数：def CDLKICKING(open, high, low, close)

36. 函数名：CDLKICKINGBYLENGTH

名称：Kicking - bull/bear determined by the longer marubozu（由较长缺影线决定的反冲形态）

简介：二日K线模式，与反冲形态类似，较长缺影线决定价格的涨跌。

函数：def CDLKICKINGBYLENGTH(open, high, low, close)

37. 函数名：CDLSTALLEDPATTERN

名称：Stalled Pattern（停顿形态）

简介：三日K线模式，上涨趋势中，第二日为长阳线，第三日开盘于前一日收盘价附近，短阳线，预示着上涨结束。

函数：def CDLSTALLEDPATTERN(open, high, low, close)

38. 函数名: CDLSTICKSANDWICH

名称: Stick Sandwich (条形三明治)

简介: 三日K线模式, 第一日为长阴线, 第二日为阳线, 开盘价高于前一日收盘价, 第三日开盘价高于前两日最高价, 收盘价与第一日收盘价相同。

函数: `def CDLSTICKSANDWICH(open, high, low, close)`

39. 函数名: CDLTAKURI

名称: Takuri (Dragonfly Doji with very long lower shadow, 探水竿)

简介: 一日K线模式, 大致与蜻蜓十字相同, 下影线长度长。

函数: `def CDLTAKURI(open, high, low, close)`

40. 函数名: CDLTASUKIGAP

名称: Tasuki Gap (跳空并列阴阳线)

简介: 三日K线模式, 分上涨和下跌, 以上升为例, 前两日为阳线, 第二日跳空, 第三日为阴线, 收盘价于缺口中, 上升趋势持续。

函数: `def CDLTASUKIGAP(open, high, low, close)`

41. 函数名: CDLTHRUSTING

名称: Thrusting Pattern (插入)

简介: 二日K线模式, 与颈上线类似, 下跌趋势中, 第一日为长阴线, 第二日开盘价跳空, 收盘价略低于前一日实体中部, 与颈上线相比实体较长, 预示着趋势持续。

函数: `def CDLTHRUSTING(open, high, low, close)`

42. 函数名：CDLTRISTAR

名称：Tristar Pattern（三星）

简介：三日K线模式，由三个十字组成，第二日的十字必须高于或者低于第一日和第三日，预示着反转。

函数：`def CDLTRISTAR(open, high, low, close)`

43. 函数名：CDLUNIQUE3RIVER

名称：Unique 3 River（奇特三河床）

简介：三日K线模式，下跌趋势中，第一日为长阴线，第二日为锤头，最低价创新低，第三日开盘价低于第二日收盘价，收阳线，收盘价不高于第二日收盘价，预示着反转，第二日下影线越长可能性越大。

函数：`def CDLUNIQUE3RIVER(open, high, low, close)`

44. 函数名：CDLUPSIDEGAP2CROWS

名称：Upside Gap Two Crows（向上跳空的两只乌鸦）

简介：三日K线模式，第一日为阳线，第二日跳空以高于第一日最高价开盘，收阴线，第三日开盘价高于第二日，收阴线，与第一日比仍有缺口。

函数：`def CDLUPSIDEGAP2CROWS(open, high, low, close)`

45. 函数名：CDLXSIDEGAP3METHODS

名称：Upside/Downside Gap Three Methods（上升 / 下降跳空三法）

简介：五日K线模式，以上升跳空三法为例，上涨趋势中，第一日为长阳线，第二日为短阳线，第三日为跳空阳线，第四日为阴线，开盘价与收盘价于前两日实体内，第五日为长阳线，收盘价高于第一日收盘价，预示股价上升。

函数：def CDLXSIDEGAP3METHODS(open, high, low, close)

下面对函数做一个演示。

【程序5-8】

```
from gm.api import *
import talib
import numpy as np
import matplotlib.pyplot as plt

set_token("9e02621e7fcf850731def2e8a9de1fdbd5b21ad6")

data = history_n(symbol="SZSE.399006", frequency="1d", count=100, end_time="2018-12-31", fields="high,low,close,open,volume", fill_missing="last", adjust=ADJUST_PREV, df=True)

close = np.asarray(data["close"].values)
open = np.asarray(data["open"].values)
high = np.asarray(data["high"].values)
low = np.asarray(data["low"].values)
volume = np.asarray(data["volume"].values).astype(np.double)

cdl2crows = talib.CDL2CROWS(high, low, close, volume)
print(cdl2crows)
```

【程序5-8】展示了对标的进行2只乌鸦图形的查找。当确定结果后，会输出100，而当不符合函数确定的样式图形时，会以0输出显示。

更多样式的识别还请读者参考Talib官方文档。

5.3 实战：Talib金融工具回测实战

Talib的用途在金融项目中准确地把握行情，认清所处的趋势位置。通过结合掘金量化库包可以直接对所产生的策略进行回测检验，从而找出策略方案与回测的真实性与差异性如何。

5.3.1 根据MACD变化回测2017年盈利情况

MACD是股市中最常用的指标，一般认为当dif值由负变为正时为购买金融产品的时机，而dif值由正变为负时为卖出时机。通过Talib的MACD结合量化掘金类库可以达到择时对股票进行购买并计算收益的目的。

在前面介绍掘金量化库包的时候，通过设定不同的回测主程序可以对选定的内容进行回测。

1. 进行程序的初始化

```
def init(context):  
    context.symbol = "SZSE.300296"  
    context.frequency = "1d"  
    context.fields = "open,high,low,close"  
    context.volume = 200  
    schedule(schedule_func=algo,date_rule="1d",time_rule="09:35:00")
```

init函数是回测程序的初始化程序，在这里进行数据的初始化，从本代码段中可以看到，这里通过context初始化了目标代码、时间周期以及所需要读取的字典。最后通过schedule函数对回测的执行规则做了设定，选择了回测算法以及时间规则。

再提醒一点，在初始化过程中，可以添加一些额外的程序进行初始化工作。例如，将机器学习中的回归分析、深度学习中的模型检测

填入init函数中进行预处理。

2. 主算法的设定

回测策略的关键是算法的设定。根据不同的金融产品对象设置不同的分析算法从而决定买卖点。

本算法中，MACD买卖点的核心就是获取前一日的MACD值，当其大于0时候进行金融产品的买入，而当MACD小于0时进行卖出。

想要计算MACD值，首先就要获取给定时间段的值，在这里使用量化掘金库包中的history_n函数获取前一段时间count周期内的数据。

```
now = context.now
last_day = get_previous_trading_date("SZSE",now)
data =
history_n(symbol=context.symbol,frequency=context.frequency,count=35,end_time=
now,
fields=context.fields,fill_missing="last",adjust=ADJUST_PREV,df=True)
```

代码段中，首先通过context.now函数获取回测阶段当天的数值，通过get_previous_trading_date函数获取相同市场上一个交易日的日期。

history_n函数获取截至当前日期上一个交易日的的所有周期数据，并以pandas格式返回。

```
open = np.asarray((data["open"].values))
high = np.asarray((data["high"].values))
low = np.asarray((data["low"].values))
close = np.asarray((data["close"].values))
```

获取的数据需要通过NumPy库包进行数据处理，这里通过调用关键字的方法获取各个值的序列，为下一步的计算做准备。

MACD值的计算就显得较为简单，直接调用Talib库包中的函数值计算即可。

```
macd, _, _ = talib.MACD(close)
macd = macd[-1]
```

注意

这里的MACD返回的是3个数值序列，根据需要只要第一个即可。而macd[-1]指的是获取当前序列最后一个数值，即根据上一个交易日获取的MACD值进行计算。

判断MACD值的情况，调用掘金量化库包进行买卖，其方法如下：

```
if macd > 0:
    order_volume(symbol=context.symbol, volume=context.volume,
                  side=PositionSide_Long,
                  order_type=OrderType_Market, position_effect=PositionEffect_Open)
    print("买入")
elif macd < 0:
    order_volume(symbol=context.symbol, volume=context.volume,
                  side=PositionSide_Short,
                  order_type=OrderType_Market, position_effect=PositionEffect_Close)
    print("卖出")
```

3. 主方法的设定

主方法中主要对账户信息、Python代码的名称、执行的类型、起止时间、复权方法进行设定。代码段如下：

```
if __name__ == "__main__":
    run(
        strategy_id='*****',
        filename='MACD 不同时间.py',
        mode=MODE_BACKTEST,
        token='9e02621e7fcf850731def2e8a9de1fdbd5b21ad6',
        backtest_start_time='2017-01-01 09:00:00',
        backtest_end_time='2017-12-31 15:00:00',
        backtest_initial_cash=20000,
        backtest_adjust=ADJUST_PREV
    )
```

在主方法中还对起始金额做了设定，这里为了简便，设置起始金额为2万元。

【程序5-9】

```

from gm.api import *
import numpy as np
import talib

def init(context):

    context.symbol = "SZSE.300296"
    context.frequency = "1d"
    context.fields = "open,high,low,close"
    context.volume = 200
schedule(schedule_func=algo,date_rule="1d",time_rule="09:35:00")

def algo(context):

    now = context.now
    last_day = get_previous_trading_date("SZSE",now)
    data = history_n(symbol=context.symbol,frequency=context.frequency,count=35,
end_time=now,fields=context.fields,fill_missing="last",adjust=ADJUST_PREV,df=True)

    open = np.asarray((data["open"].values))
    high = np.asarray((data["high"].values))
    low = np.asarray((data["low"].values))
    close = np.asarray((data["close"].values))
    macd,_,_ = talib.MACD(close)
    macd = macd[-1]

    if macd > 0:
        order_volume(symbol=context.symbol, volume=context.volume,
side=PositionSide_Long,
                        order_type=OrderType_Market,
position_effect=PositionEffect_Open)
        print("买入")
    elif macd < 0:
        print("卖出")
        order_volume(symbol=context.symbol, volume=context.volume,
side=PositionSide_Short,
                        order_type=OrderType_Market,
position_effect=PositionEffect_Close)

if __name__ == "__main__":
    run(
        strategy_id='*****',
        filename='MACD 不同时间.py',
        mode=MODE_BACKTEST,
        token='9e02621e7fcf850731def2e8a9de1fdbd5b21ad6',
        backtest_start_time='2017-01-01 09:00:00',
        backtest_end_time='2017-12-31 15:00:00',
        backtest_initial_cash=20000,
        backtest_adjust=ADJUST_PREV
    )

```


【程序5-9】完整地展示了策略的例子，结合掘金量化客户端可以完整地看到结果，如图5-7所示。



图5-7 回测结果

从总览上可以看到，使用此MACD简易策略，完整的年化收入为15.05%，而最大回测约为14%，夏普比率为0.65。

这样看来，这个策略可以使用，但是运营的收益率不太理想。细心的读者可能还注意到，这里不包括设置手续费和滑点率这些更细节的东西。

从图5-8对比标的的涨幅可以看到，对于使用MACD进行的买卖策略实际上不能达到既有标的的年化涨幅，这点是很可惜的。



图5-8 MACD买入与股票涨幅对比

4. 更改主算法

既然使用MACD算法取得的回测成绩不满意，那么就需要对既定的策略进行检讨。

回测策略的核心是算法的使用，在回测代码中，使用的是MACD作为买卖判定的指标。除此之外，TaLib还有其他一些计算函数用于帮助回测的检定。

RSI（相对强弱指数）是通过比较一段时期内的平均收盘涨数和平均收盘跌数来分析市场买沽盘的意向和实力，从而做出未来市场的走势。RSI是在1978年6月由Wells Wider创制的一种技术指标，它通过特定时期内股价的变动情况计算市场买卖力量对比，来判断股票价格内部本质强弱、推测价格未来的变动方向。

一般真实使用时判定当RSI高于70时，股票可以被视为超买，是卖出的时候。当RSI低于30时，股票可以被视为超卖，是买入的时候。核心代码如下：

```
rsi = talib.RSI(close)
rsi = rsi[-1]
if rsi < 30:
    order_volume(symbol=context.symbol, volume=context.volume,
                 side=PositionSide_Long,
                 order_type=OrderType_Market,
                 position_effect=PositionEffect_Open)
    print("买入")
elif rsi > 70:
    print("卖出")
    order_volume(symbol=context.symbol, volume=context.volume,
                 side=PositionSide_Short,
                 order_type=OrderType_Market,
                 position_effect=PositionEffect_Close)
```

通过TaLib计算RSI值，针对不同的时期进行买卖交易。

【程序5-10】

```
from gm.api import *
import numpy as np
import talib
def init(context):
    context.symbol = "SZSE.300296"
    context.frequency = "1d"
    context.fields = "open,high,low,close"
    context.volume = 200

    schedule(schedule_func=algo, date_rule="1d", time_rule="09:35:00")
def algo(context):
    now = context.now
    last_day = get_previous_trading_date("SZSE", now)
```

```

    data = history_n(symbol=context.symbol, frequency=context.frequency,
count=35,
                    end_time=now, fields=context.fields, fill_missing="last",
adjust=ADJUST_PREV, df=True)

    open = np.asarray((data["open"].values))
    high = np.asarray((data["high"].values))
    low = np.asarray((data["low"].values))
    close = np.asarray((data["close"].values))

    rsi = talib.RSI(close)
    rsi = rsi[-1]

    if rsi < 30:
        order_volume(symbol=context.symbol, volume=context.volume,
side=PositionSide_Long,
                    order_type=OrderType_Market,
position_effect=PositionEffect_Open)
        print("买入")
    elif rsi > 70:
        print("卖出")
        order_volume(symbol=context.symbol, volume=context.volume,
side=PositionSide_Short,
                    order_type=OrderType_Market,
position_effect=PositionEffect_Close)

if __name__ == "__main__":
    run(
        strategy_id='f53dcdca-dc23-11e7-a391-902b3463caf1',
        filename='RSI 买卖测定
mode=MODE_BACKTEST,
        token='9e02621e7fcf850731def2e8a9de1fdbd5b21ad6',
        backtest_start_time='2017-01-01 09:00:00',
        backtest_end_time='2017-12-31 15:00:00',
        backtest_initial_cash=20000,
        backtest_adjust=ADJUST_PREV

    )

```

待全部回测运行完毕后，最终结果如图5-9所示。



图5-9 回测结果

可以看到，通过RSI的使用，回测结果有了很大的提高，收益率由15.05%上升到20.72%，夏普比率也上升到1.36左右，如图5-10所示。



图5-10 RSI买入与股票涨幅对比

而与股票实际涨幅的对比来看，虽然提高了MACD的收益率，但是相对于对应标的的最高点并没有完全达到。因此，这个算法策略的改变并不算是完全成功。至于提高的方法还有很多，除了更改核心算法外，还有修改RSI判定参数、设置买卖的阈值点等方法，感兴趣的读者可以自行完成。

5.3.2 股价的波动范围及未来走势判定

股票未来走势以及波动的判定是股票市场中需要掌握的一个非常重要的内容，布林线指标可以用来判定股票未来走势以及波动。

布林线指标即BOLL指标，其英文全称是Bollinger Bands。布林线由约翰·布林先生创造，其利用统计原理求出股价的标准差以及信赖区间，从而确定股价的波动范围及未来走势，利用波带显示股价的安全高低价位，因而也被称为布林带。

其上下限范围不固定，随股价的滚动而变化。布林指标和麦克（MIKE）指标一样属于路径指标，股价波动在上限和下限的区间之内，这条带状区的宽窄随着股价波动幅度的大小而变化，股价涨跌幅度加大时，带状区变宽，涨跌幅度狭小盘整时，带状区则变窄。

【程序5-11】

```
from gm.api import *
import numpy as np
import talib
import matplotlib.pyplot as plt

set_token("9e02621e7fcf850731def2e8a9delfdbd5b21ad6")

symbol = 'SZSE.002129'
frequency = "3600" + "s"
end_time = "2017-12-31"

fields = "open,high,low,close,volume"
last_day = get_previous_trading_date("SZSE",end_time)

data = history_n(symbol=symbol,frequency=frequency,count=100,end_time=end_time,
fields=fields,fill_missing="last",adjust=ADJUST_PREV,df=True)

open = np.asarray((data["open"].values))
high = np.asarray((data["high"].values))
low = np.asarray((data["low"].values))
close = np.asarray((data["close"].values))
volume = np.asarray((data["volume"].values)).astype(np.double)

upperband,middleband,lowerband = talib.BBANDS(close)

plt.plot(upperband,"r",close,"b--",lowerband,"r")
plt.show()
```

talib.BBANDS函数是Talib中计算布林线的专用函数，其返回值分别为布林线的上中下轨，通过调用matplotlib库包可以很方便地将结果进行图像化展示，如图5-11所示。

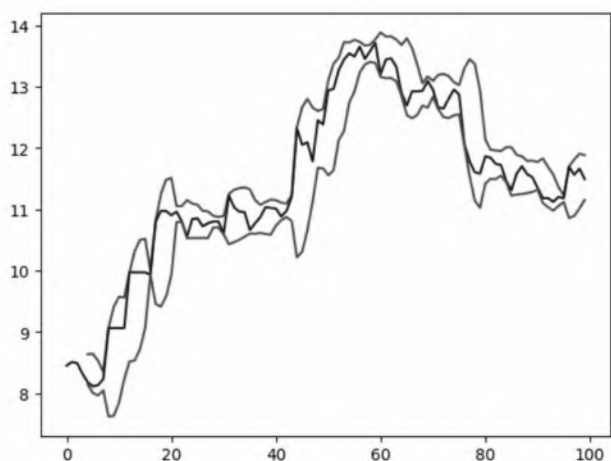


图5-11 通过布林线判断未来走势以及波动

除了基本的画出图形外，布林线还可以用于买卖点的设置，当股价高于这个波动区间时，即突破阻力线，说明股价虚高，执行卖出操作。而股价低于这个波动区间，即跌破支撑线，说明股价虚低，执行买入。

5.4 两种经典的轨道突破策略

根据均线进行价格趋势追踪是一种常用的方法，通过比较不同均线速度的改变程度来判断买入与卖出点，根据其统计特性从而形成一套完整的交易策略。

当然除了5.1节中单纯地计算不同时间的均线外，还有更多的根据统计数据做出的趋势策略。

图5-12是《财富》杂志在2005年评出的最佳交易系统的排名情况，根据特定交易策略在不同的时期表现出的优秀情况，排名也在不断变化。

轨道线是趋势线概念的延伸，当股价沿道趋势上涨到某一价位水准时会遇到阻力，回档至某一水准时价格又获得支撑，轨道线就在接高点的延长线及接低点的延长线之间上下来回，当轨道线确立后，股价就可以非常容易地找出高低价位所在，投资人可依此判断来操作股票。

历年来前十名交易系统的登榜情况			
	1993	1997	2001
1	Black or White	Aberration	Aberration
2	Culler Currency	CatScan 1	Basis II
3	DCS-II	Combo Advantage	DCS-II
4	Dollar Trader	Culler Currency	Dollar Trader
5	Pilot Trader	DCS-II	Dynamic BreakOut
6	QuadLevelTrend	Dollar Trader	Golden SX
7	Time Trend III	GrandCayman	GrandCayman
8	Ultimate II	R-Breaker	R-Breaker
9	Volpat	Time Trend III	STC S&P DayTrade
10	Wilder's Volatility	Universal LT	TrendChannel

图5-12 理念交易策略排行榜

5.4.1 Dual Thrust策略

Dual Thrust简称DT，是Michael Chalek在80年代开发的，是海外TOP10交易系统中的一个，属于开盘区间突破类交易系统，以今日开盘价加 / 减一定比例的昨日振幅确定上下轨，如图5-13所示。日内突破上轨时平空做多，突破下轨时平多做空。



图5-13 Dual Thrust展示

Dual Thrust在形式上和传统的区间突破类似。不同之处主要体现在两方面：

(1) Dual Thrust在Range的设置上，针对不同的标的引入前若干天的4个价位，使得一定时期内的Range相对稳定，可以适用于日间的趋势跟踪。

(2) Dual Thrust对不同的趋势追踪设置了不同的上轨和下轨，这里主要引入两个参数：K1和K2。当K1<K2时，买点相对容易被触发；当K1>K2时，卖点相对容易被触发。

因此，量化交易人员在使用该策略时，一方面可以参考历史数据测试的最优参数，另一方面可以根据自己对后势的判断，或其他大周期的技术指标入手，阶段性地动态调整K1和K2的值。

首先获取数据。这里需要注意的是，由于针对回测当日的开盘价进行测算，因此需要以上一个交易日的历史价格周期进行收集数据。

```
today = context.now
last_day = get_previous_trading_date("SZSE",today)
data=history_n(symbol=context.symbol,frequency=context.frequency,count=context
.count,end_time=today,fields=context.fields,fill_missing="last",adjust=ADJUST_
PREV, df=True)
```

在上面的代码段中，context.symbol在init函数中对标的进行了初始化操作，可以看到，对于每个参数都使用了context开头的参数形式，这样做的好处是可以针对不同的标的在初始化中进行参数的设置。最后以pandas形式将数据返回。

下面是Dual Thrust的核心部分：

```
high = np.asarray((data["high"].values))
low = np.asarray((data["low"].values))
close = np.asarray((data["close"].values))

hh = np.max(high)
hc = np.max(close)

lc = np.min(close)
ll = np.min(low)
```

具体算法：

（1）根据昨日的收盘计算两个值：最高价－收盘价和收盘价－最低价。之后取这两个值较大的那个，乘以k值0.7。把结果称为触发值。

（2）在当日的价格回测时间记录开盘价，然后在价格低于“开盘－触发值”时马上卖空，或在价格超过“开盘价＋触发值”时马上买入。

```

range = np.max([hh-hc,lc-ll])

data_now = current(symbols=context.symbol)[0]
data_now_open = data_now["open"]
data_now_price = data_now["price"]

range_up_price = data_now_open + context.k1 * range
if (data_now_price > range_up_price):
    order_target_percent(symbol=context.symbol, percent=0.75,
position_side=PositionSide_Long, order_type=OrderType_Market, price=0)
    context.flag = 0
    if (context.flag == context.flag_check):
        order_target_percent(symbol=context.symbol, percent=0,
position_side=PositionSide_Long, order_type=OrderType_Market, price=0)
        context.flag = 100
    context.flag += 1

```

上面的代码段中，range根据Dual Thrust算法设置了区间，之后的range_up_price对当日的区间价格进行了相对计算，设置了阈值价格。根据判断条件，当开盘价大于阈值价格时，执行买入操作；当开盘价小于阈值价格时，执行卖出操作。

【程序5-12】

```

from gm.api import *
import numpy as np
import talib

def init(context):
    context.symbol = "SHSE.600000" # sys.argv[1]
    context.frequency = "1d"
    context.fields = "open,high,low,close,volume"
    context.count = 25

    context.k1 = 0.7

    schedule(schedule_func=algo,date_rule="1d",time_rule="09:30:00")
    context.flag = 100
    context.flag_check = 2 # int(sys.argv[5])

def algo(context):

    today = context.now
    last_day = get_previous_trading_date("SZSE",today)

    data = history_n(symbol=context.symbol, frequency=context.frequency,
count=context.count,
                    end_time=last_day, fields=context.fields,
                    fill_missing="last", adjust=ADJUST_PREV, df=True)
    high = np.asarray((data["high"].values))
    low = np.asarray((data["low"].values))
    close = np.asarray((data["close"].values))

    hh = np.max(high)
    hc = np.max(close)

    lc = np.min(close)
    ll = np.min(low)

    range = np.max([hh-hc,lc-ll])
    data_now = current(symbols=context.symbol)[0]
    data_now_open = data_now["open"]
    data_now_price = data_now["price"]

    range_up_price = data_now_open + context.k1 * range
    if (data_now_price > range_up_price):
        order_target_percent(symbol=context.symbol, percent=0.75,
position_side=PositionSide_Long, order_type=OrderType_Market, price=0)
        context.flag = 0
    if (data_now_price < range_up_price):
        order_target_percent(symbol=context.symbol, percent=0,
position_side=PositionSide_Long,order_type=OrderType_Market, price=0)

if __name__ == "__main__":
    run(
        strategy_id='c08eed5a-00c2-11e8-9308-902b3463caf1',
        filename='Dual_Thrust.py',
        mode=MODE_BACKTEST,
        token="*****",
        backtest_start_time='2017-01-01 09:00:00',
        backtest_end_time='2017-12-31 15:00:00',
        backtest_initial_cash=20000,
        backtest_adjust=ADJUST_PREV
    )

```

最终回测的结果如图5-14所示。

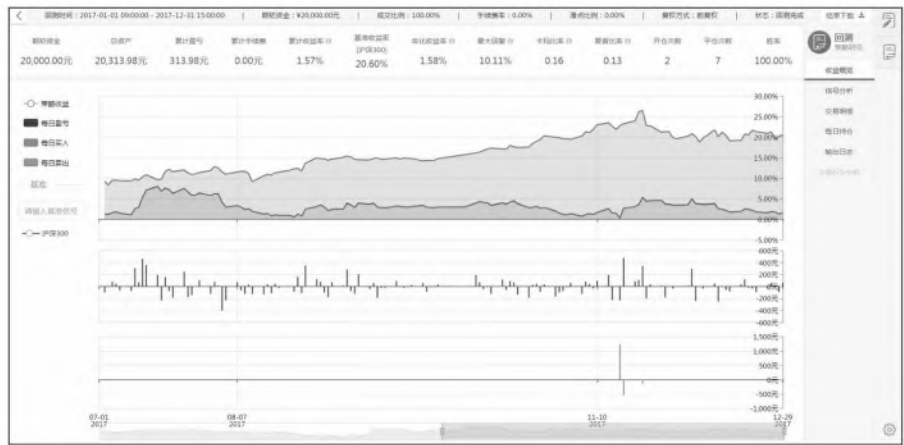


图5-14 Dual Thrust回测结果

从回测数据图上可以看到，Dual Thrust的回测结果并不好，从对开仓点更细节的分析来看（见图5-15），主要是其开仓的次数较少，而卖出点的次数过多，因此造成在判定收益时过低而回测力度比较大。



图5-15 Dual Thrust开仓点分析

改进的方法很多，例如加入一些简单的交易规则（如初始止损、跨周期的数据引用等）进行完善。具体来讲，每次以30%仓位开仓，日内突破上轨且30分钟周期的MA5>MA10就买入股票，而当日跌破下轨且30分钟周期的MA5<MA10就卖出股票。

5.4.2 Dynamic Breakout II策略

Dynamic Breakout II策略是一种动态自适应策略。所谓动态自适应策略，指的是这种系统的参数会依据市场当前的状况而自行调整。

传统的轨道线采用固定比率的价格差值来计算轨道的上下阈值，例如Talib中的BBANDS函数，通过固定采样区间标准差的计算来确定轨道的上下阈值。这样便于计算，可以获得一个稳定的计算公式。但是采样区间的选取是比较困难的，如果采样区间太短，那么在趋势比较明显的情况下会表现优秀，信号对于入场和出场的展示比较紧密。但是采样区间太短会因为信号过于敏感而使其出现得较为频繁。

在趋势不明显的市场中，采样区间设置较短会使得信号出现过多。一个较好的解决方法是把采样区间设置得更长一些，使其信号出现得不那么频繁。但是对参数进行调整并不是那么简单的一件事，最好的方法是依据当前市场本身的状况自行调整参数的值，而不是使用固定参数值的方式来进行设置。

Dynamic Breakout II策略就是采用动态自适应的方法，根据市场的波动作为衡量的标注。在震荡幅度较大的市场中，波动率自然而然较大，需要增加采样区间的值，这样不容易反复出现买卖点信号，而在趋势稳定的市场中，市场有较为明显的走势，波动率会随之变小，因此要减少采样区间的值，使得信号的出现较为敏感。

Dynamic Breakout II策略最重要的是波动率的计算。这个波动率的计算方式如下：

- (1) 计算当日的标准差。
- (2) 计算上一个交易日的标准差。
- (3) 根据公式

$$\text{deltaVolatility} = \frac{(\text{todayVolatility} - \text{yesterDayVolatility})}{\text{todayVolatility}}$$

每日比较市场波动度的增减，如果市场波动度变大10%，就把采样区间的参数值增加10%。而如果市场波动度减少10%，就把采样区间的参数值减少10%。

代码如下：

```
todayVolatility = np.std(close)
yesterDayVolatility = np.std(close[:-1])
deltaVolatility = (todayVolatility - yesterDayVolatility) / todayVolatility

lookBackDays = np.round(lookBackDays * (1 + deltaVolatility))
lookBackDays = np.min([lookBackDays, context.ceilingAmt])
lookBackDays = np.max([lookBackDays, context.floorAmt])
```

其中的todayVolatility和yesterDayVolatility是当日和上一个交易日的标准差，deltaVolatility用于计算当前的波动率，lookBackDays则用于计算采样区间，根据不同波动率对其进行增减。

context.ceilingAmt和context.floorAmt是人为设定的采样区间上下限，根据公开的数据和已有回测，一般高值设置为60，而低值设置为20较为合适。

至于买卖点的判定，当上一日的收盘价在区间上轨时买入，而收盘价低于区间下轨时则卖出。

【程序5-13】

```
from gm.api import *
import numpy as np
import talib

def init(context):

    context.symbol = "SZSE.300296"
    context.frequency = "1d"
    context.fields = "open,high,low,close,volume"
    context.ceilingAmt = 60
    context.floorAmt = 20
    context.bolBandTrig = 1

    schedule(schedule_func=algo,date_rule="1d",time_rule="09:30:00")

lookBackDays = 30
```



```

def algo(context):

    global lookBackDays

    today = context.now
    last_day = get_previous_trading_date("SZSE",today)

    data = history_n(symbol=context.symbol, frequency = context.frequency,
count=int(lookBackDays),
                        end_time=last_day, fields=context.fields,
                        fill_missing="last", adjust=ADJUST_PREV, df=True)

    close = np.asarray((data["close"].values))
    high = np.asarray((data["high"].values))
    low = np.asarray((data["low"].values))

    todayVolatility = np.std(close)
    yesterdayVolatility = np.std(close[:-1])
    deltaVolatility = (todayVolatility - yesterdayVolatility) / todayVolatility

    lookBackDays = np.round(lookBackDays * (1 + deltaVolatility))
    lookBackDays = np.min([lookBackDays,context.ceilingAmt])
    lookBackDays = np.max([lookBackDays,context.floorAmt])

    MidLine = np.average(close)
    Band = np.std(close)

    upBand = MidLine + context.bolBandTrig * Band
    lowBand = MidLine - context.bolBandTrig * Band

    buyPoint = np.max(high)
    sellPoint = np.min(low)
    current_data = current(symbols= context.symbol)
    current_price = current_data[0]["price"]

    if ((close[-1] > upBand) and (current_price > buyPoint)):
        order_percent(symbol=context.symbol, percent=1,
side=OrderSide_Buy,position_effect = PositionEffect_Open ,
order_type=OrderType_Market, price=0)

    if ((close[-1] < lowBand) and (current_price < sellPoint)):
        order_percent(symbol=context.symbol, percent=0,
side=OrderSide_Sell,position_effect = PositionEffect_Open ,
order_type=OrderType_Market, price=0)

if __name__ == "__main__":

```

```
run(  
    strategy_id='c08eed5a-00c2-11e8-9308-902b3463caf1',  
    filename='Dynamic_Breakout_II.py',  
    mode=MODE_BACKTEST,  
    token='*****',  
    backtest_start_time='2017-01-01 09:00:00',  
    backtest_end_time='2017-12-31 15:00:00',  
    backtest_initial_cash=20000,  
    backtest_adjust=ADJUST_PREV  
)
```

具体回测请读者自行完成。

5.5 小结

本章介绍了Talib的基本使用方法，Talib是金融分析中最常用的技术分析指标工具，通过图表或技术指标的记录研究市场过去及现在的行为反应，以推测未来价格的变动趋势。

但是从更深一个层次来说，仅仅依靠技术分析指标对走势做出预测并不可行，因为决定一个公司股票价格的最终因素是其内在的价值，而技术分析只关心证券市场本身的变化，不考虑会对其产生某种影响的经济方面、政治方面等各种外部的因素。

后续的介绍将以基本面分析为主，这也是本书的重点。

第6章

多因子策略

技术分析与基本面分析是股票价格分析最基础也是最经典的两个部分。

技术分析是针对股票的特定走势与价格趋势做出分析策略，主要根据某个特定时间段的开盘价、收盘价、最高价与最低价以及特定时间段的成交量等基本因素做出分析结果。

基本面分析是基于公司本身的素质，对影响公司估价走势的基本因子进行处理和排序，生成可以被量化统计的结果数据。

6.1 一个奇怪的问题

为什么要“买买买”（见图6-1）？

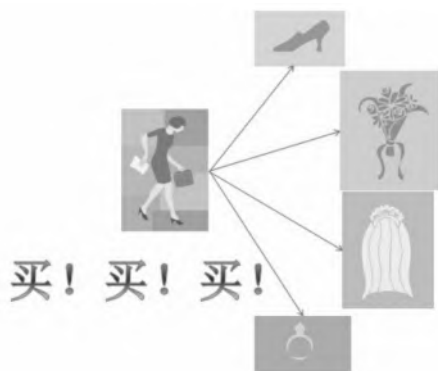


图6-1 买买买

相信读者在买东西之前一定会综合考虑要买的这个东西的形状大小、颜色类型甚至是重量是否符合自己的要求。也就是说，读者在最终决定要买卖一个商品之前，一定会认真比较某个商品的某些特点，还有可能会列出不同的备选方案，从而最终决定买入哪个商品。

那么有一个非常奇怪的问题，为什么大多数人在买入一只股票的时候都没有任何思考的过程，甚至是在一秒钟之内就做出了买卖的决定呢？

下面将给出解释。

6.1.1 因子是什么

因子是什么？

通俗地说，因子就是在选择一只股票时，能够帮助你做出决定的一个因素。例如，如果你觉得净利润增长率大于25%的公司就是一个好公司，那么只需要关注净利润增长率大于25%的公司即可。这个净利润增长率大于25%就是因子。

这只是举一个例子，真正影响公司股价、推动资金进入和股价波动的因子是各种各样的，有基本面、消息面、政策面以及关于研报和分析师预测的预期等。

下面举例来说，对于国内A股，长时间能够表现出色的因子主要有哪些。

1. 规模因子：小市值因子

无论是在A股市场，还是港台以及欧美市场，小市值因子一直存在，并且小市值因子的逻辑很简单易懂。市值越小，其操纵涨跌所需要的资金越少，股票被操纵的可能性就越大。

2. 技术因子：动量反转因子

动量反转因子是典型的技术因子，指的是单个标的涨跌幅度的反转效应明显。简单地说，就是涨得多了就要跌，跌得多了就要涨，远离要回归，价格归于价值。这个其实就是在股市里说的补涨补跌。

3. 预测因子：预测收入的因子

预测因子指的是能够预测下一个财务周期的各种因子。预测收入增长的越高越好，净利润增长的越高越好，支出费用越低越好。这些都是能够帮助决定股价的一些因子。

一般来说，因子包括9类，即规模因子、估值因子、成长因子、盈利因子、动量反转因子、交投因子、波动因子、股东因子、一致性预测因子。

- 规模因子，如总市值、流通市值、自由流通市值。
- 估值因子，如市盈率（TTM）、市净率、市销率、市现率、企业价值倍数。
- 成长因子，如营业收入同比增长率、营业利润同比增长率、归属于母公司的净利润同比增长率、经营活动产生的现金流金额同比增长率。
- 盈利因子，如净资产收益率（ROE）、总资产报酬率（ROA）、销售毛利率、销售净利率。
- 动量反转因子，如前一个月涨跌幅、前2个月涨跌幅、前3个月涨跌幅、前6个月涨跌幅。
- 交投因子，如前一个月日均换手率。
- 波动因子，如前一个月的波动率、前一个月的振幅。
- 股东因子，如户均持股比例、户均持股比例变化、机构持股比例变化。

- 一致性预测因子，如研报或者分析师预测当年净利润增长率、主营业务收入增长率、最近一个月预测净利润上调幅度、最近一个月预测主营业务收入上调幅度、最近一个月上调评级占比。

6.1.2 选取因子

最简单的方法是选取一些自己喜欢的因子，比如前面所说的增长率、市值、营业收入等，一个一个计算不同因子的收益，看看效果如何，效果好就留下，效果不好则剔除，再选择下一个。

下面就以前面所说的净利润增长率为基础做出因子的效果表现。

1. 从表中提取因子

对于读者使用掘金量化来说，选取因子的第一步就是从掘金量化所提供的财务数据表（掘金量化主界面→帮助中心→数据文档）中抽取特定的数据。掘金量化提供了特定的财务因子代码供选取，代码如下：

```
def get_fundamentals_n(table, symbols, end_date, fields=None,
                        filter=None, order_by=None, count=1, df=False):
```

这里函数名为get_fundamentals_n，而其中的参数分别为表名、股票池名、截止于特定的选择时间以及需要选择的因子等，count是周期数，每个标的选择几个数据，df指的是生成的数据以pandas的形式进行返回。

例如，从掘金量化主界面→帮助中心→数据文档→财务数据→衍生财务指标中选择净利润增长率因子，代码如下：

```
_df= get_fundamentals_n(table='deriv_finance_indicator',
                        symbols=symbol_list, end_date=day_time, count=1, fields=" NPGRT
```

```
", df=True)
```

2. 对于股票池的选择

在前面提取因子时，经过查询相应的表和因子名可以选取对应的因子。但是问题在于，如果要在股票池中选择我们所需要的因子，那么首先要建立对应的股票池。

建立股票池的方法，对于读者来说，当然可以选择特定的股票来组成一个股票池，但是无特定目标时，一般选择采用股指组成的成分股来作为股票池。同样，掘金也提供了对应的函数来获取对应的股指对应的成分股。

```
def get_history_constituents(index, start_date=None,
                             end_date=None):
```

这里index代表的是股指，start_date与end_date指的是股指日期，而返回的list每项是个字典，包含的key值如下。

trade_date: 交易日期，datetime类型。

constituents: 一个字典，每只股票的symbol作为key值，weight作为value值。

举例如下：

```
symbol_list = get_history_constituents(index="SHSE.000016",
                                       start_date=now)[0].get("constituents").keys()
```

返回的是上证50成分股，结果如下：

```
[ 'SHSE.600000', 'SHSE.600016', 'SHSE.600019', 'SHSE.600028', 'SHSE.600029',
'SHSE.600030',
'SHSE.600036', 'SHSE.600048', 'SHSE.600050', 'SHSE.600104', 'SHSE.600111',
'SHSE.600309',
'SHSE.600340', 'SHSE.600518', 'SHSE.600519', 'SHSE.600547', 'SHSE.600606',
'SHSE.600837',
'SHSE.600887', 'SHSE.600919', 'SHSE.600958', 'SHSE.600999', 'SHSE.601006',
'SHSE.601088',
'SHSE.601166', 'SHSE.601169', 'SHSE.601186', 'SHSE.601211', 'SHSE.601229',
'SHSE.601288',
'SHSE.601318', 'SHSE.601328', 'SHSE.601336', 'SHSE.601398', 'SHSE.601601',
'SHSE.601628',
'SHSE.601668', 'SHSE.601669', 'SHSE.601688', 'SHSE.601766', 'SHSE.601800',
'SHSE.601818',
'SHSE.601857', 'SHSE.601878', 'SHSE.601881', 'SHSE.601985', 'SHSE.601988',
'SHSE.601989',
'SHSE.603993']
```

下面将选取的成分股与因子组合在一起，要求选取当日的上证50成分股的增长率。

【程序6-1】

```
from gm.api import *
import datetime

set_token("715f84a04a4574e2e77491d62f7c8bbc0c663b12")
day_time,hour_and_mins=str(datetime.datetime.now().strftime('%Y-%m-%d
%H:%M:%S')).split(" "):

index = "SHSE.000016"

symbol_list=get_history_constituents(index=index,start_date=day_time)[0].get("
constituents").keys()

df=
get_fundamentals_n(table="deriv_finance_indicator",symbols=symbol_list,end_dat
e=day_time,
fields="NPGRT",count=1,df=True)

print(df.head(5))
```


【程序6-1】中首先导入了掘金量化的程序包，而datetime是获取当前时间的辅助包，可以通过函数获取当前系统对应的时间并以字符串的形式打印出来，之后通过字符串分割出当前的年月日和时分秒。而symbol_list是获取当前日期上证50构成的成分股，之后将symbol_list导入对应的因子获取函数，最终结果如图6-2所示。

	symbol	pub_date	end_date	NPGRT
0	SHSE.601288	2018-04-28	2018-03-31	5.4317
1	SHSE.601166	2018-04-25	2018-03-31	4.9394
2	SHSE.600606	2018-04-24	2018-03-31	30.3047
3	SHSE.600999	2018-04-27	2018-03-31	-20.5682
4	SHSE.601668	2018-04-28	2018-03-31	14.9802

图6-2 上证50成分股股票增长率情况

读者可以尝试在不同的表以及使用不同的因子来获取不同的因子状态。

3. 对因子进行排序

对于单因子净利润增长率来说，当然是越高越好。

下面对因子进行排序，在这里并不是简单地对数据进行排序，而是需要对决定因子的股票代码进行排序，代码如下：

```
df=
get_fundamentals_n(table="deriv_finance_indicator",symbols=sy
mbol_list,
end_date=day_time,fields="NPGRT",count=1,df=True)
df = df.sort_values(["NPGRT"],ascending=False)
print(df.head(5))
```

这里是直接对pandas数据进行排序，sort_value函数是对pandas内部的数据进行排序的函数，ascending=False决定了排序方式是由大到小的，最终结果如图6-3所示。

	symbol	pub_date	end_date	NPGRT
25	SHSE.600050	2018-04-21	2018-03-31	374.7468
12	SHSE.603993	2018-04-28	2018-03-31	159.5361
19	SHSE.601628	2018-04-27	2018-03-31	119.8406
22	SHSE.601601	2018-04-28	2018-03-31	87.5500
47	SHSE.601857	2018-04-28	2018-03-31	78.1014

图6-3 排序后的上证50成分股股票增长率情况

还有一种方法是产生对应的字典，之后对字典进行排序，部分代码如下：

```
df=
get_fundamentals_n(table="deriv_finance_indicator",symbols=symbol_list,end_date=day_time,
fields="NPGRT",count=1,df=True)

npgrt = df["NPGRT"].values

symbol_and_npgrt = {}

for _symbol,_npgrt in zip(symbol_list,npgrt):
    symbol_and_npgrt[_symbol] = _npgrt
symbol_and_npgrt = sorted(symbol_and_npgrt.items(),key = lambda
x:x[1],reverse = False) #这里是由小到大排列
for _ in symbol_and_npgrt:
    print(_[0])
```

这里首先获取了装载NPGRT数据的pandas，之后建立了一个symbol与NPGRT对应的字典，sorted函数用于对数据进行自动排序，这里根据字典的value对数据排序，之后将排序后的值返回到数据字典中。具体结果请读者自行打印查看。

6.1.3 单因子选股轮动测试

下面使用选定的因子进行选股轮动。在使用某个策略的时候，最重要的是确定策略所使用的方案，本节中所使用的方案如下：

- 股票池选用上证50的成分股。
- 选择股票池中净利润增长率排名前5的5家公司。
- 每个月进行调仓，调出排名跌出前5的公司，重新买入进入排名前5的公司。

1. init格式化对象

使用掘金框架进行回测设计，第一步就是设定格式化对象，其中最重要的是设置回测的方式。

```
def schedule(schedule_func, date_rule, time_rule):
```

schedule函数定义了回测的内容，schedule_func是回测主调用函数，回测主内容写在此处，date_rule是基于回测的时间周期，time_rule是每个时间周期中具体执行的时间点。代码段如下：

```
from gm.api import *
import os

def init(context):

    context.index = "SHSE.000016"
    context.num = 5
    schedule(schedule_func=algo, date_rule='1w', time_rule='09:31:00')
```

最上端的from导入了掘金的程序包，os是操作系统需要的包，在后面会介绍。init函数是回测框架的初始化函数，可以设置大量初始化内容。

2. algo函数

algo是回测框架的主要算法框架，这里定义了每个时间周期需要进行的策略的主要内容。在本策略中，为了简单起见，首先在每个frequency（设定的策略运行周期）上进行清仓操作，之后获取当前时

间段的上证50成分股，下一步获取成分股对应的因子数据，之后对因子进行排序，买入排名前5的标的内容。

```
def algo(context):
    now = context.now

    order_close_all()

    symbol_list= get_history_constituents(index=context.index, start_date=now)[0].
    get("constituents").keys()
    df = get_fundamentals_n(table="deriv_finance_indicator",
    symbols=symbol_list, end_date=now, fields="NPGRT",count=1, df=True)
    df = df.sort_values(["NPGRT"], ascending=False)

    target_list = df["symbol"].values
    target_list = target_list[:context.num]

    for symbol in target_list:
        order_target_percent(symbol=symbol, percent=1. / context.num,
    order_type=OrderType_Market, position_side=PositionSide_Long)
```

这里order_close_all为了简略起见，直接清仓。在真实的实盘交易时，为了节省交易费用等，需要判断持仓情况，这里先不采用。

【程序6-2】

```

from gm.api import *
import os

def init(context):

    context.index = "SHSE.000016"
    context.num = 5
    schedule(schedule_func=algo, date_rule='1m', time_rule='09:31:00')
def algo(context):
    now = context.now
    order_close_all()

symbol_list= get_history_constituents(index=context.index, start_date=now)[0].
get("constituents").keys()
    df = get_fundamentals_n(table="deriv_finance_indicator",
symbols=symbol_list, end_date=now, fields="NPGRT",count=1, df=True)
    df = df.sort_values(["NPGRT"], ascending=False)

    target_list = df["symbol"].values
    target_list = target_list[:context.num]

    for symbol in target_list:
        order_target_percent(symbol=symbol, percent=1. / context.num,
order_type=OrderType_Market, position_side=PositionSide_Long)

def on_backtest_finished(context, indicator):
    print(indicator)
if __name__ == "__main__":
    run(
        strategy_id='9f53e15f-6870-11e8-801c-4cedfb681747',
        filename=(os.path.basename(__file__)),
        mode=MODE_BACKTEST,
        token='715f84a04a4574e2e77491d62f7c8bbc0c663b12',
        backtest_start_time=start_time,
        backtest_end_time='2018-06-25 15:00:00',
        backtest_initial_cash=10000000,
        backtest_adjust=ADJUST_PREV
    )

```

具体代码运行结果如图6-4所示。

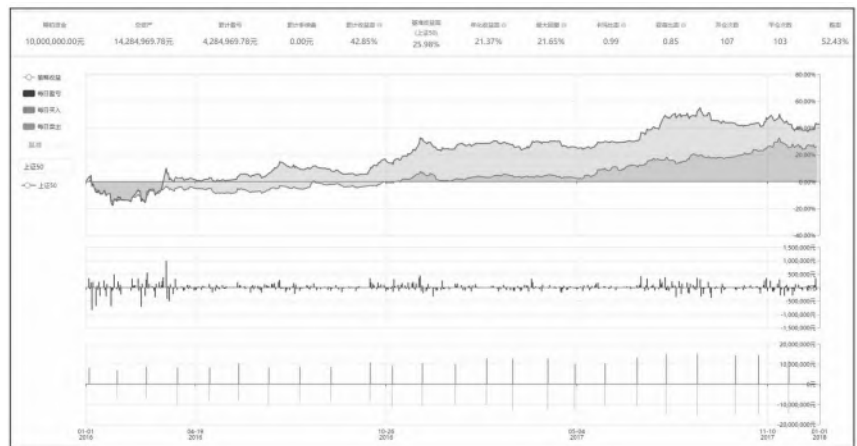


图6-4 选取净利润增长率排前5的回测结果

可以看到，通过选取净利润增长率前5的股票进行回测，其累计收益率为42.85%，而同期上证50的基准收益率为25.98%，可以说获得了约17%的额外收益。

下面展示一个反例，这里是作者随机选择的一个因子，单因子测试的结果如图6-5所示。

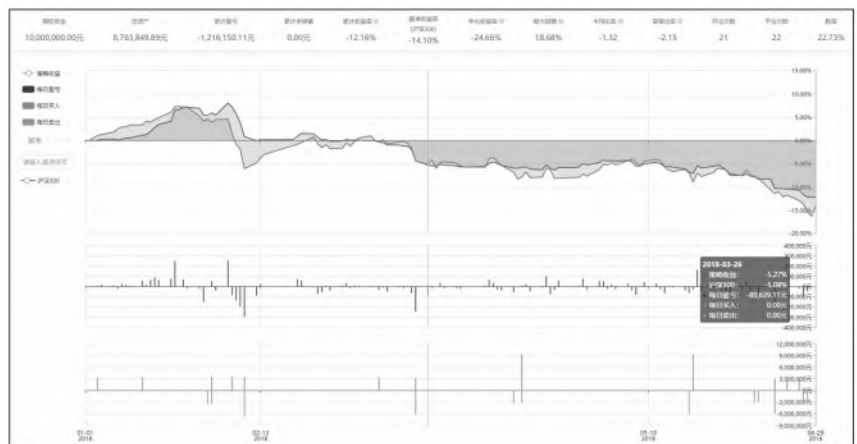


图6-5 一个反例因子的测试结果

从图6-5可以看到，选择因子时，好的因子可以有效地提高收入，让使用者能够获取丰厚的收益，而坏的因子非但不会带来收益，反而会促使使用者产生额外亏损。

怎么选取一个好的因子呢？这点在下一节会专门讲述。

6.2 因子的量化选择

关于因子的作用，在6.1节已经做了专门的讲述。随之而来的问题是因子的选择。究竟什么样的因子会带来额外收益，而什么样的因子会带来额外损失呢？图6-6所示为好的因子与坏的因子所带来的效果。



图6-6 好的因子与坏的因子

从传统的方法来看，对于不同的候选因子，在模型设定的初期，计算每个周期每只正常交易股票的该因子的大小，按从小到大的顺序对样本股票进行排序，并平均分为 n 个组合，一直持有到周期末，在下个周期初再按同样的方法重新构建 n 个组合并持有到周期末，一直重复到模型结束。

6.2.1 基于IC值的多因子计算方法

随着金融行业的发展，在金融系统中产生能够表述某一方面问题的因子越来越多，大概可以分为估值与市值类因子、偿债能力和资本结构类因子、分析师预期类因子、均线型因子、成交量型因子、成长能力类因子、每股指标因子、现金流指标因子、盈利能力和收益质量

类因子、能量型因子、营运能力类因子、超买超卖型因子、趋势型因子这些大类。

大类因子进一步细分下去，目前来说会形成5000个左右的可观测因子，因此要从中找出有效因子，一个有效而合适的测试方法必不可少。

目前最常用的就是因子IC的计算方法。因子IC法来自于多因子模型的打分法，指的是选用若干能够对股票未来时间段收益产生预测作用的因子，根据每个因子在对应位置的状况给出股票在该位置上的得分，然后按照一定的权重将各个因子的得分相加，从而得到该股票各个因子的最终得分。

因此可以看到，在打分模型中，各个因子的权重设定和计算非常重要，即使是非常好的因子，在权重配置下也可能会有不好的结果。

IC的计算公式如下：

$$r_{12} = \frac{\sum_{k=1}^n (x_{1k} - \bar{x}_1)(x_{2k} - \bar{x}_2)}{\sqrt{\sum_{k=1}^n (x_{1k} - \bar{x}_1)^2 \sum_{k=1}^n (x_{2k} - \bar{x}_2)^2}}$$

从公式中可以看到，IC的计算公式实际上就是不同序列的相关系数的计算，那么IC值的计算用一句话解释就是：“IC值为因子与对应的下期收益率之间的关系”。下面通过Python对IC进行计算。假设不同的股票在某个时间段对应的因子值和下期的收益率如表6-1所示。

表6-1 股票因子与收益率表

	因子 A	因子 B	下期收益率
股票 a	6	2	0.02
股票 b	1	4	0.004
股票 c	2	8	0.005
股票 d	7	9	0.007

读者可以手动计算此时的IC值，但是Python中的NumPy工具包提供了更为方便的计算方法，即corrcoef函数：

```
corr = np.corrcoef()
```

完整代码如下：

【程序6-3】

```
import numpy as np

factor_A = [6,1,2,7]
return_profit = [0.02,0.004,0.005,0.007]

corr = np.corrcoef(factor_A,return_profit)
print(corr)
```

打印结果如下：

```
[[ 1.          0.593641]
 [ 0.593641  1.        ]]
```

其中，0.593641是两个序列的相关系数，即为其对应的IC值。读者可以自行计算因子B对应的IC值。

注意

无论是因子A还是因子B，其相关系数对应计算的都是下个时间周期的收益率。

6.2.2 基于IC值的多因子计算方法（续）

除了直接求相关系数的计算方法外，还有一种IC值的计算方法，即回归法。简单来说，就是用该期的股票收益率对上期的因子值做回归，并用该期的因子值预测下期股票的收益率，取下期预测的收益率和下期实际收益率的相关系数。

下面使用表6-1来介绍基于回归法的IC值求解。

1. 计算因子与收益率的回归系数

对于回归的计算方法有很多种，Python提供了专门的工具包，帮助使用者进行回归系数的计算。所需要的包如下：

```
import statsmodels.api as sm
import numpy as np
```

其使用方法如下：

```
x = np.asarray([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
y = 10 + 4 * x1
x_ = sm.add_constant(x)
model = sm.OLS(y, x_)
results = model.fit()
print(results.params)
```

其中，x是生成的一个序列变量，而y为方程的计算结果，具体打印结果如下：

```
[10.  4.]
```

这里result.params对应的是两个结果，第一个为noise值，第二个为对应的weight值。

使用回归计算因子与收益率的代码段如下：

```
factor_1 = [6,1,2,7]
return_profit = [0.02,0.004,0.005,0.007]

x_ = sm.add_constant(factor_1)
y = return_profit

model = sm.OLS(y, x_)
results = model.fit()
print(results.params)
```

而最终打印结果如下：

```
[ 0.003  0.0015]
```

2. 计算下一期股票收益率

在得到了对应的回归系数后，下一步是计算对应的预期股票收益率。代码段如下：

```
pre_return_profit = results.params[0] + np.asarray(factor_1)
*
results.params[1]
```

其结果如下：

```
[ 0.012  0.0045  0.006  0.0135]
```

3. 计算预期收益率与真实收益率的相关系数

最后一步是计算预期收益率与真实收益率之间的相关系数。代码段如下：

```
corr = np.corrcoef(pre_return_profit,return_profit)
print(corr)
```

完整代码如【程序6-4】所示。

【程序6-4】

```
import statsmodels.api as sm
import numpy as np

factor_1 = [6,1,2,7]
return_profit = [0.02,0.004,0.005,0.007]

x_ = sm.add_constant(factor_1)
y = return_profit
model = sm.OLS(y, x_)
results = model.fit()
pre_return_profit = results.params[0] + np.asarray(factor_1) *
results.params[1]

corr = np.corrcoef(pre_return_profit,return_profit)
print(corr)
```

最终结果如下：

```
[[ 1.          0.593641]
 [ 0.593641  1.        ]]
```

可以看到，使用两种方法计算出的相关系数的值是相同的。本例中参数的计算相对简单，读者可以采用更为复杂的数据来计算对应的数据值。

补充说明一点，对于不同的股票，由于其市值不同，因子的影响往往差异很大。例如，最基本的市盈率因子，往往会集中于银行等体量比较大的板块中。在对因子的IC值进行计算之前，首先要根据不同的条件做回归分析，取残差作为一个替代值。这里不展开介绍，相关代码段如下：

```

def fast_batch_get_neutralized_factor(symbol_list, factor, now, more_is_better):

last_day = get_previous_trading_date("SHSE", now)
_symbol_list = symbol_list
symbol_list = []
for _ in _symbol_list:
    symbol_list.append(_)
TOTMKTCAP = []
#根据市值进行中性化处理
data = get_fundamentals(table='trading_derivative_indicator',
symbols=symbol_list, start_date=last_day, end_date=last_day,
fields="TOTMKTCAP")

if len(data) == len(symbol_list):
    for _ in data:
        try:
            TOTMKTCAP.append(_["TOTMKTCAP"])
        except:TOTMKTCAP.append(1e+15)
else:
    for symbol in symbol_list:
        try:
            _data = get_fundamentals(table='trading_derivative_indicator',
symbols=symbol, _date=last_day, end_date=last_day, fields="TOTMKTCAP")
            TOTMKTCAP.append(_data[0]["TOTMKTCAP"])
        except:

```

```

TOTMKTCAP.append(1e+15)

factor_with_table = {}
for line in open("../tools/单因子表列.txt", encoding='UTF-8'):
    line = line.strip('\n')[0:]
    lines = line.split(",")
    factor_with_table[lines[1]] = lines[0]
table_name = factor_with_table[factor]

factor_value_list = []
data = get_fundamentals(table=table_name, symbols=symbol_list,
start_date=last_day,
                        end_date=last_day, fields=factor)

if len(data) == len(symbol_list):
    for _ in data:
        try:
            factor_value_list.append(_[factor])
        except:
            if more_is_better == True:
                factor_value_list.append(-999)
            else:
                factor_value_list.append(999)
    else:
        for symbol in symbol_list:
            try:
                _data = get_fundamentals(table=table_name, symbols=symbol,
start_date=last_day, end_date=last_day, fields=factor)
                factor_value_list.append(_data[0][factor])
            except:
                if more_is_better == True:
                    factor_value_list.append(-999)
                else:
                    factor_value_list.append(999)

TOTMKTCAP = fast_detal_edge(TOTMKTCAP)    #这个是 x
factor_value_list = fast_detal_edge(factor_value_list)    #这个是 y

_TOTMKTCAP = sm.add_constant(TOTMKTCAP)
model = sm.OLS(factor_value_list, _TOTMKTCAP)
results = model.fit()

d_value_dict = {}

```

```

for i in range(len(symbol_list)):
    d_value = factor_value_list[i] - ( results.params[1] * TOTMKTCAP[i] +
results.params[0])
    d_value_dict[symbol_list[i]] = d_value

return d_value_dict

```

因子IC（信息系数）代表因子预测股票收益的能力。IC具体为全部股票在调仓周期期初的排名和本调仓周期收益排名的线性相关度。IC的均值是多个调仓周期IC的平均值。

IC越大，表明是排名越靠前的股票，未来收益排名也越靠前。IC的理论最大值为1，但对于多时间段的IC均值来说，当其值大于0.05时，就可以认为因子是有效的阿尔法因子，大于0.1时，就可以认为因子是特别好的阿尔法因子。当IC均值接近0时，因子可被认为是无效因子。当IC均值小于-0.05时，因子可以被认为是反向有效因子，只要将排名次序颠倒过来，因子就是正向的阿尔法因子。简而言之，IC均值的绝对值越大，因子越有效。

因子IR（信息比率）代表因子在历史上表现的稳定性，公式如下：

$$IR = \frac{IC_{\text{mean}}}{IC_{\text{std}}}$$

因子在不同历史时期的表现有可能差别很大，有的时候表现得很好，有的时候表现得很差，表现在IC上，就是IC的波动率很大。假设IC均值一定，IC的波动率越小，因子表现得越稳定，IR就越大。

6.2.3 因子IC值计算的目标，等权法因子值的合成

对于单个因子的测试来说，直接使用掘金程序进行测试即可，如果回测人员觉得有必要选用更多的因子一起更新使用的內容的话，需要对因子进行合成，生成一个新的数量因子。

对于选定的因子，最常用的方法是按等权重进行合成，如表6-2所示。

表6-2 股票多因子与收益率表

	因子 A	因子 B	因子 C	下期收益率
股票 a	6	2	1	0.02
股票 b	1	4	5	0.004
股票 c	2	8	6	0.005
股票 d	7	9	2	0.007

表6-2在表6-1的基础上做了修改，增加了一个因子C。下面根据不同的值计算对应的IC值。其代码如下：

【程序6-5】

```
import numpy as np

factor_A = [6,1,2,7]
factor_B = [2,4,8,9]
factor_C = [1,5,6,2]
return_profit = [0.02,0.004,0.005,0.007]

corr = np.corrcoef(factor_A,return_profit)
print("factor_A:",corr[0][1])

corr = np.corrcoef(factor_B,return_profit)
print("factor_B:",corr[0][1])

corr = np.corrcoef(factor_C,return_profit)
print("factor_C:",corr[0][1])
```

打印结果如下：


```
factor_A: 0.593641001409
factor_B: -0.651000828374
factor_C: -0.790625442073
```

可以看到对应不同的因子值，其IC值也不相同，因子A的IC值约为0.59，而因子B和因子C对应的IC值约为-0.65和-0.79。此时，因子B和因子C被视为反向因子，因此在选取时需要对其进行辨别。

如果从表中的3个因子中选出两个组成一个新的因子，根据上面的计算方法，选择因子A和因子C是最为合适的。需要注意的是，由于因子C是负相关的，因此在计算时需要将其转成正值计算，代码如下：

```
factor_total = np.asarray(factor_A) + np.asarray(factor_C) * -1
```

这里 factor_A 与 factor_C 共同组成了一个新的因子，factor_total：

```
[ 5 -4 -4  5]
```

下面计算新的因子与收益率之间的IC值，代码如下：

```
factor_total = np.asarray(factor_A) + np.asarray(factor_C) * -1

corr = np.corrcoef(return_profit, factor_total)
print(corr[0][1])
```

打印结果如下：

```
0.69853547
```

可以看到，相较于单独的因子A，因子A和因子C合成的因子，其IC值正向增加了0.1，因此可以认为合成的因子能够提高对收益率的反映。

注意

这里选用等权法进行计算，由于因子A是正相关的，可以认为值越大越好，其权值为+1；而由于因子C为负相关的，可以认为对收益率的影响越小越好，从而权值被设定为-1。至于在计算多因子时到底是越大越好还是越小越好，需要综合认定和对最终结果的判定，一定要在同一个模型中统一判断标准。

在前面6.1节已经介绍了，除了基本面因子外，还可以根据不同的条件生成各种有代表性的因子，这点在下一节会介绍。

6.3 实战：基于成长因子的模型测试

成长模型是一种以公司未来成长为基础的选股模型。

成长投资理论由现代投资理论的开路先锋之一，Philip A. Fisher（费雪）最早提出，他在1959年的名著《怎样选择成长股》中对该投资理念进行了详细阐述，解释了成长股的标准、如何寻找成长股、怎样把握时机获利等一系列重要的问题，为广大投资者称道。

在实际操作中，成长型投资通常是在对经济周期和行业景气分析的基础上，结合股票基本面情况，包括收益率和增长率等属性的分析，来评估和选择成长型股票。

在量化形式上，成长型投资主要是通过ROE、ROA、ROIC、营业收入增长率、主营业务利润率等参数来挖掘成长性相对更高的股票。

6.3.1 模型说明

以ROIC-NPG两个指标来构建成长模型。根据我们对各项指标的重新梳理，现将ROIC列为质量指标，另外构建包括EBITG（息税前收益增长率）、NPG（净利润增长率）、MPG（主营利润增长率）、GPG（毛利润增长率）、OPG（营业利润增长率）、OCG（经营现金流增长率）、NAG（净资产增长率）、EPSG（每股收益增长率）、ROEG（净资产收益率增长率）、GMPG（毛利率增长率）10个考核公司成长能力的指标。

1. 因子的构建形式

在多因子计算的初始，需要建立一个能够存储因子的矩阵，Python中对于矩阵的建立，一般使用的是pandas工具包，而对于选定的因子，建立一个可以容纳其数据的字段名即可，代码如下：

```
df = pd.DataFrame([])
df["symbol"] = symbol_list
df["EBITG"] = -999
df["NPG"] = -999
df["TAG"] = -999 # MPG用TAG代替
df["GPG"] = -999
df["OPG"] = -999
df["OCG"] = -999
```

注意

在掘金数据集中，MPG数据被TAG代替，这里只需要建立TAG代码段即可。

可能读者注意到，在这里其值被定义为-999，这样做的目的是，在读取数据时，为了防止有数据缺失或者出错，将默认值设置成现实数据中可能会遇到的最小值。这样做的好处是在后续计算时可以自动将出错的数据的计算结果降为最差结果。

2. 因子的获取

对于需要使用的因子，掘金量化也提供了对应的基本面获取函数：

```
def get_fundamentals_n(table, symbols, end_date, fields=None,
                        filter=None, order_by=None, count=1, df=False):
```

其中，table是对应的表名，field是字段名，order_by是排序方式，如果有一些需要过滤的字段，可以使用filter进行字段过滤。

首先看本模型中的第二个因子—NPG。通过查表可得，第一个需要查询的字段NPG在掘金数据库中的字段名为NPGRT，其在deriv_finance_indicator表中，因此根据get_fundamentals_n函数可以写成如下形式：

```
get_fundamentals(table='deriv_finance_indicator',
                  symbols=symbol_list,
                  start_date=day_time,
                  end_date=day_time, fields="NPGRT", df=True)
```

需要注意的是，对于部分数据有可能会产生缺失，因此在读取时需要对数据的数量进行判断，如果是正常读取，就可以正常结束，而当数据出现缺失时，可使用默认的最小值进行填充，代码如下：

```

_df = get_fundamentals(table='deriv_finance_indicator', symbols=symbol_list,
start_date=day_time, end_date=day_time, fields="NPGRT", df=True)
if len(_df) == len(symbol_list):
df["NPG"] = _df["NPGRT"]
else:
for number in range(len(symbol_list)):
try:
_df = get_fundamentals(table='deriv_finance_indicator',
symbols=symbol_list[number], start_date=day_time, end_date=day_time,
fields="NPGRT")
_NPG = tools.get_data_value(_df, "NPGRT")
df.iloc[number, 2] = _NPG[0]
except:
df.iloc[number, 2] = -999

```

这里首先使用symbol_list批量化获取数据值，之后判断读取的数据有没有缺失，如果产生缺失，就逐个获取对应的因子数据，而对于缺失的因子数据，使用-999进行填充。

前面介绍了第二个因子NPG的获取，而对于第一个因子EBITG（息税前收益增长率），掘金数据表中并没有直接给出，需要读者对数据进行处理，自行计算得出，分别使用本期的EBITMARGIN（息税前利润率）与上期的值计算，即：

```

EBITG = ((now_EBITMARGIN - last_EBITMARGIN) /
last_EBITMARGIN)

```

对于不同的数据值获取方式，掘金同时提供了不同的基本面获取函数，代码如下：

```

def get_fundamentals_n(table, symbols, end_date, fields=None,
filter=None,order_by=None, count=1, df=False):

```

get_fundamentals_n函数的意思是查询基本面财务数据中每个股票在end_date的前n条，本处只需要获取两条即可。

```

for number in range(len(symbol_list)):
try:
    _df = get_fundamentals_n(table='deriv_finance_indicator',
symbols=symbol_list[number], end_date=day_time, count=2, fields="EBITMARGIN")
    now_EBITMARGIN = _df[0]["EBITMARGIN"]
    last_EBITMARGIN = _df[1]["EBITMARGIN"]
    EBITG = ((now_EBITMARGIN - last_EBITMARGIN) / last_EBITMARGIN)
    df.iloc[number, 1] = EBITG
except:
    df.iloc[number, 1] = -999

```

字段中的EBITG分别取了本期和上期的值，其差值与上期的数据比值可以认为是所需要的息税前利润率。

取全部因子的代码如下：

```

# symbol_list = tools.read_MSCI()
df = pd.DataFrame([])
df["symbol"] = symbol_list
df["EBITG"] = -999
df["NPG"] = -999
df["TAG"] = -999 # MPG 取不到，用 TAG 代替
df["GPG"] = -999
df["OPG"] = -999
df["OCG"] = -999
df["vol"] = 0
# 求出 EBITG
for number in range(len(symbol_list)):
    try:
        _df = get_fundamentals_n(table='deriv_finance_indicator',

```

```

symbols=symbol_list[number], end_date=day_time, count=2, fields="EBITMARGIN")
    now_EBITMARGIN = _df[0]["EBITMARGIN"]
    last_EBITMARGIN = _df[1]["EBITMARGIN"]
    EBITG = ((now_EBITMARGIN - last_EBITMARGIN) / last_EBITMARGIN)
    df.iloc[number, 1] = EBITG
except:
    df.iloc[number, 1] = -999

# 求 NPG
_df = get_fundamentals(table='deriv_finance_indicator', symbols=symbol_list,
start_date=day_time, end_date=day_time, fields="NPGRT", df=True)
if len(_df) == len(symbol_list):
    df["NPG"] = _df["NPGRT"]
else:
    for number in range(len(symbol_list)):
        try:
            _df = get_fundamentals(table='deriv_finance_indicator',
symbols=symbol_list[number], start_date=day_time, end_date=day_time,
fields="NPGRT")
            _NPG = tools.get_data_value(_df, "NPGRT")
            df.iloc[number, 2] = _NPG[0]
        except:
            df.iloc[number, 2] = -999

# 求出 TAG
_df = get_fundamentals(table='deriv_finance_indicator', symbols=symbol_list,
start_date=day_time, end_date=day_time, fields="TAGRT", df=True)
if len(_df) == len(symbol_list):
    df["TAG"] = _df["TAGRT"]
else:
    for number in range(len(symbol_list)):
        try:
            _df = get_fundamentals(table='deriv_finance_indicator',
symbols=symbol_list[number], start_date=day_time, end_date=day_time,
fields="TAGRT")
            _NPG = tools.get_data_value(_df, "TAGRT")

            df.iloc[number, 2] = _NPG[0]
        except:
            df.iloc[number, 2] = -999

# 求 GPG
for number in range(len(symbol_list)):

```

```

try:
    _df = get_fundamentals_n(table='deriv_finance_indicator',
symbols=symbol_list[number], end_date=day_time, count=2, fields="OPGPMARGIN")

    now_EBITMARGIN = _df[0]["OPGPMARGIN"]
    last_EBITMARGIN = _df[1]["OPGPMARGIN"]
    EBITG = ((now_EBITMARGIN - last_EBITMARGIN) / last_EBITMARGIN)

    df.iloc[number, 4] = EBITG
except:
    df.iloc[number, 4] = -999

# 求 OPG
for number in range(len(symbol_list)):
    try:
        _df = get_fundamentals_n(table='deriv_finance_indicator',
symbols=symbol_list[number], end_date=day_time, count=2, fields="OPPRORT")

        now_EBITMARGIN = _df[0]["OPPRORT"]
        last_EBITMARGIN = _df[1]["OPPRORT"]
        EBITG = ((now_EBITMARGIN - last_EBITMARGIN) / last_EBITMARGIN)
        df.iloc[number, 5] = EBITG
    except:
        df.iloc[number, 5] = -999

# 求 OCG
for number in range(len(symbol_list)):
    try:
        _df = get_fundamentals_n(table='deriv_finance_indicator',
symbols=symbol_list[number], end_date=day_time, count=2, fields="NCFPS")
        now_EBITMARGIN = _df[0]["NCFPS"]
        last_EBITMARGIN = _df[1]["NCFPS"]
        EBITG = ((now_EBITMARGIN - last_EBITMARGIN) / last_EBITMARGIN)
        df.iloc[number, 6] = EBITG
    except:
        df.iloc[number, 6] = -999

```

生成的因子矩阵如图6-7所示。

	symbol	EBITG	NPG	TAG	GPG	OPG	OCG
0	SHSE.600000	-999.0000	-1.0856	-7.7215	-0.7566	0.0357	-1.2163
1	SHSE.600008	-0.0696	-17.8317	19.6606	-0.7582	-0.2473	-0.1166
2	SHSE.600009	-0.0718	28.6179	20.3458	-0.7122	-0.0216	-0.3339
3	SHSE.600010	0.3434	222.1856	22.1764	-0.7143	0.2254	-1.7664
4	SHSE.600011	0.2523	86.9642	15.1118	-0.6618	1.0049	-8.9287
5	SHSE.600015	-999.0000	5.2328	1.3626	-0.7681	-0.0322	-0.9754
6	SHSE.600016	-999.0000	32.6025	-6.8159	-0.6997	0.1790	-0.9932
7	SHSE.600018	-0.3022	12.8480	6.7102	-0.7977	-0.2261	-1.3802
8	SHSE.600019	0.1064	16.9055	12.7369	-0.7288	0.2013	-0.9414
9	SHSE.600023	-0.3734	13.5005	7.2002	-0.8625	-0.4635	-0.6455
10	SHSE.600025	-0.0086	-0.4916	2.5441	-0.7827	-0.2988	0.4431
11	SHSE.600028	0.3171	374.7468	8.5934	-0.7446	0.3757	-2.4729
12	SHSE.600029	-0.0603	-11.6128	12.8822	-0.6878	0.6151	-1.4010
13	SHSE.600030	-999.0000	-6.8055	20.4395	-0.7770	-0.0059	-1.2780

图6-7 部分数据因子矩阵

可以看到，数据中有一系列的-999，主要是存在数据的缺失问题。

3. 矩阵因子的计算与排序

对于生成的数据集，下面提取出对应的数据组成矩阵，代码如下：

```
df = df.dropna()
df_factor = df.iloc[:,1:]
df_factor = np.asmatrix(df_factor)
```

因为第一个数据是symbol，在计算时不需要，所以可以从第一列开始存储。

```
-999.0000  -1.0856  -7.7215 -0.7566  0.0357  -1.2163
-0.0696  -17.8317  19.6606 -0.7582  -0.2473  -0.1166
-0.0718   28.6179  20.3458 -0.7122  -0.0216  -0.3339
0.3434   222.1856  22.1764 -0.7143  0.2254  -1.7664
0.2523   86.9642  15.1118 -0.6618  1.0049  -8.9287
-999.0000   5.2328   1.3626 -0.7681  -0.0322  -0.9754
```

下面对生成的矩阵进行因子计算。在本策略模型中，各个因子对收益率的影响可以认为越大越好，因此可以统一地将全部权重值设成-1，这样做的好处是在后续的计算中，可以调用默认方法将值从小

到大进行排列。当然，读者也可以将其全部设置成+1，只需要统一规格即可。

```
weight_mat = np.asmatrix(weight)
res = np.dot(df_factor, weight_mat)
df["score"] = (res)
df = (df.sort_values(["score"]))
```

对于矩阵计算，这里调用了NumPy中的矩阵计算方法，res就是最终计算的结果，将其重新作为一个因子合并到df数据集中，打印结果如图6-8所示。

	symbol	EBITG	NPG	TAG	GPG	OPG	OCG		score
0	SHSE.600000	-999.0000	-1.0856	-7.7215	-0.7566	0.0357	-1.2163	49	-5.9173
1	SHSE.600008	-0.0696	-17.8317	19.6606	-0.7582	-0.2473	-0.1166	262	-5.1157
2	SHSE.600009	-0.0718	28.6179	20.3458	-0.7122	-0.0216	-0.3339	242	-4.6576
3	SHSE.600010	0.3434	222.1856	22.1764	-0.7143	0.2254	-1.7664	40	-4.6171
4	SHSE.600011	0.2523	86.9642	15.1118	-0.6618	1.0049	-8.9287	215	-4.6115
5	SHSE.600015	-999.0000	5.2328	1.3626	-0.7681	-0.0322	-0.9754	272	-4.4601
6	SHSE.600016	-999.0000	32.6025	-6.8159	-0.6997	0.1790	-0.9932	234	-4.4385
7	SHSE.600018	-0.3022	12.8480	6.7102	-0.7977	-0.2261	-1.3802	117	-4.4302
8	SHSE.600019	0.1064	16.9055	12.7369	-0.7288	0.2013	-0.9414	252	-4.4071
9	SHSE.600023	-0.3734	13.5005	7.2002	-0.8625	-0.4635	-0.6455	80	-4.3782

图6-8 带有结果的因子矩阵

可以看到，这里对于不同的标的，综合获取了各个因子后，利用矩阵计算从而获得了一个最终的计算值。下面对生成的结果根据股票的最终得分进行排序。

```
df = (df.sort_values(["score"]))
symbol_list = []
for _ in df["symbol"].values:
    symbol_list.append(_)
```

pandas中有专门用来排序的函数，sort_values是根据pandas的某个因子进行排序。

4. 核心选股模型的编写

核心选股模型的代码如下：

```
def 成长模型(index,now):
    symbol_list = tools.get_symbol_list(index,now)

    last_day = get_previous_trading_date("SHSE",now)
    day_time = last_day
    #symbol_list = tools.read_MSCI()
    df = pd.DataFrame([])
    df["symbol"] = symbol_list
    df["EBITG"] = -999
    df["NPG"] = -999
    df["TAG"] = -999 #MPG 取不到，用 TAG 代替
    df["GPG"] = -999
    df["OPG"] = -999
    df["OCG"] = -999
    #求出 EBITG
    for number in range(len(symbol_list)):
        try:
            _df = get_fundamentals_n(table='deriv_finance_indicator',
symbols=symbol_list[number],end_date=day_time, count=2,fields="EBITMARGIN")

            now_EBITMARGIN = _df[0]["EBITMARGIN"]
```

```

last_EBITMARGIN = _df[1]["EBITMARGIN"]
EBITG = ((now_EBITMARGIN-last_EBITMARGIN)/last_EBITMARGIN)

df.iloc[number,1] = EBITG
except:df.iloc[number,1] = -999

#求 NPG
_df = get_fundamentals(table='deriv_finance_indicator', symbols=symbol_list,
start_date=day_time,end_date=day_time, fields="NPGRT",df=True)
if len(_df) == len(symbol_list):
    df["NPG"] = _df["NPGRT"]
else:
    for number in range(len(symbol_list)):
        try:
            _df = get_fundamentals(table='deriv_finance_indicator',
symbols=symbol_list[number], start_date=day_time,
                                end_date=day_time, fields="NPGRT")
            _NPG = tools.get_data_value(_df,"NPGRT")

            df.iloc[number, 2] = _NPG[0]
        except:
            df.iloc[number, 2] = -999

#求出 TAG
_df = get_fundamentals(table='deriv_finance_indicator', symbols=symbol_list,
start_date=day_time,end_date=day_time, fields="TAGRT",df=True)
if len(_df) == len(symbol_list):
    df["TAG"] = _df["TAGRT"]
else:
    for number in range(len(symbol_list)):
        try:
            _df = get_fundamentals(table='deriv_finance_indicator',
symbols=symbol_list[number], start_date=day_time,
                                end_date=day_time, fields="TAGRT")
            _NPG = tools.get_data_value(_df,"TAGRT")

            df.iloc[number, 2] = _NPG[0]
        except:
            df.iloc[number, 2] = -999

#求 GPG
for number in range(len(symbol_list)):
    try:
        _df = get_fundamentals_n(table='deriv_finance_indicator',

```

```
symbols=symbol_list[number],end_date=day_time, count=2,fields="OPGPMARGIN")
```

```
now_EBITMARGIN = _df[0]["OPGPMARGIN"]
last_EBITMARGIN = _df[1]["OPGPMARGIN"]
EBITG = ((now_EBITMARGIN-last_EBITMARGIN)/last_EBITMARGIN)
```

```
df.iloc[number,4] = EBITG
except:df.iloc[number,4] = -999
```

```
#求 OPG
```

```
for number in range(len(symbol_list)):
    try:
        _df = get_fundamentals_n(table='deriv_finance_indicator',
symbols=symbol_list[number],end_date=day_time, count=2,fields="OPPRORT")
```

```
now_EBITMARGIN = _df[0]["OPPRORT"]
last_EBITMARGIN = _df[1]["OPPRORT"]
EBITG = ((now_EBITMARGIN-last_EBITMARGIN)/last_EBITMARGIN)
df.iloc[number,5] = EBITG
except:df.iloc[number,5] = -999
```

```
#求 OCG
```

```
for number in range(len(symbol_list)):
    try:
        _df = get_fundamentals_n(table='deriv_finance_indicator',
symbols=symbol_list[number],end_date=day_time, count=2,fields="NCFPS")
now_EBITMARGIN = _df[0]["NCFPS"]
last_EBITMARGIN = _df[1]["NCFPS"]
EBITG = ((now_EBITMARGIN-last_EBITMARGIN)/last_EBITMARGIN)
df.iloc[number,6] = EBITG
except:df.iloc[number,6] = -999
```

```
df = df.dropna()
df_factor = df.iloc[:,1:]
df_factor = np.asmatrix(df_factor)
```

```
#先进行列归一化，然后对每行进行标准化处理
```

```
df_factor = preprocessing.MinMaxScaler().fit_transform(df_factor)
weight = [[-1],[-1],[-1],[-1],[-1],[-1]]
weight_mat = np.asmatrix(weight)
res = np.dot(df_factor, weight_mat)
df["score"] = (res)
```

```
df = (df.sort_values(["score"]))
```

```
print(df)
symbol_list = []
for _ in df["symbol"].values:
    symbol_list.append(_)

return symbol_list
```

这里的代码段成长模型首先解析指数的构成成分股，之后根据需要对所有的成分股进行排序并打分，接着对打分的结果进行排序，返回经过排序后的股票列表。

6.3.2 使用模型进行回测

以ROIC-NPG两个指标来构建成长模型。根据我们对各项指标的重新梳理，现将ROIC列为质量指标，另外构建了EBITG、NPG和MPG。一个最简单的策略就是每日买入排序后的股票标的前N名，使用的是上证50作为股票池，核心选股代码在6.3.1小节已经给出。这里为了简化测试，取前5名继续排序，代码如下：

【程序6-6】

```

from 挖掘成长 import 成长模型_挖掘成长伏藏 as fun
import os

def init(context):
    schedule(schedule_func=algo, date_rule='1d', time_rule='09:31:00')
    context.index = "SHSE.000016"

    context.num = 5

def algo(context):
    now = context.now
    last_day = get_previous_trading_date("SHSE", now)

    order_close_all()

    """核心选股代码"""
    _symbol_list = fun.成长模型(context.index, now)
    """核心选股代码"""
    symbol_list = []
    for symbol in _symbol_list:
        symbol_list.append(symbol)

    # 取前多少个标的
    target_list = symbol_list[:context.num] # 无黑名单存在, 卖掉以后又可以买回

    for symbol in target_list:
        order_target_percent(symbol=symbol, percent=1. / context.num,
order_type=OrderType_Market, position_side=PositionSide_Long)
def on_backtest_finished(context, indicator):
    print(indicator)
if __name__ == "__main__":
    run(
        strategy_id='9f53e15f-6870-11e8-801c-4cedfb681747',
        filename=(os.path.basename(__file__)),
        mode=MODE_BACKTEST,
        token='715f84a04a4574e2e77491d62f7c8bbc0c663b12',
        backtest_start_time="2016-01-10 09:30:00",
        backtest_end_time='2018-06-10 15:00:00',
        backtest_initial_cash=10000000,
        backtest_adjust=ADJUST_PREV
    )

```

具体结果如图6-9所示。

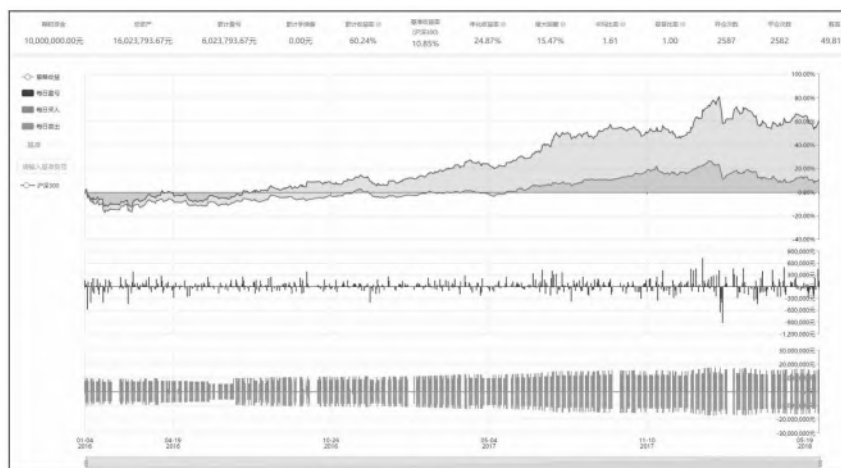


图6-9 成长模型回测结果

从图6-9可以看到回测结果并不是很好，因为这里省略了部分选股策略和算法，这也给读者留下了极大的优化空间，希望读者可以在此基础上做出更改。

6.4 霍华·罗斯曼的投资模型

华尔街永远不缺传奇与经典，对于纵横华尔街的一些传奇投资人，自有其生存与发展之道。霍华·罗斯曼就是其中之一。下面使用掘金API重现其投资模型和方法。

6.4.1 霍华·罗斯曼简介

霍华·罗斯曼（Howard Rothman）是华尔街著名的投资分析公司——远见投资顾问公司（Vision Investment Advisor, VIA）的首席投

资分析师和总裁（1997～2002年），也曾为美国折扣券商协会（the National Introducing Brokers Association）的主席及创始会员。

他于1990年一月被折扣券商会员（Introducing Broker Membership）推荐进入国家期货协会董事会（the National Futures Association Board of Directors）担任董事，并连任达3届。目前为期货产业协会的纪律委员会（the Law and Compliance Division of the Futures Industry Association）会员。

霍华·罗斯曼的投资方式强调“在正确的时机买入并紧握持股”与“选择优质股票并长期持有”（“buy right and hold tight” and “buy strong and hold long”）。

他认为只要在适当的价格买入稳定且持续成长获利的股票，不需要经常更换持股，投资报酬率必然指日可期。由此可见，霍华·罗斯曼的投资哲学虽然简单，却充分显现了首席投资分析师的谨慎风范。

6.4.2 霍华·罗斯曼的投资模型

霍华·罗斯曼强调其投资风格在于为投资大众建立均衡且以成长为导向的投资组合。其选股方式偏好大型股、管理良好且领导产业趋势以及产生实际报酬率的公司，不仅重视公司产生现金的能力，也强调有稳定成长能力的重要性。

下面介绍几个要点。

- （1）总市值大于等于50亿美元。
- （2）良好的财务结构。
- （3）较高的股东权益报酬。
- （4）拥有良好且持续的自由现金流量。

- (5) 稳定持续的营收成长率。
- (6) 优于比较指数的盈余报酬率。

将其投资模型用程序语言组织起来：

- (1) 总市值 $\geq$ 市场平均值*1.0。
- (2) 最近一季流动比率 $\geq$ 行业平均值。
- (3) 近四季股东权益报酬率 $\geq$ 市场平均值。
- (4) 最近三年CPS $\geq$ 0且最近四季CPS $\geq$ 0。
- (5) 近四季营收成长率为6%至30%。
- (6) 近四季盈余成长率为8%至50%。

6.4.3 对霍华·罗斯曼模型的分析

对于组织起来的投资模型并不能直接使用，最好的方式是根据已有的条件逐步分析和进行下去。

投资模型共有6个条件，其中条件5和条件6可以认为是自身的判定条件，可以首先被选择或剔除。

代码如下：

```
# (5) 近四季营收成长率为 6%至 30%。          由高到低 NPGRT
# (6) 近四季盈余成长率为 8%至 50%。          净利润      由高到低 TAGRT
df = get_fundamentals(table='deriv_finance_indicator', symbols=symbol_list,
start_date=now, end_date=now, fields="NPGRT,TAGRT", filter="NPGRT > 6 and
NPGRT < 30 and TAGRT > 8 and TAGRT < 50", df=True)

symbol_list = []
for _ in df["symbol"].values:
    symbol_list.append(_)
```

接着对剩余的4个条件进行分析，从字面上看，这4个条件均为行业或者股票标的池之间的比较，因子可以共同作为一个选择形式分析。更为细分地看，对于条件1和条件2，一般情况下，总市值和流动比率并没有特定的要求，也就是说，并不需要对其进行详细的比较，因此可以作为一个单独的选股条件进行选择。

代码如下：

```
# (1) 总市值≥市场平均值*1.0
# (2) 最近一季流动比率≥市场平均值
df = get_fundamentals(table='trading_derivative_indicator',
symbols=symbol_list, start_date=now, end_date=now, fields="TOTMKTCAP,PCTTM",
df=True)
TOTMKTCAP_mean = df["TOTMKTCAP"].mean()
PCTTM_mean = df["PCTTM"].mean()
df = df[df["TOTMKTCAP"] > TOTMKTCAP_mean]
df = df[df["PCTTM"] > PCTTM_mean]

symbol_list = []
for _ in df["symbol"].values:
    symbol_list.append(_)
```

至于条件3和条件4，可以认为是对不同股票标的之间的比较，因此可以对其进行传统的多因子排序，默认其值越大越好，代码如下：

```

_df = get_fundamentals(table='deriv_finance_indicator', symbols=symbol_list,
start_date=now,
                        end_date=now,fields="ROEAVGCUT,FCFEPS", df=True)

if len(_df) == len(symbol_list):
    df["ROEAVGCUT"] = _df["ROEAVGCUT"]
    df["FCFEPS"] = _df["FCFEPS"]
else:
    for number in range(len(symbol_list)):
        try:
            _df = get_fundamentals(table='deriv_finance_indicator',
symbols=symbol_list[number], start_date=now, end_date=now,
fields="ROEAVGCUT,FCFEPS")
            _factor_value = tools.get_data_value(_df, "ROEAVGCUT")
            df.iloc[number, 1] = _factor_value[0]

            _factor_value = tools.get_data_value(_df, "FCFEPS")
            df.iloc[number, 2] = _factor_value[0]
        except:
            df.iloc[number, 1] = np.mean(df["ROEAVGCUT"])
            df.iloc[number, 2] = np.mean(df["FCFEPS"])

```

最后对求出的条件3和条件4进行排序，这里还是使用矩阵化乘积的方式进行。

霍华·罗斯曼的完整选股模型如下：

```

def get_symbol_list(index, now):
    _symbol_list = tools.get_symbol_list(index, now)
    symbol_list = []
    bank_list = tools.get_symbol_list("SHSE.000947", now)
    stock_company_list = tools.get_symbol_list("SZSE.399975", now)

    for _ in _symbol_list:
        if (_ not in bank_list) and (_ not in stock_company_list):
            symbol_list.append(_)
    # (1) 总市值≥市场平均值*1.0
    # (2) 最近一季流动比率≥市场平均值
    df = get_fundamentals(table='trading_derivative_indicator',
symbols=symbol_list, start_date=now, end_date=now,
                        fields="TOTMKTCAP,PCTTM", df=True)
    TOTMKTCAP_mean = df["TOTMKTCAP"].mean()
    PCTTM_mean = df["PCTTM"].mean()
    df = df[df["TOTMKTCAP"] > TOTMKTCAP_mean]
    df = df[df["PCTTM"] > PCTTM_mean]

    symbol_list = []
    for _ in df["symbol"].values:
        symbol_list.append(_)

    # (5) 近四季营收成长率为 6%至 30%。                由高到低 NPGRT
    # (6) 近四季盈余成长率为 8%至 50%。                净利润                由高到低 TAGRT
    df = get_fundamentals(table='deriv_finance_indicator', symbols=symbol_list,
start_date=now, end_date=now,

```

```

        fields="NPGRT,TAGRT", filter="NPGRT > 6 and NPGRT < 30
and TAGRT > 8 and TAGRT < 50", df=True)

symbol_list = []
for _ in df["symbol"].values:
    symbol_list.append(_)
df = pd.DataFrame([])
df["symbol"] = symbol_list
df["ROEAVGCUT"] = -999
df["FCFEPS"] = -999
_df = get_fundamentals(table='deriv_finance_indicator', symbols=symbol_list,
start_date=now,
                        end_date=now,
                        fields="ROEAVGCUT,FCFEPS", df=True)

if len(_df) == len(symbol_list):
    df["ROEAVGCUT"] = _df["ROEAVGCUT"]
    df["FCFEPS"] = _df["FCFEPS"]
else:
    for number in range(len(symbol_list)):
        try:
            _df = get_fundamentals(table='deriv_finance_indicator',
symbols=symbol_list[number],
                                start_date=now,
                                end_date=now, fields="ROEAVGCUT,FCFEPS")
            _factor_value = tools.get_data_value(_df, "ROEAVGCUT")
            df.iloc[number, 1] = _factor_value[0]

            _factor_value = tools.get_data_value(_df, "FCFEPS")
            df.iloc[number, 2] = _factor_value[0]
        except:
            df.iloc[number, 1] = np.mean(df["ROEAVGCUT"])
            df.iloc[number, 2] = np.mean(df["FCFEPS"])

df = df.dropna()
df_factor = df.iloc[:, 1:]
df_factor = np.asmatrix(df_factor)

# 先进行列归一化，然后在对每行进行标准化处理
df_factor = preprocessing.MinMaxScaler().fit_transform(df_factor)

weight = [[-1], [-1]]
weight_mat = np.asmatrix(weight)
res = np.dot(df_factor, weight_mat)
df["score"] = (res)

df = (df.sort_values(["score"]))
symbol_list = []
for _ in df["symbol"].values:
    symbol_list.append(_)

return symbol_list

```

请读者自行建立选股框架进行验证。

注意

这里模型的使用是基于作者对霍华·罗斯曼投资模型的理解，读者需要通过自身的消化和理解做出合适自己的模型。例如，作者在分析时使用1、2、5、6作为选择条件，3和4作为多因子排序条件，而完全可以将1、2、3、4作为多因子排序条件共同进行下一步的选取。这点请读者自行完成。

6.5 小结

本章介绍了量化选股中最基础的多因子选股策略，从一个简单的例子介绍了因子的选择和对一个股票标的产生的影响。之后介绍了使用IC值计算某个时间段的因子与下一个时间段股票收益之间的关系，作用是通过比较和计算不同的因子对下一个时间段股票收益的影响，从而选出较为适合的相关因子。

因子的选取方法有很多，但是万变不离其宗，其根本就是需要找到当前的因子与下期股票的涨势相互之间的关系，这点请读者牢记。

成长模型是读者接触到的第一个较为完整的选股模型，核心是使用特定的因子综合起来计算最终因子得分，这里是使用等权法计算的，也是最为传统的方法。而在回测框架上，使用的是最为简单的模型框架，选择模型时没有考虑止损和止盈相关的影响。而这些都是现实中需要极其注意的地方。

本章介绍了多因子入门的知识，主要是基本面选股，但是在实际使用时，除了基本面因子外，还有很多技术面因子，这点将在下一章向读者介绍。

第7章

带技术指标的多因子策略

多因子策略除了基本面因子外，还有大量的技术面因子。

技术面因子作为多因子策略的重要组成部分，已经越来越多地被量化基金和程序设计人员所使用。其主要作用是作为基本面选股的一个重要补充，解决了基本面选股策略中对买入时机的选择有所欠缺的问题。

技术面选股和基本面选股是一个问题的不同方面，它们共同补充从而构成了多因子选股的完整策略，如图7-1所示。



图7-1 完整的多因子选股策略

7.1 技术面多因子介绍

相对于基本面因子，技术面因子的设计和组合是多种多样的，甚至可以说是天马行空、无奇不有的。因为基本面因子本质来源于公司

的财务、供销、利润以及产出等，而技术面因子的组合计算更多涉及数学、统计等。本节主要介绍一些常用的数学技术性指标。

7.1.1 101个技术因子

2015年12月，WorldQuant LLC公布了101个阿尔法表达式（见图7-2），并声称其中80%的因子仍然在实盘中使用。国内众多量化爱好者如获至宝，蜂拥实现并验证其因子的有效性。

虽然从检验结果来看，公布的大部分因子在中国股市的效果平平或已失效，但是其因子的构造形式却值得深入研究。论文中说到，目前量化交易中使用的因子数量早已成百上千，甚至达到了数百万的数量级。这已远远不是通过人工搜索并试验有效因子所能企及的。

也就是说，如何通过机器实现这些阿尔法因子的挖掘、测试和应用才是更有意义的，也是大势所趋。



```
Alpha#1: (rank(Ts_ArgMax(SignedPower(((returns < 0) ? stddev(returns, 20) : close), 2.), 5)) - 0.5)
Alpha#2: (-1 * correlation(rank(delta(log(volume), 2)), rank(((close - open) / open)), 6))
Alpha#3: (-1 * correlation(rank(open), rank(volume), 10))
Alpha#4: (-1 * Ts_Rank(rank(low), 9))
Alpha#5: (rank((open - (sum(vwap, 10) / 10))) * (-1 * abs(rank((close - vwap)))))
Alpha#6: (-1 * correlation(open, volume, 10))
Alpha#7: ((adv20 < volume) ? ((-1 * ts_rank(abs(delta(close, 7)), 60)) * sign(delta(close, 7))) : (-1 * 1))
```

图7-2 101个技术选股因子

有兴趣的读者可以找文章的原文来阅读，本书只实现其中3个因子，目的是给读者一个大概的引导，起到抛砖引玉的作用。

1. 因子16:5日内最高值与量之间的相关系数的倒数

其代码如下：

```
def alpha16(symbol,now):

data = history_n(symbol=symbol,frequency="1d",end_time=now,count=5
,fields="close,volume",df=True)
    vloume = data["volume"].values
    close = data["close"].values
    corr = np.corrcoef(vloume,close)
    return -corr
```

其中的data是获取数据后生成的一个数据表，volume和close是其中对应的数据。在获取数据后，调用NumPy计算两个数据列对应的相关系数并返回相关系数的倒数。

2. 因子53: $((close - low) - (high - close)) / ((close - low) \text{ 的9日之前的值} - \text{当前值})$

其代码如下：

```
def alpha53(symbol,now):
    last_day = get_previous_trading_date("SHSE",now)
    data = history_n(symbol=symbol, frequency="1d", end_time=last_day, count=9,
fields="high,low,close", df=True)
    high = data["high"].values
    low = data["low"].values
    close = data["close"].values
    result = (2 * close[-1] - high[-1] - low[-1])/(high[-1] - low[-1])
    return result
```

这里需要注意的是，对于需要计算的序列，其中的日期是至关重要的，此因子的计算是采用昨天的收盘价与9日前的价差进行比较，因此使用的日期为上一个交易日为宜。

顺便说一下，在这里可能是读者第一次接触到未来函数这个概念，因为在回测时一般是默认计算上一个交易日的买卖点，因此建议读者在使用时使用last_day。这样能够较好地防止有未来函数参与回测中。

3. 因子78：对股票过去5天的成交量和最高价进行排名，再取相关系数

其代码如下：

```
from scipy.stats import rankdata

def alpha53(symbol,now):
    last_day = get_previous_trading_date("SHSE", now)
    data = history_n(symbol=symbol, frequency="1d", end_time=last_day, count=5,
fields="volume,high", df=True)
    high = data["high"].values
    volume = data["low"].values
    high_rank = rankdata(high)
    volume_rank = rankdata(volume)
    corr = np.corrcoef(high_rank,volume_rank)
    return corr
```

这里需要说明的是，在scipy工具包中的rankdata函数用以对数据按从大到小的顺序进行排序，之后再对排序后的数据计算其相关系数。

除此之外还有不少技术因子，见表7-1。

表7-1 其他技术因子

流动性	STOM	月度平均换手率：最近一个月的交易量/流通股数
	STOQ	季度平均换手率：最近一季度的交易量/流通股数
	STOS	半年平均换手率：最近半年的交易量/流通股数
	STOA	年度平均换手率：最近一年的交易量/流通股数
	STOM_barra	Barra 因子：公式为 $\ln\left(\sum_{i=1}^T \frac{V_i}{S_t}\right)$ ， V_i 为 t 日成交金额， S_t 为 t 日流动市值
	STOQ_barra	Barra 因子：公式为 $\ln[\frac{1}{T} \sum_{t=1}^T \exp(STOM_t)]$ ， $T=63$ 个交易日
	STOA_barra	Barra 因子：公式为 $\ln[\frac{1}{T} \sum_{t=1}^T \exp(STOM_t)]$ ， $T=244$ 个个交易日
	Ins	机构持股比例：机构持股变动/总股本
	Ins_c	机构持股比例变动
	Size	总市值
	non-linear-size	Barra 因子：中等市值，将总市值的对数与总市值立方的对数回归得到残差，再对残差做标准化处理
	Market cap	流通市值
	MSM	一个月换手率变动：最近 1 个月换手率/最近 1 年换手率
	MSQ	季度换手率变动：最近 3 个月换手率/最近 1 年换手率
	MSS	半年换手率变动：最近 6 个月换手率/最近 1 年换手率
动量	RSTR_barra	Barra 因子： $\sum_{t=L}^{T+L} w_t [\ln(1+r_t)]$ ， $L=21$ ， $T=500$ ，半衰期 120 日
	RSTR_m24	$\sum_{t=L}^{T+L} w_t [\ln(1+r_t)]$ ， $L=1$ ， $T=240$ ，半衰期 120 日
	RSTR_m12	$\sum_{t=L}^{T+L} w_t [\ln(1+r_t)]$ ， $L=1$ ， $T=240$ ，半衰期 60 日
	RSTR_m6	$\sum_{t=L}^{T+L} w_t [\ln(1+r_t)]$ ， $L=1$ ， $T=120$ ，半衰期 30 日
	RSTR_m3	$\sum_{t=L}^{T+L} w_t [\ln(1+r_t)]$ ， $L=1$ ， $T=60$ ，半衰期 15 日
	RSTR_m1	$\sum_{t=L}^{T+L} w_t [\ln(1+r_t)]$ ， $L=1$ ， $T=20$ ，半衰期 5 日
	RS_1	最新收盘价/21 个交易日前收盘价
	RS_3	最新收盘价/63 个交易日前收盘价
	RS_6	最新收盘价/126 个交易日前收盘价
	RS_12	最新收盘价/252 个交易日前收盘价
	Alpha	Alpha 系数，个股收益率序列与沪深 300 指数收益率序列以半衰期指数加权，得到 Alpha 系数，半衰期为 60 日

波动率	DASTD	Barra 因子，年度平均波动率，累计日波动率以半衰期指数加权，半衰期为 40 日
	CMRA	Barra 因子，年度收益率波动
	HSIGMA	Barra 因子，sigma，个股收益率序列与沪深 300 指数收益率序列以半衰期指数加权，得到残差，对残差求标准差得到 sigma，半衰期为 60 日
	Beta	Barra 因子，贝塔系数，个股收益率序列与沪深 300 指数收益率序列以半衰期指数加权，得 Beta 系数，半衰期为 60 日
	Yieldvol_1	月度日收益率波动率；一个月日收益率标准差
	Yieldvol_3	季度日收益率波动率；三个月日收益率标准差
	Yieldvol_6	半年日收益率波动率；半年日收益率标准差
	High_low_1	月度股价波动；最高价/最低价（最近一个月内价格）
	High_low_3	季度股价波动；最高价/最低价（最近三个月内价格）
	High_low_6	半年股价波动；最高价/最低价（最近六个月内价格）
	High_low_12	全年股价波动；最高价/最低价（最近十二个月内价格）
	VOL_1	成交量月度波动率；1 月波动率标准差
	VOL_3	成交量季度波动率；3 月波动率标准差
	VOL_6	成交量半年波动率；6 月波动率标准差
	VOL_12	成交量年度波动率；12 月波动率标准差

7.1.2 基于Talib的技术因子重写

Talib库有很多现成的工具和计算工具，但是某些技术指标的确定和国内还是有差别的，例如MACD和RSI等函数。在这里按国内的指标重写了部分函数，方便读者在回测时使用。

1. SMA简单移动平均

算术移动平均线是简单而普遍的移动平均线。平均线是指算术平均数，计算方法为一组数字相加，除以该组数据的组成个数，其中每一个给定时限在计算平均值时的权重均相等。

```
def SMA_CN(close, timeperiod) :
    return reduce(lambda x, y: ((timeperiod - 1) * x + y) / timeperiod, close)
```

2. RSI指标

RSI是根据一定时期内上涨点数和涨跌点数之和的比率制作出的一种技术曲线，能够反映出市场在一定时期内的景气程度。RSI由威尔斯·威尔德（Welles Wilder）最早应用于期货买卖，后来人们发现在众多的图表技术分析中，强弱指标的理论 and 实践极其适合股票市场的短线投资，于是被用于股票升跌的测量和分析中。

该分析指标的设计是以三条线来反映价格走势的强弱的，这种图形可以为投资者提供操作依据，非常适合做短线差价操作。

```
def RSI_CN(close, timeperiod):
    diff = map(lambda x, y: x - y, close[1:], close[:-1])
    diffGt0 = map(lambda x: 0 if x < 0 else x, diff)
    diffABS = map(lambda x: abs(x), diff)
    diff = np.array(diff)
    diffGt0 = np.array(diffGt0)
    diffABS = np.array(diffABS)
    diff = np.append(diff[0], diff)
    diffGt0 = np.append(diffGt0[0], diffGt0)
    diffABS = np.append(diffABS[0], diffABS)
    rsi = map(lambda x: SMA_CN(diffGt0[:x], timeperiod) / SMA_CN(diffABS[:x],
timeperiod) * 100,
              range(1, len(diffGt0) + 1))

    return np.array(rsi)
```

3. KDJ指标

KDJ指标又叫随机指标，是一种相当新颖、实用的技术分析指标，它起先用于期货市场的分析，后被广泛用于股市的中短期趋势分析，是期货和股票市场上最常用的技术分析工具。

随机指标KDJ一般用于股票分析的统计体系，根据统计学原理，通过一个特定的周期（常为9日、9周等）内出现过的最高价、最低价及最后一个计算周期的收盘价及这三者之间的比例关系，来计算最后一

个计算周期的未成熟随机值RSV，然后根据平滑移动平均线的方法来计算K值、D值与J值，并绘成曲线图来研判股票走势。

```
def KDJ_CN(high, low, close, fastk_period, slowk_period, fastd_period):
    kValue, dValue = talib.STOCHF(high, low, close, fastk_period,
    fastd_period=1, fastd_matype=0)

    kValue = np.array(map(lambda x: SMA_CN(kValue[:x], slowk_period), range(1,
    len(kValue) + 1)))
    dValue = np.array(map(lambda x: SMA_CN(kValue[:x], fastd_period), range(1,
    len(kValue) + 1)))

    jValue = 3 * kValue - 2 * dValue

    func = lambda arr: np.array([0 if x < 0 else (100 if x > 100 else x) for x
    in arr])

    kValue = func(kValue)
    dValue = func(dValue)
    jValue = func(jValue)

    return kValue, dValue, jValue
```

4. MACD指标

MACD称为指数平滑移动平均线，是从双指数移动平均线发展而来的，由快的指数移动平均线（EMA12）减去慢的指数移动平均线（EMA26）得到快线DIF，再用“ $2 \times (\text{快线DIF} - \text{慢线DIF})$ 的9日加权移动均线DEA”得到MACD柱。

MACD的意义和双移动平均线基本相同，即由快、慢均线的离散、聚合表征当前的多空状态和股价可能的发展变化趋势，但阅读起来更方便。当MACD从负数转向正数时，是买的信号。当MACD从正数转向负数时，是卖的信号。

当MACD以大角度变化，表示快的移动平均线和慢的移动平均线的差距非常迅速地拉开，代表一个市场大趋势的转变。

```
def MACD_CN(close, fastperiod = 12, slowperiod = 26, signalperiod = 9):  
    macdDIFF, macdDEA, macd = talib.MACDEXT(close, fastperiod=fastperiod,  
fastmatype=1,slowperiod=slowperiod, slowmatype=1, signalperiod=signalperiod,  
signalmatype=1)  
    macd = macd * 2  
    return macdDIFF, macdDEA, macd
```

下面使用MACD指标对股票进行买卖，代码如下：

【程序7-1】

```

import numpy as np
from gm.api import *
import os
from tools import Talib_CN_基于Talib实现国内行情软件一致 as talib_cn

def init(context):
    schedule(schedule_func=algo, date_rule='1d', time_rule='09:31:00')

    context.index = "SHSE.000300"

def algo(context):
    now = context.now
    last_day = get_previous_trading_date("SHSE", now)

    data =
history_n(context.index, frequency="1d", count=30, end_time=last_day, fields="close", df=True)
    close = data["close"].values
    macdDIFF, macdDEA, macd = talib_cn.MACD_CN(close)

    if macd[-1] > 0 and macd[-2] < 0:
        order_target_percent(symbol=context.index, percent=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
    elif macd[-1] < 0 and macd[-2] > 0:
        order_target_percent(symbol=context.index, percent=0,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)

def on_backtest_finished(context, indicator):
    print(indicator)
if __name__ == "__main__":
    run(
        strategy_id='9f53e15f-6870-11e8-801c-4cedfb681747',
        filename=(os.path.basename(__file__)),
        mode=MODE_BACKTEST,
        token='715f84a04a4574e2e77491d62f7c8bbc0c663b12',
        backtest_start_time='2016-01-05 09:30:00',
        backtest_end_time='2018-06-23 15:00:00',
        backtest_initial_cash=100000000,
        backtest_adjust=ADJUST_PREV
    )

```

这里的核心程序是使用MACD对收盘价进行判定，当MACD线上穿0线时买入，当MACD下穿0线时卖出，而采用的股票对象是沪深300指数，


```
def get_symbol_list(index,now):

    try:
        symbol_list = get_history_constituents(index=index,
start_date=now)[0].get("constituents").keys()
        symbol_list_not_suspended = get_history_instruments(symbols=symbol_list,
start_date=now, end_date=now)
        symbol_list = [item['symbol'] for item in symbol_list_not_suspended if
not item['is_suspended']]

        _symbol_list = symbol_list
        symbol_list = []
        for _ in _symbol_list:
            symbol_list.append(_)
    except:print(index)
    return symbol_list
```

首先使用get_history_constituents函数获取指数的历史成分股，之后判断当时没有停牌的股票。而symbol_list是作为存放的股票列表，重新通过一个检索将数据读出到symbol_list中并返回。

当然，读者也可以自行建立待测股票池。

2. 选股买入策略的编写

下面对选股策略进行编写。在这里的核心是对过去14天收盘价标准差小于0.05的股票进行个股买入。

其代码如下：

```
data =
history_n(symbol,frequency="1d",fields="close",count=14,end_time=last_day)
close = tools.get_data_value(data,"close")
std = np.std(close)
```

这里需要判定标准差小于0.05。

3. 整体策略的编写

下面对整体策略进行编写，除了买入策略外，还有卖出策略，在这里需要不停地对股票进行判断，检测当时的股票价格是否符合卖出条件。

本策略中，卖出条件为收益30%或者损失7%，代码如下：

```
positions = context.account().positions()
for position in positions:
    symbol = position['symbol']
    current_price = current(symbol)[0]["price"]
    if (current_price > context.symbol_buy_pirce[symbol] * 1.3) or
(current_price < context.symbol_buy_pirce[symbol] * 0.97):
        order_target_percent(symbol=symbol, percent=0,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
```

最终整体代码如下：

【程序7-2】

[illegible]

```

last_day = get_previous_trading_date("SHSE",now)
day_time ,hour_and_mins = str(now).split(" ")

symbol_list = []
if hour_and_mins == "09:31:00":
    _symbol_list = tools.get_symbol_list(context.index,now)

    for symbol in _symbol_list:
        data =
history_n(symbol,frequency="1d",fields="close",count=14,end_time=last_day)
        close = tools.get_data_value(data,"close")
        std = np.std(close)
        if std < context.threshold:
            symbol_list.append(symbol)
    print("-----",now," ",symbol_list,"-----")

    for symbol in symbol_list:
        volume = history_n(symbol=symbol, frequency="60s", count=2,
fields='volume', fill_missing='Last',
            end_time=now, df=True)['volume'].values
        if volume[1]/volume[0] >= 5:
            current_price = current(symbol)[0]["price"]
            last_price = history_n(symbol=symbol, frequency="1d", count=1,
fields='close', fill_missing='Last',
                end_time=now, df=True)['close'].values
            if np.log(current_price/last_price) > 0:
                order_target_percent(symbol=symbol, percent=1. / (context.num),
order_type=OrderType_Market,
                    position_side=PositionSide_Long)
                context.symbol_buy_pirce[symbol] = current_price

if __name__ == "__main__":
    run(
        strategy_id='9f53e15f-6870-11e8-801c-4cedfb681747',
        filename=(os.path.basename(__file__)),
        mode=MODE_BACKTEST,
        token='715f84a04a4574e2e77491d62f7c8bbc0c663b12',
        backtest_start_time="2018-01-01 09:30:00",
        backtest_end_time='2018-01-15 15:00:00',
        backtest_initial_cash=10000000,
        backtest_adjust=ADJUST_PREV
    )

```


在这里需要注意的是，本策略使用的是按分钟回测的on_bar的结果，这样做测试的时间会非常长。

最终结果如图7-4所示。

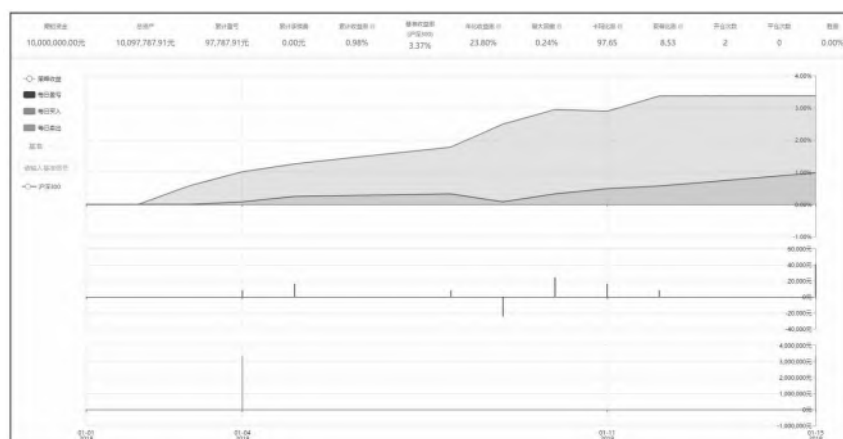


图7-4 按分钟回测的数据结果

7.2 较为复杂的技术因子

除了传统的技术因子外，随着计算能力的增强，还有更多的因子被人为地发现和产生，下面将介绍一些较为复杂的因子。

说明

本节略微有一些难度，读者可以先跳过本节从7.3节开始学起，相对来说，7.3节因子的常用性和普及程度要高于本节所介绍的，但是常用的因子更容易失效。

7.2.1 阻力支撑相对强度因子介绍

1. 阻力支撑相关概念

阻力位是指指标价格上涨时可能遇到的压力，即交易者认为卖方力量开始反超买方，从而价格难以继续上涨或从此回调下跌的价位；支撑位则是交易者认为买方力量开始反超卖方，从而止跌或反弹上涨的价位。

常见的确定阻力支撑位的方法有，布林带上下轨突破策略（突破上轨建仓买入，突破下轨卖出平仓）和均线策略（如超过20日均线建仓买入，低于20日均线卖出平仓）。然而，布林带突破策略在震荡期间出现了持续亏损，均线策略交易成本巨大，且在震荡期间的回撤很大。

2. 阻力支撑相对强度

阻力支撑相对强度（Resistance Support Relative Strength, RSRS）是另一种阻力位与支撑位的运用方式，它不再把阻力位与支撑位当作一个定值，而是看作一个变量，反映了交易者对目前市场状态顶底的一种预期判断。

下面按照不同市场状态分类来说明支撑阻力相对强度的应用逻辑。

（1）市场在上涨牛市中

如果支撑明显强于阻力，牛市持续，价格加速上涨。

如果阻力明显强于支撑，牛市可能即将结束，价格见顶。

（2）市场在震荡中

如果支撑明显强于阻力，牛市可能即将启动。

如果阻力明显强于支撑，熊市可能即将启动。

（3）市场在下跌熊市中

如果支撑明显强于阻力，熊市可能即将结束，价格见底。

如果阻力明显强于支撑，熊市持续，价格加速下跌。

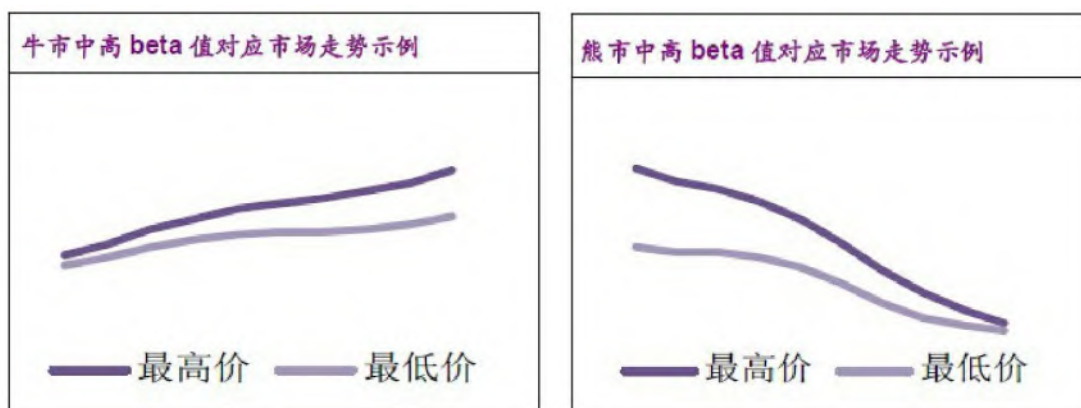
3. 阻力支撑相对强度的计算方法

每日最高价和最低价是一种阻力位与支撑位，它是当日全体市场参与者的交易行为所认可的阻力与支撑。一个很自然的想法是建立最高价和最低价的线性回归，并计算出斜率，即：

$$high = \alpha + \beta \cdot low + \epsilon, \epsilon \sim N(0, \sigma^2)$$

当斜率值很大时，支撑强度大于阻力强度。在牛市中阻力渐小，上方上涨空间大；在熊市中支撑渐强，下跌势头欲止。

当斜率值很小时，阻力强度大于支撑强度。在牛市中阻力渐强，上涨势头渐止；在熊市中支撑渐弱，下方下跌空间渐大，如图7-5所示。



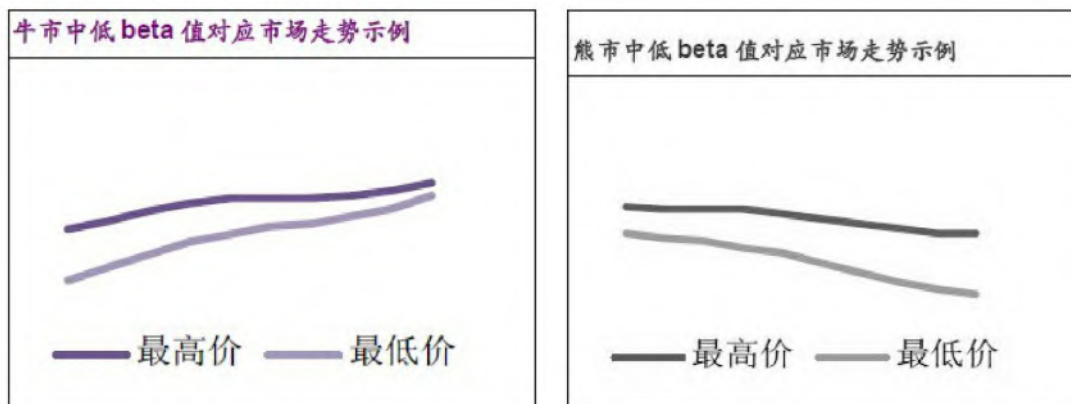


图7-5 不同斜率（Beta值）对应市场走势

RSRS因子的基本计算代码如下：

```

def get_rsrs_weight_not_support(symbol,now,length = 600,window = 18):
    last_day = get_previous_trading_date("SZSE",now)
    last_last_day = get_previous_trading_date("SZSE",last_day)
    data = history_n(symbol=symbol, frequency="1d", count=length + window,
                      end_time=last_last_day,
                      fields="open,high,low,close",
                      fill_missing="last", df=True)
    data_high = (data["high"].values)
    data_low = (data["low"].values)
    #这里是直到前天的 RSRS 序列值
    rsrs_weights = []
    for len_ in range(len(data) - window + 1):
        high_ = data_high[len_:len_ + window]
        low_ = data_low[len_:len_ + window]

        low_ = sm.add_constant(low_)
        model_ = sm.OLS(high_, low_)
        results_ = model_.fit()
        weight_ = (results_.params[1])

        # rsrs_weights.append(weight_) #原始 weight
        rsrs_weights.append(weight_) # 这里是加上权重的 weight

    last_day_data = history_n(symbol=symbol, frequency="1d", count=window,
                               end_time=last_day,
                               fields="open,high,low,close", fill_missing="last",
                               df=True)

    high_ = last_day_data["high"].values
    low_ = last_day_data["low"].values

    low_ = sm.add_constant(low_)
    model_ = sm.OLS(high_, low_)
    results_ = model_.fit()
    weight_ = (results_.params[1])

    z_score = (weight_ - np.mean(rsrs_weights)) / np.std(rsrs_weights)

    return z_score

```

这里的基本思想是计算高值与低值之间的斜率值，即：

- 取前M日的最高价与最低价序列 ($M = 600$)。
- 将两列数据进行OLS回归。
- 取前M日的斜率时间序列，计算当日斜率的标准分。

7.2.2 改进的RSRS因子与回测数据

实际上，在使用RSRS进行因子值计算时，其量化效果很大程度上受拟合本身效果的影响。我们将RSRS标准分与决定系数相乘得到RSRS修正标准分，以此降低绝对值很大但拟合效果很差的RSRS标准分对策略的影响。

RSRS标准分在预测性上的改善效果主要体现于标准分左侧，由于A股更多地使用做多策略，因此左侧预测性改善对择时策略帮助并不大。一个改进的策略是将RSRS修正标准分与RSRS斜率值相乘得到RSRS右偏标准分。

改进后的RSRS计算方法如下：

```

#斜率越大，支撑强弱（RSRS）越强，反之越弱
def get_rsrs_weight_classic(symbol,now,length = 200,window = 18):
    last_day = get_previous_trading_date("SHSE", now)

    ans = []
    ans_rightdev = []

    data = history_n(symbol, frequency="1d", count=length, end_time=last_day,
fields="high,low")

    highs = msci_tools.get_data_value(data, "high")
    lows = msci_tools.get_data_value(data, "low")
    for i in range(len(highs))[window:]:
        data_high = highs[i - window + 1:i + 1]
        data_low = lows[i - window + 1:i + 1]
        X = sm.add_constant(data_low)
        model = sm.OLS(data_high, X)
        results = model.fit()
        ans.append(results.params[1])
        # 计算 r2
        ans_rightdev.append(results.rsquared)

    # 计算标准化的 RSRS 指标
    # 计算均值序列
    section = ans[-length:]
    # 计算均值序列
    mu = np.mean(section)
    # 计算标准化 RSRS 指标序列
    sigma = np.std(section)
    zscore = (section[-1] - mu) / sigma
    # 计算右偏 RSRS 标准分
    zscore_rightdev = zscore * ans[-1] * ans_rightdev[-1]

    return (zscore_rightdev)

```

下面使用右偏RSRS对大盘进行回测择时。

【程序7-3】

```

from gm.api import *
import os

from MSCI_tools import 计算RSRS as RSRS
def init(context):
    schedule(schedule_func=algo, date_rule='1d', time_rule='09:31:00')

    context.index = "SHSE.000300"

def algo(context):
    now = context.now
    last_day = get_previous_trading_date("SHSE", now)

    rsrs_weight = RSRS.get_rsrs_weight_classic(context.index, last_day)

    if rsrs_weight > 0.7:
        order_target_percent(symbol=context.index, percent=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
    elif rsrs_weight < -0.7:
        order_target_percent(symbol=context.index, percent=0,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)

def on_backtest_finished(context, indicator):
    print(indicator)
if __name__ == "__main__":
    run(
        strategy_id='9f53e15f-6870-11e8-801c-4cedfb681747',
        filename=(os.path.basename(__file__)),
        mode=MODE_BACKTEST,
        token='715f84a04a4574e2e77491d62f7c8bbc0c663b12',
        backtest_start_time='2016-01-10 09:30:00',
        backtest_end_time='2018-07-01 15:00:00',
        backtest_initial_cash=10000000,
        backtest_adjust=ADJUST_PREV
    )

```

回测结果如图7-6所示。

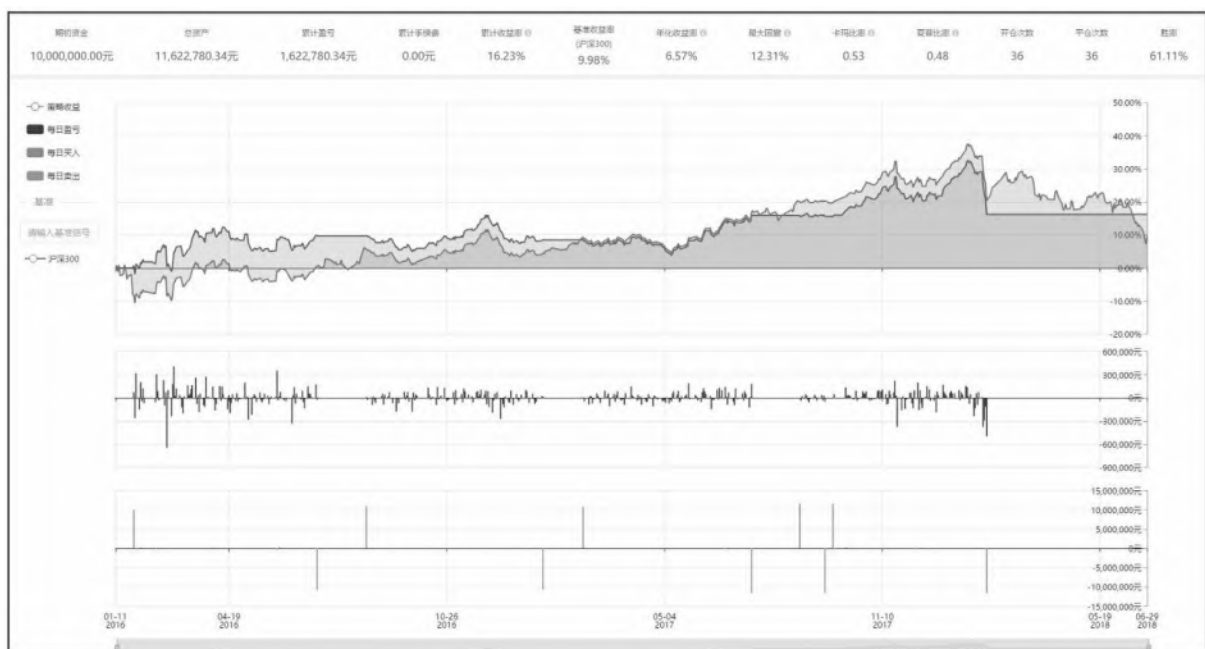


图7-6 RSRS因子回测结果

对于沪深300的择时回测可以看到，在前半段因子走出了一个独立行情，可以较好地针对走势做出调整。顺便说一下，通过回测图可以看到，RSRS的因子在本次测试的前半段能够较好地走出独立行情，能够较好地锁住因子，而在后半段仅仅反映出对应标的的走势，这是一种因子失效的表现。

7.2.3 价差偏离度因子介绍

价差偏离度因子本质上是一个相对意义上的反转因子，价差偏离度低，近期跑输其同类股票，股票相对处于低位，有向上回复的动力，有正的预期超额收益；价差偏离度越高，股票处于相对高位，后期有回调的压力。

价差偏离度和传统的市值因子、估值因子相关性弱，通过因子分层后分组和Fama-Macbeth回归可以发现，传统的1个月反转和3个月反

转可以被价差偏离度替代，而价差偏离度因子信息来源相对独立，不能被其他的常见因子所解释。

因此可以知道，价差偏离度因子的主要作用是捕捉股票相对其同类型（板块）股票的高估低估程度。低估的股票之后会补涨，高估的股票之后有补跌的趋势。

下面介绍价差偏离度因子的使用。

（1）价差偏离度因子指的是股票池内股票标的过去250个交易日的复权收盘价，要求计算涨跌幅及250天涨跌幅的Pearson相关系数，而“股票间的距离”使用的是皮尔逊相关系数，用系数值代替不同的股票之间的距离。

公式如下：

Distance=两股票过去250个交易日涨跌幅的Pearson相关系数
distance=1 - 两股票过去250个交易日涨跌幅的Pearson相关系数

（2）为每一只股票取出距离最近的10只股票distance_min10，等权构建这10只股票的特征组合，计算特征组合的净值价格，作为参考价格（ReferencePrice）。

（3）取出股票池中每只股票的收盘价（closePrice），与参考价格取对数后作差，得到对数价差（PriceSpread）。

公式如下：

$$\begin{aligned} \text{PriceSpread} &= \ln((\text{closePrice}) \\ &\quad - \ln(\text{ReferencePrice})) \end{aligned}$$

（4）对每只股票算过去60个交易日的PriceSpread，并对其标准化得到SpreadBias。公式如下：

$\text{SpreadBias} = \text{PriceSpread} - \text{mean}(\text{PriceSpread}) \text{std}(\text{PriceSpread})$
 $\text{SpreadBias} = \text{PriceSpread} - \text{mean}(\text{PriceSpread}) \text{std}(\text{PriceSpread})$

股票池中距离的获取:

```
"" , num = 10):

    start_date = ago.get_previous_date(now, 250)
    last_day = get_previous_trading_date("SZSE", now)

    df = pd.DataFrame([])
    _symbol_list = []
    for symbol in symbol_list:
        try:
            data = history(symbol=symbol, frequency=frequency,
start_time=start_date, end_time=last_day, fields="close",
                            fill_missing="last", adjust=ADJUST_PREV, df=True)
            if len(data) == count:
                _symbol_list.append(symbol)
                close = (data["close"].values)
                close = np.diff(close)/close[:-1]
                data_close = pd.DataFrame({symbol: close})
                if len(df) == 0:
                    df = (data_close)
                else:
                    df = pd.concat([df, data_close], axis=1)
        except:
            pass
    symbol_list = _symbol_list

    corr = df.corr()
    symbol_and_corr_list={}
    for _ in range(len(symbol_list)):
        line_1 = (corr.ix[:,_])

        line_1 = line_1.sort_values(ascending=False) #由高到低排序

        res = (line_1.index)[0:num]
        symbol_and_corr_list[symbol_list[_]] = res
    return symbol_and_corr_list
```

对于因子值的最终计算：

```
def get_SpreadBias(symbol_and_corr_list,now):

    last_day = get_previous_trading_date("SZSE",now)
    symbol_and_ReferencePrice = {}

    symbol_and_PriceSpread = {}

    for symbol in symbol_and_corr_list:
        symbol_list = symbol_and_corr_list[symbol]

        PriceSpread_arr = []
        for n in range(60):
            target_day = ago.get_previous_date(last_day,n)
            close_arr = []
            for _ in symbol_list:
                data = history(symbol=_, frequency="1d", start_time=target_day,
                                end_time=target_day, fields="close",
                                fill_missing="last", adjust=ADJUST_PREV, df=True)
                close_arr.append(data["close"].values)
            data = history_n(symbol=symbol, frequency="1d", count=1,
                                end_time=target_day, fields="close",
                                fill_missing="last", adjust=ADJUST_PREV, df=True)
            close = data["close"].values
            PriceSpread = np.log(close[-1]) - np.log(np.mean(close_arr))
            PriceSpread_arr.append(PriceSpread)
        symbol_and_PriceSpread[symbol] = (PriceSpread_arr[-1] -
            np.mean(PriceSpread_arr))/np.std(PriceSpread_arr)
    return symbol_and_PriceSpread
```

此因子计算需要大量的时间，因此建议读者自行斟酌是否继续完成。

根据价差偏离度的定义可以知道，该因子实际上度量了个股相对其相似的股票集价格的偏离程度，是一个相对意义上的反转因子。价差偏离度低，说明股票近期跑输其特征组合，相对估值较低，有向上回复的动力，有正的预期超额收益；反之，价差偏离度较高，则说明该股票相对估值较高，后期有回调的压力。这里要注意的是，该因子

有效的前提是股票基本面没有发生重大变化，若近期内发生重大变化使得价格发生改变，则价差偏离度因子很有可能会失效。

7.3 简单的技术性因子——波动率因子

读者看完了7.2节中部分因子的计算与写法，可能对相关内容比较难以理解，其实大多数的内容也是仅仅涉及了回归分析，这应该是大学本科一二年级的课程。

本节将继续介绍一个最常用的技术性因子——波动率因子，难度会有所降低，但是会更加具有普遍性和适用性。

7.3.1 波动率因子介绍

“低波动率影响现象”，从字面上解释为低风险和低波动率的股票有更高的风险调整后的回报率。

然而这一理论与传统金融学的思想相悖。传统金融学认为，风险和收益之间存在权衡，高风险往往带来高收益，低风险则对应低收益，因此在传统金融学背景下，低风险低波动率的股票也能带来更高的回报率这一说法似乎难以成立。

在论文《The Volatility Effect: Lower Risk without Lower Return》中，作者给出了对于这一现象的几种解释：

- 低波动的资产投资组合需要杠杆来匹配市场风险，进而充分利用低风险股票所特有的绝对回报，但是现在对于杠杆存在制度限制。

- 低效投资过程。管理者通常对高波动的股票支付过高，以期待在牛市中得到更高回报。
- 私人投资者的行为偏差。

而在实际中，波动率因子又根据波动的相对条件分为“自波动”与“相对波动（Beta）”值，这是波动率计算的两个最基础的方面。其主要区别是，“自波动”主要计算标的自身的时间序列的波动性，而“相对波动”主要是计算标的的时间序列与相对的时间序列之间的波动性。

其代码实现如下：

1. 对数收益率自波动的计算方法

自波动的计算方法是根据每个时间段的收益率的标准差进行计算，代码如下：

```
def get_volatility_wind(symbol, now, count):  
  
    last_day = get_previous_trading_date("SHSE", now)  
    data = history_n(symbol=symbol, frequency="1d", count=count,  
  
end_time=last_day, fields="high, low, close", fill_missing="last")  
  
    close = msci_tools.get_data_value(data, "close")  
    close = pd.DataFrame(close)  
    stocks_change = close.apply(lambda x: np.log(x) - np.log(x.shift(1)))  
    # 计算 30, 60, 90 日波动率, 年化之  
    daily_vol = stocks_change.std()  
    annual_vol = daily_vol * 252 ** 0.5  
    return annual_vol.values[0]
```

其主要思想是计算收益率的对数时对其进行年化处理。
get_data_value函数用于从数据集中提取收盘价时间序列，代码如下：

```
#这段代码用于获取返回的 data 数据中的对应因子的 list
def get_data_value(data, factor):
    result = []
    for _ in data:
        result.append(_[factor])
    return result
```

2. 简单自波动的计算方法

还有一种较为常用的自波动的计算方法是简单波动率算法。其来源于自身的相关系数与变异系数的计算值与结果，代码如下：

```
def get_volatility_normal(symbol, now, count):

    last_day = get_previous_trading_date("SHSE", now)
    data = history_n(symbol=symbol, frequency="1d", count=count,

end_time=last_day, fields="high, low, close", fill_missing="last")
    close = msci_tools.get_data_value(data, "close")

    res = np.var(close) / np.mean(close)
    res = np.sqrt(res)
    return res
```

这两种都是计算特定标的的时间序列的方法，而对于其他标的（一般是上证指数或者沪深300等指数）的波动性计算，由下列方法完成。

3. 使用收盘价进行的Beta值计算方法

使用不同标的的收盘价作为计算的时间序列是最简单和最常用的方法，代码如下：

```

import statsmodels.api as sm

def get_beta_weight_1_not_support(symbol,now,count,market_index =
"SHSE.000001"):
    """
    一般用单个股票资产的历史收益率对同期指数（大盘）收益率进行回归，回归系数就是 Beta 系数。
    """
    last_day = get_previous_trading_date("SHSE",now)
    market_data = history_n(symbol=market_index, frequency="1d", count=count,
                             end_time=last_day,fields="high,low,close",fill_missing="last")

    market_close = msci_tools.get_data_value(market_data,"close")

    symbol_data = history_n(symbol=symbol, frequency="1d", count=count,
                             end_time=last_day,fields="high,low,close",fill_missing="last")

    symbol_close = msci_tools.get_data_value(symbol_data,"close")

    market_close = sm.add_constant(market_close)
    model_ = sm.OLS(symbol_close, market_close)
    results_ = model_.fit()
    weight_ = (results_.params[1])
    return weight_

```

这里使用上证指数作为对应的标的，在实际中可以根据需要修改成沪深300或者创业板等特定标的。

注意

这里使用了statsmodels工具包，因此在使用之前需要先导入对应的包。

4. 使用收益率的Beta值计算方法

除了绝对收盘价之间值的计算外，还有一种使用收益率作为Beta值的计算方法，代码如下：


```

def get_beta_weight_2(symbol,now,count,market_index = "SHSE.000001"):
    """
    一般用单个股票资产的历史收益率对同期指数（大盘）收益率进行回归，回归系数就是 Beta 系数。
    """
    last_day = get_previous_trading_date("SHSE",now)
    market_data = history_n(symbol=market_index, frequency="1d", count=count,
                             end_time=last_day,
                             fields="high,low,close",
                             fill_missing="last")

    market_close = msci_tools.get_data_value(market_data,"close")

    market_close_ratio = []
    for i in range(1,len(market_close)):
        ratio = (market_close[i] - market_close[i-1])/market_close[i-1]
        ratio = (0.99 ** (len(market_close) - i)) * ratio
        market_close_ratio.append(ratio)

    symbol_data = history_n(symbol=symbol, frequency="1d", count=count,
                             end_time=last_day,
                             fields="high,low,close",
                             fill_missing="last")

    symbol_close = msci_tools.get_data_value(symbol_data,"close")

    symbol_close_ratio = []
    for i in range(1,len(symbol_close)):
        ratio = (symbol_close[i] - symbol_close[i-1])/symbol_close[i-1]
        ratio = (0.99 ** (len(market_close) - i)) * ratio
        symbol_close_ratio.append(ratio)

    market_close_ratio = sm.add_constant(market_close_ratio)
    model_ = sm.OLS(symbol_close_ratio, market_close_ratio)
    results_ = model_.fit()
    weight_ = (results_.params[1])
    return weight_

```

以上4种是较为常见的波动率的计算方法，除此之外，还有更多的波动率的计算方法，下一节进行介绍。

7.3.2 更多的波动率因子

除了7.3.1小节介绍的波动率因子外，还有更多的波动率因子可以使用，作者总结了几个有效因子供读者学习。需要注意的是，这里的price_data是生成的对应的pandas格式，按需要包括最高价、最低价以及收盘价。

(1) Parkinson (1980) 估计量采用交易时段最高价和最低价两个价格数据，利用极差进行估计。该估计量使价格波动区间在一定假设下比基于收盘价的估计量更能有效地估计回报波动率。

```
def Parkinson_get_estimator(price_data, window=30, trading_periods=252,
                             clean=True):

    rs = (1.0 / (4.0 * math.log(2.0))) * ((price_data['High'] /
price_data['Low']).apply(np.log)) ** 2.0

    def f(v):
        return trading_periods * v.mean() ** 0.5

    result = rs.rolling(
        window=window,
        center=False
    ).apply(func=f)

    if clean:
        return result.dropna()
    else:
        return result
```

(2) Garman-Klass (1980) 利用交易时段最高价、最低价和收盘价三个价格数据进行估计，该估计量通过将估计量除以调整因子来纠正存在的偏差，以便得到方差的无偏估计。

```

def GarmanKlass_get_estimator(price_data, window=30, trading_periods=252,
clean=True):

    log_hl = (price_data['High'] / price_data['Low']).apply(np.log)
    log_cc = (price_data['Close'] / price_data['Close'].shift(1)).apply(np.log)
    rs = 0.5 * log_hl ** 2 - (2 * math.log(2) - 1) * log_cc ** 2

    def f(v):
        return (trading_periods * v.mean()) ** 0.5

    result = rs.rolling(window=window, center=False).apply(func=f)
    if clean:
        return result.dropna()
    else:
        return result

```

(3) Parkinson和Garman-Klass估计量之所以能提高估计效率，是因为它们依赖于一些不适用于真实市场的假设，尤其是价格服从不带漂移项的几何布朗运动以及交易是连续的假设。Rogers-Satchell在一定程度上放宽了这些限制条件，引入了带有漂移项的更优的估计量。

```

def RogersSatchell_get_estimator(price_data, window=30, trading_periods=252,
clean=True):

    log_ho = (price_data['High'] / price_data['Open']).apply(np.log)
    log_lo = (price_data['Low'] / price_data['Open']).apply(np.log)
    log_co = (price_data['Close'] / price_data['Open']).apply(np.log)

    rs = log_ho * (log_ho - log_co) + log_lo * (log_lo - log_co)

    def f(v):
        return trading_periods * v.mean() ** 0.5

    result = rs.rolling(
        window=window,
        center=False
    ).apply(func=f)

    if clean:
        return result.dropna()
    else:
        return result

```

(4) Garman-Klass (1980) 估计量无法解决价格序列中存在跳空开盘的情况。Yang-Zhang (2000) 推导出了适用于价格跳空开盘的估计量，本质上是各种估计量的加权平均。

```

def YangZhang_get_estimator(price_data, window=30, trading_periods=252,
clean=True):

    log_ho = (price_data['High'] / price_data['Open']).apply(np.log)
    log_lo = (price_data['Low'] / price_data['Open']).apply(np.log)
    log_co = (price_data['Close'] / price_data['Open']).apply(np.log)

    log_oc = (price_data['Open'] / price_data['Close'].shift(1)).apply(np.log)
    log_oc_sq = log_oc ** 2

    log_cc = (price_data['Close'] / price_data['Close'].shift(1)).apply(np.log)
    log_cc_sq = log_cc ** 2

    rs = log_ho * (log_ho - log_co) + log_lo * (log_lo - log_co)

    close_vol = log_cc_sq.rolling(
        window=window,
        center=False
    ).sum() * (1.0 / (window - 1.0))
    open_vol = log_oc_sq.rolling(
        window=window,
        center=False
    ).sum() * (1.0 / (window - 1.0))
    window_rs = rs.rolling(
        window=window,
        center=False
    ).sum() * (1.0 / (window - 1.0))

    k = 0.34 / (1 + (window + 1) / (window - 1))
    result = (open_vol + k * close_vol + (1 - k) * window_rs).apply(np.sqrt) *
math.sqrt(trading_periods)

    if clean:
        return result.dropna()
    else:
        return result

```

(5) 使用偏度Skew代替波动率。

```
def Skew_get_estimator(price_data, window=30, clean=True):
    log_return = (price_data['Close'] /
price_data['Close'].shift(1)).apply(np.log)

    result = log_return.rolling(
        window=window,
        center=False
    ).skew()

    if clean:
        return result.dropna()
    else:
        return result
```

(6) 使用峰度Kurtosis代替波动率。

```
def Kurtosis_get_estimator(price_data, window=30, clean=True):
    "峰度 Kurtosis 代替波动率"
    log_return = (price_data['Close'] /
price_data['Close'].shift(1)).apply(np.log)

    result = log_return.rolling(
        window=window,
        center=False
    ).kurt()

    if clean:
        return result.dropna()
    else:
        return result
```

7.4 实战：一个回测成功率100%的中长线买卖例子

对于股票投资，特别是中长线投资，成功率是非常重要的，任何人都不希望经过长时间的投资等待，原先的一笔投资产生亏损。下面

给读者分享一个回测中成功率能够达到100%的选股策略，主要依据的是技术选股，如图7-7所示。

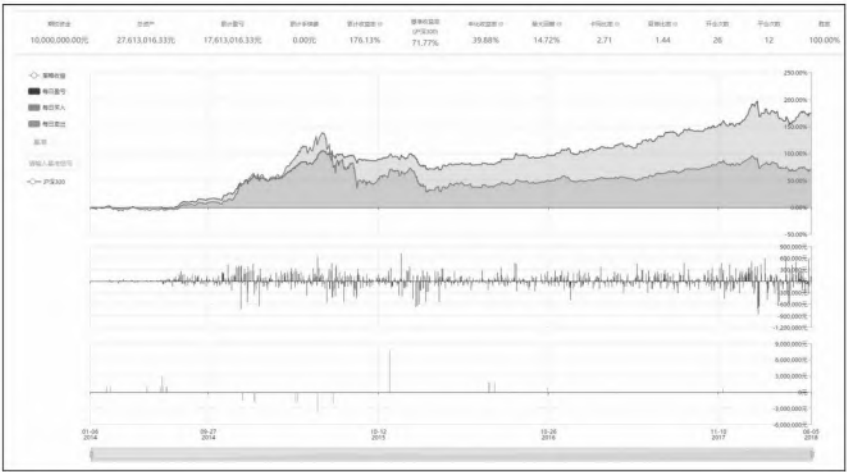


图7-7 回测成功率100%的选股策略

7.4.1 技术指标的设计

传统的选股方式主要依靠技术指标，这也是长期以来广大股民从市场中获取收益的方式。技术指标更多的是涉及对收盘价以及交易量的波动性判断。

首先是买入点的指标，代码如下：

（1）定义60天的波动率小于10%：

```

def buy_check_price_amplitude(context, symbol, count = 60, threshold = 0.1):

    now = context.now
    last_day = get_previous_trading_date("SHSE", now)

    #amplitude = beta_and_vol.get_volatility_normal(symbol,last_day,count=count)

    data =
history_n(symbol,frequency="1d",count=count,end_time=last_day,fields="open,high,low,close")
    close = msci_tools.get_data_value(data,"close")
    open = msci_tools.get_data_value(data, "open")
    high = msci_tools.get_data_value(data, "high")
    low = msci_tools.get_data_value(data, "low")

    amplitude = (np.max(high) - np.min(low))/close[0]

    if amplitude < threshold:
        return True
    else:
        return False

```

(2) 买入点一地量（连续18周成交量小于最近4年内天量的10%）：

```

def buy_check_mean_volume_small(context,
symbol,week_count_short=18 ,week_count_long = 150, threshold = 0.1):

    now = context.now
    last_day = get_previous_trading_date("SHSE", now)

    data = history_n(symbol, frequency="1d", count=week_count_long * 5,
end_time=last_day, fields="volume")
    volume = msci_tools.get_data_value(data,"volume")
    max_volume_last = np.max(volume[-week_count_short*5:])
    max_volume = np.max(volume)

    if max_volume_last < max_volume * 0.1:
        return True
    else:
        return False

```


(3) 单日成交量大于该股的前五日移动平均成交量的2.5倍，大于前10日移动平均成交量的3倍：

```
def buy_check_volume_increase(context, symbol):  
  
    now = context.now  
    last_day = get_previous_trading_date("SHSE", now)  
  
    data = history_n(symbol, frequency="1d", count=11, end_time=last_day,  
fields="volume")  
    volume = msci_tools.get_data_value(data, "volume")  
    h = volume  
    volume = h['volume'][-1]  
    volume_mean_5 = np.mean(h['volume'][-6:-1])  
    volume_mean_10 = np.mean(h['volume'][-11:-1])  
    if volume > volume_mean_5 * 2.5 and volume > volume_mean_10 * 3:  
        return True  
    else:  
        return False
```

(4) 找到昨天之前成交量大于昨天的成交量（0.8倍），这个区间的天数大于30天，昨天单日成交量大于该区间的平均成交量的2倍，区间价格波动小于10%：

```

def buy_check_volume_increase_and_price_amplitude(context, symbol):

    now = context.now
    last_day = get_previous_trading_date("SHSE", now)

    data = history_n(symbol, frequency="1d", count=270, end_time=last_day,
fields="open,high,low,close,volume")

    close = msci_tools.get_data_value(data,"close")
    open = msci_tools.get_data_value(data, "open")
    high = msci_tools.get_data_value(data, "high")
    low = msci_tools.get_data_value(data, "low")
    volume = msci_tools.get_data_value(data,"volume")

    volume_yesterday = volume[-1]

    # 昨日收盘价要高于开盘价
    if close[-1] < open[-1]:
        return False

#找到昨天之前成交量大于昨天的成交量（0.8 倍）的那个日期，这个日期到今天的区间的天数大于 30 天
    start = 0
    end = -1
    for i in range(1, len(volume)):
        index = -(i + 1)
        if volume[index] > volume[-1] * 0.8:
            start = index + 1
            break
    if start == 0 or (end - start) < 30:
        return False
    # 昨天单日成交量大于该区间的平均成交量的 2 倍
    volume_mean = np.mean(volume[start:end])
    volume_mean_5 = np.mean(volume[-6:-1])
    if volume_yesterday < volume_mean * 2.5 or volume_yesterday < volume_mean_5
* 2.5:
        return False

    # 区间价格波动小于 10%
    price_max = max(high[start:end])
    price_min = min(low[start:end])
    price_amplitude = (price_max - price_min) / open[start]
    # print 'min=%s, max=%s, open=%s, amplitude=%s' % (price_min, price_max,
h['open'][start], price_amplitude)
    if price_amplitude > 0.1:
        return False

    return True

```

(5) 对于卖出点的指标设定，卖出点的判定代码如下：

```
def sell_check(context, symbol):  
    if sell_check_mean_price(context, symbol, 0.1) and  
sell_check_turnover_ratio(context, symbol):  
        return True  
    if sell_check_mean_price(context, symbol, 0.2):  
        return True  
    return False
```

(6) 卖出点—均线（5日线超过10日线10%，10日线超过30日线10%）：

```

def sell_check_mean_price(context, symbol, threshold = 0.1):

    now = context.now
    last_day = get_previous_trading_date("SHSE", now)

    data = history_n(symbol, frequency="1d", count=31, end_time=last_day,
fields="open,high,low,close,volume")

    close = msci_tools.get_data_value(data, "close")
    open = msci_tools.get_data_value(data, "open")
    high = msci_tools.get_data_value(data, "high")
    low = msci_tools.get_data_value(data, "low")
    volume = msci_tools.get_data_value(data, "volume")

    # 求 5 日线、10 日线、30 日线
    mean_5 = np.mean(close[-5:])
    mean_10 = np.mean(close[-10:])
    mean_30 = np.mean(close[-30:])
    diff_5_10 = (mean_5 - mean_10) / mean_10
    diff_5_30 = (mean_5 - mean_30) / mean_30
    diff_10_30 = (mean_10 - mean_30) / mean_30

    # 求昨天的 5 日线、10 日线、30 日线
    yes_mean_5 = np.mean(close[-6:-1])
    yes_mean_10 = np.mean(close[-11:-1])
    yes_mean_30 = np.mean(close[-31:-1])
    yes_diff_5_10 = (yes_mean_5 - yes_mean_10) / yes_mean_10
    yes_diff_5_30 = (yes_mean_5 - yes_mean_30) / yes_mean_30
    yes_diff_10_30 = (yes_mean_10 - yes_mean_30) / yes_mean_30

    return diff_5_30 > threshold

```

(7) 卖出点—换手率:

```

def sell_check_turnover_ratio(context, symbol):

    now = context.now
    last_day = get_previous_trading_date("SHSE", now)
    #换手率
    _df = get_fundamentals(table='trading_derivative_indicator', symbols=symbol,
start_date=last_day,
                                end_date=last_day, fields='TURNRATE')

    TURNRATE = msci_tools.get_data_value(_df, "TURNRATE")
    return TURNRATE[0]>15

#卖出点 RSRS
from MSCI_tools import 计算RSRS as RSRS
def sell_check_rsrs(context, symbol):
    threshold = -0.7
    now = context.now
    zscore_rightdev = RSRS.get_rsrs_weight_classic(symbol, now)

    if zscore_rightdev < threshold:
        return True
    else:
        return False

```

(8) 庄股值计算（庄股值：能够无量涨停的或无量杀跌的）：

```

def cow_stock_value(context, symbol):
    now = context.now
    _pb = get_fundamentals(table='trading_derivative_indicator', symbols=symbol,
start_date=now,
                                end_date=now, fields='PB,NEGOTIABLEMV')
    pb = msci_tools.get_data_value(_pb, "PB")
    NEGOTIABLEMV = (msci_tools.get_data_value(_pb,
"NEGOTIABLEMV"))[0]/100000000

    if NEGOTIABLEMV > 100:
        return 0
    num_fall = fall_money_day_3line(context, symbol, 120, 20, 60, 160)
    num_cross = money_5_cross_60(context, symbol, 120, 5, 160)
    return (num_fall * num_cross) / (pb * (NEGOTIABLEMV ** 0.5))

```

(9) 3条移动平均线计算缩量：

```

def fall_money_day_3line(context, symbol, n, n1=20, n2=60, n3=120):
    now = context.now
    last_day = get_previous_trading_date("SHSE", now)

    data = history_n(symbol, frequency="1d", count=n+n3, end_time=last_day,
fields="cum_volume")
    stock_m = msci_tools.get_data_value(data, "cum_volume")
    i=0
    count=0
    while i<n:
        money_MA200=np.mean(stock_m[i:n3-1+i])
        money_MA60=np.mean(stock_m[i+n3-n2:n3-1+i])
        money_MA20=np.mean(stock_m[i+n3-n1:n3-1+i])
        if money_MA20<=money_MA60 and money_MA60<=money_MA200:
            count=count+1
            i=i+1
    return count

```

(10) 计算脉冲，成交额5日穿60日：

```

def money_5_cross_60(context, symbol, n, n1 = 5, n2 = 60):
    now = context.now
    last_day = get_previous_trading_date("SHSE", now)

    data = history_n(symbol, frequency="1d", count=n+n2+1, end_time=last_day,
fields="cum_volume")
    stock_m = msci_tools.get_data_value(data, "cum_volume")
    i=0
    count=0
    while i<n:
        money_MA60=np.mean(stock_m[i+1:n2+i])
        money_MA60_before=np.mean(stock_m[i:n2-1+i])
        money_MA5=np.mean(stock_m[i+1+n2-n1:n2+i])
        money_MA5_before=np.mean(stock_m[i+n2-n1:n2-1+i])
        if (money_MA60_before-money_MA5_before)*(money_MA60-money_MA5)<0:
            count=count+1
            i=i+1
    return count

```

7.4.2 回测的设计

下面对回测进行设计，从前面的内容可以得知，这里的买入点和卖出点都有特定的标的，买入标的主要需要满足3个条件：

- 找到昨天之前成交量大于昨天的成交量（0.8倍），这个区间的天数大于30天。
- 昨天单日成交量大于该区间的平均成交量的2倍。
- 区间价格波动小于10%。

而卖出标的更需要满足以下条件之一：

- 5日均线值超过30日均线值的10%且换手率大于15。
- 5日均线值超过30日均线值的20%。

完整代码如下。

【程序7-4】

```
import numpy as np
from gm.api import *
import os
from MSCI_tools import msci_tools
```

```

from 中长线买入卖出点选择 import tools

def init(context):
    schedule(schedule_func=algo, date_rule='1d', time_rule='09:31:00')

    context.num = 10 #最大持股数

def algo(context):

    now = context.now

    sell_symbol_list = []
    buy_symbol_list = []

    #过滤卖出股票
    positions = context.account().positions()
    for position in positions:
        symbol = position['symbol']
        if tools.sell_check(context, symbol):
            order_target_percent(symbol=symbol, percent=0,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
            sell_symbol_list.append(symbol)
            print(now, " ", "卖出: ", symbol)
    symbol_list = msci_tools.get_symbol_list("SHSE.000922", now)

    for symbol in symbol_list:
        try:
            if (symbol not in sell_symbol_list) and
(tools.buy_check(context, symbol)) :
                order_target_percent(symbol=symbol, percent=1./context.num,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
                print(now, " ", "买入: ", symbol)
        except:pass

def on_backtest_finished(context, indicator):
    print(indicator)

if __name__ == "__main__":
    run(
        strategy_id='9f53e15f-6870-11e8-801c-4cedfb681747~',
        filename=(os.path.basename(__file__)),
        mode=MODE_BACKTEST,

```



```
token='715f84a04a4574e2e77491d62f7c8bbc0c663b12',  
backtest_start_time="2014-01-05 09:30:00",  
backtest_end_time='2018-06-05 15:00:00',  
backtest_initial_cash=10000000,  
backtest_adjust=ADJUST_PREV  
)
```

buy_check与sell_check分别是买入点与卖出点的设置，这里只有通过买入点的判定才能够买入，读者需要注意一个技巧，就是加入了一个黑名单的机制，在买入前还需要对黑名单进行判定，这个技巧可以说非常有用。

7.5 小结

基本面因子和技术面因子是相辅相成的，共同组成了一个完整的选股策略。

本章主要从技术面因子入手，做出了多种最常用的波动率因子的变动，主要涉及自波动与相对波动（Beta）的计算，这也是最常用的技术面因子。

在学习完基本面因子和技术面因子的构成方法后，第8章内容将围绕这方面展开。作者将带领读者完成一个关于指数强化基金的配置。

人人都是基金经理，你也可以！

第8章

人人都是基金经理——中证红利指数增强策略

人人都是基金经理。

对于基金经理这个词读者可能非常熟悉，那些掌管几十亿甚至于上百亿资金，在证券市场运筹帷幄、指点江山、挥斥方遒、好不潇洒的股市健儿就是人们印象中的基金经理。

作为一个基金经理，一般要求具有金融相关专业硕士以上教育背景，具备良好的数学基础和扎实的经济学理论功底，如有海外留学经历或获得CFA证书，则将更具竞争力。基金经理的工作是负责决定该基金的组合和投资策略，投资组合是按照基金说明书的投资目标去选择，投资策略由该基金经理决定。

做基金经理难吗？

是的，做一个基金经理很难。

人人都可以成为基金经理吗？

是的，人人都可以成为基金经理。

通过学习相关的内容与投资组合策略，任何人都可以成为一名合格的基金经理。本章将以中证红利为基础介绍通过构建指数增强基金，在这个过程中将全面复习以前学到的内容，综合运用基本面因子和多种技术因子，并且通过最终的回测验证基金组合的收益和成果。

注意

本章只是一个指数增强组合的例子，如果读者将其用于真实的股票组合和操作，还需要进一步细致和专业地对其进行分析和处理。

8.1 中证红利指数基金介绍

本节从理论开始介绍，如果读者已经有金融相关从业经验，那么可以忽略本节。

8.1.1 红利指数基金的由来

红利策略最初叫作“狗股策略”（Dogs of the Dow Theory），是由美国基金经理迈克尔·奥希金斯在1991年提出的。该策略的具体做法是每年年底在道琼斯工业平均指数成分股中找出10只股息率最高的股票，新年买入，一年后按股息率高低更新股票池，如此循环往复。

如果将股息率最高的股票按一定方法编制成指数，该指数就叫作“红利指数”，围绕红利指数进行的一系列投资方式，例如指数定投、指数长持等，都属于“红利策略”的范畴。

据投资百科（Investopedia）统计，狗股策略在1957年～2003年期间的年均回报率为14.3%，高于道指11%的年均回报率。而在1973年～1996年，狗股策略年均回报率达20.3%，高于道指同期15.8%的平均

回报率。数据证明，狗股 / 红利策略长期来看是能够战胜市场平均的。

8.1.2 中证红利简介

中证指数有限公司2005年5月12日宣布，为反映沪深证券市场高股息股票的整体走势，为投资者提供新的业绩基准和投资标的，将于5月26日正式发布中证红利指数。

中证红利指数挑选了上海、深圳交易所中现金股息率高、分红比较稳定、具有一定规模及流动性的100只股票作为样本，以反映A股市场高红利股票的整体状况和走势。基日为2004年12月31日，基点为1000点。中证红利指数简称“中证红利”，指数代码为000922（上海）/399922（深圳）。

目前来说，中证红利指数的主要权重行业集中在钢铁、电力、石化、汽车、航运、煤炭、有色等。此外，对于A股市场而言，有持续分红能力的公司是稀缺资源，只占整个市场的三成左右。分给投资人的真金白银显示了这类公司良好的财务状况与内在价值。

从近5年的走势来看（见图8-1），自2015年6月A股自高点回落进入震荡市以来，中证红利指数长期跑赢大盘。值得一提的是，随着价值的回归，在大盘蓝筹和行业龙头备受市场关注的同时，红利投资策略的优势更为明显。

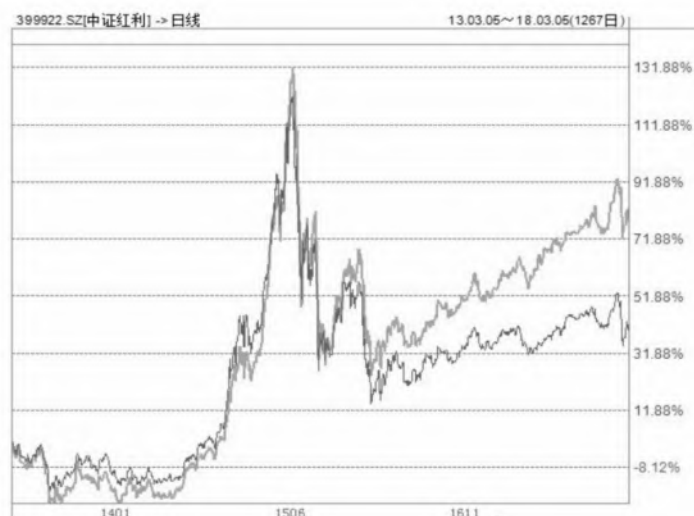


图8-1 中证红利基金走势

从历史数据来看（见图8-2），中证红利指数5年来的市盈率（TTM）均未超过10倍，而A股大盘5年来的平均市盈率（TTM）为14.02倍。相比之下，红利指数的估值仍处在相对较低的位置，投资价值明显。

交易时间	市盈率（TTM）	市净率
2017.12.29	9.57	1.22
2016.12.30	9.06	1.07
2015.12.31	9.17	1.21
2014.12.31	9.36	1.40
2013.12.31	6.39	1.05

图8-2 中证红利市盈率与市净率

中证红利指数的行业权重主要分布在金融地产、可选消费、工业、公共事业等聚集着大量白马蓝筹价值股的行业，如图8-3所示。而同时，A股市场追踪中证红利的市场有多个指数基金，从成立至今也取得了不少成就。

行业权重分布

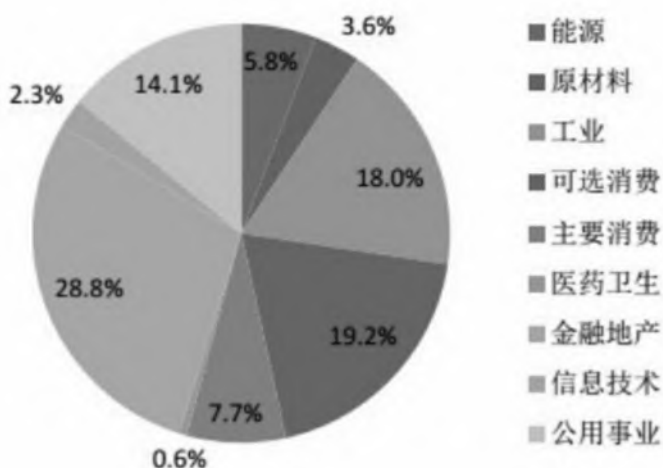


图8-3 中证红利指数行业权重分布

8.2 基于中证红利的指数增强基金策略的构建

对于不同的指数特点和关系的内容，构建基于其内容的指数基金，最为合适的方法就是尽可能地组合出能够反映基金特点的组合策略。

从前文分析来看，中证红利选择沪深两市中现金股息率高、分红稳定、具有一定规模及流动性的100只股票，其特点就是稳定性强、业绩较好，同时具有一定的规模。但是由于要求其稳定性，对其个股的成长性并没有太大要求。

因此，基于分析可以得出，对于中证红利，主要有以下几个方面要进行关注。

- 股息率：股息率（Dividend Yield Ratio）是一年的总派息额与当时市价的比例。以占股票最后销售价格的百分数表示年度股息，该指标是投资收益率的简化形式。股息率是股息与股票价格之间的比率。在投资实践中，股息率是衡量企业是否具有投资价值的重要标尺之一。
- 市净率：市净率指的是每股股价与每股净资产的比率。市净率可用于股票投资分析，一般来说市净率较低的股票，投资价值较高，相反则投资价值较低。但在判断投资价值时，还要考虑当时的市场环境以及公司经营情况、盈利能力等因素。
- 市盈率：市盈率（Price Earnings ratio, 即P/E ratio）也称“本益比”“股价收益比率”或“市价盈利比率”。
- 贝塔值：与大盘的联动波动率。
- 自相关波动率：自身的波动率。

基于以上分析，本节开始构建基于中证红利的指数增强基金，综合运用以上指标，搭建一个能够获取高收益、高稳态、低风险的收益组合。

8.2.1 中证红利策略的构建方法

通过前面的学习可以知道，对于组合的构建最基本的就是找到若干只符合要求的股票，组合成一个股票池，这个股票池反映的是构建者的认知与能力。

基本面因子的获取

基本面因子是中证红利构成因子的基础，在这里主要选用市净率、市盈率以及股息率作为基本的选股要求，代码如下：

```
df = get_fundamentals(table='trading_derivative_indicator',  
symbols=symbol_list,start_date=last_day,end_date=last_day,fields='PETTM,DY,PB',  
df=True)
```

除此之外，还需要添加波动率因子，前面在介绍技术因子时对其也有涉及，这里根据需要采用了两个技术因子：Beta值与自相关波动率，代码如下：


```

#这里使用收益进行计算
def get_beta_weight_2(symbol,now,count,market_index = "SHSE.000001"):
    """
    一般用单个股票资产的历史收益率对同期指数（大盘）收益率进行回归，回归系数就是 Beta 系数。
    """
    last_day = get_previous_trading_date("SHSE",now)
    market_data = history_n(symbol=market_index, frequency="1d", count=count,
                             end_time=last_day,
                             fields="high,low,close",
                             fill_missing="last")

    market_close = msci_tools.get_data_value(market_data,"close")

    market_close_ratio = []
    for i in range(1,len(market_close)):
        ratio = (market_close[i] - market_close[i-1])/market_close[i-1]
        ratio = (0.99 ** (len(market_close) - i)) * ratio
        market_close_ratio.append(ratio)

    symbol_data = history_n(symbol=symbol, frequency="1d", count=count,
                             end_time=last_day,
                             fields="high,low,close",
                             fill_missing="last")

    symbol_close = msci_tools.get_data_value(symbol_data,"close")

    symbol_close_ratio = []
    for i in range(1,len(symbol_close)):
        ratio = (symbol_close[i] - symbol_close[i-1])/symbol_close[i-1]
        ratio = (0.99 ** (len(market_close) - i)) * ratio
        symbol_close_ratio.append(ratio)

    market_close_ratio = sm.add_constant(market_close_ratio)
    model_ = sm.OLS(symbol_close_ratio, market_close_ratio)
    results_ = model_.fit()
    weight_ = (results_.params[1])
    return weight_

```

而Beta值的计算如下：

```
#波动率的简单方法
def get_volatility_normal(symbol,now,count):

    last_day = get_previous_trading_date("SHSE",now)
    data = history_n(symbol=symbol, frequency="1d", count=count,

end_time=last_day,fields="high,low,close",fill_missing="last")
    close = msci_tools.get_data_value(data,"close")

    res = np.var(close)/np.mean(close)
    res = np.sqrt(res)
    return res
```

这里为了方便起见，波动率采用了收益率的计算方法，而Beta值的计算采用了简单的计算方法，即变异系数的平方根。完整代码如下：

```

def bonus_cn(now,index = "SHSE.000922"):

    last_day = get_previous_trading_date("SHSE",now)

    symbol_list = get_history_constituents(index=index,
start_date=last_day)[0].get("constituents").keys()
    symbols_not_suspended = get_history_instruments(symbols=symbol_list,
start_date=last_day, end_date=now)
    symbols_not_suspended = [item['symbol'] for item in symbols_not_suspended if
not item['is_suspended']]

    df = get_fundamentals(table='trading_derivative_indicator',
symbols=symbol_list,start_date=last_day,end_date=last_day,fields='PETTM,DY,PB',
df=True)

    df = df.dropna()
    symbol_list = []
    for _ in df["symbol"].values:
        symbol_list.append(_)

    beta_list = []
    volatility_list = []
    for symbol in symbol_list:
        beta = fun.get_beta_weight_2(symbol,now,count=20)
        beta_list.append(beta)

        volatility = fun.get_volatility_normal(symbol,now,count=60)
        volatility_list.append(volatility)

    df["beta"] = beta_list
    df["volatility"] = volatility_list

    df_factor = df.iloc[:,3:]
    df_factor = np.asmatrix(df_factor)
    weight_mat = np.asmatrix([[1],[-1],[1],[1],[1]])
    res = np.dot(df_factor,weight_mat)
    df["score"] = (res)

    df = (df.sort_values(["score"]))
    symbol_list = []
    for _ in df["symbol"].values:
        symbol_list.append(_)

    return symbol_list

```

这里的代码段采用了模块法，index是中证红利的指数代码，这样做的好处是可以其他指数代码代替当前的中证红利。之后根据设定的技术因子和基本面因子获取对应的数据值，构建出因子矩阵。

对于构建出的因子矩阵，下一步是计算出对应的因子值。目前来说，采用的是等权重的方法，根据对应的因子数据高低对股份的影响是正相关还是负相关，把权重值简单设置成1或-1。

8.2.2 策略回测与优化

由于一些原因，作者不能公开相应的写法，因此这里只放出一张回测图供大家参考，如图8-4所示。

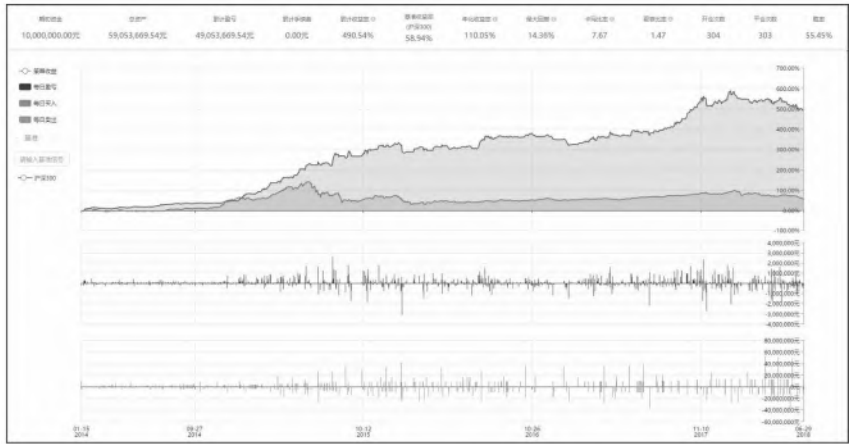


图8-4 中证红利回测结果

提示，对于不同的数据，其权重值并不一定相同，这在增加了回测难度的同时也增加了优化空间。这点需要读者自行完成。

8.3 小结

构建一个基于指数的指数增强组合的步骤如下：

(1) 根据自己的目标找到对应特征的指数。

(2) 综合采用多个因子增强指数的特征。

本章中采用中证红利作为基本示例，演示了构建一个基于已有指数的指数增强基金。对于不同的指数，如果需要构建对应的指数增强组合，最重要的任务是根据其标的特征采用多个因子增强所关注的方向，这也是构建指数增强组合的基础。

除了中证红利指数外，掘金网站提供了更多的指数作为解析的标的，如图8-5所示。

指数代码	指数名称	启用日期	退市日期	成分股开始日期	成分股结束日期	有没有
SHSE.000001	上证指数	1991-07-15	—	2011-09-30	至今	有
SHSE.000002	A股指数	1992-02-21	—	2011-09-30	至今	有
SHSE.000003	B股指数	1992-08-17	—	2011-09-30	至今	有
SHSE.000004	工业指数	1993-05-03	—	2011-09-30	至今	有
SHSE.000005	商业指数	1993-05-03	—	2011-09-30	至今	有
SHSE.000006	地产指数	1993-05-03	—	2011-09-30	至今	有
SHSE.000007	公用指数	1993-05-03	—	2011-09-30	至今	有
SHSE.000008	综合指数	1993-05-03	—	2011-09-30	至今	有

图8-5 部分其他指数

除了一些特定的指数外，大多数掘金网站上的指数都可以被特定解析，解析的函数在前面也有介绍。而对于特定指数的特点和偏向，通过查阅相关说明也是可以了解的。具体组建哪种形态的股票组合，这点需要“基金经理”来决定，也就是股票组合创建人通过自己的认知和股票组合的侧重点来设计组合。

关于多因子策略的学习这里就告一段落，下一章将带领读者学习量化策略的另一大方面，即回归分析，这也是最常用的量化处理的方法之一。

这里股票池的构建还是使用对指数的解析，在后续的学习中还会介绍使用多种方法构建满足需要的股票池，这也是后面学习的一个重点。

第9章

掘金量化——回归分析基础

前面作者用3章向读者介绍了量化策略中一个非常重要的策略分支——多因子策略。

所谓多因子策略，就是根据各个因子的大小对股票进行打分，然后按照一定的权重加权得到一个总分，最后根据总分再对股票进行筛选。对于多因子模型的评价而言，实际上通过评分法回测出的股票组合收益率，就能够对备选的选股模型做出优劣评价。

这种方法的优点是相对比较稳健，不容易受到极端值的影响。但是多因子策略需要对各个因子的权重做一个相对比较主观的设定，这也是多因子策略在实际模型评价过程中比较困难和需要模型求取的关键点所在，同时这也能够获得优化的空间。多因子策略模型图如图9-1所示。



图9-1 多因子策略模型图

本章开始将介绍另一种常用的量化策略——回归分析，这也是最为常用的一种分析方法，它可以用在股价的回归分析上，同时也可以对基本面数据进行分析，这点在之后的讲解中会介绍。

9.1 回归分析基础

回归分析预测法是在分析市场现象自变量和因变量之间相关关系的基础上，建立变量之间的回归方程，并将回归方程作为预测模型，根据自变量在预测期的数量变化来预测因变量关系，大多表现为相关关系。因此，回归分析预测法是一种重要的市场预测方法。

当对股市或者其他金融市场现象未来的发展状况和水平进行预测时，如果能将影响预测对象的主要因素找到，并且能够取得其数量资料，就可以采用回归分析预测法进行预测。回归分析预测法是一种具体的、行之有效的、实用价值很高的常用预测方法，多用于中短期预测。

9.1.1 回归法简介

在处理测量数据时，经常要研究变量之间的关系。变量之间的关系一般分为两种，一种是完全确定关系，即函数关系；另一种是相关关系，即变量之间既存在着密切联系，但又不能由一个或多个变量的值求出另一个变量的值。

例如，学生对于高等数学、概率与统计、普通物理的学习，会对统计物理的学习产生影响，它们虽然存在着密切的关系，但很难从前几门功课的学习成绩来精确地求出统计物理的学习成绩。对于彼此联系比较紧密的变量，人们总希望建立一定的公式，以便变量之间互相

推测。回归分析的任务就是用数学表达式来描述相关变量之间的关系。

- 多元回归是指一个因变量（预报对象）、多个自变量（预报因子）的回归模型。基本方法是根据各变量的值算出交叉乘积和。
- 这种包括两个或两个以上自变量的回归称为多元回归。应用此方法可以加深对定性分析结论的认识，并得出各种要素间的数量依存关系，从而进一步揭示出各要素间内在的规律。
- 一般来说，多元回归过程能同时提供多个备选的函数关系式，并提供每个关系式对实验数据的理解能力，研究者可以结合自己的理论预期，据此做出选择。

回归分析预测法有多种类型。依据相关关系中自变量的个数不同进行分类，可分为一元回归分析预测法和多元回归分析预测法。在一元回归分析预测法中，自变量只有一个，而在多元回归分析预测法中，自变量有两个以上。依据自变量和因变量之间的相关关系不同，可分为线性回归预测和非线性回归预测。

9.1.2 一元线性回归

回归分析只涉及两个变量的，称为一元回归分析（见图9-2）。一元回归的主要任务是根据两个相关变量中的一个变量去估计另一个变量，被估计的变量称为因变量，可设为 Y ；估计出的变量称为自变量，设为 X 。回归分析就是要找出一个数学模型 $Y=f(X)$ ，使得从 X 估计 Y 可以用一个函数式去计算。

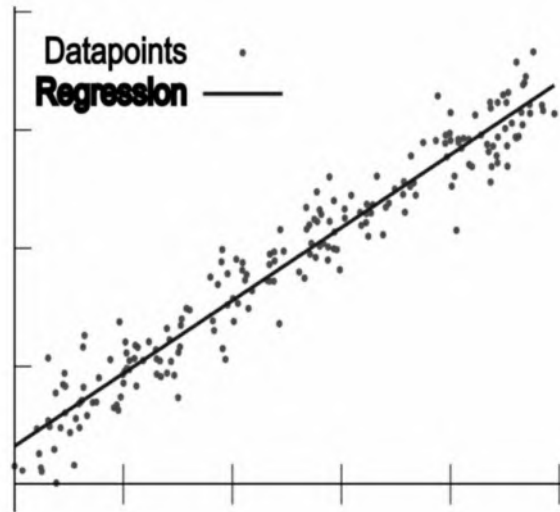


图9-2 一元线性回归

当 $Y=f(X)$ 的形式是一个直线方程时，称为一元线性回归。这个方程一般可表示为 $Y=A+BX$ 。根据最小平方法或其他方法，可以从样本数据中确定常数项 A 与回归系数 B 的值。 A 、 B 确定后，有一个 X 的观测值，就可得到一个 Y 的估计值。回归方程是否可靠，估计的误差有多大，都要经过显著性检验和误差计算。有无显著的相关关系以及样本的大小等，是影响回归方程可靠性的因素。

一元线性回归方程的形式

如果只有一个自变量 X ，而且因变量 Y 和自变量 X 之间的数量变化关系呈近似线性关系，就可以建立一元线性回归方程，由自变量 X 的值来预测因变量 Y 的值，这就是一元线性回归预测。

如果因变量 Y 和自变量 X 之间呈线性相关，也就是说，对于自变量 X 的某一个值，因变量 Y 对应的取值不是唯一确定的，而是有很多可能的取值，它们分布在一条直线的上下，这是因为 Y 还受除自变量以外的其他因素的影响。这些因素的影响大小和方向都是不确定的，通常用一个随机变量（记为 ϵ ）来表示，也称为随机扰动项。于是， Y 和 X 之间的依存关系可表示为：

$$y = a + \beta x + \varepsilon$$

上面就是总体的一元线性回归模型。其中 a, β 是常数。随机扰动项 ε 是无法直接观测的随机变量。一般为了便于求解，将上述公式表示为：

$$y = a + bx$$

a 和 b 分别为总体回归方程参数 $\alpha \mid \beta$ 的估计量， a 是样本回归方程的常数项，也就是样本回归直线在 Y 轴上的截距，表示除自变量 X 以外的其他因素对因变量 Y 的平均影响量； b 是样本回归系数，即样本回归直线的斜率，表示自变量 X 每增加一个单位，因变量 Y 的平均增加量。

换句话说，回归算法是一种基于已有数据的预测算法，其目的是研究数据特征因子与结果之间的因果关系。举个经典的例子，表9-1为某地区房屋面积与价格之间的对应表。

表9-1 某地区房屋面积与价格对应表

价格（千元）	面积（平方米）
200	105
165	80
184.5	120
116	70.8
270	150

为了简单起见，在该表中只计算了一个特征值（房屋的面积）和一个结果数据（房屋的价格），因此可以使用数据集构建一个直角坐标系，如图9-3所示。

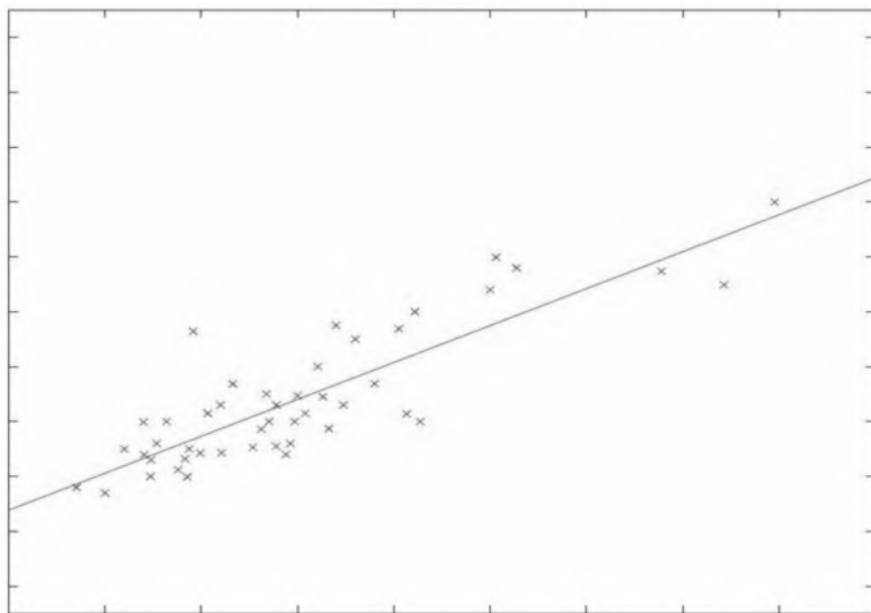


图9-3 房屋面积与价格回归表

由图9-3可见，数据集的目的是建立一个线性方程组，能够对所有的点距离无限地接近，即价格能够根据房屋的面积大小决定。

同时，可以据此得到一个线性方程组：

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

更进一步，如果将其设计为一个多元线性回归的计算模型，例如添加一个新的变量：独立卧室数，那么数据表如表9-2所示。

表9-2 某地区房屋面积与价格对应表

价格（千元）	面积（平方米）	卧室（个）
200	105	3
165	80	2
184.5	120	2
116	70.8	1
270	150	4

据此得到的线性方程组为：

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x$$

可以看到，回归计算的建模能力是非常强大的，其可以根据每个特征去计算结果，能够较好地体现特征值的影响。同时，从上面的内容可以看到，每个回归模型都可以由一个回归函数表现出来，这样能够较好地表现出特征与结果之间的关系。

9.1.3 多元线性回归

除了一元线性回归外，研究一个因变量与两个或两个以上自变量的回归，称为多元线性回归，是反映一种现象或事物的数量依多种现象或事物的数量的变动而相应地变动的规律，建立多个变量之间线性或非线性数学模型数量关系式的统计方法。

多元线性回归的基本原理和基本计算过程与一元线性回归相同，但由于自变量个数多，计算相当麻烦，一般在实际中应用时都要借助统计软件。这里只介绍多元线性回归的一些基本问题。

但由于各个自变量的单位可能不一样，比如一个消费水平的关系式中，工资水平、受教育程度、职业、地区、家庭负担等因素都会影响到消费水平，而这些影响因素（自变量）的单位显然是不同的，因此自变量前系数的大小并不能说明该因素的重要程度，简单来说，同样的工资收入，如果用元为单位就比用百元为单位所得的回归系数要小，但是工资水平对消费的影响程度并没有变，所以得想办法将各个自变量化到统一的单位上来。

举例来说， x_1 、 x_2 、 x_3 ……是N个可以精确测量或者可控制的变量，如果变量y与多个x值的联系是线性的，那么进行N次试验后，可

以获得N组数据，相互关系如下：

$$\begin{aligned}y_1 &= b_0 + b_1x_{11} + b_2x_{12} + \cdots + b_px_{1p} + \varepsilon_1 \\y_2 &= b_0 + b_1x_{21} + b_2x_{22} + \cdots + b_px_{2p} + \varepsilon_2 \\&\vdots \\y_n &= b_0 + b_1x_{n1} + b_2x_{n2} + \cdots + b_px_{np} + \varepsilon_n\end{aligned}$$

其中的多个b是待测参数， ε 是随机因子对y的影响。

9.1.4 回归法的解法——最小二乘法详解

最小二乘法（LS算法）是一种数学优化技术，也是一种回归分析的常用解法。它通过最小化误差的平方和寻找数据的最佳函数匹配。利用最小二乘法可以简便地求得未知的数据，并使得这些求得的数据与实际数据之间误差的平方和最小。最小二乘法还可用于曲线拟合，其他一些优化问题也可通过最小化能量或最大化熵用最小二乘法来表达。

下面通过一个图示向读者演示最小二乘法的原理，如图9-4所示。

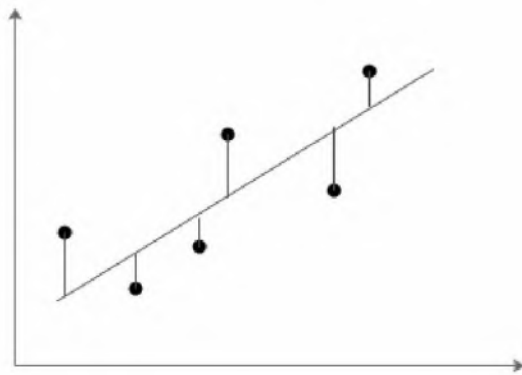


图9-4 最小二乘法的原理

从图9-4可以看到，若个点依次分布在向量空间中，如果希望找出一条直线和这些点达到最佳匹配，最简单的方法就是希望这些点到

直线的值最小，即下面的最小二乘法实现公式最小。

$$f(x) = ax + b$$
$$\delta = \sum (fx_i - y_i)^2$$

这里直接应用真实值与计算值之间的差的平方和，这种差值有个专门的名称为“残差”。基于此，表达残差的方式有以下三种。

- 范数：残差绝对值的最大值 $\max_{1 \leq i \leq m} |r_i|$ ，即所有数据点中残差距离的最大值。
- L1-范数：绝对残差和 $\sum_{i=1}^m |r_i|$ ，即所有数据点残差距离之和。
- L2-范数：残差平方和 $\sum_{i=1}^m r_i^2$ 。

可以看到，所谓的最小二乘法也就是L2-范数的一个具体应用。通俗地说，就是看模型计算的结果与真实值之间的相似性。

因此，最小二乘法可由如下公式定义：

对于给定的数据 (x_i, y_i) ($i=1, \dots, m$)，在确定的假设空间H中，求解 $f(x) \in H$ ，使得残差 $\delta = \sum (f(x_i) - y_i)^2$ 的L2-范数最小。

看到这里，可能有读者会提出疑问，这里的 $f(x)$ 又该如何表示呢？实际上，函数 $f(x)$ 是一条多项式曲线：

$$f(x, w) = w_0 + w_0 x + w_0 x^2 + w_0 x^3 + \dots + wx^n$$

继续讨论下去，最小二乘法就是找到这么一组权重w，使得 $\delta = \sum f(x_i) - y_i)^2$ 最小。那么问题就来了，如何能使得最小二乘法最小？

对于求出最小二乘法的结果，通过数学上的微积分处理方法，这是一个求极值的问题，这里只需要对权值依次求偏导数，最后令偏导数为0，即可求出极值点。

$$\begin{aligned}\frac{\partial f}{\partial w_0} &= 2 \sum_1^m (w_0 + w_1 w_i - y_i) = 0 \\ \frac{\partial f}{\partial w_1} &= 2 \sum_1^m (w_0 + w_1 w_i - y_i) x_i = 0 \\ &\vdots \\ \frac{\partial f}{\partial w_n} &= 2 \sum_1^m (w_0 + w_n w_i - y_i) x_i = 0\end{aligned}$$

具体实现最小二乘法的代码如下。

【程序9-1】


```

import numpy as np
from matplotlib import pyplot as plt

A = np.array([[5],[4]])
C = np.array([[4],[6]])
B = A.T.dot(C)
AA = np.linalg.inv(A.T.dot(A))
l=AA.dot(B)
P=A.dot(l)
x=np.linspace(-2,2,10)
x.shape=(1,10)
xx=A.dot(x)
fig = plt.figure()
ax= fig.add_subplot(111)
ax.plot(xx[0,:],xx[1,:])
ax.plot(A[0],A[1], 'ko')

ax.plot([C[0],P[0]], [C[1],P[1]], 'r-o')
ax.plot([0,C[0]], [0,C[1]], 'm-o')

ax.axvline(x=0,color='black')
ax.axhline(y=0,color='black')

margin=0.1
ax.text(A[0]+margin, A[1]+margin, r"A",fontsize=20)
ax.text(C[0]+margin, C[1]+margin, r"C",fontsize=20)
ax.text(P[0]+margin, P[1]+margin, r"P",fontsize=20)
ax.text(0+margin,0+margin,r"O",fontsize=20)
ax.text(0+margin,4+margin, r"y",fontsize=20)
ax.text(4+margin,0+margin, r"x",fontsize=20)
plt.xticks(np.arange(-2,3))
plt.yticks(np.arange(-2,3))

ax.axis('equal')
plt.show()

```

最终结果如图9-5所示。

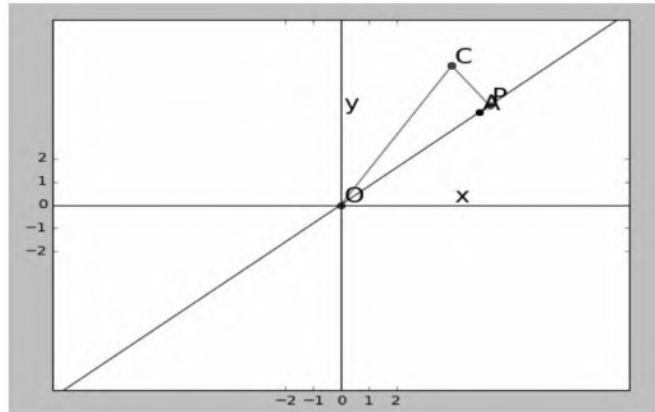


图9-5 最小二乘法拟合曲线

在【程序9-1】中，详细演示了最小二乘法的计算，但是在实际应用中并不需要每次使用最小二乘法进行计算时都需要列出详细的方法，Python有专门的工具包为读者提供计算最小二乘法的方法。

代码段如下：

```
import statsmodels.api as sm
import numpy as np

def 一元线性回归():
    x1 = np.asarray([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
    y = 10 + 4 * x1

    x_ = sm.add_constant(x1)
    model = sm.OLS(y, x_)
    results = model.fit()

    print(results.params)
```

这里首先生成了一个序列，之后根据这个序列使用自定义函数生成一个序列值，statsmodels是Python中的统计学工具包。add_constant生成一个一元的待估参数函数，OLS是使用Python中的最小二乘法去估算生成的函数模型，最后打印结果如下：

```
[10.  4.]
```

这里的打印结果是系数和扰动噪音。读者可以自行替换相关内容。需要注意的是，扰动噪音被放在params中的第一个参数处。

而对于多元线性回归算法，代码如下：

```
import statsmodels.api as sm
import numpy as np

def 多元线性回归():
    x1 = np.asarray([1,2,3,4,5,6,7,8,9,10])
    x2 = np.asarray([3,2,1,5,4,6,7,6,9,2])

    y = 10 + 4 * x1 + 3 * x2
    x_ = np.column_stack((x1, x2))
    x_ = sm.add_constant(x_)

    model = sm.OLS(y,x_)
    results = model.fit()
    print(results.params)
```

这里的参数设置与一元线性回归类似，column_stack是NumPy组建的一个多元回归函数，add_constant是添加扰动噪音的函数，最后还是通过最小二乘法计算出最小函数的结果。

9.2 回归分析的一些其他计算方法

除了常用的OLS方法之外，还有其他的计算线性回归的方法，例如梯度下降算法和使用Python的TensorFlow工具包计算回归分析。

9.2.1 梯度下降算法与使用TensorFlow计算线性回归

除了最常用的OLS算法外，梯度下降算法也是一种常用的求解回归方程的算法。在介绍随机梯度下降算法之前，给大家讲一个道士下山的故事。

图9-6所示为一个模拟随机梯度下降算法的演示图。为了便于理解，作者将其比喻成道士想要出去游玩的一座山。

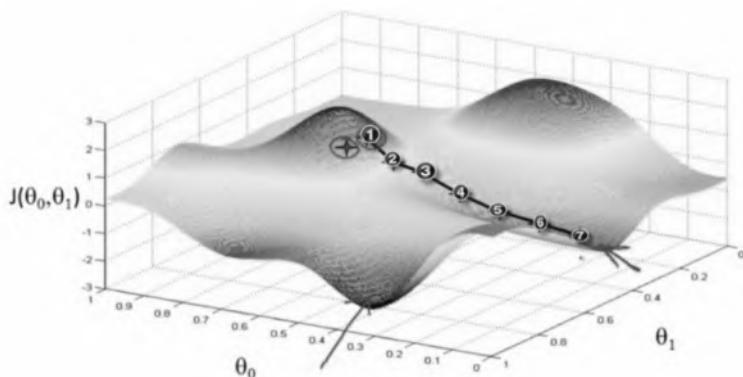


图9-6 模拟随机梯度下降算法的演示图

设想道士有一天和道友一起到一座不太熟悉的山上去玩，在兴趣盎然中很快登上了山顶。但是天有不测，下起了雨。如果这时需要道士和其同来的道友以最快的速度下山，那么怎么办呢？

最好的办法是顺着坡度最陡峭的地方走下去。但是由于不熟悉路，道士在下山的过程中，每走过一段路程，就需要停下来观望，从而选择最陡峭的下山路。这样一路走下来的话，可以在最短时间内走到底。

图9-6中的路线可以近似地表示为：

$$\textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{3} \rightarrow \textcircled{4} \rightarrow \textcircled{5} \rightarrow \textcircled{6} \rightarrow \textcircled{7}$$

每个数字代表每次停顿的地点，这样只需要在每个停顿的地点选择最陡峭的下山路即可。

这就是道士下山的故事。随机梯度下降算法与此类似，如果想要使用最迅捷的方法，最简单的办法就是在下降一个梯度的阶层后，寻找一个当前获得的最大坡度继续下降。这就是随机梯度算法的原理。

从上面的例子可以看出，随机梯度下降算法就是不停地寻找某个节点中下降幅度最大的那个趋势进行迭代计算，直到将数据收缩到符合要求的范围为止。通过数学公式计算的话，公式如下：

$$f(\theta) = \theta_0 x_0 + \theta_1 x_1 + \dots + \theta_n x_n = \sum \theta_i x_i$$

前面介绍最小二乘法的时候，说明了直接求解最优化变量的方法，也介绍了在求解过程中的前提条件是要求计算值与实际值的偏差的平方最小。

但是在随机梯度下降算法中，对于系数需要不停地求解出当前位置下最优化的数据。通过数学方式表达的话，就是不停地对系数 θ 求偏导数，公式如下：

$$\frac{\partial}{\partial \theta} f(\theta) = \frac{\partial}{\partial \theta} \frac{1}{2} \sum (f(\theta) - y_i)^2 = (f(\theta) - y) x_i$$

公式中， θ 会向着梯度下降最快的方向减少，从而推断出 θ 的最优解。

因此，可以说随机梯度下降算法最终被归结为通过迭代计算特征值从而求出最合适的值。 θ 求解的公式如下：

$$\theta = \theta - \alpha (f(\theta) - y_i) x_i$$

公式中， α 是下降系数，用较为通俗的话表示就是用以计算每次下降的幅度大小。系数越大，每次计算中的差值越大，而系数越小，差值越小，但是计算时间也相对延长。

随机梯度下降算法模型如图9-7所示。

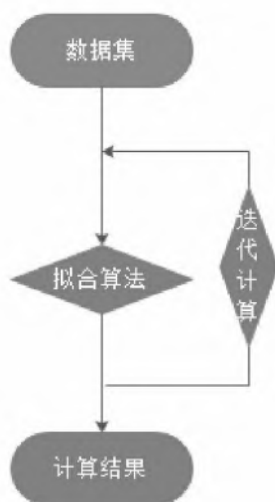


图9-7 随机梯度下降算法过程

从图9-7中可以看到，实现随机梯度下降算法的关键是拟合算法的实现。而本例的拟合算法实现较为简单，通过不停地修正数据值从而达到数据的最优值。

随机梯度下降算法在神经网络特别是机器学习中应用较广，但是由于其天生的缺陷，噪音较多，使得在计算过程中并不是都向着整体最优解的方向优化，往往可能只是一个局部最优解。因此，为了克服这些困难，一个最好的办法是增大数据量，在不停地使用数据进行迭代处理的时候，能够确保整体的方向是全局最优解，或者最优结果在全局最优解附近。

【程序9-2】

```

x = [(2, 0, 3), (1, 0, 3), (1, 1, 3), (1, 4, 2), (1, 2, 4)]
y = [5, 6, 8, 10, 11]

epsilon = 0.002

alpha = 0.02
diff = [0, 0]
max_itor = 1000
error0 = 0
error1 = 0
cnt = 0
m = len(x)

theta0 = 0
theta1 = 0
theta2 = 0

while True:
    cnt += 1

    for i in range(m):
        diff[0] = (theta0 * x[i][0] + theta1 * x[i][1] + theta2 * x[i][2]) - y[i]
        theta0 -= alpha * diff[0] * x[i][0]
        theta1 -= alpha * diff[0] * x[i][1]
        theta2 -= alpha * diff[0] * x[i][2]

    error1 = 0
    for lp in range(len(x)):
        error1 += (y[lp] - (theta0 + theta1 * x[lp][1] + theta2 * x[lp][2])) **
2 / 2
    if abs(error1 - error0) < epsilon:
        break
    else:
        error0 = error1

print('theta0 : %f, theta1 : %f, theta2 : %f, error1 : %f' % (theta0, theta1,
theta2, error1))
print('Done: theta0 : %f, theta1 : %f, theta2 : %f' % (theta0, theta1, theta2))
print('迭代次数: %d' % cnt)

```

最终结果打印如下：

```
theta0 : 0.100684, theta1 : 1.564907, theta2 : 1.920652, error1 : 0.569459  
Done: theta0 : 0.100684, theta1 : 1.564907, theta2 : 1.920652  
迭代次数: 2118
```

从结果上看，这里需要迭代2118次即可获得最优解。

而使用TensorFlow计算一元线性回归和多元线性回归的方法如下：


```

#使用 TensorFlow 进行一元线性回归
def linear_regression(real):

    xs = []
    for _ in range(len(real)):
        xs.append(_)
    xs = np.reshape(xs, (-1,1))
    ys = real
    ys = np.reshape(ys, (-1, 1))

    x = tf.placeholder(tf.float32, [None, 1])
    y_ = tf.placeholder(tf.float32, [None, 1]) # y_为测试集结果数据

    weight = tf.Variable(tf.zeros([1, 1]))
    bias = tf.Variable(tf.zeros([1]))

    #y = tf.matmul(x, weight) + bias #不用矩阵乘法
    y = tf.multiply(weight,x) + bias #简单的线性乘法

    loss = tf.reduce_mean(tf.square(y - y_)) #批量线性回归用这个损失函数
    train_step = tf.train.AdamOptimizer(0.00001).minimize(loss)

    init = tf.global_variables_initializer()

    flag = True

    with tf.Session() as sess:
        sess.run(init)

        count = 0
        loss_temp = 0
        loss_flag = 0
        while flag:
            feed = {x: xs, y_: ys}
            sess.run(train_step, feed_dict=feed)
            loss_res = sess.run(loss, feed_dict=feed)

            if count%10000 == 0:
                print("正在运行次数为:",count," Loss 为: ",loss_res)

            if loss_temp == loss_res:
                loss_flag += 1
                if loss_flag > 10:
                    flag = False
            count += 1
            loss_temp = loss_res

        weight = sess.run(weight)
        bias = sess.run(bias)

    return weight[0][0],bias[0]

```

这里使用TensorFlow进行一元线性回归，而使用TensorFlow进行多元线性回归的代码如下：

```

#使用 TensorFlow 进行多元线性回归
def linear_regression(real_Y,real_MAT):

m,n = np.shape(real_MAT)

    real_Y = np.reshape(real_Y, (m, 1))

    real_MAT = np.reshape(real_MAT, (-1,n))
    x_ = tf.placeholder(tf.float32, [None, n])
    y_ = tf.placeholder(tf.float32, [m, 1]) # y_为测试集结果数据

    weight = tf.Variable(tf.ones([n, 1]))
    bias = tf.Variable(tf.ones([1,1]))

    y = tf.matmul(x_, weight) + bias

    loss = tf.reduce_mean(tf.square(y - y_)) #批量线性回归用这个损失函数
    train_step = tf.train.AdamOptimizer(0.01).minimize(loss)

    init = tf.global_variables_initializer()

    flag = True
    with tf.Session() as sess:
        sess.run(init)

        count = 0
        loss_temp = 0
        while flag:

            feed = {x_: real_MAT, y_: real_Y}
            sess.run(train_step, feed_dict=feed)

            loss_res = sess.run(loss, feed_dict=feed)
            loss_res = np.nan_to_num(loss_res)
            # if count%10000 == 0:
            #     print("正在运行次数为:",count," Loss 为: ",loss_res)

            if (loss_temp == loss_res) or (loss_res < 0.001):
                flag = False
            count += 1
            loss_temp = loss_res

        weight = sess.run(weight)
        bias = sess.run(bias)
        return weight,bias

```

9.2.2 线性回归的姐妹——逻辑回归

最后将少用数学公式而采用浅显易懂的方法解释一些量化策略中用到的基本理论和算法。下面将顺带介绍一下线性回归的姐妹算法——逻辑回归。本小节内容的难度较大，读者可以不看数学理论部分，9.4节将以实例的方式进行展示。

逻辑回归主要应用于分类领域，其主要作用是对不同性质的数据进行分类标识。逻辑回归是在线性回归的算法上发展起来的，它提供一个系数 θ ，并对其求值。基于此，可以较好地提供理论支持和不同的算法，轻松地对数据集进行分类。

图9-8是房屋面积与价格回归示意图，在这里使用逻辑回归算法对房屋价格进行了分类。可以看到，其被较好地分成了两部分，这也是在计算时要求区分的内容。

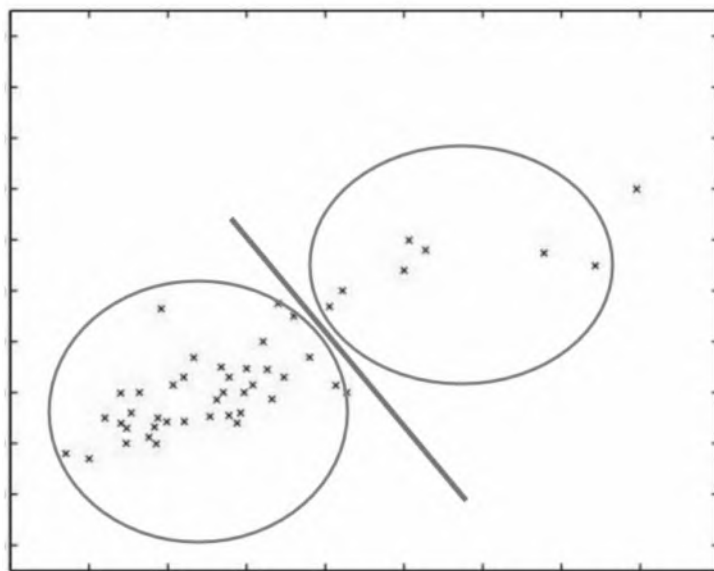


图9-8 房屋面积与价格回归示意图

逻辑回归的具体公式如下：

$$f(x) = \frac{1}{1 + \exp(-\theta^T x)}$$

与线性回归相同，这里的 θ 是逻辑回归的参数，即回归系数。如果将其进一步变形，使其能够反映二元分类问题，那么公式为：

$$f(y = 1|x, \theta) = \frac{1}{1 + \exp(-\theta^T x)}$$

这里 y 值是由已有的数据集中的数据和 θ 共同决定的。而实际上，这个公式求的是在满足一定条件时，计算最终数据被归于某一类的概率。通过公式表示为：

$$\log(x) = \ln\left(\frac{f(y = 1|x, \theta)}{f(y = 0|x, \theta)}\right) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

通过这个逻辑回归倒推公式可以看到，最终逻辑回归的计算可以转化成由数据集的特征向量与系数 θ 共同完成，然后求得其加权和，得到最终的判断结果。

由前面的数学分析来看，最终逻辑回归问题也可以称为对系数 θ 的求值问题。这里读者只需要知道原理即可。

9.3 实战：回归分析——短时间开盘价与收盘价之间的关系

下面使用一个简单的实例进行介绍。

对于某只股票来说，在短时期内的开盘价和收盘价总有一定的关系，那么如何利用短期的开盘价与收盘价之间的关系去对当日的收盘价进行预测呢？图9-9是一个股价的线性回归图。

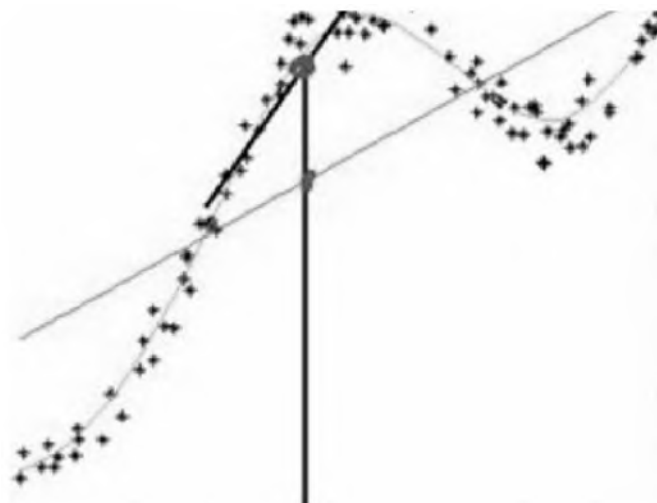


图9-9 股价的线性回归

9.3.1 量化策略基本思路与简单实现

首先要设计出简单的量化策略，对于使用回归分析进行开盘价和收盘价预测也是如此。基于本例中只是对回归分析做一个简单的演示，设计的量化策略如下：

- 用前7日的开盘价和收盘价做线性拟合。
- 根据当日的开盘价去预测当日的收盘价。
- 如果当日开盘价比当日预测的收盘价低就买入，否则卖出。

下面是对策略的实现。在实现策略之前，最好通过一个个程序段对主要策略予以实现。在实际工作中也是如此，首先完善主策略，之后将其包装成只需要传参的函数，代码如下：

【程序9-3】

```

from gm.api import *
import datetime
import numpy as np
import statsmodels.api as sm

set_token("e8978d765c4822e5a85fcaa73e044065cf17b58b")
day_time, hour_and_mins = str(datetime.datetime.now().strftime('%Y-%m-%d
%H:%M:%S')).split(" ")

symbol = "SZSE.000651"
data = history_n("SZSE.000651", frequency="1d", end_time=
day_time, count=7, fields="close,
open", df=True)

close = data["close"].values
open = data["open"].values

open = sm.add_constant(open)

model = sm.OLS(close, open)

results = model.fit()
bias, weight = (results.params)

print(bias, " ", weight)

```

这里使用的是查询方法，打印结果如下：

```
19.9680323625  0.562732442978
```

可以看到通过线性回归分析最终得到一个系数组，分别是线性回归对应的bias和weight，通过拟合函数的构建可以知道，最终结果应该如下：

$$\text{close} = \text{bias} + \text{open} \times \text{weight}$$

这样可以通过计算获取最终的预测值。

下面将主策略以函数的形式包装起来，通过程序可以发现主策略主要涉及两个主要参数：symbol与data，因此可以将这两个参数独立

出来作为形参传入，代码如下：

```
def get_predict_close(symbol, now, count = 7):  
  
    last_day = get_previous_trading_date("SHSE", now)  
    data =  
history_n(symbol, frequency="1d", end_time=last_day, count=count, fields="close, open",  
df=True)  
  
    close = data["close"].values  
    open = data["open"].values  
  
    open = sm.add_constant(open)  
  
    model = sm.OLS(close, open)  
  
    results = model.fit()  
    bias, weight = (results.params)  
  
    return bias, weight
```

需要注意的是，这里在传参的时候还额外传入了count参数，这个的参数被设置成默认的数字7，读者也可以设置成其他参数。

9.3.2 使用掘金量化实现回测

下面使用掘金量化对策略进行回测，代码如下：

【程序9-4】


```

import numpy as np
from gm.api import *
import fun
import os
import csv

import datetime
def init(context):
    schedule(schedule_func=algo, date_rule='1d', time_rule='09:31:00')

    context.symbol = "SHSE.601318"

def algo(context):
    now = context.now

    bias,weight = fun.get_predict_close(context.symbol,now,count=7)

    data = history_n(context.symbol, frequency="1d", end_time=now, count=1,
fields="open", df=True)
    pre_close = bias + weight * data["open"].values
    if pre_close > data["open"].values:
        order_target_percent(symbol=context.symbol, percent=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
    else:
        order_target_percent(symbol=context.symbol, percent=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
def on_backtest_finished(context, indicator):
    pass

if __name__ == "__main__":
    run(
        strategy_id='9f53e15f-6870-11e8-801c-4cedfb681747',
        filename=(os.path.basename(__file__)),
        mode=MODE_BACKTEST,
        token='e8978d765c4822e5a85fcaa73e044065cf17b58b',
        backtest_start_time="2018-01-15 09:30:00",
        backtest_end_time='2018-07-10 15:00:00',
        backtest_initial_cash=10000000,
        backtest_adjust=ADJUST_PREV
    )

```

在本例中，设置的是当预测值大于开盘价时就买入，而预测的收盘价小于开盘价时就卖出，测试结果如图9-10所示。

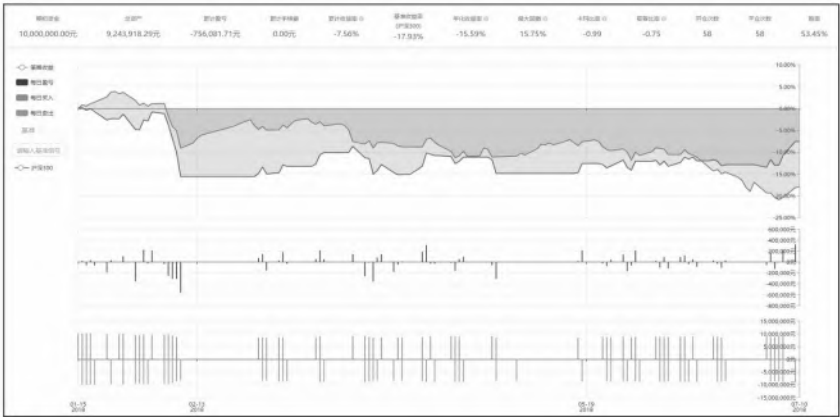


图9-10 线性回归的回测结果

可以看到这里的胜率并不高，并没有获得较高的额外收益，读者有可能想到，如果将其变化，目前预测的收盘价高于开盘价，而将其转换成预测的收盘价低于开盘价进行买入。代码有了一些变化：

【程序9-5】

```

import numpy as np
from gm.api import *
import fun
import os
import csv

import datetime
def init(context):
    schedule(schedule_func=algo, date_rule='1d', time_rule='09:31:00')

    context.symbol = "SHSE.601318"

def algo(context):
    now = context.now
    order_close_all()
    bias,weight = fun.get_predict_close(context.symbol,now,count=7)

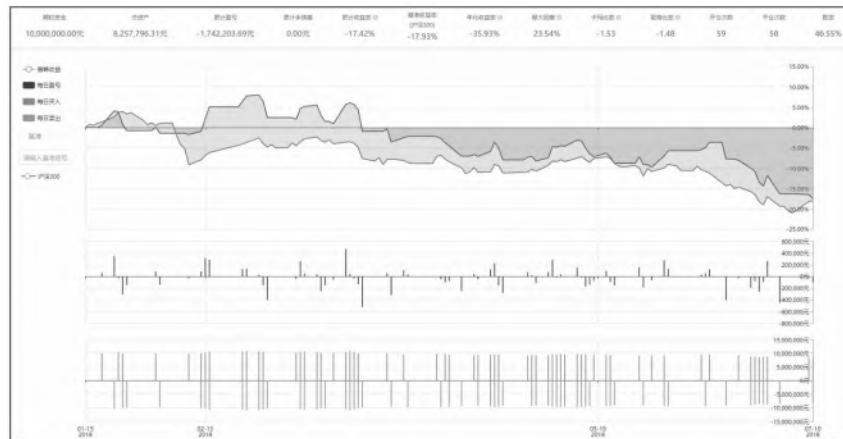
    data = history_n(context.symbol, frequency="1d", end_time=now, count=1,
fields="open", df=True)
    pre_close = bias + weight * data["open"].values
    if pre_close < data["open"].values:
        order_target_percent(symbol=context.symbol, percent=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)

def on_backtest_finished(context, indicator):
    pass

if __name__ == "__main__":
    run(
        strategy_id='9f53e15f-6870-11e8-801c-4cedfb681747',
        filename=(os.path.basename(__file__)),
        mode=MODE_BACKTEST,
        token='e8978d765c4822e5a85fcaa73e044065cf17b58b',
        backtest_start_time="2018-01-15 09:30:00",
        backtest_end_time='2018-07-10 15:00:00',
        backtest_initial_cash=10000000,
        backtest_adjust=ADJUST_PREV
    )

```

回测结果如图9-11所示。



可能回测结果不像读者想象的那样，能够获取较高的收益，对于胜率的比较，图9-10和图9-11的胜率分别为53.45%和46.55%，这样在一起的和为100%。

但是问题在于，无论采用哪种方式均无法获取正收益，这里截取并显示601318所对应的股票走势图，代码如下：

```
data = history("SHSE.601318",frequency="1d",start_time="2018-01-15",end_time=day_time,fields="close",df=True)

close = data["close"].values
plt.plot(close)
plt.show()
```

显示结果如图9-12所示。

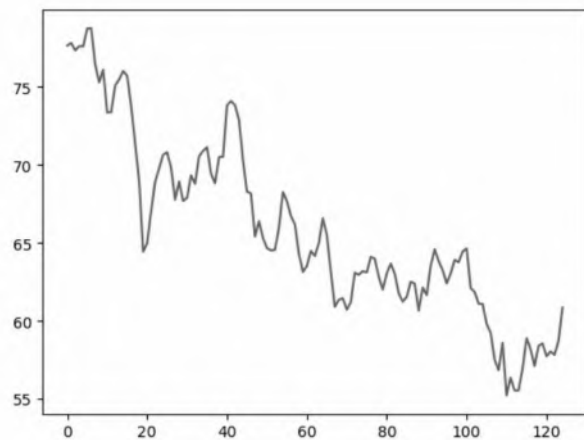


图9-12 601318走势图

可以看到，从2018年至今，601318的走势一直以向下走为主，这样的走势对于任何买入和卖出指标的量化交易来说都是危险的。因此，建议读者买卖股票除了技术指标外，还需要对目标标的进行选择。

选择的好坏永远大于技术择时。

9.4 买还是卖——逻辑回归帮你做决定

买还是卖，这是一个问题（见图9-13）。



图9-13 买还是卖

对于大多数投资者来说，买或者不买实际上是一种经验的总结，其更多的是通过历史数据的分析做出判断。那么能否提供一种方法，通过模拟历史数据来对未来做出判断，这就是逻辑回归。

9.4.1 逻辑回归是一种分类算法

前面对逻辑回归已经有了一个大致的介绍，如果有读者跳过了其中的数学部分也没关系，你需要记住的就是逻辑回归并不是数值回归，逻辑回归是一种分类算法。

逻辑回归是一种广义的线性回归分析模型，常用于数据挖掘、疾病自动诊断、经济预测等领域。例如，探讨引发疾病的危险因素，并根据危险因素预测疾病发生的概率等。

以胃癌病情分析为例，选择两组人群，一组是胃癌组，另一组是非胃癌组，两组人群必定具有不同的体征与生活方式等。因此，因变量就为是否是胃癌，值为“是”或“否”，自变量就包括很多了，如年龄、性别、饮食习惯、幽门螺杆菌感染等。

自变量既可以是连续的，也可以是分类的。然后通过logistic回归分析可以得到自变量的权重，从而可以大致了解到底哪些因素是导致胃癌的危险因素。同时，利用该权值可以根据危险因素预测一个人患癌症的可能性。

逻辑回归的主要用途如下。

- 寻找危险因素：寻找某一疾病的危险因素；
- 预测：如果已经拟合出回归模型，那么可以根据模型预测在不同自变量的情况下发生某病或某种情况的概率有多大。
- 判别：根据逻辑回归模型判断某人属于某病或属于某种情况的概率有多大。

这是逻辑回归最常用的三个用途，实际上逻辑回归的用途是极为广泛的，逻辑回归几乎已经成了流行病学和医学中最常用的分析方法，因为它与多重线性回归相比有很多的优势，后面会对该方法进行详细的阐述。实际上还有很多分类方法，只不过逻辑回归是最成功也是应用最广的方法。

9.4.2 逻辑回归的TensorFlow实现

TensorFlow在前面已经介绍过了，这里将以实际应用为主，介绍使用TensorFlow完成以前一日的开盘价、收盘价、最高价、最低价和成交量为因子的逻辑回归。

第一步：数据格式

在构建模型结构时暂时不需要输入数据，但需要制定几个输入特征。从前面介绍的影响因子可以看出，输入特征有以下5个。

- 前一日的开盘价：open。
- 前一日的收盘价：close。
- 前一日的最高价：high。
- 前一日的最低价：low。
- 前一日的成交量：volume。

而最后一个量为flag，这里做了规定，即当日收盘价高于开盘价时，即将值设置成1，反之为0。

第二步：数据的获取

下面使用掘金获取数据。

通过分析可以知道，由于在设计策略时已经做出规定，通过计算前一日的5个特征值去对第二日的涨跌进行计算，因此数据的获取代码如下：

```
last_day = get_previous_trading_date("SHSE",now)
last_last_day = get_previous_trading_date("SHSE",last_day)

df = history_n(symbol, frequency="1d", end_time=last_last_day, count=60,
fields="open,close,high,low,volume", df=True)
```

这里需要说明，函数中的now是当前的日期，通过get_previous_trading_date函数获取前一日的日期并进一步获取前两日的日期。这样做的好处在于，数据的计算是根据前日的数据对第二天的涨跌进行判断，而最终的判定因子还是以前一日的涨跌为主。生成的数据如下：

	close	high	low	open	volume
0	3843.4885	3854.4292	3769.8787	3769.8787	11421952500
1	3828.7014	3838.6921	3817.0320	3824.5930	7817261200
2	3755.4941	3828.7129	3744.8853	3823.3921	8715064100
3	3756.8765	3776.9751	3714.6443	3771.0369	8953172600
4	3763.6460	3787.3025	3744.5476	3769.9250	8330103500

这里是以沪深300的指数数据为例进行判定的。

下面对指标进行判定。前面已经介绍过了，对于指标的判定采用测试当日的数据涨跌情况作为判定指标，上涨设为1，而下跌设为0，代码如下：

```
df["flag"] = 0
for _ in range(1, len(df)):
    res = (df.loc[_ - 1, "close"]) - (df.loc[_ - 1, "open"])
    if (res) > 0:
        df.loc[_ , "flag"] = 1
```

可以看到，对于获取的数据集，根据第二日的涨跌设置成1或者0，并且将其设置成pandas中的flag数据，结果如下：

	close	high	low	open	volume	flag
0	3843.4885	3854.4292	3769.8787	3769.8787	11421952500	0
1	3828.7014	3838.6921	3817.0320	3824.5930	7817261200	1
2	3755.4941	3828.7129	3744.8853	3823.3921	8715064100	1
3	3756.8765	3776.9751	3714.6443	3771.0369	8953172600	0
4	3763.6460	3787.3025	3744.5476	3769.9250	8330103500	0

这里需要注意的是，flag是所要计算的第二日的涨跌幅，因此整体数据的长度要小于原始数据长度。

第三步：TensorFlow中逻辑回归模型的设计

在TensorFlow中，对于逻辑回归设计了便于使用的模型，代码如下：

```
classifier = tf.estimator.DNNClassifier(
    # 这个模型接受哪些输入的特征
    feature_columns=symbol_feature_columns,
    # 包含两个隐藏层，每个隐藏层包含 10 个神经元
    hidden_units=[128, 128],
    # 最终结果要分成的几类
    n_classes=2)
```

这里的 `tf.estimator.DNNClassifier` 函数指的是使用了TensorFlow中自带的逻辑回归函数，`hidden_units`参数使用list设置了隐藏层中的神经元个数以及最终的分类数。

```
def train_func(train_x, train_y):  
    dataset = tf.data.Dataset.from_tensor_slices((dict(train_x), train_y))  
    dataset = dataset.shuffle(1000).repeat().batch(100)  
    return dataset
```

`train_func`是准备训练集函数，这里设置了训练参数和训练结果值，并且对数据进行了打乱处理，最终将打乱后的结果返回。

```
classifier.train(  
    input_fn=lambda: train_func(train_x, train_y),  
    steps=1000)
```

下面是训练函数，这里使用上文的训练数据集进行输出。

```
def predict_input_fn(features, labels, batch_size):  
    features = dict(features)  
    if labels is None:  
        # No labels, use only features.  
        inputs = features  
    else:  
        inputs = (features, labels)  
    dataset = tf.data.Dataset.from_tensor_slices(inputs)  
  
    assert batch_size is not None, "batch_size must not be None"  
    dataset = dataset.batch(batch_size)  
    return dataset
```

这里是输出函数，将待测数据放入函数中，使用训练好的模型和参数进行测试。

整体代码段如下：

```
def get_check_flag(symbol,now):

    last_day = get_previous_trading_date("SHSE",now)
    last_last_day = get_previous_trading_date("SHSE",last_day)

    df = history_n(symbol, frequency="1d", end_time=last_last_day, count=60,
fields="open,close,high,low,volume", df=True)

    df["flag"] = 0

    for _ in range(1, len(df)):
        res = (df.loc[_ - 1, "close"]) - (df.loc[_ - 1, "open"])
        if (res) > 0:
            df.loc[_, "flag"] = 1

    train_x, train_y = df, df.pop('flag')
```

```

symbol_feature_columns = []
for key in train_x.keys():
    symbol_feature_columns.append(tf.feature_column.numeric_column(key=key))

classifier = tf.estimator.DNNClassifier(
    # 这个模型接受哪些输入的特征
    feature_columns=symbol_feature_columns,
    # 包含两个隐藏层，每个隐藏层包含 10 个神经元
    hidden_units=[128, 128, 128],
    # 最终结果要分成几类
    n_classes=2)

def train_func(train_x, train_y):
    dataset = tf.data.Dataset.from_tensor_slices((dict(train_x), train_y))
    dataset = dataset.shuffle(1000).repeat().batch(100)
    return dataset

classifier.train(
    input_fn=lambda: train_func(train_x, train_y),
    steps=1000)

def predict_input_fn(features, labels, batch_size):
    features = dict(features)
    if labels is None:
        # No labels, use only features.
        inputs = features
    else:
        inputs = (features, labels)
    dataset = tf.data.Dataset.from_tensor_slices(inputs)

    assert batch_size is not None, "batch_size must not be None"
    dataset = dataset.batch(batch_size)
    return dataset

df = history_n(symbol, frequency="1d", end_time=last_day, count=1,
fields="open,close,high,low,volume", df=True)
test =
[df["open"].values, df["close"].values, df["high"].values, df["low"].values, df["v
olume"].values]

test_data = pd.DataFrame({'open': test[0], 'close': test[1], 'high': test[2],
'low': test[3], 'volume': test[4]}, index=[0])

```

```
predict_result = []
predictions = classifier.predict(
    input_fn=lambda:predict_input_fn(test_data,labels=None,batch_size=1))
for predict in predictions:
    predict_result.append(predict['probabilities'].argmax())

return (predict_result)
```

通过模型训练前60日的涨跌情况判断当日的涨跌情况，并将数据返回。若返回值是1，则为买入信号，若返回值为0，则为卖出信号。

9.4.3 使用TensorFlow的逻辑回归进行回测

下面使用逻辑回归的策略进行数据回测。使用的是沪深300指数作为买卖标的，代码如下：

【程序9-6】

```

import numpy as np
from gm.api import *
import 单股票的逻辑回归分析 as fun
import os
import csv

import datetime
def init(context):
    schedule(schedule_func=algo, date_rule='1d', time_rule='09:31:00')

    context.symbol = "SHSE.000300"

def algo(context):
    now = context.now
    order_close_all()
    flag = fun.get_check_flag(context.symbol, now)
    print(flag)

    if flag:
        order_target_percent(symbol=context.symbol, percent=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
def on_backtest_finished(context, indicator):
    pass

if __name__ == "__main__":
    run(
        strategy_id='9f53e15f-6870-11e8-801c-4cedfb681747',
        filename=(os.path.basename(__file__)),
        mode=MODE_BACKTEST,
        token='e8978d765c4822e5a85fcaa73e044065cf17b58b',
        backtest_start_time="2018-01-15 09:30:00",
        backtest_end_time='2018-07-10 15:00:00',
        backtest_initial_cash=10000000,
        backtest_adjust=ADJUST_PREV
    )

```

这里导入的主策略判定就是前面所设置的逻辑回归模型，根据值的判定从而决定数据的买卖。最终结果如图9-14所示。

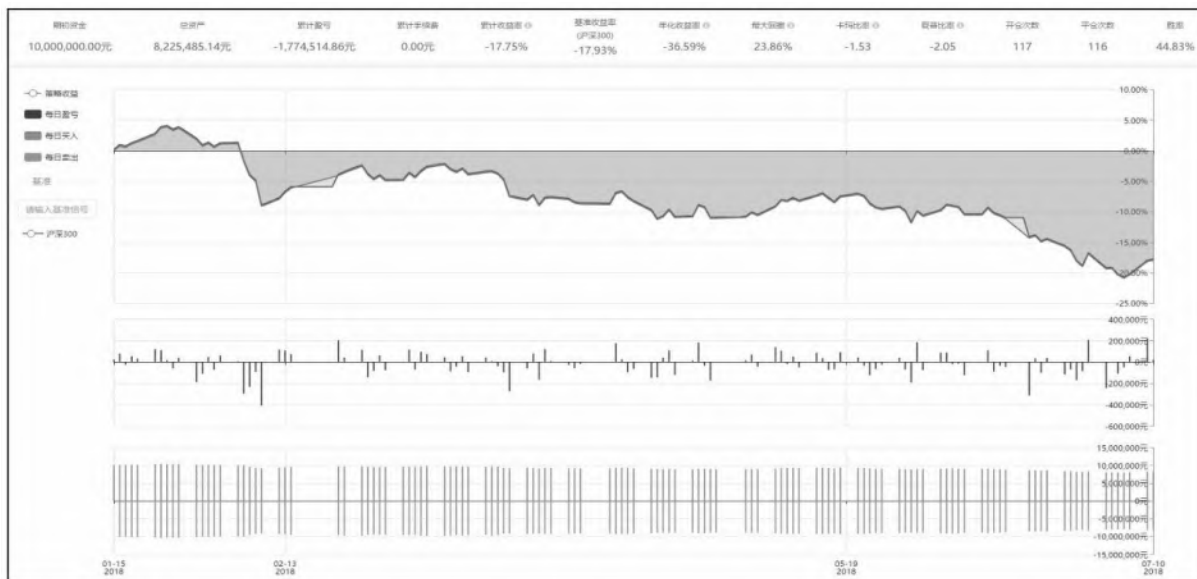


图9-14 逻辑回归的回测结果

这里可以看到，回测曲线非常好地拟合了沪深300的走势，但是由于在回测期间沪深300为下跌状态，因此其收益为负。

此时，如果把买入指标和卖出指标互换，那么回测结果如图9-15所示。

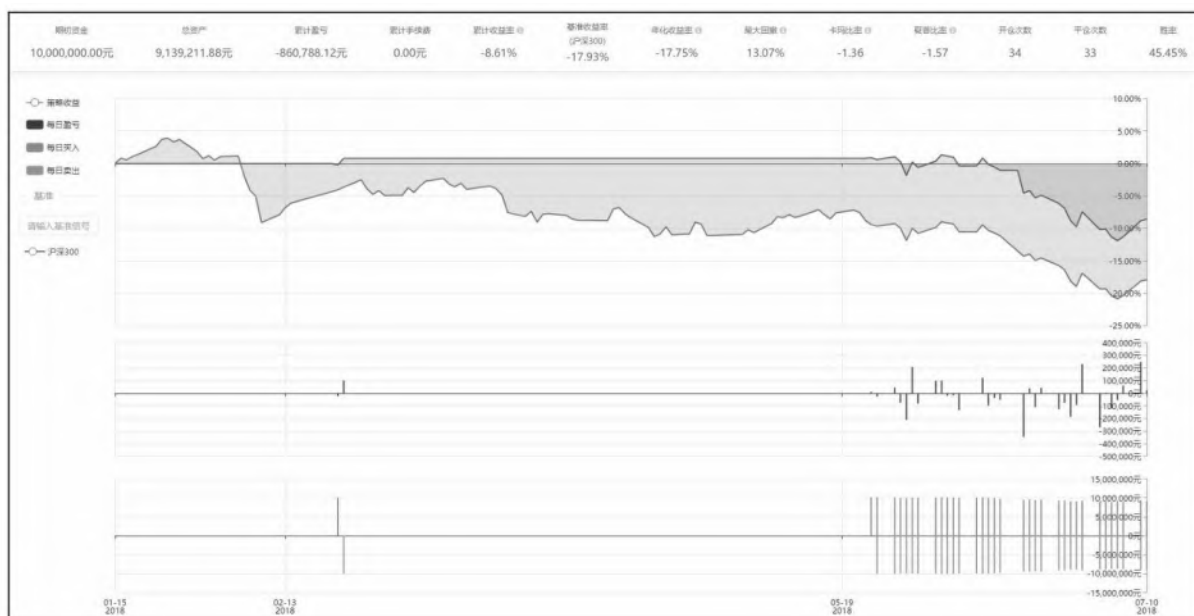


图9-15 买卖指标互换之后的回测结果

这里可以看到，对于买卖指标进行互换，收益并没有改善多少，因此可以说，简单改变买卖指标进行股票的买卖对于收益并没有多少影响。

这里有一点需要注意，对于胜率来说，图9-14和图9-15的胜率和为90%，而不是前面所写的100%，这里可能是产生了数据缺失，请读者注意。

9.5 机器学习策略——支持向量机

本节将简略介绍支持向量机的用法。

支持向量机是机器学习的一种，相对于线性回归和逻辑回归来说，支持向量机实际上是一种有监督的机器学习算法。将机器学习算法应用于股市分类中，与上一节的逻辑回归类似，实际上也是通过训练模型学习不同因子对未来的走势的影响去做出判断。

9.5.1 支持向量机的基本概念

支持向量机解决的是有监督的二元分类问题（supervised binary classification）。

例如，对于股票市场来说，对于某只股，希望根据它的属性以及一系列已经分类好的历史样本，将这个新样本分到两个不同的目标类中的某一类。

股市的涨跌识别就是一个例子：一只股票在股市中每日的行情要么涨要么跌。因此，对于股票的走势判定，我们希望机器学习算法能

够自动对它分类。为此，我们首先通过人工对大量的历史数据进行标识，然后用这些标识后的历史数据对机器学习算法进行训练。

在处理这类分类问题时，SVM的作用对象是样本的特征空间（feature space），它是一个有限维度的向量空间，每个维度对应着样本的一个特征，而这些特征组合起来可以很好地描述被分类的样本。

为了对新的样本分类，SVM算法会根据历史数据在特征空间内构建一个超平面（hyperplane），它将特征空间线性分割为两部分，对应着分类问题的两类，分别位于超平面的两侧。构建超平面的过程就是模型训练过程。对于一个给定的新样本，根据它的特征值，它会被放在超平面两侧中的某一侧，这便完成了分类。不难看出，SVM是一个非概率的线性分类器。这是因为SVM模型回答的是非此即彼的问题，新样本会被确定地分到两类中的某一类。

在数学表达上，每一只股票的历史样本点由一个 (X, y) 元组表示，其中粗体的 X 是特征向量，即 $X = (x_1, \dots, x_p)$ ，而其中每一个 x_j 代表股票的某一个特征，而 y 代表该样本的已知分类（通常用+1和0表示涨和跌）。SVM会根据这些给定的历史数据来训练算法的参数，找到最优的线性超平面。

理想情况下，这个超平面可以将两类涨跌完美的分开（即没有错分的情况）。对于给定的训练数据，可以将它们完美分开的超平面很可能不是唯一的，比如一个超平面稍微旋转一个角度，便得到一个仍然能够完美分割的超平面。在众多能够实现分类的超平面中，只有一个是最优的。

在实际应用中，很多数据并非是线性可分的。SVM的强大之处在于，它不仅仅局限于是一个高维空间的线性分类器。它通过非线性的核函数（kernel functions）把原始的特征空间映射到更高维的特征空间（可以是无限维的），在高维空间中再将这些样本点线性分割。

高维空间的线性分割对应着原始特征空间的非线性分割，因此在原始特征空间中生成了非线性的决策边界。此外，这么做并不以增加计算机的计算负担为代价。因此，SVM相当高效。

9.5.2 使用支持向量机进行回测

对于这部分内容，由于难度关系，只给出回测代码，有兴趣的读者可以自行研究完成。

策略的判定：

- 选取7个特征变量组成滑动窗口长度为15天的训练集，随后训练一个二分类（上涨 / 下跌）的支持向量机模型。
- 若没有仓位，则在每个星期一的时候输入标的股票近15个交易日的特征变量进行预测，并在预测结果为上涨的时候购买标的。
- 若已经持有仓位，则在盈利大于10%的时候止盈，在星期五损失大于2%的时候止损。
- 特征变量为：收盘价 / 均值、现量 / 均量、最高价 / 均价、最低价 / 均价、现量、区间收益率、区间标准差。

【程序9-7】

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
from datetime import datetime
import numpy as np
from gm.api import *
import sys

try:
    from sklearn import svm
except:
    print('请安装 scikit-learn 库和带 mkl 的 numpy')
    sys.exit(-1)

def init(context):
    # 订阅浦发银行的分钟 bar 行情
    context.symbol = 'SHSE.600000'
    subscribe(symbols=context.symbol, frequency='60s')
    start_date = '2016-03-01' # SVM 训练起始时间
    end_date = '2017-06-30' # SVM 训练终止时间
    # 用于记录工作日
```

```

# 获取目标股票的 daily 历史行情
recent_data = history(context.symbol, frequency='ld', start_time=start_date,
end_time=end_date, fill_missing='last',
                        df=True)

days_value = recent_data['bob'].values
days_close = recent_data['close'].values
days = []
# 获取行情日期列表
print('准备数据训练 SVM')
for i in range(len(days_value)):
    days.append(str(days_value[i])[0:10])

x_all = []
y_all = []
for index in range(15, (len(days) - 5)):
    # 计算三星期共 15 个交易日相关数据
    start_day = days[index - 15]
    end_day = days[index]
    data = history(context.symbol, frequency='ld', start_time=start_day,
end_time=end_day, fill_missing='last',
                    df=True)

    close = data['close'].values
    max_x = data['high'].values
    min_n = data['low'].values
    amount = data['amount'].values
    volume = []
    for i in range(len(close)):
        volume_temp = amount[i] / close[i]
        volume.append(volume_temp)
    close_mean = close[-1] / np.mean(close) # 收盘价/均值
    volume_mean = volume[-1] / np.mean(volume) # 现量/均量
    max_mean = max_x[-1] / np.mean(max_x) # 最高价/均价
    min_mean = min_n[-1] / np.mean(min_n) # 最低价/均价
    vol = volume[-1] # 现量
    return_now = close[-1] / close[0] # 区间收益率
    std = np.std(np.array(close), axis=0) # 区间标准差
    # 将计算出的指标添加到训练集 x
    # features 用于存放因子
    features = [close_mean, volume_mean, max_mean, min_mean, vol, return_now,
std]

    x_all.append(features)

# 准备算法需要用到的数据
for i in range(len(days_close) - 20):
    if days_close[i + 20] > days_close[i + 15]:

```

```

        label = 1
    else:
        label = 0
    y_all.append(label)
x_train = x_all[: -1]
y_train = y_all[: -1]
# 训练 SVM
context.clf = svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto',
coef0=0.0, shrinking=True, probability=False,
                        tol=0.001, cache_size=200, verbose=False, max_iter=-1,
                        decision_function_shape='ovr', random_state=None)
context.clf.fit(x_train, y_train)
print('训练完成!')
def on_bar(context, bars):
    bar = bars[0]
    # 获取当前年月日
    today = bar.bob.strftime('%Y-%m-%d')
    # 获取数据并计算相应的因子
    # 于星期一的 09:31:00 进行操作
    # 当前 bar 的工作日
    weekday = datetime.strptime(today, '%Y-%m-%d').isoweekday()
    # 获取模型相关的数据
    # 获取持仓
    position = context.account().position(symbol=context.symbol,
side=PositionSide_Long)
    # 如果 bar 是新的星期且没有仓位, 就开始预测
    if not position and weekday == 1:
        # 获取预测用的历史数据
        data = history_n(symbol=context.symbol, frequency='1d', end_time=today,
count=15,
                        fill_missing='last', df=True)
        close = data['close'].values
        train_max_x = data['high'].values
        train_min_n = data['low'].values
        train_amount = data['amount'].values
        volume = []
        for i in range(len(close)):
            volume_temp = train_amount[i] / close[i]
            volume.append(volume_temp)
        close_mean = close[-1] / np.mean(close)
        volume_mean = volume[-1] / np.mean(volume)
        max_mean = train_max_x[-1] / np.mean(train_max_x)
        min_mean = train_min_n[-1] / np.mean(train_min_n)

```

```

    vol = volume[-1]
    return_now = close[-1] / close[0]
    std = np.std(np.array(close), axis=0)
    # 得到本次输入模型的因子
    features = [close_mean, volume_mean, max_mean, min_mean, vol, return_now,
std]

    features = np.array(features).reshape(1, -1)
    prediction = context.clf.predict(features)[0]
    # 若预测值为上涨, 则开仓
    if prediction == 1:
        # 获取昨收盘价
        context.price = close[-1]
        # 把浦发银行的仓位调至 95%
        order_target_percent(symbol=context.symbol, percent=0.95,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
        print('SHSE.600000 以市价单开多仓到仓位 0.95')
    # 当涨幅大于 10%, 平掉所有仓位止盈
    elif position and bar.close / context.price >= 1.10:
        order_close_all()
        print('SHSE.600000 以市价单全平多仓止盈')
    # 当时间为周五并且跌幅大于 2%时, 平掉所有仓位止损
    elif position and bar.close / context.price < 1.02 and weekday == 5:
        order_close_all()
        print('SHSE.600000 以市价单全平多仓止损')
if __name__ == '__main__':
    '''
    strategy_id 策略 ID, 由系统生成
    filename 文件名, 请与本文件名保持一致
    mode 实时模式:MODE_LIVE 回测模式:MODE_BACKTEST
    token 绑定计算机的 ID, 可在系统设置-密钥管理中生成
    backtest_start_time 回测开始时间
    backtest_end_time 回测结束时间
    backtest_adjust 股票复权方式不复权:ADJUST_NONE 前复权:ADJUST_PREV 后复
权:ADJUST_POST
    backtest_initial_cash 回测初始资金
    backtest_commission_ratio 回测佣金比例
    backtest_slippage_ratio 回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',

```

```

backtest_start_time='2017-07-01 09:00:00',
backtest_end_time='2017-10-01 09:00:00',
backtest_adjust=ADJUST_PREV,
backtest_initial_cash=10000000)

```

测试结果如图9-16所示。

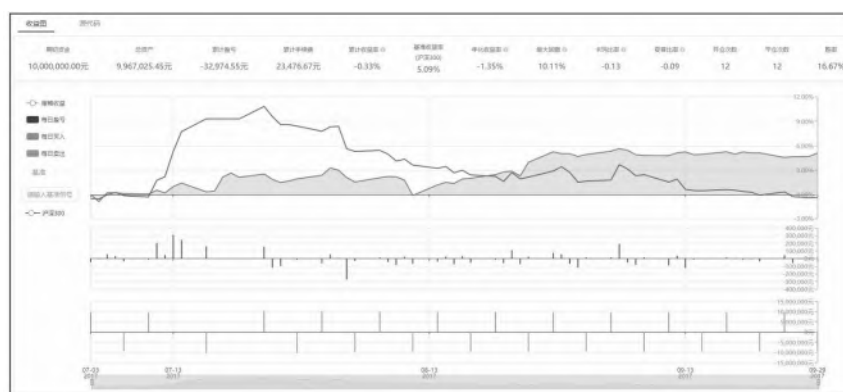


图9-16 支持向量机回测结果

9.6 小结

本章带领大家进行了线性回归的基础学习，介绍了一元回归和多元回归的基本概念，讲解了回归分析的计算方法。这些内容可能有些难度，希望读者量力而行。

本章使用了一个简单的例子去实践线性回归的拟合方法。虽然这是一个非常简单的例子，但是这个例子可以变化的部分很多，例如可以加入前一日的高低点判定以及将其改成逻辑回归，就像9.4节中的示例一样。

9.5节的支持向量机采用的是机器学习的方法，这也是未来发展的趋势。

本章主要介绍了一些常用的策略算法，无论是线性回归、逻辑回归，还是最后的支持向量机，都是通过建立模型拟合影响因子对真实结果的量化模拟，涉及开盘价、闭盘价、最高价、最低价以及交易量等基本面数据，并且涉及若干“技术性”因子。

第10章

回归模型的经典应用

回归分析是最常用的量化策略之一。前面在回归模型的基础讲解中，对使用Python进行回归模型计算做了介绍，包括使用回归模型对股价进行预测以及使用逻辑回归帮助股票买卖者进行决策等。

事实上，更多的回归模型应用于对股票的基本面数据进行分析，例如经典的CAPM模型、Fama-French三因子模型以及最新的PB_ROE模型。这些都是已经应用于现实中的金融市场并获得认同和较好的收益的经典模型。图10-1所示为回归模型的示例。

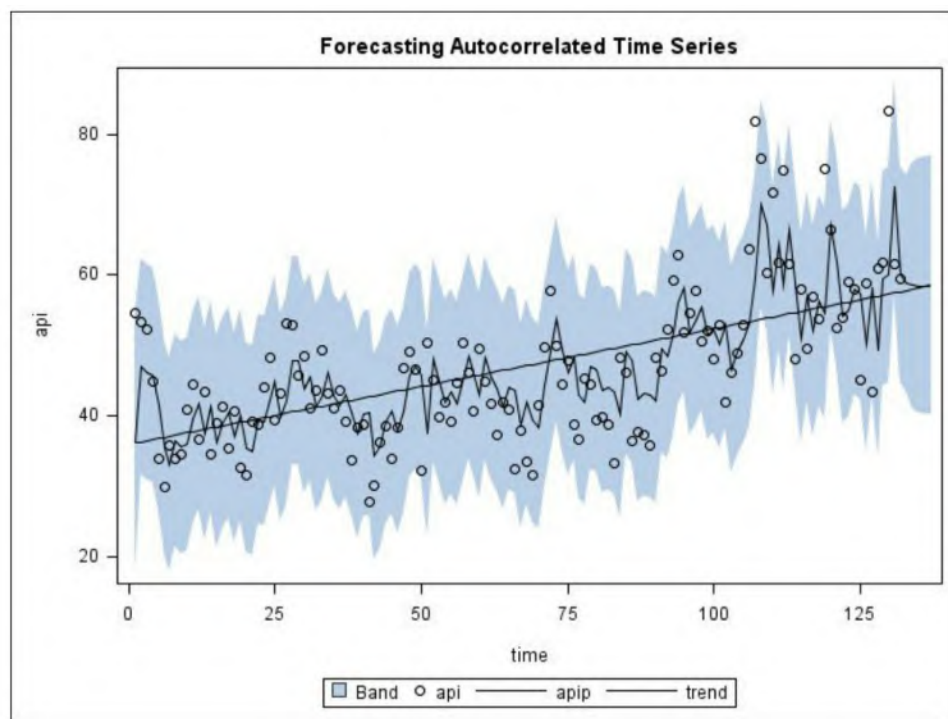


图10-1 回归模型

本章着重介绍Fama-French三因子模型和PB_ROE模型，这也是目前常用的模型方法，CAPM模型由于一些问题被Fama-French模型所替代，因此并不重点介绍，只做简单的讲解。

10.1 CAPM模型简介

资本资产定价模型（Capital Asset Pricing Model, CAPM）是由美国学者夏普（William Sharpe）、林特纳（John Lintner）、特里诺（Jack Treynor）和莫辛（Jan Mossin）等人于1964年在资产组合理论和资本市场理论的基础上发展起来的。

证券市场中资产的预期收益率与风险资产之间的关系，以及均衡价格的形成原因，是现代金融市场价格的理论支柱，广泛应用于投资决策和公司理财领域。

CAPM资产定价模型假设所有投资者都按马科维茨的资产选择理论进行投资，对期望收益、方差和协方差等的估计完全相同，投资人可以自由借贷。基于这样的假设，CAPM资产定价模型研究的重点在于探求风险资产收益与风险的数量关系，即为了补偿某一特定程度的风险，投资者应该获得多少的报酬率。

10.1.1 CAPM定价模型的提出

马科维茨（Markowitz）的分散投资与效率组合投资理论，第一次以严谨的数理工具向人们展示了一个风险厌恶的投资者在众多风险资产中构建最优资产组合的方法。应该说，这一理论带有很强的规范（normative）意味，告诉投资者应该如何进行投资选择。

但问题是，在20世纪50年代，即便有了当时刚刚诞生的电脑的帮助，在实践中应用马科维茨的理论仍然是一项烦琐、令人生厌的高难度工作；或者说，与投资的现实世界脱节得过于严重，进而很难完全被投资者采用。美国普林斯顿大学的鲍莫尔（William Baumol）在其1966年的一篇探讨马科维茨—托宾体系的论文中就谈到，按照马科维茨的理论，即使以较简化的模式，要从1500只证券中挑选出有效率的投资组合，当时每运行一次电脑需要耗费150美元~300美元，而如果要执行完整的马科维茨运算，所需的成本至少是前述金额的50倍，而且还必须有一个前提，就是分析师必须能够持续且精确地估计标的证券的预期报酬、风险及相关系数，否则整个运算过程将变得毫无意义。

正是由于这一问题的存在，从20世纪60年代初开始，以夏普、林特纳和莫辛为代表的经济学家开始从实证的角度出发，探索证券投资的现实，即马科维茨的理论在现实中的应用能否得到简化。如果投资者都采用马科维茨资产组合理论选择最优资产组合，那么资产的均衡价格将如何在收益与风险的权衡中形成？或者说，在市场均衡状态下，资产的价格如何依风险而确定？

这些学者的研究直接导致了CAPM的产生。作为基于风险资产期望收益均衡基础的预测模型之一，CAPM阐述了在投资者都采用马科维茨的理论进行投资管理的条件下市场均衡状态的形成，把资产的预期收益与预期风险之间的理论关系用一个简单的线性关系表达出来，即认为资产的预期收益率与衡量该资产风险尺度的 β 值之间存在正相关关系。

应该说，作为一种阐述风险资产均衡价格决定的理论，单一指数模型或以之为基础的CAPM不仅大大简化了投资组合选择的运算过程，使马科维茨的投资组合选择理论朝现实世界的应用迈进了一大步，而且也使得证券理论从以往的定性分析转入定量分析，从规范性转入实

证性，进而对证券投资的理论研究和实际操作，甚至整个金融理论与实践的发展都产生了巨大影响，成为现代金融学的理论基础。

当然，近几十年，作为资本市场均衡理论模型关注的焦点，CAPM的形式已经远远超越了夏普、林特纳和莫辛提出的传统形式，有了很大的发展，如套利定价模型、跨时资本资产定价模型、消费资本资产定价模型等，目前已经形成了一个较为系统的资本市场均衡理论体系。

10.1.2 CAPM定价模型的公式与假设

当资本市场达到均衡时，风险的边际价格是不变的，任何改变市场组合的投资所带来的边际效果是相同的，即增加一个单位的风险所得到的补偿是相同的。按照 β 的定义，代入均衡资本市场条件下，得到资本资产定价模型。

1. CAPM的定价模型

定价模型如下：

$$R_a = r_f + \beta_a \times (r_m - r_f)$$

- r_f (Risk Free Rate) 是无风险回报率（相对于一年期国债）。
- β_a 是证券的Beta系数。
- r_m 是市场期望的回报率 (Expected Market Return)。
- $r_m - r_f$ 是股票市场溢价 (Equity Market Premium, EMP)。

CAPM公式中的右边第一个是无风险收益率，其一般使用一年期国债。如果股票投资者需要承受额外的风险，那么将需要在无风险回报率的基础上多获得相应的溢价。股票市场溢价就等于市场期望回报率

减去无风险回报率。证券风险溢价就是股票市场溢价和一个 β 系数的乘积。

2. CAPM定价模型的假设

但是事实上，CAPM定价模型需要大量的假设，归类如下：

- 投资者希望财富越多越好，效用是财富的函数，财富又是投资收益率的函数，因此可以认为效用为收益率的函数。
- 投资者能事先知道投资收益率的概率分布为正态分布。
- 投资风险用投资收益率的方差或标准差标识。
- 影响投资决策的主要因素为期望收益率和风险两项。
- 投资者都遵守主宰原则（Dominance Rule），即同一风险水平下，选择收益率较高的证券；同一收益率水平下，选择风险较低的证券。
- 可以在无风险折现率 R 的水平下无限制地借入或贷出资金。
- 所有投资者对证券收益率概率分布的看法一致，因此市场上的效率边界只有一条。
- 所有投资者具有相同的投资期限，而且只有一期。
- 所有的证券投资可以无限制地细分，在任何一个投资组合里可以含有非整数股份。
- 买卖证券时没有税负及交易成本。
- 所有投资者可以及时免费获得充分的市场信息。
- 不存在通货膨胀，且折现率不变。
- 投资者具有相同预期，即他们对预期收益率、标准差和证券之间的协方差具有相同的预期值。

上述假设表明：第一，投资者是理性的，而且严格按照马科维茨模型的规则进行多样化的投资，并将从有效边界的某处选择投资组

合；第二，资本市场是完美 / 完全市场，没有任何摩擦阻碍投资。

但是问题在于，这些假设太过于完美主义，任何一个现实中的交易者或者现实中的股票交易场所都不可能达到CAPM的要求。

3. CAPM资本资产定价模型的优缺点

(1) 优点

- CAPM最大的优点在于简单、明确。它把任何一种风险证券的价格都划分为三个因素：无风险收益率、风险的价格和风险的计算单位，并把这三个因素有机结合在一起。
- CAPM的另一优点在于它的实用性。它使投资者可以根据绝对风险而不是总风险来对各种竞争报价的金融资产做出评价和选择。这种方法已经被金融市场上的投资者广为采纳，用来解决投资决策中的一般性问题。

(2) 局限性

当然，CAPM也不是尽善尽美的，它本身存在着一定的局限性，表现在：

- 假设市场处于完善的竞争状态。但是，实际操作中完全竞争的市场是很难实现的，“做市”时有发生。
- 假设投资者的投资期限相同且不考虑投资计划期之后的情况。但是，市场上的投资者数目众多，他们的资产持有期间不可能完全相同，而且现在进行长期投资的投资者越来越多，所以这个假设就变得不那么现实了。
- 假设投资者可以不受限制地以固定的无风险利率借贷，这一点也是很难办到的。

- 假设市场无摩擦。但实际上，市场存在交易成本、税收和信息不对称等问题。
- “理性人假设”和“一致预期假设”。这两个假设也是一种理想状态。

10.1.3 CAPM中Beta的定义

CAPM中的Beta值与在前面用于衡量股票变动性的Beta值相似，这里请读者注意。

按照CAPM的规定，Beta系数是用以度量一项资产系统风险的指针，是用来衡量一种证券或一个投资组合相对总体市场的波动性（volatility）的一种风险评估工具。

也就是说，如果一只股票的价格和市场的价格波动性是一致的，那么这只股票的Beta值就是1。如果一只股票的Beta值是1.5，就意味着当市场上升10%时，该股票价格上升15%；而市场下降10%时，股票的价格亦会下降15%。

Beta值是通过统计分析同一时期市场每天的收益情况以及单个股票每天的价格收益来计算的。1972年，经济学家费歇尔·布莱克（Fischer Black）、迈伦·斯科尔斯（Myron Scholes）等在他们发表的论文《资本资产定价模型：实例研究》中，通过研究1931年到1965年纽约证券交易所股票价格的变动，证实了股票投资组合的收益率和它们的Beta值间存在着线性关系，如图10-2所示。

当Beta值处于较高位置时，投资者便会因为股份的风险高而相应地提升股票的预期回报率。例如，如果一个股票的Beta值是2.0，无风险回报率是3%，市场回报率（Market Return）是7%，那么市场溢价（Equity Market Premium）就是4%（7%-3%），股票风险溢价（Risk

Premium) 为8% ($2 \times 4\%$, 用Beta值乘以市场溢价), 那么股票的预期回报率则为11% ($8\% + 3\%$, 即股票的风险溢价加上无风险回报率)。

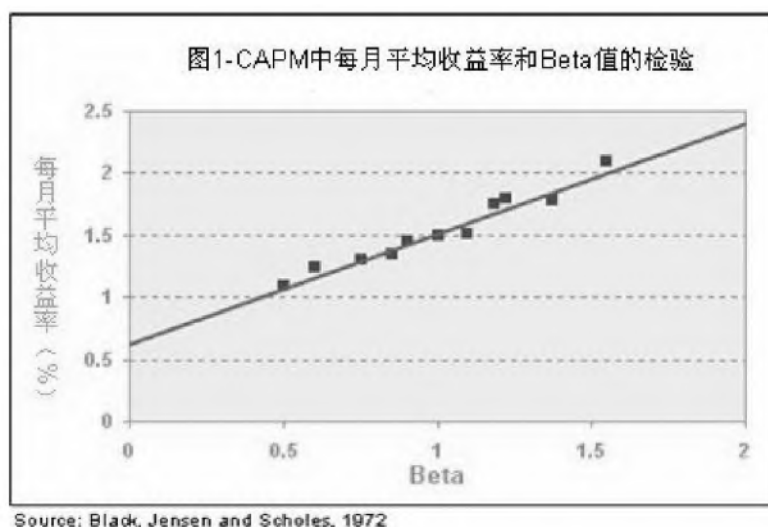


图10-2 CAPM中的Beta值

理论上来说, 一个风险投资者需要得到的溢价可以通过CAPM计算出来。换句话说, 我们可以通过CAPM知道当前股票的价格是否与其回报相吻合。

但是事实上, 在计算某些股票时, 由于缺乏历史数据, 其Beta值不易估计。此外, 由于经济不断发展变化, 各种证券的Beta值也会产生相应的变化, 依靠历史数据估算出的Beta值对未来的指导作用也要打折扣。因此, Beta值的计算有时候难以确认。

对于CAPM模型的设计和回测, 这里不再详细介绍, 有兴趣的读者可自行完成。

10.2 Fama-French三因子模型

CAPM作为一种资产定价模型统治了金融市场很长一段时间。直到后来，Fama和French（见图10-3）两个人研究股票超额收益率的时候发现了一个神奇的现象：有两类股票的历史平均收益率会高于一般人们所预测的收益率。它们是小公司股票以及具有较高股权账面市值比的股票。

Fama和French分别分析了原因，他们认为：

- 市值比较小的公司通常规模比较小，公司相对而言没那么稳定，因此风险较大，需要获得更高的收益来补偿。
- 此外就是账面市值比。账面市值比就是账面的所有者权益除以市值（以下简称B/M）。若B/M较高，则说明市场上对公司的估值比公司自己的估值更低。这些公司一般都是销售状况或者盈利能力不是十分好的公司，因此相对于低B/M的公司来说需要更高的收益来补偿。

一般对于股票收益的解释认为收益风险同源。市场风险是唯一能给股票带来超额收益的风险。但是基于以上两个事实的研究发现，除了市场风险外，Fama-French认为市场上还存在市值风险、账面市值比风险等，据此建立的模型被称为“Fama-French三因子模型”。本节旨在深入浅出地介绍三因子模型的思想并提供一个实用的三因子策略。



图10-3 Fama和French

10.2.1 Fama-French模型的基础公式

1992年，Fama和French对美国股票市场决定不同股票回报率差异的因素的研究发现，对于传统的股票模型来说，股票市场的Beta值不能解释不同股票回报率的差异。在此基础上，有实证研究表明，股票市值、账面市值比、财务杠杆（leverage）和市盈率的倒数（E/P）等指标可以很好地解释股票收益。

于是，Fama和French在1992年发表了一篇文章，把这些因子都考虑进去，研究1963年～1990年在NYSE、AMEX和NASDAQ交易的股票（除金融类股票外）的平均收益和这些因子的关系。在横截面回归（cross-section regression）后发现，在独立检验4者对平均收益的影响时，4者都表现出了很强的解释能力；而在进行多变量回归时，市值和账面市值比这两个因子吸收了另外两个因子的解释能力，成为解释平均收益的决定性变量。公式如下：

$$R_i = a_i + b_i R_m + s_i E(\text{SMB}) + l_i E(\text{HMI}) + \varepsilon_i$$

这里的 R_i 指的是股票相对无风险投资（一年期国债）的期望额外收益率。

$$R_i = E(r_i - r_f)$$

此外，公式中还有一个额外参数 R_m ，为市场相对无风险投资的期望超额收益率。

$E(\text{SMB})$ 是小市值公司相对大市值公司股票的期望超额收益率， $E(\text{HMI})$ 则是高B/M公司股票比起低B/M公司股票的期望超额收益率，而最后一项是回归残差项。

10.2.2 Fama-French模型的实现与回测

根据上文的介绍，使用Fama-French模型需要对涉及的因子进行相关分析，总结如下：

- 计算市场收益率、个股的账面市值比和市值，并对后两个进行分类。
- 根据分类得到的组合分别计算其市值加权收益率、SMB和HML。
- 对各个股票进行回归（假设无风险收益率等于0）得到Alpha值。
- 选取Alpha值小于0，且将Alpha值由小到大排列后，值最小的10只股票进入标的池。
- 平掉不在标的池的股票并等权买入在标的池的股票。

使用掘金量化的代码实现如下：

【程序10-1】

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
import numpy as np
from gm.api import *
from pandas import DataFrame

def init(context):
    # 每月第一个交易日的 09:40 定时执行 algo 任务
    schedule(schedule_func=algo, date_rule='1m', time_rule='09:40:00')
    print(order_target_percent(symbol='SHSE.600000', percent=0.5,
order_type=OrderType_Market,
                                position_side=PositionSide_Long))

    # 数据滑窗
    context.date = 20
    # 设置开仓的最大资金量
    context.ratio = 0.8
    # 账面市值比的大/中/小分类
    context.BM_BIG = 3.0
    context.BM_MID = 2.0
    context.BM_SMA = 1.0
    # 市值大/小分类
    context.MV_BIG = 2.0
    context.MV_SMA = 1.0
```

```

# 计算市值加权的收益率，MV 为市值的分类，BM 为账目市值比的分类
def market_value_weighted(stocks, MV, BM):
    select = stocks[(stocks.NEGOTIABLEMV == MV) & (stocks.BM == BM)]
    market_value = select['mv'].values
    mv_total = np.sum(market_value)
    mv_weighted = [mv / mv_total for mv in market_value]
    stock_return = select['return'].values
    # 返回市值加权的收益率的和
    return_total = []
    for i in range(len(mv_weighted)):
        return_total.append(mv_weighted[i] * stock_return[i])
    return_total = np.sum(return_total)
    return return_total

def algo(context):
    # 获取上一个交易日的日期
    last_day = get_previous_trading_date(exchange='SHSE', date=context.now)
    # 获取沪深 300 成分股
    context.stock300 = get_history_constituents(index='SHSE.000300',
start_date=last_day,

end_date=last_day)[0]['constituents'].keys()
    # 获取当天有交易的股票
    not_suspended = get_history_instruments(symbols=context.stock300,
start_date=last_day, end_date=last_day)
    not_suspended = [item['symbol'] for item in not_suspended if not
item['is_suspended']]
    fin = get_fundamentals(table='tq_sk_finindic', symbols=not_suspended,
start_date=last_day, end_date=last_day,
fields='PB,NEGOTIABLEMV', df=True)
    # 计算账面市值比，为 P/B 的倒数
    fin['PB'] = (fin['PB'] ** -1)
    # 计算市值的 50%的分位点，用于后面的分类
    size_gate = fin['NEGOTIABLEMV'].quantile(0.50)
    # 计算账面市值比的 30%和 70%分位点，用于后面的分类
    bm_gate = [fin['PB'].quantile(0.30), fin['PB'].quantile(0.70)]
    fin.index = fin.symbol
    x_return = []
    # 对未停牌的股票进行处理
    for symbol in not_suspended:
        # 计算收益率
        close = history_n(symbol=symbol, frequency='1d', count=context.date + 1,

```

```

end_time=last_day, fields='close',
                                skip_suspended=True, fill_missing='Last',
adjust=ADJUST_PREV, df=True)['close'].values
    stock_return = close[-1] / close[0] - 1
    pb = fin['PB'][symbol]
    market_value = fin['NEGOTIABLEMV'][symbol]
    # 获取[股票代码, 股票收益率, 账面市值比的分类, 市值的分类, 流通市值]
    if pb < bm_gate[0]:
        if market_value < size_gate:
            label = [symbol, stock_return, context.BM_SMA, context.MV_SMA,
market_value]
        else:
            label = [symbol, stock_return, context.BM_SMA, context.MV_BIG,
market_value]
        elif pb < bm_gate[1]:
            if market_value < size_gate:
                label = [symbol, stock_return, context.BM_MID, context.MV_SMA,
market_value]
            else:
                label = [symbol, stock_return, context.BM_MID, context.MV_BIG,
market_value]
        elif market_value < size_gate:
            label = [symbol, stock_return, context.BM_BIG, context.MV_SMA,
market_value]
        else:
            label = [symbol, stock_return, context.BM_BIG, context.MV_BIG,
market_value]
    if len(x_return) == 0:
        x_return = label
    else:
        x_return = np.vstack([x_return, label])
    stocks = DataFrame(data=x_return, columns=['symbol', 'return', 'BM',
'NEGOTIABLEMV', 'mv'])
    stocks.index = stocks.symbol
    columns = ['return', 'BM', 'NEGOTIABLEMV', 'mv']
    for column in columns:
        stocks[column] = stocks[column].astype(np.float64)
    # 计算 SMB.HML 和市场收益率
    # 获取小市值组合的市值加权组合收益率
    smb_s = (market_value_weighted(stocks, context.MV_SMA, context.BM_SMA) +
        market_value_weighted(stocks, context.MV_SMA, context.BM_MID) +
        market_value_weighted(stocks, context.MV_SMA, context.BM_BIG)) / 3
    # 获取大市值组合的市值加权组合收益率

```

```

smb_b = (market_value_weighted(stocks, context.MV_BIG, context.BM_SMA) +
        market_value_weighted(stocks, context.MV_BIG, context.BM_MID) +
        market_value_weighted(stocks, context.MV_BIG, context.BM_BIG)) / 3
smb = smb_s - smb_b
# 获取大账面市值比组合的市值加权组合收益率
hml_b = (market_value_weighted(stocks, context.MV_SMA, 3) +
        market_value_weighted(stocks, context.MV_BIG, context.BM_BIG)) / 2
# 获取小账面市值比组合的市值加权组合收益率
hml_s = (market_value_weighted(stocks, context.MV_SMA, context.BM_SMA) +
        market_value_weighted(stocks, context.MV_BIG, context.BM_SMA)) / 2
hml = hml_b - hml_s
close = history_n(symbol='SHSE.000300', frequency='1d', count=context.date
+ 1,
                    end_time=last_day, fields='close', skip_suspended=True,
                    fill_missing='Last', adjust=ADJUST_PREV,
df=True)['close'].values
market_return = close[-1] / close[0] - 1
coff_pool = []
# 对每只股票进行回归获取其 alpha 值
for stock in stocks.index:
    x_value = np.array([[market_return], [smb], [hml], [1.0]])
    y_value = np.array([stocks['return'][stock]])
    # OLS 估计系数
    coff = np.linalg.lstsq(x_value.T, y_value)[0][3]
    coff_pool.append(coff)
# 获取 alpha 最小并对小于 0 的 10 只股票进行操作 (若少于 10 只, 则全部买入)
stocks['alpha'] = coff_pool
stocks = stocks[stocks.alpha < 0].sort_values(by='alpha').head(10)
symbols_pool = stocks.index.tolist()
positions = context.account().positions()
# 平不在标的池的股票
for position in positions:
    symbol = position['symbol']
    if symbol not in symbols_pool:
        order_target_percent(symbol=symbol, percent=0,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
        print('市价单平不在标的池的', symbol)
# 获取股票的权重
percent = context.ratio / len(symbols_pool)
# 买在标的池中的股票
for symbol in symbols_pool:
    order_target_percent(symbol=symbol, percent=percent,

```

```

order_type=OrderType_Market,
                position_side=PositionSide_Long)
    print(symbol, '以市价单调多仓到仓位', percent)
if __name__ == '__main__':

    run(strategy_id='63e92f59-1386-11e8-bbe9-902b3463caf1',
        filename='多因子选股.py',
        mode=MODE_BACKTEST,
        token='a71a8083b68e73817e93f7f196b030482abe5939',
        backtest_start_time='2017-01-03 08:00:00',
        backtest_end_time='2017-12-31 16:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=1000000,
        )

```

这里根据Fama-French模型的思想对市值进行分类，分成高、中、低三部分，之后根据市值对其进行计算。这是最常用的一种市值计算方法。最终结果如图10-4所示。

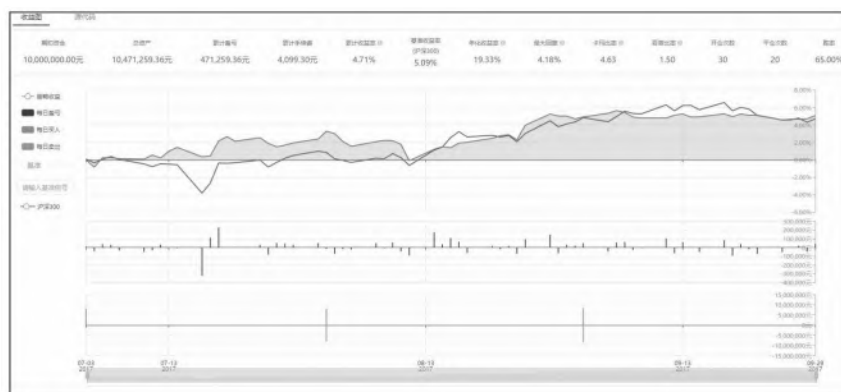


图10-4 Fama-French模型回测结果

可以看到，相对在回测时期，Fama-French模型获得了比较好的成功率，可以较好地获取超额收益。

三因子模型对投资界有深远的影响，将股票按市值和账面市值比这样的特征进行划分，就是其中一例。股票按照市值大小划分为小盘股、中盘股和大盘股，按账面市值比划分为价值型、平衡型和成长

型。而衍生的股票指数的编制方式、基金持股风格的划分也受到三因子模型的影响。

Fama-French在1993年提出三因子模型之后，Carhart在1997年提出了动量因子（Momentum）从而得到四因子模型，Fama-French 2015年在三因子的基础上继续增加了两个因子：盈利能力因子RMW和投资因子CMA，得到五因子模型。三因子模型开启了人们对因子投资的研究，随着对因子的认识不断扩展，除了有风格因子外，还有策略因子，比如介于被动投资和主动投资之间的Smart Beta策略。

10.3 PB-ROE回归模型的使用

PB-ROE是另一种回归模型。

众所周知，选股的核心思想在于寻找价格低于内在价值的股票，从而获取未来价格修复的收益。具体来看，符合价值投资理念的标的首先要具有良好的基本面，其次要有合理的价格。PB-ROE选股模型便是价值投资中的一种经典方法。该模型中，ROE作为基本面指标，用于衡量公司质地是否优良，PB作为估值指标，用于衡量当前价格是否低估。

10.3.1 PB-ROE模型介绍

PB-ROE是英文Price/Book ratio-Return On Equity的缩写。其中PB与ROE的含义如下：

$$PB = \text{股价} / \text{账面价值}$$

$$ROE = [(\text{净利润} - \text{优先股股利}) / \text{期初普通股股东权益}] * 100\%$$

账面价值的含义是：总资产－无形资产－负债－优先股权益。而所谓的账面价值，是公司解散清算的价值。因为如果公司清算，那么债要先还，无形资产则将不复存在，而优先股的优先权之一就是清算的时候先分钱。

一般来说，用每股净资产来代替账面价值，PB就用市净率来替代，而ROE又叫股权收益率，ROE使用净资产收益率来替代。

因此，可以简单地理解PB-ROE就是“市净率－净资产收益率”。这个模型的意思是，净资产收益率高的股票，其实净率也应该高些。从图10-5（可参看本书下载包中的图片文件）可以看到，某年的A股市场中，根据PB-ROE回归图示化的结果，大多数的股票都在回归曲线的上方，从模型上看，其值都被高估。

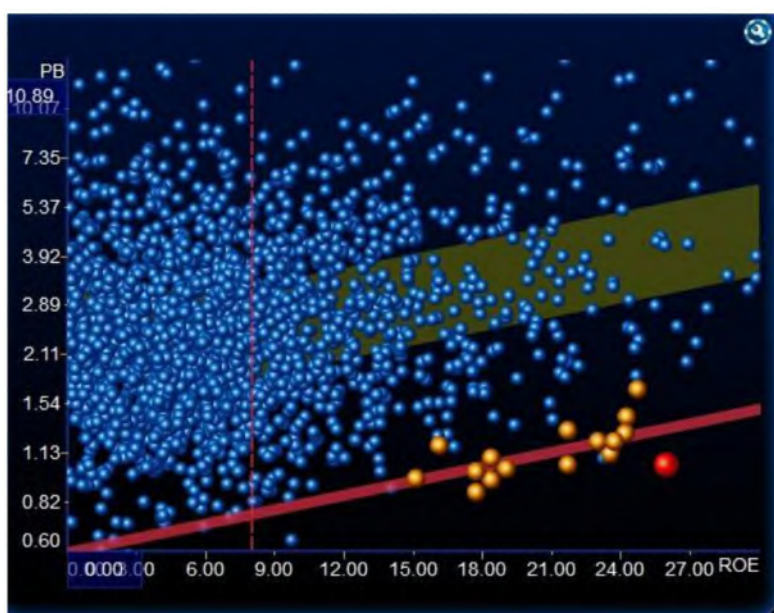


图10-5 201X年市场PB-ROE模型图示

但是对于严重落在PB-ROE回归模型曲线下方的标的股票，并不能简单地认为就是被严重低估，而要综合考察其对应当时的基本面和外部条件是否有严重变化，这样才能选择到好的标的。

10.3.2 PB-ROE模型的实现

下面使用掘金量化实现PB-ROE模型。

第一步：数据表中PB值的获取

10.3.1小节介绍了PB-ROE模型的基本概念，对于模型所要使用的数据，掘金量化数据库中有完整的数据存档，在这里直接使用即可。

通过查表可得，ROE使用的是ROEAVGCUT，其在deriv_finance_indicator中。而PB在trading_derivative_indicator表中，查询函数如下：

```
def get_fundamentals(table, symbols, start_date, end_date,
fields=None,filter=None, order_by=None, limit=1000,
df=False):
```

这里需要注意的是，table是对应的表名，而symbols是批量使用股票列表值同时获取对应的数据。

第二步：获取对应股票列表的PB值

```

_df = get_fundamentals(table='trading_derivative_indicator',
symbols=symbol_list, start_date=last_day, end_date=last_day,fields="PB",
df=True)
    if len(_df) == len(symbol_list):
        df["PB"] = _df["PB"]
    else:
        for number in range(len(symbol_list)):
            try:
                _df = get_fundamentals(table='trading_derivative_indicator',
symbols=symbol_list[number], start_date=last_day,end_date=last_day,
fields="PB")
                _factor_value = tools.get_data_value(_df, "PB")

                df.iloc[number, 1] = _factor_value[0]
            except:
                df.iloc[number, 1] = np.mean(df["PB"])

```

这里首先使用get_fundamentals函数读取对应的股票列表的PB值，之后对读取的数据进行确认，这样做的目的是防止数据产生确实值，如果测试中的数据长度不等于股票列表的长度，就使用for循环单独获取对应的PB值，如果产生确实值，就使用已有列表的均值对其进行填充。

而ROE数据的获取与PB获取一样，代码段如下：

```

_df = get_fundamentals(table='deriv_finance_indicator', symbols=symbol_list,
start_date=last_day, end_date=last_day, fields="ROEAVG", df=True)
    if len(_df) == len(symbol_list):
        df["ROE"] = _df["ROEAVG"]
    else:
        for number in range(len(symbol_list)):
            try:
                _df = get_fundamentals(table='deriv_finance_indicator',
symbols=symbol_list[number], start_date=last_day,end_date=last_day,
fields="ROEAVG")
                _factor_value = tools.get_data_value(_df, "ROEAVG")

                df.iloc[number, 2] = _factor_value[0]
            except:
                df.iloc[number, 2] = np.mean(df["ROE"])

```

下面使用前面介绍的OLS方法对数据进行回归分析，代码如下：

```
df = df.dropna()
pb_ = df["PB"].values # 这是Y
roe_ = df["ROE"].values # 这是X

roe = []
for _ in roe_:
    roe.append(_)
pb = []
for _ in pb_:
    pb.append(_)

_roe = sm.add_constant(roe)

model = sm.OLS(pb, _roe)
results = model.fit()

bias, weight = (results.params)
```

首先清理缺失值，之后提取对应的PB与ROE值组成对应数列，OLS回归计算对应的系数组成相应的回归曲线，如图10-6所示。

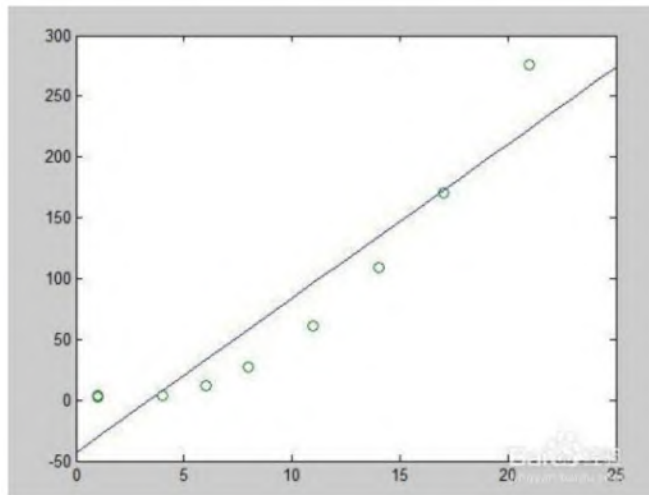


图10-6 经过OLS拟合后的PB-ROE曲线

得到对应的参数后，计算单只股票真实PB与拟合PB的残差，代码如下：

```
df["new_PB"] = bias + df["ROE"]*weight
df["d_value"] = df["PB"] - df["new_PB"]    #这里 d_value 是残差，大于 0 是高估，小于 0
是低估

df = (df.sort_values(["d_value"]))

symbol_list = []

for _ in df["symbol"].values:
    symbol_list.append(_)
```

这里计算的是对应的残差值，一般认为残差值大于0是股票被高估，而残差值小于0是被低估。按照残差值由小到大排列，返回列表。

PB_ROE核心代码如下：

```
def PB_ROE 选股(_symbol_list, now):
    last_day = get_previous_trading_date("SHSE", now)
```

```

symbol_list = []
bank_list = []
stock_company_list = []
house_company_list = []
wine_list = []
for _ in _symbol_list:
    if (_ not in bank_list) and (_ not in stock_company_list) and (_ not in
house_company_list) and (_ not in wine_list):
        symbol_list.append(_)

    _df = get_fundamentals(table='deriv_finance_indicator', symbols=symbol_list,
start_date=last_day, end_date=last_day, fields='TAGRT,NPGRT', filter="NPGRT >
21 and TAGRT > 20", df=True)
    _df = _df.dropna()
    symbol_list = []
    for _ in _df["symbol"].values:
        symbol_list.append(_)
    white_list = []
    symbol_list = symbol_list + white_list

df = pd.DataFrame([])
df["symbol"] = symbol_list

df["PB"] = -999
df["ROE"] = -999

# 求 PB
_df = get_fundamentals(table='trading_derivative_indicator',
symbols=symbol_list, start_date=last_day, end_date=last_day, fields="PB",
df=True)
if len(_df) == len(symbol_list):
    df["PB"] = _df["PB"]
else:
    for number in range(len(symbol_list)):
        try:
            _df = get_fundamentals(table='trading_derivative_indicator',
symbols=symbol_list[number], start_date=last_day, end_date=last_day,
fields="PB")
            _factor_value = tools.get_data_value(_df, "PB")

            df.iloc[number, 1] = _factor_value[0]
        except:
            df.iloc[number, 1] = np.mean(df["PB"])

```

```

# 求 ROE
_df = get_fundamentals(table='deriv_finance_indicator', symbols=symbol_list,
start_date=last_day, end_date=last_day, fields="ROEAVG", df=True)
if len(_df) == len(symbol_list):
    df["ROE"] = _df["ROEAVG"]
else:
    for number in range(len(symbol_list)):
        try:
            _df = get_fundamentals(table='deriv_finance_indicator',
symbols=symbol_list[number], start_date=last_day, end_date=last_day,
fields="ROEAVG")
            _factor_value = tools.get_data_value(_df, "ROEAVG")

            df.iloc[number, 2] = _factor_value[0]
        except:
            df.iloc[number, 2] = np.mean(df["ROE"])

df = df.dropna()
pb_ = df["PB"].values # 这是 Y
roe_ = df["ROE"].values # 这是 X

roe = []
for _ in roe_:
    roe.append(_)
pb = []
for _ in pb_:
    pb.append(_)

_roe = sm.add_constant(roe)

model = sm.OLS(pb, _roe)
results = model.fit()
bias, weight = (results.params)
# print(bias, " ", weight)
df["new_PB"] = bias + df["ROE"] * weight
df["d_value"] = df["PB"] - df["new_PB"] # 这里 d_value 是残差, 大于 0 是高估, 小于 0 是低估
df = (df.sort_values(["d_value"]))
symbol_list = []

for _ in df["symbol"].values:
    if (_ not in bank_list) and (_ not in stock_company_list) and (_ not in
house_company_list) and (_ not in wine_list):

```



```
symbol_list.append(_)\n\nreturn symbol_list
```

可能有读者会注意到，这里在导入股票列表的时候，分别设置了3个黑名单，即证券、银行、地产和酒类（地产和酒类合并在一起）相关股票，属于这三大板块就自动剔除。

这样做的目的是，根据研究，股票市场PB-ROE产生不匹配的股票标的大多数集中在证券和银行方面，而酒类由于长期不被投资者关注，也容易产生不匹配的现象。

10.3.3 基于上证180的股票回测

前面已经介绍过，不同的指数由于其代表标的的不同，其入选的指标和格式也不尽相同。

上证180指数（又称上证成份指数，指数代码000010）是上海证券交易所对原上证30指数进行了调整并更名而成的，其样本股是在所有A股股票中抽取最具市场代表性的180种样本股票，自2002年7月1日起正式发布。

作为上证指数系列核心的上证180指数的编制方案，目的在于建立一个反映上海证券市场的概貌和运行状况、具有可操作性和投资性、能够作为投资评价尺度及金融衍生产品基础的基准指数。

下面将使用上证180进行回测工作。

第一步：解析股票池

进行回测工作的第一步是对指数构成的股票进行解析，掘金量化提供了专用的函数对其进行解析，代码如下：

```

def get_symbol_list(index,now):
    try:
        symbol_list = get_history_constituents(index=index,
start_date=now)[0].get("constituents").
keys()
        symbol_list_not_suspended = get_history_instruments(symbols=symbol_list,
start_date=now, end_date=now)
        symbol_list = [item['symbol'] for item in symbol_list_not_suspended if
not item['is_suspended']]

        _symbol_list = symbol_list
        symbol_list = []
        for _ in _symbol_list:
            symbol_list.append(_)
    except:print(index)
    return symbol_list

```

这里get_history_constituents函数根据日期对指数进行解析，之后判断当日的个股停牌状态，接着使用一个list列表将数据读取出来并返回。

第二步：PB_ROE的核心选股代码

PB_ROE核心选股代码采用上一小节中PB_ROE排序的计算方法，在这里添加了对酒类、地产和银行等部分需要剔除的标的的代码，同时增加了白名单功能，加入原本不在解析指数内的股票。

```

def PB_ROE 选股(_symbol_list, now):
    last_day = get_previous_trading_date("SHSE", now)

    symbol_list = []
    bank_list = tools.get_symbol_list("SHSE.000947", now)
    stock_company_list = tools.get_symbol_list("SZSE.399975", now)
    house_company_list = tools.get_symbol_list("SHSE.000948", now)
    wine_list = tools.get_symbol_list("SZSE.399987", now)
    for _ in _symbol_list:
        if (_ not in bank_list) and (_ not in stock_company_list) and (_ not in
house_company_list) and (_ not in wine_list):
            symbol_list.append(_)

    _df = get_fundamentals(table='deriv_finance_indicator', symbols=symbol_list,
start_date=last_day, end_date=last_day, fields='TAGRT,NPGRT', filter="NPGRT >
21 and TAGRT > 20", df=True)
    _df = _df.dropna()
    symbol_list = []
    for _ in _df["symbol"].values:
        symbol_list.append(_)

    white_list = []
    symbol_list = symbol_list + white_list

    df = pd.DataFrame([])
    df["symbol"] = symbol_list

    df["PB"] = -999
    df["ROE"] = -999

    # 求 PB
    _df = get_fundamentals(table='trading_derivative_indicator',
symbols=symbol_list, start_date=last_day, end_date=last_day, fields="PB",
df=True)
    if len(_df) == len(symbol_list):
        df["PB"] = _df["PB"]

```

```

else:
    for number in range(len(symbol_list)):
        try:
            _df = get_fundamentals(table='trading_derivative_indicator',
symbols=symbol_list[number], start_date=last_day, end_date=last_day,
fields="PB")
            _factor_value = tools.get_data_value(_df, "PB")

            df.iloc[number, 1] = _factor_value[0]
        except:
            df.iloc[number, 1] = np.mean(df["PB"])

# 求 ROE
_df = get_fundamentals(table='deriv_finance_indicator', symbols=symbol_list,
start_date=last_day, end_date=last_day, fields="ROEAVG", df=True)
if len(_df) == len(symbol_list):
    df["ROE"] = _df["ROEAVG"]
else:
    for number in range(len(symbol_list)):
        try:
            _df = get_fundamentals(table='deriv_finance_indicator',
symbols=symbol_list[number], start_date=last_day, end_date=last_day,
fields="ROEAVG")
            _factor_value = tools.get_data_value(_df, "ROEAVG")

            df.iloc[number, 2] = _factor_value[0]
        except:
            df.iloc[number, 2] = np.mean(df["ROE"])

df = df.dropna()
pb_ = df["PB"].values # 这是 Y
roe_ = df["ROE"].values # 这是 X

roe = []
for _ in roe_:
    roe.append(_)
pb = []
for _ in pb_:
    pb.append(_)

_roe = sm.add_constant(roe)
model = sm.OLS(pb, _roe)
results = model.fit()

```

```
bias, weight = (results.params)
# print(bias, " ", weight)

df["new_PB"] = bias + df["ROE"] * weight
df["d_value"] = df["PB"] - df["new_PB"] # 这里d_value是残差, 大于0是高估, 小于0是低估
df = (df.sort_values(["d_value"]))
symbol_list = []

for _ in df["symbol"].values:
    if (_ not in bank_list) and (_ not in stock_company_list) and (_ not in
house_company_list) and (_ not in wine_list):
        symbol_list.append(_)

return symbol_list
```

第三步：使用PB_ROE进行回测

下面使用PB_ROE为核心进行股票回测，代码如下：

【程序10-2】

```
import numpy as np
from gm.api import *
import fun
import os
import csv
from MSCI_tools import msci_tools as tools

import datetime
back_thresord = 0.97

def init(context):
    schedule(schedule_func=algo, date_rule='1d', time_rule='09:31:00')

    context.time_now = str(str(datetime.datetime.now()).strftime('%Y-%m-%d
%H:%M:%S')).split(" "))
    print(context.time_now)
    context.symbol_high = {}
    context.symbol_low = {}
    context.num = 8 #测试 7 8 9 最好, 夏普对应 0.82    0.91    0.89
    context.count = 1

    context.black_list = {}
    context.flag = True
```

```

def algo(context):

    now = context.now
    last_day = get_previous_trading_date("SHSE", now)

    target_list = tools.get_symbol_list("SHSE.000010",now)
    """核心选股代码"""
    _symbol_list = fun.get_target_list(target_list, now)
    """核心选股代码"""

    symbol_list = []
    for symbol in _symbol_list:
        symbol_list.append(symbol)

    # 取前多少个标的
    target_list = symbol_list[:context.num]

    # -----市价单平不在标的池的
    positions = context.account().positions()
    # 如标的池有仓位，平不在标的池的股票仓位

    for position in positions:
        symbol = position['symbol']

        if (symbol not in target_list):
            order_target_percent(symbol=symbol, percent=0,
order_type=OrderType_Market,position_side=PositionSide_Long)

    positions = context.account().positions()
    # 判断已经持有的股
    holded_symbol = []
    for position in positions:
        symbol = position['symbol']
        holded_symbol.append(symbol)

    for symbol in target_list:
        if (symbol not in holded_symbol):
            data = history_n(symbol, frequency="1d", count=2, end_time=now,
fields="close,open", df=True)
            open = data["open"].values
            close = data["close"].values
            if open[-1] < close[0] * 1.08: # 这里是对涨停板进行甄别，就用这个，用多少天

```

的运算太慢

```
context.symbol_high[symbol] = 0
context.symbol_low[symbol] = 9999
order_target_percent(symbol=symbol, percent=(1. /
(context.num))*0.95, order_type=OrderType_Market,
                        position_side=PositionSide_Long)

else:
    pass

def on_backtest_finished(context, indicator):
    print(context.time_now, " ", indicator)

if __name__ == "__main__":
    run(
        strategy_id='e98538bc-9378-11e8-9fbc-902b3463caf1',
        filename=(os.path.basename(__file__)),
        mode=MODE_BACKTEST,
        token='e8978d765c4822e5a85fcaa73e044065cf17b58b',
        backtest_start_time="2017-01-01 09:30:00",
        backtest_end_time='2017-12-31 15:00:00',
        backtest_initial_cash=10000000,
        backtest_adjust=ADJUST_PREV
    )
```

这里使用上证180对2017年全年的数据进行回测，每日对前5标的股票进行确定，当股票超出前5后剔除，买入新进入的股票。回测结果如图10-7所示。

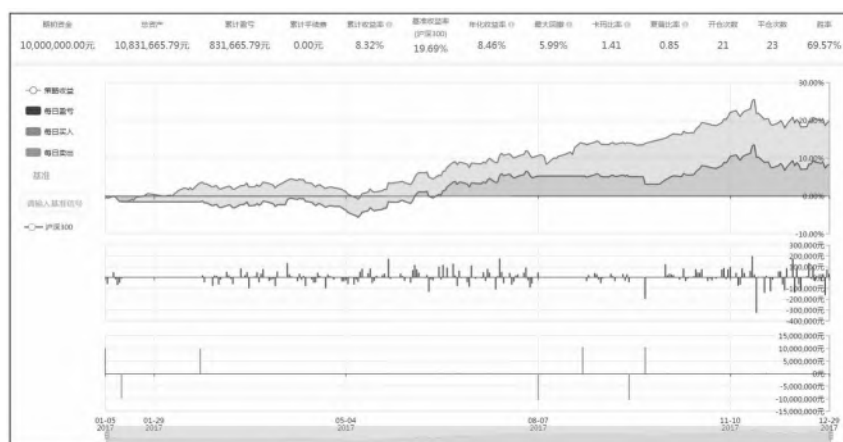


图10-7 2017年PB_ROE为核心的数据回测结果

可以看到，对于使用上证180进行回测，其并没有跑赢当年的沪深300指数，这可能由于当时属于“漂亮50”的权重股涨幅期间造成的普通股涨幅不大。

更多指数回测请读者自行完成。

10.3.4 使用自定义股票池的PB-ROE回测

上证180是基于已有的指数组合进行股票回测，但是对于不同的投资者来说，根据自己对股票的认识和要求，选择的股票标的也有所不同，因此对于不同的投资者来说，可以根据不同的认知组合不同的股票池。

对于不同的股票组合，一般采用条件选股的模式进行组合。本小节中核心股票的选择方法是PB-ROE模型，因此在选择上以ROE为主。

第一步：获取全部A股代码

实际上，对于全部3000多只A股来说，部分缺乏流动性或者ST以及其他原因不能够被大多数投资者认可的股票并不具有一定的投资价值，因此在选择上可以剔除。在这里选用的是基于中证组合的A股入选标的的股票，指数代码如下：

```
index_list_中证规模 = {  
    "SHSE.000010": "上证 180", "SHSE.000016": "上证 50", "SHSE.000300": "沪深 300",  
    "SHSE.000903": "中证 100", "SHSE.000904": "中证 200", "SHSE.000905": "中证 500",  
    "SHSE.000906": "中证 800", "SHSE.000907": "中证 700", "SHSE.000852": "中证 1000"}  
}
```

根据指数解析股票构成组合的代码如下：

```

#这里是获取全 A 股的股票列表
def get_symbol_set(now):
    index_list = index_lists.index_list_中证规模
    symbol_list = []

    for index in index_list:
        _symbol_list = tools.get_symbol_list(index, now)
        symbol_list += _symbol_list

    bank_list = tools.get_symbol_list("SHSE.000947", now)
    stock_company_list = tools.get_symbol_list("SZSE.399975", now)
    house_company_list = tools.get_symbol_list("SHSE.000948", now)
    wine_list = tools.get_symbol_list("SZSE.399987", now)
    symbol_set = set()
    for _ in symbol_list:
        if (_ not in bank_list) and (_ not in stock_company_list) and (_ not in
house_company_list) and (
            _ not in wine_list):
            symbol_set.add(_)
    print("symbol_set 建立完毕")
    return symbol_set

```

为了剔除重复的股票代码，使用了set数据集作为存储容器，其好处是不会产生重复项。

第二步：ROE/ROIC的方差小于阈值

ROE的中文解释是净资产收益率，ROIC是资本回报率，其比值应该在1的上下浮动，并且方差小于一个给定的阈值。这里还需要注意的是，ROE和ROIC是基于股票标的基本面公布的数据，每个季度更新一次，因此这里的count是向前多少个季度的意思。

代码如下：

```

#第一步：获取 ROE/ROIC
symbol_ROE_ROIC_list = []
for symbol in symbol_set:
    try:
        _df = get_fundamentals_n(table='deriv_finance_indicator', symbols=symbol,
count=count, end_date=last_day, fields="ROEAVGCUT,ROIC", df=True)

        if len(_df) < count - 2:
            pass
        else:
            _df = _df.dropna()
            _df["盈利性"] = _df["ROEAVGCUT"] / _df["ROIC"]

            factor = _df["盈利性"].values

            dta = []

            for _ in factor:
                dta.append(_)

            dta_std = np.std(dta)

            if dta_std < std:
                symbol_ROE_ROIC_list.append(symbol)
    except:pass

```

第三步：PETTM在一个正常水平，即 $11 < \text{PETTM} < 27$

PETTM是滚动市盈率的意思，一般要求正常的市盈率在11~27之间，代码如下：

```

df = get_fundamentals_n(table='trading_derivative_indicator',
symbols=symbol_PSTTM_list, count=1, end_date=last_day, fields="PETTM", df=True)
df = df.dropna()
df = df[df["PETTM"] > 11]
df = df[df["PETTM"] < 27]

symbol_PETTM_list = []
for symbol in df["symbol"].values:
    symbol_PETTM_list.append(symbol)

```

第四步：NPGRT和TAGRT增长率复合要求

```
df = get_fundamentals_n(table='deriv_finance_indicator',
symbols=symbol_PETTM_list, count=1, end_date=last_day, fields="TAGRT,NPGRT",
df=True)

df = df.dropna()
df = df[(df["TAGRT"] > 10) & (df["TAGRT"] < 39) & (df["NPGRT"] > 10) &
(df["NPGRT"] < 39) ]

symbol_GRT_list = []
for symbol in df["symbol"].values:
    symbol_GRT_list.append(symbol)

target_list = symbol_GRT_list
```

全部股票池构建代码如下：

```
def get_target_list_base(now , count = 14, std = 0.11):

    last_day = get_previous_trading_date("SHSE",now)

    symbol_set = get_symbol_set(last_day)
    target_list = []

    #第一步：获取ROE/ROIC std 校园
    symbol_ROE_ROIC_list = []
    for symbol in symbol_set:
        try:
            _df = get_fundamentals_n(table='deriv_finance_indicator',
symbols=symbol, count=count, end_date=last_day,
fields="ROEAVGCUT,ROIC", df=True)

            if len(_df) < count - 2:
                pass
            else:
                _df = _df.dropna()
                _df["盈利性"] = _df["ROEAVGCUT"] / _df["ROIC"]
```

```

        factor = _df["盈利性"].values
        dta = []
        for _ in factor:
            dta.append(_)
        dta_std = np.std(dta)
        if dta_std < std:
            symbol_ROE_ROIC_list.append(symbol)
    except:pass

#第二步：这里假设 PETTM 在正常水平，即 11 < PETTM < 27
df = get_fundamentals_n(table='trading_derivative_indicator',
symbols=symbol_ROE_ROIC_list, count=1, end_date=last_day, fields="PETTM",
df=True)
df = df.dropna()
df = df[df["PETTM"] > 11]
df = df[df["PETTM"] < 27]

symbol_PSTTM_list = []
for symbol in df["symbol"].values:
    symbol_PSTTM_list.append(symbol)

#第三步：NPGRT TAGRT
df = get_fundamentals_n(table='deriv_finance_indicator',
symbols=symbol_PETTM_list, count=1, end_date=last_day, fields="TAGRT,NPGRT",
df=True)

df = df.dropna()
df = df[(df["TAGRT"] > 10) & (df["TAGRT"] < 39) & (df["NPGRT"] > 10) &
(df["NPGRT"] < 39) ]

symbol_GRT_list = []
for symbol in df["symbol"].values:
    symbol_GRT_list.append(symbol)

target_list = symbol_GRT_list

return target_list

```

需要注意的是，这里股票池是作者根据自己的认知所构建的，count为向前多少个季度的计数。

第五步：建立自定义股票池

自定义股票池中使用了ROE与ROIC的比值，这两个真实值是在季报中出现的，因此更新时采用的是每月更新一次，即选择特定的日期作为股票池更新日期，日期选择如下：

```
data_list_2014 = [  
    "2017-01-13", "2017-02-15", "2017-03-15", "2017-04-14", "2017-05-15",  
    "2017-06-15",  
    "2017-07-14", "2017-08-15", "2017-09-15", "2017-10-13", "2017-11-15",  
    "2017-12-15",  
    "2018-01-15", "2018-02-14", "2018-03-15", "2018-04-13", "2018-05-15",  
    "2018-06-15",  
    "2018-07-13"]
```

每月15日更新数据，但是需要注意的是，对于15日当日休市的股票，作者选择前一个交易日的数据进行更新。

【程序10-3】

```

from gm.api import *
from MSCI_tools import msci_tools as tools
import numpy as np
from MSCI_tools import 指数列表 as index_lists
#这里是获取全 A 股的股票列表
def get_symbol_set(now):
    index_list = index_lists.index_list_中证规模
    symbol_list = []

    for index in index_list:
        _symbol_list = tools.get_symbol_list(index, now)
        symbol_list += _symbol_list

    bank_list = tools.get_symbol_list("SHSE.000947", now)
    stock_company_list = tools.get_symbol_list("SZSE.399975", now)
    house_company_list = tools.get_symbol_list("SHSE.000948", now)
    wine_list = tools.get_symbol_list("SZSE.399987", now)
    symbol_set = set()
    for _ in symbol_list:
        if (_ not in bank_list) and (_ not in stock_company_list) and (_ not in
house_company_list) and (
            _ not in wine_list):
            symbol_set.add(_)
    print("symbol_set 建立完毕")
    return symbol_set
def get_target_list_base(now , count = 14, std = 0.11):

    last_day = get_previous_trading_date("SHSE",now)

```

```

symbol_set = get_symbol_set(last_day)
target_list = []

#第一步: 获取 ROE/ROIC std 校园
symbol_ROE_ROIC_list = []
for symbol in symbol_set:
    try:
        _df = get_fundamentals_n(table='deriv_finance_indicator',
symbols=symbol, count=count, end_date=last_day,
                                fields="ROEAVGCUT,ROIC", df=True)

        if len(_df) < count - 2:
            pass
        else:
            _df = _df.dropna()
            _df["盈利性"] = _df["ROEAVGCUT"] / _df["ROIC"]

            factor = _df["盈利性"].values

            dta = []

            for _ in factor:
                dta.append(_)

            dta_std = np.std(dta)

            if dta_std < std:
                symbol_ROE_ROIC_list.append(symbol)
    except:pass

#print(len(symbol_ROE_ROIC_list)) 155

#第二步: 这里假设 PETTM 在正常水平, 即 11 < PETTM < 27
df = get_fundamentals_n(table='trading_derivative_indicator',
symbols=symbol_ROE_ROIC_list, count=1, end_date=last_day,
                        fields="PETTM", df=True)

df = df.dropna()
df = df[df["PETTM"] > 11]
df = df[df["PETTM"] < 27]

symbol_PSTTM_list = []
for symbol in df["symbol"].values:
    symbol_PSTTM_list.append(symbol)

```



```

#第三步: NPGRT TAGRT
df = get_fundamentals_n(table='deriv_finance_indicator',
symbols=symbol_PETTM_list, count=1, end_date=last_day,
                        fields="TAGRT,NPGRT", df=True)

df = df.dropna()
df = df[(df["TAGRT"] > 10) & (df["TAGRT"] < 39) & (df["NPGRT"] > 10) &
(df["NPGRT"] < 39) ]

symbol_GRT_list = []
for symbol in df["symbol"].values:
    symbol_GRT_list.append(symbol)

target_list = symbol_GRT_list

return target_list
data_list_2016 = [
    "2016-08-15", "2016-09-14", "2016-10-14", "2016-11-15", "2016-12-15",
    "2017-01-13", "2017-02-15", "2017-03-15", "2017-04-14", "2017-05-15",
    "2017-06-15",
    "2017-07-14", "2017-08-15", "2017-09-15", "2017-10-13", "2017-11-15",
    "2017-12-15",
    "2018-01-15", "2018-02-14", "2018-03-15", "2018-04-13", "2018-05-15",
    "2018-06-15",
    "2018-07-13"]

if __name__ == "__main__":

    set_token("e8978d765c4822e5a85fcaa73e044065cf17b58b")

    data_and_target_list = []

    for data in data_list_2016:
        target_list = get_target_list_base(data)
        while len(target_list) == 0:
            data = get_previous_trading_date("SZSE", data)
            target_list = get_target_list_base(data)
        data_and_target_list.append(target_list)
        print(data, " ", len(target_list), " ", target_list)

    print("-----")

```

```
file_name = "data_and_target_list_2016" + ".npy"  
np.save(file_name, data_and_target_list)
```

这里根据2016年至2018年4月的数据做出股票池，之后将其存储为data_and_target_list_2016.npy文件，读者可以自行完成。

第六步：使用自定义股票池进行PB_ROE模型选股

下面使用自定义的PB_ROE模型对股票池进行回测。首先需要注意的是，由于使用按月组建股票池的策略，因此选择每日对数据进行判定回测的话，需要对日期进行判定，代码如下：

【程序10-4】

```

import numpy as np
from gm.api import *
import fun
import os
import csv
from MSCI_tools import msci_tools as tools

import datetime
data_list_2016 = [
    "2016-08-15", "2016-09-14", "2016-10-14", "2016-11-15", "2016-12-15",
    "2017-01-13", "2017-02-15", "2017-03-15", "2017-04-14", "2017-05-15",
    "2017-06-15",
    "2017-07-14", "2017-08-15", "2017-09-15", "2017-10-13", "2017-11-15",
    "2017-12-15",
    "2018-01-15", "2018-02-14", "2018-03-15", "2018-04-13", "2018-05-15",
    "2018-06-15",
    "2018-07-13"]

file_data = "data_and_target_list_2016.npy"
arr_list = np.load(file_data)
back_thresord = 0.97
def init(context):
    schedule(schedule_func=algo, date_rule='1d', time_rule='09:31:00')

    context.time_now = str(str(datetime.datetime.now()).strftime('%Y-%m-%d
%H:%M:%S')).split(" "))
    print(context.time_now)
    context.symbol_high = {}
    context.symbol_low = {}
    context.num = 8
    context.count = 1

```

```

context.black_list = {}
context.record_file = "record.csv"

context.data_and_symbol_list = {}
for _1,_2 in zip(data_list_2016, arr_list):
    context.data_and_symbol_list[_1] = _2

context.counter = -1
context.flag = True

def algo(context):

    now = context.now
    day_time, hour_and_mins = str(now).split(" ")

    last_day = get_previous_trading_date("SHSE", now)
    targrt_day = str(day_time).split("-")[-1]

    if int(targrt_day) > 15:
        if context.flag:
            context.counter += 1
            context.flag = False
    else:
        context.flag = True

    target_list = context.data_and_symbol_list[data_list_2016[context.counter]]
    print(day_time," ",data_list_2016[context.counter]) #不要打印了
    """核心选股代码"""
    _symbol_list = fun.get_target_list(target_list, now)
    """核心选股代码"""

    # 这里并没有缩小股票池，将其移到下面取标的之后才缩小股票池
    symbol_list = []
    for symbol in _symbol_list:
        symbol_list.append(symbol)

    # 取前多少个标的
    target_list = symbol_list[:context.num] # 无黑名单存在，卖掉以后又可以买回

    # -----市价单平不在标的池的
    positions = context.account().positions()
    # 如标的池有仓位，平不在标的池的股票仓位

```

```

for position in positions:
    symbol = position['symbol']

    if (symbol not in target_list):
        order_target_percent(symbol=symbol, percent=0,
order_type=OrderType_Market,position_side=PositionSide_Long)

positions = context.account().positions()
# 判断已经持有的股
holded_symbol = []
for position in positions:
    symbol = position['symbol']
    holded_symbol.append(symbol)

for symbol in target_list:
    if (symbol not in holded_symbol):
        data = history_n(symbol, frequency="1d", count=2, end_time=now,
fields="close,open", df=True)
        open = data["open"].values
        close = data["close"].values
        if open[-1] < close[0] * 1.08: # 这里是对涨停板进行甄别,就用这个,用多少天
的运算太慢
            context.symbol_high[symbol] = 0
            context.symbol_low[symbol] = 9999
            order_target_percent(symbol=symbol, percent=(1. /
(context.num))*0.95, order_type=OrderType_Market,
                position_side=PositionSide_Long)
        else:
            pass

def on_backtest_finished(context, indicator):
    print(context.time_now," ",indicator)
    res = [context.time_now,file_data,back_thresord,indicator["sharp_ratio"],
indicator["max_drawdown"], indicator["pnl_ratio"],
        indicator["pnl_ratio_annual"],
        indicator]
    writer = csv.writer(open(context.record_file, 'a+', encoding='utf8',
newline=''))
    writer.writerow(res)

if __name__ == "__main__":
    run(

```

```
strategy_id='e98538bc-9378-11e8-9fbc-902b3463caf1',
filename=(os.path.basename(__file__)),
mode=MODE_BACKTEST,
token='e8978d765c4822e5a85fcaa73e044065cf17b58b',
backtest_start_time="2016-09-03 09:30:00",
backtest_end_time='2018-04-20 15:00:00',
backtest_initial_cash=10000000,
backtest_adjust=ADJUST_PREV
)
```

这里需要注意作者使用的编程技巧，在init函数中形成对应于日期的字典格式文件，之后在每日回测的主程序中对日期进行判定，早于当月15日就使用上一次组建的股票池，而晚于15日则使用当前的股票池。回测结果请读者自行完成。

10.4 小结

本章主要讨论了回归模型的基础和用法，这是目前最常用的量化分析方法之一。同时，本章还综合运用多因子选股模型为回归分析组成提供了股票池。

可以看到，对于不同的投资者来说，其资金关注的侧重点不同，那么组成的股票池也会有千差万别的选择，这里建议选择那些长期稳定高增长的股票，会给投资者带来丰厚的收益。

多因子和回归分别是量化投资里最主要也是最重要的两个部分，其分别代表对未来的预期以及对现实的反映，这两种量化分析方法相辅相成，既可以交互，也可以独立使用，其决定于投资者的认知能力。

下一章会介绍一个“另类”的量化投资方法，即“配对交易”，有兴趣的读者可以深入学习。

第11章

配对交易的魔力

大家小时候应该都玩过跷跷板，两个小朋友分作两边，一头翘起来，一头沉下去，一头再翘起来，一头再沉下去，最后两个小朋友都玩得很高兴。

如果说股市交易有一种工具能够像跷跷板这样，有两个“跷跷板”标的，当一头的标的价格低的时候，去买入价格低的投资产品；而当买入的标的价格抬起时，卖出高价股票，买入另一头的低价股票，等待低的那头被抬起后卖出。

可能这里的表述不是很正确，将其引申为本章的配对交易来说，就是基于两只股票的价差来盈利，对于不同的两只股票，一般而言会存在一定的价差，而由于其具有特定的相关性，股价的价差长期来看是保持一种稳定的状态。

但是在实际市场上，由于交易员的非理性行为或者突发事件的存在，打破了这种稳定的价差。

也就是说，两只标的（股票）的价差长期稳定，但是在某个特定事件会出现价差扩大的现象。配对交易就是利用这种价差变化的机会进行牟利，卖出相对被高估的股票，买入相对被低估的股票，等价差减少，股价波动重新趋于稳定时，结束交易并获取收益。

配对策略是一种经典的量化交易策略，配对交易首先要找到合适的配对交易资产标的，而在筛选配对资产的过程中，需要充分掌握一些基本的统计学知识。

11.1 配对交易的基本理论

配对交易实际上就是一种基于统计学的统计建模过程。

进行配对策略最常用的是统计学方面的知识，这里涉及相关系数、均值方差等方面的内容，相信读者应该不陌生。但是除此之外，还有一个新的概念——“协整”。这是为配对交易引入的在计量经济学中特定出现的一个新的名称。本节将对可能涉及的概念进行统一介绍。

11.1.1 相关性分析

相关性的统计与分析是经济学中常用的一种方法。相关性是指当两个因素之间存在联系时，一个典型的表现是：一个变量会随着另一个变量变化。相关又分成正相关和负相关两种情况。

本节我们不讨论正负的相关性，而是要看不同的数据集和中相关性的绝对量。相关性的绝对值越大，可以认为相关性越高。一般来说，取绝对值后，0~0.09为没有相关性，0.1~0.3为弱相关，0.3~0.5为中等相关，0.5~1.0为强相关。

这里不介绍具体的相关性公式，只介绍使用的方法和代码设计。

第一步：创建数据集

为了简便起见，这里首先使用恒定的线性方程表示两组数据集：

```
arr_a = np.arange(0,100,1)
arr_b = 3 * arr_a + 1
```

这里使用了NumPy作为数据的创建库，之后从0~100中取出100个数据组成一个数组：


```
[0,1,2,3...]
```

而arr_b是通过arr_a计算出的数据集，这里采用一个线性方程对数据进行计算，并存储数据计算结果。

第二步：计算相关系数

对于相关系数的计算，NumPy中提供了专门的函数corrcoef，用于对其内容的相关性进行计算，其结果打印如下：

```
[[ 1.  1.]  
 [ 1.  1.]
```

除了常用的NumPy之外，Python还提供了其他方法计算数据的相关系数。pandas是另一种常用的计算库文件，同样，也提供了对不同数据集之间的相关度的计算方法。

```
import pandas as pd  
arr_a = pd.DataFrame({"arr_a":arr_a})  
arr_b = pd.DataFrame({"arr_b":arr_b})
```

这里首先导入了pandas库包，之后根据需要生成了对应的DataFrame文件，并将其起名为对应的名称。

这样还不行，对于pandas来说，相关系数并不是由pandas根函数提供的，而是由DataFrame对象内置的计算器计算出的。因此，需要将不同的DataFrame数组组建成一个新的DataFrame数据集。

```
arr = pd.concat([arr_a,arr_b],axis=1)
```

concat是连接函数，将不同的数据进行连接，axis是内置参数，用以决定从哪里进行连接，这里由于需要计算不同的数组之间的相关系数，因此选择将数据连接成不同的行，计算不同行之间的相关系数。

```
arr = pd.concat([arr_a, arr_b], axis=1)
corr = arr.corr()
print(corr)
```

这里通过concat函数生成了一个新的数据集。需要注意的是，这里在生成新的数据集时，行名称也是被保留的。

通过内置的函数，corr可以计算数据集之间的相关系数，最终打印结果如下：

```
      arr_a  arr_b
arr_a    1.0    1.0
arr_b    1.0    1.0
```

这里可以看到，除了给出结果外，还加上了对应的名称，这样可以极大地帮助选取对应的数值进行计算。

通过上面的例子可以看到，对于不同的数组，根据相关性计算可以得到对应的相关系数。现在换一部分值重新计算其对应的相关性。

【程序11-1】

```
import numpy as np
import pandas as pd

arr_a = np.arange(0,100,1)
arr_b = np.random.randint(0,100,100)

arr_a = pd.DataFrame({"arr_a":arr_a})
arr_b = pd.DataFrame({"arr_b":arr_b})

arr = pd.concat([arr_a, arr_b], axis=1)
corr = arr.corr()

print(corr)
```

【程序11-1】中，arr_a的数值没有变化，之后arr_b通过一个随机函数生成了100个100以内的随机数。其他相关性的计算方法没有变

化，最后打印结果如下：

```
      arr_a    arr_b
arr_a 1.000000 0.060579
arr_b 0.060579 1.000000
```

可以看到，对于采用随机的配对数据，相关系数变得很小，基本上可以忽略不计。因此可以说，两组数据并没有相关性。

11.1.2 均值、方差与协方差

均值又称为平均数，是表示一组数据集中趋势的量数，是指在一组数据中所有数据之和再除以这组数据的个数。它是反映数据集中趋势的一项指标。

NumPy中也提供了计算均值的函数。

【程序11-2】

```
import numpy as np

arr_a = np.arange(0,100,1)
mean = np.mean(arr_a)
print(mean)
```

最终打印结果如下：

```
49.5
```

在使用概率论和统计方差衡量随机变量或一组数据时，方差用于度量离散程度。统计学中，方差用来度量随机变量和其数学期望（均值）之间的偏离程度。

【程序11-3】

```
import numpy as np

arr_a = np.arange(0,100,1)
var = np.var(arr_a)
print(var)
```

最终打印结果如下：

```
833.25
```

可以看到，数据的方差很大，这表明对于数据来说，数值并不是在均值上下均匀分布，而是具有很强的扩散性。

顺便提一下，更多的时候比较数据的均衡性使用的是标准差，也就是方差的平方根。

标准差在概率统计中最常用于统计分布程度上的测量。标准差定义是总体各单位标准值与其平均数离差平方的算术平均数的平方根。它反映组内个体间的离散程度。

【程序11-4】

```
import numpy as np

arr_a = np.arange(0,100,1)
std = np.std(arr_a)
print(std)
```

请读者自行打印完成。

均值描述的是样本集合的中间点，它告诉我们的信息是很有限的，而标准差描述的是样本集合的各个样本点到均值的距离的平均。

例如[0, 8, 12, 20]和[8, 9, 11, 12]，两个集合的均值都是10，但显然两个集合差别很大，计算两者的标准差，前者是8.3，后者是1.8，显然后者较为集中，故其标准差小一些，标准差描述的就是这种“散布度”。

协方差是关于如何调节协变量对因变量的影响效应，从而更加有效地分析实验处理效应的一种统计技术，也是对实验进行统计控制的一种综合方差分析和回归分析的方法。

相对于用于描述一维数据的标准差和均值，协方差主要用于描述多维数据的数据集之间的关系，其定义公式也是仿照方差的公式而来的。同样使用pandas计算不同的数据集之间的协方差，代码如下：

【程序11-5】

```
import numpy as np
import pandas as pd

arr_a = np.arange(0,100,1)
arr_b = np.random.randint(0,100,100)

arr_a = pd.DataFrame({"arr_a":arr_a})
arr_b = pd.DataFrame({"arr_b":arr_b})

arr = pd.concat([arr_a,arr_b],axis=1)
cov = arr.cov()

print(cov)
```

这里同样是随机生成了两组数据集，之后使用DataFrame函数生成既定格式的数据，再将其组合生成一个新的数据组。

cov是DataFrame自带的分析函数，用于分析不同的数据集之间的协方差。打印结果如下：

```
      arr_a      arr_b
arr_a  841.666667  -8.045455
arr_b  -8.045455  834.825354
```

总结一下，协方差、相关系数是紧密相关的，二者都是用来描述两个连续变量的线性相关关系。它们的不同点在于：

- 协方差只表示线性相关的方向，取值正无穷到负无穷。也就是说，协方差为正值，说明一个变量变大，另一个变量也变大；取负值说明一个变量变大，另一个变量变小；取0说明两个变量没有相关关系。需要注意的是，协方差的绝对值不反映线性相关的程度。
- 相关系数不仅表示线性相关的方向，还表示线性相关的程度，取值 $[-1, 1]$ 。也就是说，相关系数为正值，说明一个变量变大，另一个变量也变大；取负值说明一个变量变大，另一个变量变小；取0说明两个变量没有相关关系。同时，相关系数的绝对值越接近1，线性关系越显著。通常情况下，取绝对值后， $0 \sim 0.09$ 为没有相关性， $0.1 \sim 0.3$ 为弱相关， $0.3 \sim 0.5$ 为中等相关， $0.5 \sim 1.0$ 为强相关。

11.2 协整性的判定与检验

配对交易是一种基于统计建模分析的交易策略，它是通过计算不同股票之间是否具有同样的趋势与走势进行股票交易的策略。

在股市交易中，由于人们的心理作用以及行业内部的周期相似性，同行业中的股票往往具有相同的趋势，同时涨跌受与本行业相关的消息等因素的影响。即使股价在表现上有所偏离，但是最终还是会趋于一致。这种性质被称为“协整性”。

11.2.1 协整性

协整性的解释较为复杂，这里先看两只股票在2015年的走势图，如图11-1所示。



图11-1 两只股票2015年的走势图

从图11-1中可以看到，两只股票非常相似，在跌的时候一起下跌，而涨的时候会有共同上涨的趋势。更进一步可以看到，两只股票的价差较为平稳，其在变化过程中，前进的方向也是一致的。这种性质被称为“协整性”。

在继续探讨协整性之前，提出一个新词——平稳性。简单来说，平稳性就是在一个时间序列中不随外界噪音改变而能够保持稳定不变的性质。

图11-2中有张图，可以看到上图即使随着时间变动，其值也始终围绕着均值上下波动，形成一个框体；而下图是一个非平稳序列，它并不能形成一个有效的框体，均值也是随时间而变动的。

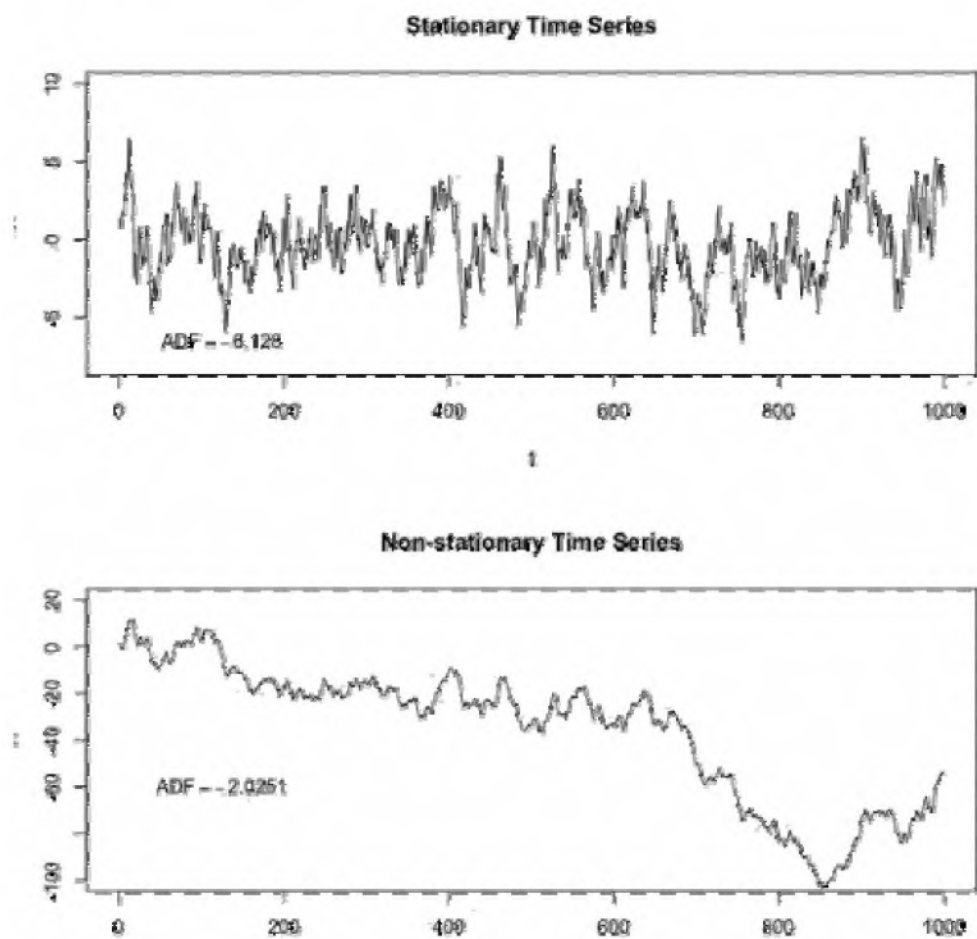


图11-2 平稳序列和非平稳序列图

前面阐述了协方差与相关系数之间的关系，这里还要说明协整性与相关系数之间的关系。虽然它们比较相似，但是表述的却不是同样的内容，相关系数是对两组数据集直接的相关性进行计算，而协整性是在不同的数据之间计算其差值并对其差值进行分析。

前面介绍了可能会用到的协整性的理论，关于相关性、均值、协方差等的计算和公式，这里就不再阐述了。下一小节主要介绍关于协整性的检验和计算，会涉及一些公式，如果读者觉得理解有困难，可予以略过。

11.2.2 平稳性的检验方法

协整性的第一步是对序列进行平稳性检验。

一般来说，平稳性分为严平稳性和弱平稳性。严平稳性是指一个序列的分布函数始终不变，而弱平稳性是指序列具有不变的统计常量。一般说的平稳性是弱平稳性。在时间序列分析中，常用“单位根检验”来判断一个过程是否为弱平稳性。

具体单位根检验的方法已经超出了本书的范围，有兴趣的同学可以参考“时间序列分析”等方面的图书。

下面介绍如何使用Python进行单位根检验。直接编写程序进行单位根检验的方法固然可取，但是Python中有专门的库包进行单位根检验。statsmodels是一个专门用于统计建模的工具集，它包含时间序列的单位根检验函数，因此可以直接对其进行调用。

为了更好地解释协整性的验证和处理，这里采用分布代码段的形式向读者说明。

第一步：创建数据集

这里使用NumPy创建一个非平稳数据集，采用一个新的函数cumsum。cumsum是一个计算累计数的函数，比如 $\text{cumsum}(c(1, 2, 3, 4, 5)) = (1, 1+2, 1+2+3, 1+2+3+4, \dots) = (1, 3, 6, 10, \dots)$ ，这点请读者自行测试。

【程序11-6】

```
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm

arr_a = np.random.normal(0, 1, 250)
arr_a = np.cumsum(arr_a) + 50

for i in range(250):
    arr_a[i] = arr_a[i] - i / 10
plt.plot(X)
plt.show()
```

arr_a和arr_b是生成的两个数组，arr_a有一个向上的趋势，而arr_b在趋势中加上扰动，其生成结果如图11-3所示。



图11-3 随机非平稳序列

第二步：检测序列的平稳性

在前面已经介绍过了，对平稳性检测需要先对其进行单位根检测，statsmodels提供的adfuller函数可以很好地完成单位根检测的计算。其代码如下：

```
import statsmodels.api as sm
st = sm.tsa.stattools.adfuller(arr_a)
```

```
print(st)
```

打印结果如下：

```
(-0.032544928790406935, 0.95575645193568282, 0, 249, {'1%': -3.4568881317725864, '5%': -2.8732185133016057, '10%': -2.5729936189738876}, 705.93885714074645)
```

这里有5个“,”（大括号内是一个整体），将打印结果分成6部分，第一个参数为“Test Statistic”，第四个参数为测试的关键参数点。

对于判定结论，读者需要记住的是，若第一个参数小于第四个参数中的“1%”所对应的值，则可认为序列没有单位根，为平稳序列；若第一个参数大于第四个参数中的“10%”所对应的值，则可认为序列存在单位根，为非平稳序列。完整代码如下：

【程序11-7】

```
import numpy as np
import statsmodels.api as sm

arr_a = np.random.normal(0, 1, 250)
arr_a = np.cumsum(arr_a) + 50

for i in range(250):
    arr_a[i] = arr_a[i] - i / 10

st = sm.tsa.stattools.adfuller(arr_a)

if st[0] > st[4]['10%']:
    print('存在单位根, 为非平稳序列')
elif st[0] < st[4]['1%']:
    print('没有单位根, 为平稳序列')
```

打印结果如下：

```
存在单位根, 为非平稳序列
```

第三步：采用“差分法”重建序列

对于非平稳序列转化为平稳序列，最简单的办法就是“差分法”。非平稳序列往往一次到两次差分之后，就会变成平稳序列。

那么什么是差分呢？很难吗？

“差分法”就是序列后一点的值减去当前点的值，用公式表示为 $y_t - y_{(t-1)}$ 。值得注意的是，每一次差分之后，都会少一个序列值。

经过一次差分计算的序列称为“一阶差分”，以此类推，经过n次差分计算的序列称为“n阶序列”。

NumPy中专门计算差分的函数为diff。

```
arr_b = np.diff(arr_a)
```

第四步：重新进行平稳性验证

经过以上分析，重新对创建的序列进行平稳性分析，代码如下：

【程序11-8】

```

import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm

arr_a = np.random.normal(0, 1, 250)
arr_a = np.cumsum(arr_a) + 50

for i in range(250):
    arr_a[i] = arr_a[i] - i / 10

st = sm.tsa.stattools.adfuller(arr_a)
if st[0] > st[4]['10%']:
    print('arr_a 存在单位根, 为非平稳序列')
elif st[0] < st[4]['1%']:
    print('arr_a 没有单位根, 为平稳序列')

arr_b = np.diff(arr_a)
st = sm.tsa.stattools.adfuller(arr_b)

if st[0] > st[4]['10%']:
    print('arr_b 存在单位根, 为非平稳序列')
elif st[0] < st[4]['1%']:
    print('arr_b 没有单位根, 为平稳序列')

plt.subplot(211)
plt.plot(arr_a)
plt.subplot(212)
plt.plot(arr_b)
plt.show()

```

打印结果如下：

```

arr_a存在单位根, 为非平稳序列
arr_b没有单位根, 为平稳序列

```

可以看到，随机生成的arr_a为非平稳序列，而经过差分法处理后的序列arr_b被修正为平稳序列，走势图如图11-4所示。

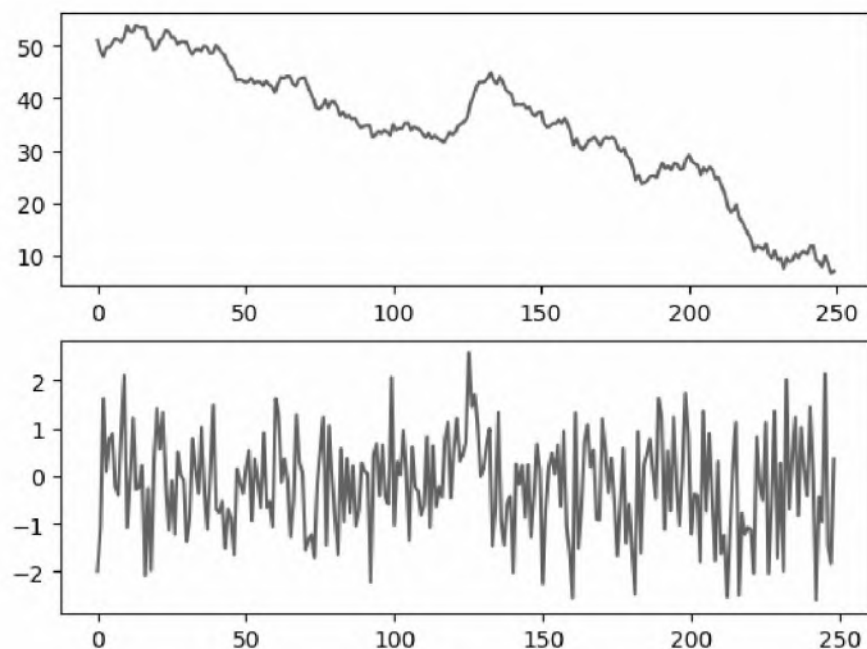


图11-4 非平稳序列转化为平稳序列

图11-4中，上图为随机生成的非平稳序列，而下图是经过差分处理的平稳序列，从图中可以很容易看出，上图的趋势并不确定，只是有一个向下的趋势，这样找到一个将其包围的框体较难，而图11-4下图序列的震荡是在一个框体中，因此具有平稳性。

11.3 配对交易

前面介绍了基于单一序列的平稳性的判定，下面介绍配对交易的协整判断及其完整的代码。

11.3.1 配对交易的算法

本节介绍配对交易的具体计算问题。对于配对交易，首先需要掌握其算法步骤，步骤如下：

(1) 找出具有较强相关性的股票，建立配对股票模式。这样一般要求其相对系数大于0.95。

(2) 分别检验配对标的的平稳性，一般来说，股票都不会直接是平稳性序列，因此在其基础上检验配对的股票是否具有同阶差分协整性。

(3) 做系数回归分析，确立系数和截距值。

(4) 用单位根检验新生成的值残差是否平稳，若平稳，则两个时序是协整的，否则结束，实验失败。

(5) 根据策略编写代码。

11.3.2 提取股票的相关性

计算不同序列的协整性，第一步就是计算不同序列之间的相关性，只有相关系数较大的股票才能进行下一步的协整性判断。

为了简便起见，这里选择银行板块中的“建设银行”与“农业银行”。需要注意，所针对的标的是经过作者精心选择的，现在可能已经不适用，读者从本例中得到更多的是思路。

第一步：导入所使用的库包，创建回归分析代码

这里所使用的库包较多，因此第一步需要分析不同的股票标的相关系数。而量化掘金网站上提供了相关行业的分类代码，如图11-5所示。

代码	名称
A01	农业
A02	林业
A03	畜牧业
A04	渔业
A05	农、林、牧、渔业
B06	煤炭开采和洗选业
B07	石油和天然气开采业
B08	黑色金属矿采选业
B09	有色金属矿采选业
B11	开采辅助活动
C13	农副食品加工业
C14	食品制造业

图11-5 证监会二级行业

从获取的二级行业代码可以获取全部证券编号。而根据获取的代码可以获取相关的行业股票代码，完整代码如下：

【程序11-9】

```
import numpy as np
from gm.api import *
import pandas as pd

set_token("a71a8083b68e73817e93f7f196b030482abe5939")

symbols = get_industry("I65")
df = pd.DataFrame()

for symbol in symbols:
    try:
        data = history_n(symbol=symbol,frequency="1d",count=42,end_time="2018-12-30",fields="close",fill_missing="last",adjust=ADJUST_PREV,df=True)
        data_close = pd.DataFrame({symbol:data["close"]})
        if len(df) == 0:
            df = (data_close)
        else:df = pd.concat([df,data_close], axis=1)
    except:pass

corr = df.corr()

print ('相关系数大于 0.9 的组合对数有' +str((np.sum(corr.values.ravel())>0.9) - len(symbols))/2)+'对')
print ('相关系数大于 0.95 的组合对数有' +str((np.sum(corr.values.ravel())>0.95) - len(symbols))/2)+'对')

print(corr[corr>0.95])
```


从【程序11-9】中可以看到，首先从掘金量化数据库中读取了其对应的数据，之后按DataFrame生成对应的数据格式，使用生成的DataFrame数据集自带的数据相关性计算数据集内部的数据相关性。

打印结果如下：

相关系数大于0.9的组合对数有2050.0对
相关系数大于0.95的组合对数有601.0对

需要说明的是，这里为了演示配对程序，选择了软件和信息技术服务行业，此行业有197只股票代码，计算其不同的相关性。代码 `print (corr[corr>0.95])` 打印出了所有相关系数大于0.95的股票，结果如图11-6所示。

	SHSE.600289	SHSE.600536	SHSE.600556	SHSE.600571	SHSE.600588
SHSE.600289	1.0000	NaN	NaN	NaN	NaN
SHSE.600536	NaN	1.0000	NaN	NaN	NaN
SHSE.600556	NaN	NaN	1.000	NaN	NaN
SHSE.600571	NaN	NaN	NaN	1.0000	NaN
SHSE.600588	NaN	NaN	NaN	NaN	1.0000
SHSE.600602	NaN	NaN	NaN	NaN	NaN
SHSE.600654	0.9503	NaN	NaN	NaN	NaN
SHSE.600701	NaN	NaN	NaN	NaN	NaN
SHSE.600718	NaN	0.9548	NaN	NaN	NaN
SHSE.600728	NaN	NaN	NaN	NaN	NaN
SHSE.600892	NaN	NaN	NaN	NaN	NaN
SHSE.900901	NaN	NaN	NaN	NaN	NaN
SHSE.600406	NaN	NaN	NaN	NaN	NaN
SHSE.600410	NaN	NaN	NaN	NaN	NaN
SHSE.600446	NaN	NaN	NaN	NaN	NaN
SHSE.600476	NaN	0.9586	NaN	0.9585	NaN

图11-6 相关系数大于0.95的部分数据

可以看到，这里不同的股票标的之间有不同的相关系数，根据条件值显示了部分相关系数大于0.95的值。

为了简便起见，这里并不使用这些对应的标的，而是选择常用的银行标的：建设银行（SHSE.601939）以及农业银行（SHSE.601288）。

第二步：计算特定的平稳性

在对特定的标的进行协整性检验之前，需要对两个序列分别做单位根检验，前面已经介绍过，对于不同的序列必须要求其具有协整性，如果其达不到严格要求，退一步可以要求其在进行同阶差分变换后具有协整性。（一般要求最多进行一阶差分变换。）

代码如下：

【程序11-10】

```

import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from gm.api import *

set_token("a71a8083b68e73817e93f7f196b030482abe5939")

ccb = history_n(symbol="SHSE.601939", frequency="1d", count=250,
end_time="2015-12-30", fields="close", fill_missing="last", adjust=ADJUST_PREV,
df=True)
ccb = ccb["close"].values

st = sm.tsa.stattools.adfuller(ccb)
if st[0] > st[4]['10%']:
    print('ccb 存在单位根, 为非平稳序列')
elif st[0] < st[4]['1%']:
    print('ccb 没有单位根, 为平稳序列')

ccb_ = np.diff(ccb)
st = sm.tsa.stattools.adfuller(ccb_)

if st[0] > st[4]['10%']:
    print('ccb_ 存在单位根, 为非平稳序列')
elif st[0] < st[4]['1%']:
    print('ccb_ 没有单位根, 为平稳序列')

plt.subplot(211)
plt.plot(ccb)
plt.subplot(212)
plt.plot(ccb_)
plt.show()

```

这里选择了建设银行2015年的周年数据作为数据集进行存储，由于获取的数据为pandas格式，因此还需要通过调用pandas的方法将其转化成序列问题。打印结果如下：

```

ccb存在单位根, 为非平稳序列
ccb_ 没有单位根, 为平稳序列

```

可以看到，第一次检验生成的序列，因为存在单位根，被判定为非平稳序列。而经过差分法以后，新生成的序列则没有单位根，成为

平稳序列，如图11-7所示。进一步说，生成的是一个新的一阶差分平稳序列，这个序列具有协整性。

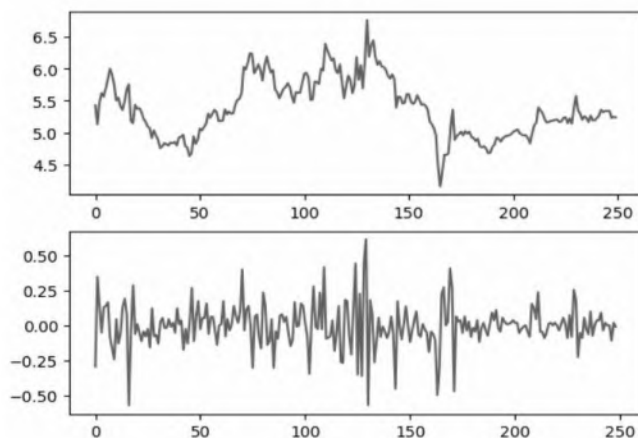


图11-7 建设银行平稳性检验

对于农业银行的平稳性检验也是如此，请读者自行验证。

11.3.3 协整系数的计算方法

第一步：协整系数的计算

协整性实际上就是采用回归分析分析两只股票价格之间存在的线性关系。

$$Y=aX+b$$

其中，a为回归系数，b为常数项。具体来看，回归的计算方式有多种，这里采用TensorFlow模块进行分析。注意，这里的TensorFlow回归分析的写法可能不容易明白，只需要记住以下代码即可：

```

import tensorflow as tf
import numpy as np
def linear_regression(real_A,real_B):

    real_A = np.reshape(real_A,(-1,1))
    real_B = np.reshape(real_B, (-1, 1))

    x_ = tf.placeholder(tf.float32, [None, 1])
    y_ = tf.placeholder(tf.float32, [None, 1]) # y_为测试集结果数据

    weight = tf.Variable(tf.ones([1, 1]))
    bias = tf.Variable(tf.ones([1]))

    y = tf.matmul(x_, weight) + bias

    loss = tf.reduce_mean(tf.square(y - y_)) #批量线性回归用这个损失函数
    train_step = tf.train.AdamOptimizer(0.0001).minimize(loss)

    init = tf.global_variables_initializer()

    flag = True
    with tf.Session() as sess:
        sess.run(init)

        count = 0
        loss_temp = 0
        while flag:

            feed = {x_: real_A, y_: real_B}
            sess.run(train_step, feed_dict=feed)
            loss_res = sess.run(loss, feed_dict=feed)
            if count%10000 == 0:
                print("正在运行次数为:",count," Loss 为: ",loss_res)

            if loss_temp == loss_res:
                flag = False
                count += 1
                loss_temp = loss_res

        weight = sess.run(weight)
        bias = sess.run(bias)

    return weight[0][0],bias[0]

```

这里不做解释，有兴趣的读者可以自行学习。代码的主要作用是使用传递进来的两个参数做出形如

$$Y=aX+b$$

的数据格式。这里默认`real_A=X`，`read_B=Y`。返回值`weight[0][0]`和`bias[0]`分别代表其对应的系数以及截距。

具体的使用也较为方便，通过导入对应的Python文件，直接调用相关的数据处理程序即可。

```
import Tensorflow批量单线性回归_双变量 as lr
...
...
...
weight,bias = lr.linear_regression(x,y)
```

因此，在处理完毕关于不同序列的平稳性检验后，可以使用基于TensorFlow的线性回归拟合方差对结果进行拟合处理。

【程序11-11】

```

import numpy as np
from gm.api import *
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
import Tensorflow 批量单线性回归_双变量 as lr

set_token("a71a8083b68e73817e93f7f196b030482abe5939")

ccb = history_n(symbol="SHSE.601939", frequency="1d", count=250,
end_time="2015-12-30", fields="close", fill_missing="last", adjust=ADJUST_PREV,
df=True)
abchina = history_n(symbol="SHSE.601288", frequency="1d", count=250,
end_time="2015-12-30", fields="close", fill_missing="last", adjust=ADJUST_PREV,
df=True)

ccv_close = ccb["close"]
abchina_close = abchina["close"]

x = ccv_close.values
y = abchina_close.values

weight,bias = lr.linear_regression(x,y)
print("weight 为: ",weight)
print("bias 为: ",bias)

y_ = weight*x + bias

plt.plot(y,"b")
plt.plot(y_,"r")
plt.show()

```

在【程序11-11】中，首先通过量化掘金的API获取了两个银行的数据集，取得2015年的日年线数据，关于每组序列相关性的检验，在前面的章节已经做了处理。

打印结果如下：

```

weight为:  0.405483
bias为:   0.889566

```

通过基于系数乘积重新获取的数据与原数据进行画图比较，如图11-8所示。

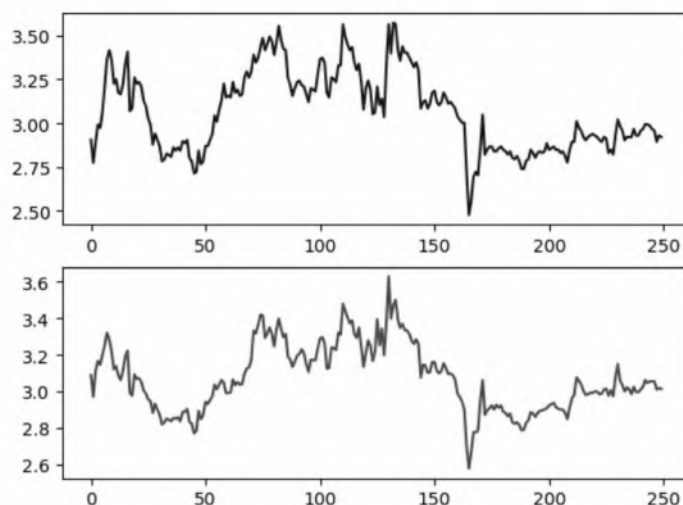


图11-8 根据相关系数重建的图像比较

第二步：残差的计算

根据算法的设计，下面对残差进行计算，代码段如下：

```
d_value = y - y_  
  
st = sm.tsa.stattools.adfuller(d_value)  
if st[0] > st[4]['10%']:  
    print(d_value 存在单位根, 为非平稳序列')  
elif st[0] < st[4]['1%']:  
    print(d_value 没有单位根, 为平稳序列')  
else: print("d_value 无法判定")
```

d_value被设置成真实的标的值与通过回归计算出的值之间的差值，之后进行单位根的计算，打印如下：

```
d_value没有单位根, 为平稳序列  
std = 0.0800697561823  
mean = -0.000215040631916
```


这里显示了3部分内容，分别为平稳序列的判定、标准差的值以及均值。

重点提醒，如果d_value被判定成存在单位根，为非平稳序列，那么全部的检验结束，绝对不能进行差分变换，没有任何意义。

最后打印出完整的残差计算结果，判定其能够成功的可行性是多少，代码如下：

```
t = sm.tsa.stattools.adfuller(d_value)
output = pd.DataFrame(index=['Test Statistic value','p-value','Lags
Used','Number of Observations Used','Critical Value(1%)','Critical
Value(5%)','Critical Value(10%)'],columns=['value'])
output.iloc[0][0] = t[0]
output.iloc[1][0] = t[1]
output.iloc[2][0] = t[2]
output.iloc[3][0] = t[3]
output.iloc[4][0] = t[4]['1%']
output.iloc[5][0] = t[4]['5%']
output.iloc[6][0] = t[4]['10%']
print(output)
```

打印结果如图11-9所示。

	value
Test Statistic value	-4.405
p-value	0.0002912
Lags Used	3
Number of Observations Used	246
Critical Value(1%)	-3.457
Critical Value(5%)	-2.873
Critical Value(10%)	-2.573

图11-9 差值的平稳性检测报告

从报告可以看出，这里的p-value值远小于0.05，因此可以认为对于残值的检测是能够被接受的。

完整的代码段如下。

【程序11-12】

```
import numpy as np
from gm.api import *
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
import Tensorflow 批量单线性回归_双变量 as lr

set_token("a71a8083b68e73817e93f7f196b030482abe5939")

ccb = history_n(symbol="SHSE.601939", frequency="1d", count=250,
end_time="2015-12-30", fields="close", fill_missing="last", adjust=ADJUST_PREV,
df=True)
abchina = history_n(symbol="SHSE.601288", frequency="1d", count=250,
```

```

end_time="2015-12-30", fields="close", fill_missing="last", adjust=ADJUST_PREV,
df=True)

ccv_close = ccb["close"]
abchina_close = abchina["close"]

x = ccv_close.values
y = abchina_close.values

weight,bias = lr.linear_regression(x,y)
print("weight 为: ",weight)
print("bias 为: ",bias)

y_ = weight*x + bias

plt.subplot(211)
plt.plot(y,"b")
plt.subplot(212)
plt.plot(y_,"r")
plt.show()

d_value = y - y_

st = sm.tsa.stattools.adfuller(d_value)
if st[0] > st[4]['10%']:
    print('ut 存在单位根,为非平稳序列')
elif st[0] < st[4]['1%']:
    print('ut 没有单位根,为平稳序列')
else:print("ut 无法判定")

mean = np.mean(d_value) #均值
std = np.std(d_value) #方差

print('std = %s'%(std))
print('mean = %s'%(mean))

t = sm.tsa.stattools.adfuller(d_value)
output = pd.DataFrame(index=['Test Statistic value','p-value','Lags
Used','Number of Observations Used','Critical Value(1%)','Critical
Value(5%)','Critical Value(10%)'],columns=['value'])
output.iloc[0][0] = t[0]
output.iloc[1][0] = t[1]
output.iloc[2][0] = t[2]

```

```
output.iloc[3][0] = t[3]
output.iloc[4][0] = t[4]['1%']
output.iloc[5][0] = t[4]['5%']
output.iloc[6][0] = t[4]['10%']
print(output)
```

11.4 配对交易的魔力

验证完所配对的股票是否可以进行配对交易，下面进入交易算法的设计与回测。因为存在配对的X股票与Y股票，所以这里采用的策略是：当残差大于2倍标准差时，买入X股票，而当残差小于-2倍的标准差时，买入Y股票。

11.4.1 前期计算

根据11.3节所对应的数据，首先计算和判定协整性。而计算所需要的数据集，可以通过“量化掘金”中的“行业概念数据”获取，如图11-10、图11-11所示。



图11-10 行业概念数据

证监会二级行业	
代码	名称
A01	农业
A02	林业
A03	畜牧业
A04	渔业
A05	农、林、牧、渔服务业
B06	煤炭开采和洗选业
B07	石油和天然气开采业

概念板块			
股票概念的分类数据，现实的二级市场交易中，经常会以“分股			
3D打印	4G概念	IPV6概念	IP变现
O2O模式	QFII重仓	ST板块	三沙概念
三网融合	上海本地	上海自贸	业绩预升
业绩预降	东亚自贸	丝绸之路	云计算
互联网金融	京津冀	低碳经济	体育概念
保险重仓	保障房	信息安全	信托重仓
充电桩	免疫治疗	养老概念	内贸规划

图11-11 行业与板块数据

需要注意的是，选择根据行业分类获取股票的数据集和根据概念板块获取数据集，它们使用的代码是不同的，代码如下：

```
symbols = get_industry(symbols) #根据概念板块获取数据集
symbols = get_concept(symbols) #根据行业分类获取数据集
```

因此，根据获取标的的不同可以在不同行业进行获取。而根据其获取不同行业股票数据集相关系数的代码如下：

```

def get_max_symbols_list(symbols = "", frequency="1d", count =
250, end_time="2017-12-30", num = 10):
    symbols = get_concept(symbols)
    df = pd.DataFrame()

    for symbol in symbols:

        try:
            data =
history_n(symbol=symbol, frequency=frequency, count=count, end_time=end_time, fiel
ds="close", fill_missing="last", adjust=ADJUST_PREV, df=True)
            if len(data) == count:
                data_close = pd.DataFrame({symbol:data["close"]})
                if len(df) == 0:
                    df = (data_close)
                else: df = pd.concat([df, data_close], axis=1)
            except: pass

    corr = df.corr()

    #找出相关性最大的股票标的
    corr_matrix = corr.as_matrix() #转化为矩阵
    corr_matrix[corr_matrix == 1 ] = 0 #将1转为0

    raw, column = corr_matrix.shape # get the matrix of a row and column
    doubel_symbols_list = []
    for n in range(num):

        _positon = np.argmax(corr_matrix) # get the index of max in the a
m, n = divmod(_positon, column)

        corr_matrix[m, n] = corr_matrix[n, m] = 0
        doubel_symbols_list.append([corr.index[m], corr.index[n]])

    return doubel_symbols_list

```

代码段中相关参数解释：symbols为股票集合，frequency为取样窗口的定义，count为窗口数量多少，end_time为结束时间，num为最终打印最多多少个配对股票。

11.4.2 协整性判断

在判断具有较高的相关性后，需要对协整性进行判断，和前面类似，代码如下：

```

def check_doubel_symbols(symbols_1, symbols_2, frequency="1d", count = 250,
end_time="2017-12-30"):
    symbols_1_data = history_n(symbol=symbols_1, frequency=frequency,
count=count, end_time=end_time, fields="close", fill_missing="last",
adjust=ADJUST_PREV, df=True)
    symbols_2_data = history_n(symbol=symbols_2, frequency=frequency,
count=count, end_time=end_time, fields="close", fill_missing="last",
adjust=ADJUST_PREV, df=True)

    symbols_1_close = symbols_1_data["close"]
    symbols_2_close = symbols_2_data["close"]

    x = symbols_1_close.values
    y = symbols_2_close.values

    weight,bias = lr.linear_regression(x,y)

y_ = weight*x + bias

    d_value = y - y_

    st = sm.tsa.stattools.adfuller(d_value)
    if st[0] > st[4]['10%']:
        print('d_value 存在单位根,为非平稳序列', symbols_1, "-", symbols_2,)
        return
    elif st[0] < st[4]['1%']:
        print('d_value 没有单位根,为平稳序列')
    else:
        print("d_value 无法判定", symbols_1, "-", symbols_2,)
        return

    mean = np.mean(d_value) #均值
    std = np.std(d_value) #方差

    t = sm.tsa.stattools.adfuller(d_value)
    output = pd.DataFrame(index=['Test Statistic value','p-value','Lags
Used','Number of Observations Used','Critical Value(1%)','Critical
Value(5%)','Critical Value(10%)'],columns=['value'])
    output.iloc[0][0] = t[0]
    output.iloc[1][0] = t[1]

```



```

output.iloc[2][0] = t[2]
output.iloc[3][0] = t[3]
output.iloc[4][0] = t[4]['1%']
output.iloc[5][0] = t[4]['5%']
output.iloc[6][0] = t[4]['10%']

record_file = '配对交易结果' + '.csv'
if t[1] < 0.05:
    print("可以配对:", symbols_1, "-", symbols_2,)

    res = [symbols_1, symbols_2, weight, bias, mean, std]
    print(res)

    writer = csv.writer(open(record_file, 'a+', encoding='utf8', newline=''))
    writer.writerow(res)

else: print("P-VALUE 不合适", symbols_1, "-", symbols_2,)

```

这里有几个需要说明的地方。

首先是参数的设计，symbol_1和symbol_2分别为需要配对的股票的代码，frequency为时间窗口的设定，count为数目设定，end_time为结束时间。

代码段内部是对标的的判断，首先根据线性回归算法算出回归系数，之后对差值进行平稳性检测，再对p值进行检测，并把符合要求的数据存入数据文件中。

11.4.3 使用量化掘金回测系统对结果进行判定

在进行完前期计算以及协整性判断后，下面使用量化掘金对数据进行判断。

第一步：数据的设定

为了简便起见，这里采用前面验证好的关于银行的配对组，使用“建设银行”与“农业银行”进行配对。

首先获取数据，采用以250个交易日的收盘价为单位的数据进行验证，由于对两个标的的协整性检测在前面已经完成，这里直接跳过协整性检测和线性回归系数的计算，结果如下：

```
weight = 0.481477
bias = 0.481829
mean = -0.000203539120501
std = 0.0754807889752
```

而这些系数的设定可以在前面的计算中获取，这里就不再重复，代码段如下：

```
from gm.api import *

def init(context):

    context.symbol_1 = "SHSE.601939"
    context.symbol_2 = "SHSE.601288"
    context.weight = 0.481477
    context.bias = 0.481829
    context.mean = -0.000203539120501
    context.std = 0.0754807889752
    schedule(schedule_func=algo, date_rule='1d', time_rule='09:31:00')

    context.flagX = False
    context.flagY = False
```

init函数是初始化数据，在这里对数据进行初始化设定，schedule是设定函数，规则定义为每日的9点31分执行算法。而context.flagX和context.flagY用于对哪只股票标的的进行买入。

第二步：策略算法的设定

算法的设定较为简单，简单地说，就是当计算差值突破阈值时进行买卖，代码如下：

```
def algo(context):
    last_day = get_previous_trading_date("SHSE", context.now)

    ccb =
history(context.symbol_1,frequency="1d",start_time=last_day,end_time=last_day,
fields="close",fill_missing="last",adjust=ADJUST_PREV,df=True)
    abchina = history(context.symbol_2, frequency="1d", start_time=last_day,
end_time=last_day, fields="close",fill_missing="last", adjust=ADJUST_PREV,
df=True)

    ccb = (ccb["close"].values)
    abchina = (abchina["close"].values)

    d_value = abchina - (ccb * context.weight + context.bias)

    if (context.flagX and d_value < context.mean + context.std) or
(context.flagY and d_value > context.mean - context.std):
        order_close_all()
        context.flagX = False
        context.flagY = False

    if d_value > context.mean + 3 * context.std:

        order_target_percent(symbol=context.symbol_1, percent=1,
position_side=PositionSide_Long, rder_type=OrderType_Market)
        context.flagX = True

    if d_value < context.mean - 3 * context.std:

        order_target_percent(symbol=context.symbol_2, percent=1,
position_side=PositionSide_Long, order_type=OrderType_Market)
        context.flagY = True
```

建仓以后，当差值回归价值曲线后，平仓全部标的。这里需要注意的是，经典的配对交易模型中是对另一个相对标记建空仓，但是由于国内股市对于空仓的建立比较烦琐，这里不建议个人使用，因此略过不写，有兴趣的同学可以自行完成。

【程序11-13】

```
# -*- coding: utf-8 -*-
from gm.api import *

def init(context):

    context.symbol_1 = "SHSE.601939"
    context.symbol_2 = "SHSE.601288"
    context.weight = 0.481477
    context.bias = 0.481829
    context.mean = -0.000203539120501
    context.std = 0.0754807889752
    schedule(schedule_func=algo, date_rule='1d', time_rule='09:31:00')

    context.flagX = False
    context.flagY = False

def algo(context):
    last_day = get_previous_trading_date("SHSE", context.now)

    ccb = history(context.symbol_1, frequency="1d", start_time=last_day
, end_time=last_day, fields="close", fill_missing="last", adjust=ADJUST_PREV, df=True)
    abchina = history(context.symbol_2, frequency="1d", start_time=last_day,
end_time=last_day, fields="close", fill_missing="last", adjust=ADJUST_PREV,
df=True)

    ccb = (ccb["close"].values)
    abchina = (abchina["close"].values)

    d_value = abchina - (ccb * context.weight + context.bias)
```

```

    if (context.flagX and d_value < context.mean + context.std) or
(context.flagY and d_value > context.mean - context.std):
        order_close_all()
        context.flagX = False
        context.flagY = False
    if d_value > context.mean + 3 * context.std:

        order_target_percent(symbol=context.symbol_1, percent=1,
position_side=PositionSide_Long,
                            order_type=OrderType_Market)
        context.flagX = True
    if d_value < context.mean - 3 * context.std:

        order_target_percent(symbol=context.symbol_2, percent=1,
position_side=PositionSide_Long,
                            order_type=OrderType_Market)
        context.flagY = True
def on_backtest_finished(context, indicator):
    print(indicator)

if __name__ == "__main__":
    run(
        strategy_id='63e92f59-1386-11e8-bbe9-902b3463caf1',
        filename='配对交易掘金检测.py',
        mode=MODE_BACKTEST,
        token='a71a8083b68e73817e93f7f196b030482abe5939',
        backtest_start_time='2016-01-03 09:00:00',
        backtest_end_time='2016-12-30 15:00:00',
        backtest_initial_cash=100000,
        backtest_adjust=ADJUST_PREV,
        backtest_slippage_ratio=0.01,
        backtest_commission_ratio=0.0005,
    )

```

【程序11-13】模拟了建仓交易的程序，这里使用的是农业银行与建设银行在2016年的回测数据，而参数的确定是根据2015年农业银行与建设银行的价差回归曲线设计的。最终结果如图11-12所示。

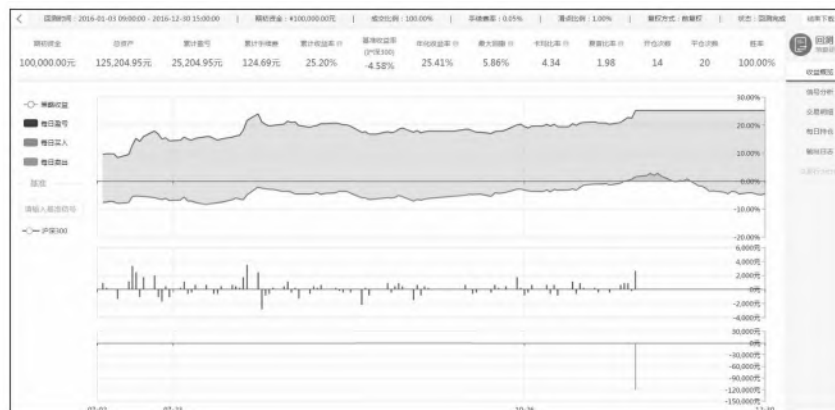


图11-12 建设银行与农业银行关于配对交易的回测数据

可以看到，整个回测收益保持在一个比较稳定的状态，在上涨阶段能够比较迅捷地获得收益，而当整体大盘趋势向下时能够保持回撤不会太大。

11.5 小结

无论黑猫还是白猫，能抓到老鼠就是好猫。

配对交易来源于期货交易策略，也是一种最基本的对冲套利交易方法。当然除此之外，还有很多好的策略没有介绍，读者可以根据自己的需要自行挖掘。

本书全面介绍了掘金量化平台与使用掘金量化进行股票回测的多种方法，希望起到抛砖引玉的作用，能够帮助读者对使用量化投资有更进一步的认识，早日建立起个人的量化投资策略。

量化投资是一门学科，也是一种“掘金”的方法，对于有志于从事这一行的从业者来说，路漫漫其修远兮，还需要更多的努力与探索。

“全彩印刷”
的编程体验

Python 王者归来

洪锦魁◎著

完整Python语法

列表、元组、字典、集合

函数与类别设计

设计与应用模块

程序除错与异常处理

文件读写与目录管理

正规表达式与文字搜索

剪贴板、Word、PDF文件

Excel、CSV、Json文件

海龟绘图与图表绘制

鼠标与键盘控制

多任务与多线程

动画、音效、游戏设计

网络爬虫、伪装浏览器

图像处理与文字辨识

机场人脸辨识系统

清华大学出版社

Python 王者归来

洪锦魁◎著



清华大学出版社
北京

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

图书在版编目（CIP）数据

Python王者归来 / 洪锦魁著. —北京：清华大学出版社，2018

ISBN 978-7-302-51334-6

I. ①P… II. ①洪… III. ①软件工具—程序设计 IV.
①TP311.561

中国版本图书馆CIP数据核字（2018）第229060号

责任编辑： 栾大成

封面设计：

责任校对： 胡伟民

责任印制：

出版发行： 清华大学出版社

网 址： <http://www.tup.com.cn>, <http://www.wqbook.com>

地 址： 北京清华大学学研大厦A座

邮 编： 100084

社 总 机： 010-62770175

邮 购： 010-62786544

投稿与读者服务： 010-62776969, c-service@tup.tsinghua.edu.cn

质量反馈： 010-62772015, zhiliang@tup.tsinghua.edu.cn

印 装 者：

经 销： 全国新华书店

开 本： 170mm×240mm

印 张： 32.5

字 数： 945千字

版 次： 2019年4月第1版

印 次： 2019年4月第1次印刷

印 数： 1~2500

产品编号： 079257-01

作者简介



洪锦魁 中国台湾省计算机专家，IT作家。

- ◆ DOS时代代表作：汇编语言、C、C++、Pascal、数据结构。
- ◆ Windows时代代表作：Windows编程、Programming、Visual Basic。
- ◆ Internet时代代表作：网页设计、HTML、JavaScript。
- ◆ 大数据时代代表作：R语言。

其作品版权被输出到中国大陆（简体中文）、马来西亚（马来文）、美国（英文）。

近年来作品：

- ◆ Java王者归来
- ◆ Python王者归来
- ◆ HTML5+CSS3王者归来

◆ R语言迈向大数据之路

其著作特色：

所有程序语法会依特性分类，同时以实用的程序范例进行解说，让读者可以事半功倍地轻松掌握相关知识。

内容简介

Python的丰富模块(module)以及广泛的应用范围,使Python成为当下最重要的计算机语言之一,本书尝试将所有常用模块与应用分门别类组织起来,相信只要读者遵循本书实例,一定可以轻松学会Python语法与应用,逐步向Python高手之路迈进,这也是撰写本书的目的。

本书以约800个程序实例讲解了:完整的Python语法,Python的输入与输出,Python的数据型态,列表(list)、元组(tuple)、字典(dict)、集合(set),函数设计,类别设计,使用系统与外部模块(module),设计自己的模块(module),文件压缩与解压缩,程序除错与异常处理,文件读写与目录管理,正则表达式(Regular Expression)与文字探勘,剪贴簿(clipboard)、Word、PDF文件处理,Excel、CSV、Json文件处理,图表绘制,电子邮件与简讯,鼠标与键盘控制,人脸识别系统,QR code制作,多任务与多线程,动画、音效、游戏设计,网络爬虫与伪装浏览器,图像处理与文字辨识,设计桃园机场出入境人脸识别系统……

前16章的内容已经足够让你打好Python基础了,如果有兴趣继续钻研,则迈向Python高手之路。为了提升阅读体验,本书为彩色印刷,在图书结构、案例选择以及代码样式上都进行了细心设计,力争呈现给读者一本与众不同的编程书。

本书适合所有对Python编程感兴趣的读者,甚至适合设计师等编程基础薄弱的岗位作为编程入门指导,同时也可以作为社会培训教材。

前言

多次与教育界的朋友相聚，谈到计算机语言的发展趋势，大家一致公认Python已经是当今最重要的计算机语言了，几乎所有知名公司，例如，Google、Facebook等皆已经将此语言列为必备计算机语言。了解到许多人想学Python，市面上的书也不少了，但是目前尚欠缺一本用简单程序实例完整讲解Python语法的书籍，造成了学习上的障碍，就这样我决定撰写一本可以用丰富实例完整讲解Python语法的入门书籍。

当完成了本书所有语法说明时，已经是第15章了，想要交稿，但是心中总是觉得欠缺什么。因为我知道Python的丰富模块(module)，广泛的应用范围，才使Python成为当下最重要的计算机语言之一，就这样我尝试将所有熟悉的模块与应用分门别类组织起来，没想到整本书完成已经是34章了。虽然花了更多时间完成这本著作，心情是愉快的，因为我相信只要读者购买本书并遵循本书实例，一定可以轻轻松松、快快乐乐地学会Python语法与应用，逐步让自己向Python高手之路迈进，这也是撰写本书的目的。

本书以约800个程序实例讲解了下列知识：

-  完整的Python语法
-  Python的输入与输出
-  Python的数据类型
-  列表(list)、元组(tuple)、字典(dict)、集合(set)
-  函数设计
-  类设计
-  使用系统与外部模块(module)
-  设计自己的模块(module)

-  文件压缩与解压缩
-  程序除错与异常处理
-  文件读写与目录管理
-  正则表达式(Regular Expression)与文字检索
-  剪贴簿(clipboard)、Word、PDF文件处理
-  Excel、CSV、Json文件处理
-  图表绘制
-  电子邮件与简讯
-  鼠标与键盘控制
-  人脸识别系统
-  QR code制作
-  多任务与多线程
-  动画、音效、游戏设计
-  网络爬虫与伪装浏览器
-  图像处理与文字辨识
-  设计桃园机场出入境人脸识别系统

其实前15章的内容已经足够让你具备Python基础知识了，如果你有兴趣继续钻研，建议你可以继续阅读迈向Python高手之路。

写过许多的计算机相关书籍，本书沿袭笔者著作的特色，程序实例丰富，相信读者只要遵循本书内容必定可以在最短时间精通网页设计，编著本书虽力求完美，但难免会有不当之处，尚祈读者不吝指正。

洪锦魁

jiinkwei@me.com

目 录

[前言](#)

[第1章 基本观念](#)

[1-1 认识Python](#)

[1-2 Python的起源](#)

[1-3 Python语言发展史](#)

[1-4 Python的应用范围](#)

[1-5 跨平台的程序语言](#)

[1-6 系统的安装与执行](#)

[1-6-1 在Windows启动与执行Python](#)

[1-6-2 在Mac OS启动与执行Python](#)

[1-7 Python 2与Python 3不相容的验证](#)

[1-8 文件的建立、存储、执行与打开](#)

[1-8-1 文件的建立](#)

[1-8-2 文件的存储](#)

[1-8-3 文件的执行](#)

[1-8-4 打开文件](#)

[1-9 程序注释](#)

[1-9-1 注释符号#](#)

[1-9-2 三个单引号或双引号](#)

[第2章 认识变量与基本数学运算](#)

[2-1 用Python做计算](#)

[2-2 认识变量](#)

[2-3 认识程序的意义](#)

[2-4 认识注释的意义](#)

[2-5 Python变量与其他程序语言的差异](#)

[2-6 变量的命名原则](#)

[2-7 基本数学运算](#)

[2-7-1 四则运算](#)

[2-7-2 余数和整除](#)

[2-7-3 次方](#)

[2-7-4 Python语言控制运算的优先级](#)

[2-8 赋值运算符](#)

[2-9 Python等号的多重指定使用](#)

[2-10 删除变量](#)

[2-11 Python的断行](#)

[2-11-1 一行有多个语句](#)

[2-11-2 将一个语句分成多行](#)

[第3章 Python的基本数据类型](#)

[3-1 type\(_\)函数](#)

[3-2 数值数据类型](#)

[3-2-1 整数与浮点数的运算](#)

[3-2-2 2进位整数与函数bin\(_\)](#)

[3-2-3 8进位整数](#)

[3-2-4 16进位整数](#)

[3-2-5 强制数据类型的转换](#)

[3-2-6 数值运算常用的函数](#)

[3-3 布尔值数据类型](#)

[3-4 字符串数据类型](#)

[3-4-1 字符串的连接](#)

[3-4-2 处理多于一行的字符串](#)

[3-4-3 逸出字符](#)

[3-4-4 强制转换为字符串](#)

[3-4-5 将字符串转换为整数](#)

[3-4-6 字符串数据的转换](#)

[3-4-7 字符串与整数相乘产生字符串复制效果](#)

[3-4-8 聪明地使用字符串加法和换行字符\n](#)

[3-4-9 字符串前加r](#)

[第4章 基本输入与输出](#)

[4-1 Python的辅助说明help\(_\)](#)

[4-2 格式化输出数据使用print\(_\)](#)

[4-2-1 函数print\(_\)](#)的基本语法

[4-2-2 格式化print\(_\)](#)输出

[4-2-3 精准控制格式化的输出](#)

[4-2-4 format\(_\)](#)函数

[4-2-5 字符串输出与基本排版的应用](#)

[4-2-6 一个无聊的操作](#)

[4-3 输出数据到文件](#)

[4-3-1 打开一个文件open\(_\)](#)

[4-3-2 使用print\(_\)](#)函数输出数据到文件

[4-4 数据输入input\(_\)](#)

[4-5 列出所有内置函数dir\(_\)](#)

[第5章 程序的流程控制使用if语句](#)

[5-1 关系运算符](#)

[5-2 逻辑运算符](#)

[5-3 if语句](#)

[5-4 if ... else语句](#)

[5-5 if ... elif ...else语句](#)

[5-6 嵌套的if语句](#)

[5-7 尚未设定的变量值None](#)

[第6章 列表\(List\)](#)

[6-1 认识列表](#)

[6-1-1 列表的基本定义](#)

[6-1-2 读取列表元素](#)

[6-1-3 列表切片\(list slices\)](#)

[6-1-4 列表索引值是-1](#)

[6-1-5 列表统计资料、最大值max\(_\)、最小值min\(_\)、总和sum\(_\)](#)

[6-1-6 列表个数len\(_\)](#)

[6-1-7 更改列表元素的内容](#)

[6-1-8 列表的相加](#)

[6-1-9 列表乘以一个数字](#)

[6-1-10 列表元素的加法运作](#)

[6-1-11 删除列表元素](#)

[6-1-12 列表为空列表的判断](#)

[6-1-13 删除列表](#)

[6-2 Python简单的面向对象观念](#)

[6-2-1 字符串的方法](#)

[6-2-2 更改字符串大小写](#)

[6-2-3 dir\(_\)获得系统内部对象的方法](#)

[6-3 获得列表的方法](#)

[6-4 增加与删除列表元素](#)

[6-4-1 在列表末端增加元素append\(_\)](#)

[6-4-2 插入列表元素insert\(_\)](#)

[6-4-3 删除列表元素pop\(_\)](#)

[6-4-4 删除指定的元素remove\(_\)](#)

[6-5 列表的排序](#)

[6-5-1 颠倒排序reverse\(_\)](#)

[6-5-2 sort\(_\)排序](#)

[6-5-3 sorted\(_\)排序](#)

[6-6 进阶列表操作](#)

[6-6-1 index\(_\)](#)

[6-6-2 count\(_\)](#)

[6-6-3 列表元素的组合join\(_\)](#)

[6-7 列表内含列表](#)

[6-7-1 再谈append\(_\)](#)

[6-7-2 extend\(_\)](#)

[6-8 列表的复制](#)

[6-8-1 列表的深复制-deep copy](#)

[6-8-2 地址的观念](#)

[6-8-3 列表的浅复制-shallow copy](#)

6-9 再谈字符串

6-9-1 字符串的索引

6-9-2 字符串切片

6-9-3 函数或方法

6-9-4 将字符串转成列表

6-9-5 切片赋值的应用

6-9-6 使用split()处理字符串

6-10 in和not in表达式

6-11 is或is not表达式

6-11-1 整数变量在内存地址的观察

6-11-2 将is和is not表达式应用在整数变量

6-11-3 将is和is not表达式应用在列表变量

6-12 enumerate对象

第7章 循环设计

7-1 基本for循环

7-1-1 for循环基本运作

7-1-2 如果程序代码区块只有一行

7-1-3 有多行的程序代码区块

7-1-4 将for循环应用在列表区间元素

7-1-5 将for循环应用在数据类别的判断

7-1-6 删除列表内所有元素

7-2 range()函数

7-2-1 只有一个参数的range()函数的应用

7-2-2 有2个参数的range()函数

7-2-3 有3个参数的range()函数

7-2-4 活用range()应用

7-2-5 列表生成(list generator)的应用

7-2-6 打印含列表元素的列表

7-2-7 生成含有条件的列表

7-3 进阶的for循环应用

7-3-1 嵌套for循环

[7-3-2 强制离开for循环-break指令](#)

[7-3-3 for循环暂时停止不往下执行 - continue指令](#)

[7-3-4 for ... else循环](#)

[7-4 while循环](#)

[7-4-1 基本while循环](#)

[7-4-2 嵌套while循环](#)

[7-4-3 强制离开while循环-break指令](#)

[7-4-4 while循环暂时停止不往下执行-continue指令](#)

[7-4-5 while循环条件表达式与对象](#)

[7-4-6 pass](#)

[7-5 enumerate对象使用for循环解析](#)

[第8章 元组\(Tuple\)](#)

[8-1 元组的定义](#)

[8-2 读取元组元素](#)

[8-3 遍历所有元组元素](#)

[8-4 修改元组内容产生错误的实例](#)

[8-5 可以使用全新定义方式修改元组元素](#)

[8-6 元组切片\(tuple slices\)](#)

[8-7 方法与函数](#)

[8-8 列表与元组数据互换](#)

[8-9 其他常用的元组方法](#)

[8-10 enumerate对象使用在元组](#)

[8-11 zip\(_\)](#)

[8-12 元组的功能](#)

[第9章 字典\(Dict\)](#)

[9-1 字典基本操作](#)

[9-1-1 定义字典](#)

[9-1-2 列出字典元素的值](#)

[9-1-3 增加字典元素](#)

[9-1-4 更改字典元素内容](#)

[9-1-5 删除字典特定元素](#)

- [9-1-6 删除字典所有元素](#)
- [9-1-7 删除字典](#)
- [9-1-8 建立一个空字典](#)
- [9-1-9 字典的复制](#)
- [9-1-10 取得字典元素数量](#)
- [9-1-11 验证元素是否存在](#)
- [9-1-12 设计字典的可读性技巧](#)

[9-2 遍历字典](#)

- [9-2-1 遍历字典的键-值](#)
- [9-2-2 遍历字典的键](#)
- [9-2-3 排序与遍历字典](#)
- [9-2-4 遍历字典的值](#)

[9-3 建立字典列表](#)

[9-4 字典内含列表元素](#)

[9-5 字典内含字典](#)

[9-6 while循环在字典的应用](#)

[9-7 字典常用的函数和方法](#)

- [9-7-1 len\(_\)](#)
- [9-7-2 fromkeys\(_\)](#)
- [9-7-3 get\(_\)](#)
- [9-7-4 setdefault\(_\)](#)
- [9-7-5 pop\(_\)](#)

[第10章 集合\(Set\)](#)

[10-1 建立集合](#)

- [10-1-1 使用大括号建立集合](#)
- [10-1-2 使用set\(_\)函数定义集合](#)
- [10-1-3 大数据与集合的应用](#)

[10-2 集合的操作](#)

- [10-2-1 交集\(intersection\)](#)
- [10-2-2 并集\(union\)](#)
- [10-2-3 差集\(difference\)](#)

10-2-4 对称差集(symmetric difference)

10-2-5 等于

10-2-6 不等于

10-2-7 是成员in

10-2-8 不是成员not in

10-3 适用集合的方法

10-3-1 add(_)

10-3-2 copy(_)

10-3-3 remove(_)

10-3-4 discard(_)

10-3-5 pop(_)

10-3-6 clear(_)

10-3-7 isdisjoint(_)

10-3-8 issubset(_)

10-3-9 issuperset(_)

10-3-10 intersection update(_)

10-3-11 update(_)

10-3-12 difference update(_)

10-3-13 symmetric difference update(_)

10-4 适用集合的基本函数操作

10-4-1 max(_)/min(_)/sum(_)

10-4-2 len(_)

10-4-3 sorted(_)

10-4-4 enumerate(_)

10-5 冻结集合frozenset

第11章 函数设计

11-1 Python函数基本观念

11-1-1 函数的定义

11-1-2 无参数无返回值的函数

11-1-3 在Python Shell执行函数

11-2 函数的参数设计

[11-2-1 传递一个参数](#)

[11-2-2 多个参数传递](#)

[11-2-3 关键词参数 参数名称=值](#)

[11-2-4 参数默认值的处理](#)

[11-3 函数返回值](#)

[11-3-1 返回None](#)

[11-3-2 简单返回数值数据](#)

[11-3-3 返回多个数据的应用](#)

[11-3-4 简单返回字符串数据](#)

[11-3-5 再谈参数默认值](#)

[11-3-6 函数返回字典数据](#)

[11-3-7 将循环应用在建立VIP会员字典](#)

[11-4 调用函数时参数是列表](#)

[11-4-1 基本传递列表参数的应用](#)

[11-4-2 在函数内修订列表的内容](#)

[11-4-3 使用副本传递列表](#)

[11-5 传递任意数量的参数](#)

[11-5-1 基本传递处理任意数量的参数](#)

[11-5-2 设计含有一般参数与任意数量参数的函数](#)

[11-5-3 设计含有一般参数与任意数量的关键词参数](#)

[11-6 递归式函数设计recursive](#)

[11-7 局部变量与全局变量](#)

[11-7-1 全局变量可以在所有函数使用](#)

[11-7-2 局部变量与全局变量使用相同的名称](#)

[11-7-3 程序设计需注意事项](#)

[11-8 匿名函数lambda](#)

[11-8-1 匿名函数lambda的语法](#)

[11-8-2 匿名函数使用与filter\(_\)](#)

[11-8-3 匿名函数使用与map\(_\)](#)

[11-9 pass与函数](#)

[11-10 type关键词应用在函数](#)

第12章 类-面向对象的程序设计

12-1 类的定义与使用

12-1-1 定义类

12-1-2 操作类的属性与方法

12-1-3 类的构造函数

12-1-4 属性初始值的设定

12-2 类的访问权限——封装(encapsulation)

12-2-1 私有属性

12-2-2 私有方法

12-3 类的继承

12-3-1 衍生类继承基类的实例应用

12-3-2 如何取得基类的私有属性

12-3-3 衍生类与基类有相同名称的属性

12-3-4 衍生类与基类有相同名称的方法

12-3-5 衍生类引用基类的方法

12-3-6 三代同堂的类与取得基类的属性super()

12-3-7 兄弟类属性的取得

12-4 多型(polymorphism)

12-5 多重继承

12-6 type与instance

12-6-1 type()

12-6-2 isinstance()

12-7 特殊属性

12-7-1 文档字符串 doc

12-7-2 name 属性

12-8 类的特殊方法

12-8-1 str ()方法

12-8-2 repr ()方法

12-8-3 iter ()方法

第13章 设计与应用模块

13-1 将自建的函数存储在模块中

13-1-1 先前准备工作

13-1-2 建立函数内容的模块

13-2 应用自己建立的函数模块

13-2-1 import模块名称

13-2-2 导入模块内特定单一函数

13-2-3 导入模块内多个函数

13-2-4 导入模块所有函数

13-2-5 使用as给函数指定替代名称

13-2-6 使用as给模块指定替代名称

13-3 将自建的类存储在模块内

13-3-1 先前准备工作

13-3-2 建立类内容的模块

13-4 应用自己建立的类模块

13-4-1 导入模块的单一类

13-4-2 导入模块的多个类

13-4-3 导入模块内所有类

13-4-4 import模块名称

13-4-5 模块内导入另一个模块的类

13-5 随机数random模块

13-5-1 randint(_)

13-5-2 choice(_)

13-5-3 shuffle(_)

13-6 时间time模块

13-6-1 time(_)

13-6-2 sleep(_)

13-6-3 asctime(_)

13-6-4 localtime(_)

13-7 系统sys模块

13-7-1 version属性

13-7-2 stdin对象

13-7-3 stdout对象

13-8 keyword模块

13-8-1 kwlist属性

13-8-2 iskeyword(_)

第14章 文件的读取与写入

14-1 文件夹与文件路径

14-1-1 绝对路径与相对路径

14-1-2 os模块与os.path模块

14-1-3 取得当前工作目录os.getcwd(_)

14-1-4 取得绝对路径os.path.abspath

14-1-5 传回特定路段相对路径os.path.relpath(_)

14-1-6 检查路径方法exist/isabs/isdir/isfile

14-1-7 文件与目录的操作mkdir/rmdir/remove/chdir

14-1-8 传回文件路径os.path.join(_)

14-1-9 获得特定文件的大小os.path.getsize(_)

14-1-10 获得特定工作目录的内容os.listdir(_)

14-1-11 获得特定工作目录内容glob

14-1-12 遍历目录树os.walk(_)

14-2 读取文件

14-2-1 读取整个文件read(_)

14-2-2 with关键词

14-2-3 逐行读取文件内容

14-2-4 逐行读取使用readlines(_)

14-2-5 数据组合

14-2-6 字符串的替换

14-2-7 数据的搜寻

14-2-8 数据搜寻使用find(_)

14-3 写入文件

14-3-1 将执行结果写入空的文件内

14-3-2 写入数值资料

14-3-3 输出多行数据的实例

14-3-4 建立附加文件

14-4 shutil模块

14-4-1 文件的复制copy(_)

14-4-2 目录的复制copytree(_)

14-4-3 文件的移动move(_)

14-4-4 文件名的更改move(_)

14-4-5 目录的移动move(_)

14-4-6 目录的更改名称move(_)

14-4-7 删除底下有数据的目录rmtree(_)

14-4-8 安全删除文件或目录send2trash(_)

14-5 文件压缩与解压缩zipFile

14-5-1 执行文件或目录的压缩

14-5-2 读取zip文件

14-5-3 解压缩zip文件

14-6 认识编码格式encode

14-6-1 中文Windows操作系统记事本默认的编码

14-6-2 utf-8编码

14-6-3 认识utf-8编码的BOM

14-7 剪贴板的应用

第15章 程序除错与异常处理

15-1 程序异常

15-1-1 一个除数为0的错误

15-1-2 撰写异常处理程序try - except

15-1-3 try - except - else

15-1-4 找不到文件的错误FileNotFoundError

15-1-5 分析单一文件的字数

15-1-6 分析多个文件的字数

15-2 设计多组异常处理程序

15-2-1 常见的异常对象

15-2-2 设计捕捉多个异常

15-2-3 使用一个except捕捉多个异常

15-2-4 处理异常但是使用Python内置的错误信息

- [15-2-5 捕捉所有异常](#)
 - [15-3 丢出异常](#)
 - [15-4 记录Traceback字符串](#)
 - [15-5 finally](#)
 - [15-6 程序断言assert](#)
 - [15-6-1 设计断言](#)
 - [15-6-2 停用断言](#)
 - [15-7 程序日志模块logging](#)
 - [15-7-1 logging模块](#)
 - [15-7-2 logging的等级](#)
 - [15-7-3 格式化logging信息输出format](#)
 - [15-7-4 时间信息asctime](#)
 - [15-7-5 format内的message](#)
 - [15-7-6 列出levelname](#)
 - [15-7-7 使用logging列出变量变化的应用](#)
 - [15-7-8 正式追踪factorial数值的应用](#)
 - [15-7-9 将程序日志logging输出到文件](#)
 - [15-7-10 隐藏程序日志logging的DEBUG等级使用CRITICAL](#)
 - [15-7-11 停用程序日志logging](#)
 - [15-8 程序除错的典故](#)
- [第16章 正则表达式\(Regular Expression\)](#)
 - [16-1 使用Python硬功夫搜寻文字](#)
 - [16-2 正则表达式的基础](#)
 - [16-2-1 建立搜寻字符串模式](#)
 - [16-2-2 使用re.compile\(_\)建立Regex对象](#)
 - [16-2-3 搜寻对象](#)
 - [16-2-4 findall\(_\)](#)
 - [16-2-5 再看re模块](#)
 - [16-2-6 再看正则表达式](#)
 - [16-3 更多搜寻比对模式](#)
 - [16-3-1 使用小括号分组](#)

[16-3-2 groups\(_\)](#)

[16-3-3 区域号码是在小括号内](#)

[16-3-4 使用管道|](#)

[16-3-5 多个分组的管道搜寻](#)

[16-3-6 使用?号做搜寻](#)

[16-3-7 使用*号做搜寻](#)

[16-3-8 使用+号做搜寻](#)

[16-3-9 搜寻时忽略大小写](#)

[16-4 贪婪与非贪婪搜寻](#)

[16-4-1 搜寻时使用大括号设定比对次数](#)

[16-4-2 贪婪与非贪婪搜寻](#)

[16-5 正则表达式的特殊字符](#)

[16-5-1 特殊字符表](#)

[16-5-2 字符分类](#)

[16-5-3 字符分类的^字符](#)

[16-5-4 正则表示法的^字符](#)

[16-5-5 正则表示法的\\$字符](#)

[16-5-6 单一字符使用通配符“.”](#)

[16-5-7 所有字符使用通配符“.*”](#)

[16-5-8 换行字符的处理](#)

[16-6 MatchObject对象](#)

[16-6-1 re.match\(_\)](#)

[16-6-2 MatchObject几个重要的方法](#)

[16-7 抢救CIA情报员-sub\(_ \)方法](#)

[16-7-1 一般的应用](#)

[16-7-2 抢救CIA情报员](#)

[16-8 处理比较复杂的正则表示法](#)

[16-8-1 将正则表达式拆成多行字符串](#)

[16-8-2 re.VERBOSE](#)

[16-8-3 电子邮件地址的搜寻](#)

[16-8-4 re.IGNORECASE/re.DOTALL/re.VERBOSE](#)

第17章 使用Python处理Word文件

17-1 从Python看Word文件结构

17-2 读取Word文件内容

17-2-1 建立docx对象

17-2-2 获得Paragraph和Run数量

17-2-3 列出Paragraph内容

17-2-4 列出Paragraph内的Run内容

17-2-5 读取和打印文件的应用

17-2-6 读取文件与适度编排输出

17-3 存储文件

17-4 建立文件内容

17-4-1 建立标题

17-4-2 建立段落Paragraph内容

17-4-3 建立Run内容

17-4-4 强制换页输出

17-4-5 插入图片

17-5 建立表格

17-5-1 建立表格内容

17-5-2 插入表格列

17-5-3 计算表格的rows和cols的长度

17-5-4 打印表格内容

17-5-5 表格的样式

17-6 Paragraph样式

17-7 Run的样式

17-8 综合应用——抢救CIA情报员

第18章 使用Python处理PDF文件

18-1 打开PDF文件

18-2 获得PDF文件的页数

18-3 读取PDF页面内容

18-4 检查PDF是否被加密

18-5 解密PDF文件

[18-6 建立新的PDF文件](#)

[18-7 PDF页面的旋转](#)

[18-8 加密PDF文件](#)

[18-9 处理PDF页面重叠](#)

[18-10 破解密码的程序设计](#)

[破解3位数字的密码](#)

[第19章 使用Python处理Excel文件](#)

[19-1 认识Excel窗口](#)

[19-2 读取Excel文件](#)

[19-2-1 打开文件](#)

[19-2-2 取得工作表worksheet名称](#)

[19-2-3 设定当前工作的工作表](#)

[19-2-4 取得工作表内容](#)

[19-2-5 取得工作表内容的栏数和行数](#)

[19-2-6 取得单元格内容](#)

[19-2-7 工作表对象ws的rows和columns](#)

[19-2-8 用整数取代域名](#)

[19-2-9 切片](#)

[19-3 写入Excel文件](#)

[19-3-1 建立Excel文件](#)

[19-3-2 存储Excel文件](#)

[19-3-3 复制Excel文件](#)

[19-3-4 建立工作表](#)

[19-3-5 删除工作表](#)

[19-3-6 写入单元格](#)

[19-3-7 将列表数据写进单元格](#)

[19-4 设定单元格的字体](#)

[19-4-1 Font\(_\)](#)

[19-4-2 字体色彩的设定](#)

[19-5 数学公式的使用](#)

[19-6 设定单元格的高度和宽度](#)

[19-7 单元格对齐方式](#)

[19-8 合并与取消合并单元格](#)

[19-8-1 合并单元格](#)

[19-8-2 取消合并单元格](#)

[19-9 建立图表](#)

[19-9-1 柱形图](#)

[19-9-2 3D柱形图](#)

[19-9-3 饼图](#)

[19-9-4 3D饼图](#)

[第20章 使用Python处理CSV文件](#)

[20-1 建立一个CSV文件](#)

[20-2 用记事本打开CSV文件](#)

[20-3 CSV模块](#)

[20-4 读取CSV文件](#)

[20-4-1 使用open\(_\)打开CSV文件](#)

[20-4-2 建立Reader对象](#)

[20-4-3 用循环列出Reader对象数据](#)

[20-4-4 用循环列出列表内容](#)

[20-4-5 使用列表索引读取CSV内容](#)

[20-4-6 DictReader\(_\)](#)

[20-5 写入CSV文件](#)

[20-5-1 打开欲写入的文件open\(_\)与关闭文件close\(_\)](#)

[20-5-2 建立writer对象](#)

[20-5-3 输出列表writerow\(_\)](#)

[20-5-4 delimiter关键词](#)

[20-5-5 写入字典数据DictWriter\(_\)](#)

[20-6 后记](#)

[第21章 网络爬虫](#)

[21-1 上网不再需要浏览器了](#)

[21-2 下载网页信息使用requests模块](#)

[21-2-1 下载网页使用requests.get\(_\)方法](#)

- [21-2-2 认识Response对象](#)
 - [21-2-3 搜索页特定内容](#)
 - [21-2-4 下载网页失败的异常处理](#)
 - [21-2-5 网页服务器阻挡造成读取错误](#)
 - [21-2-6 爬虫程序伪装成浏览器](#)
 - [21-2-7 存储下载的网页](#)
 - [21-3 检视网页原始文件](#)
 - [21-3-1 建议阅读书籍](#)
 - [21-3-2 以Microsoft浏览器为实例](#)
 - [21-3-3 源文件的重点](#)
 - [21-4 解析网页使用BeautifulSoup模块](#)
 - [21-4-1 建立BeautifulSoup对象](#)
 - [21-4-2 基本HTML文件解析——从简单开始](#)
 - [21-4-3 页标题title属性](#)
 - [21-4-4 去除卷标传回文字text属性](#)
 - [21-4-5 传回所找寻的第一个符合的标签find\(_\)](#)
 - [21-4-6 传回所找寻的所有符合的标签find_all\(_\)](#)
 - [21-4-7 认识HTML元素上下文属性与getText\(_\)](#)
 - [21-4-8 select\(_\)](#)
 - [21-4-9 卷标字符串的get\(_\)](#)
 - [21-5 网络爬虫实战](#)
 - [21-6 命令行窗口](#)
- [第22章 Selenium网络爬虫的王者](#)
- [22-1 顺利使用Selenium工具前的安装工作](#)
 - [22-1-1 安装Selenium](#)
 - [22-1-2 安装浏览器](#)
 - [22-1-3 错误的实例](#)
 - [22-1-4 驱动程序的安装](#)
 - [22-2 获得webdriver的对象类型](#)
 - [22-2-1 以Firefox浏览器为实例](#)
 - [22-2-2 以Chrome浏览器为实例](#)

[22-3 提取网页](#)

[22-4 寻找HTML文件的元素](#)

[22-5 用Python控制点选超链接](#)

[22-6 用Python填写窗体和送出](#)

[22-7 用Python处理使用网页的特殊按键](#)

[22-8 用Python处理浏览器运作](#)

[第23章 数据图表的设计](#)

[23-1 绘制简单的折线图](#)

[23-1-1 显示绘制的图形show\(_\)](#)

[23-1-2 画线plot\(_\)](#)

[23-1-3 线条宽度linewidth](#)

[23-1-4 标题的显示](#)

[23-1-5 坐标轴刻度的设定](#)

[23-1-6 修订图表的起始值](#)

[23-1-7 多组数据的应用](#)

[23-1-8 线条色彩与样式](#)

[23-1-9 刻度设计](#)

[23-1-10 图例legend\(_\)](#)

[23-1-11 保存图片文件](#)

[23-2 绘制散点图scatter\(_\)](#)

[23-2-1 基本散点图的绘制](#)

[23-2-2 绘制系列点](#)

[23-2-3 设定绘图区间](#)

[23-3 Numpy模块](#)

[23-3-1 建立一个简单的数组linspace\(_和arange\(_\)](#)

[23-3-2 绘制波形](#)

[23-3-3 建立不等宽度的散点图](#)

[23-3-4 色彩映射color mapping](#)

[23-4 随机数的应用](#)

[23-4-1 一个简单的应用](#)

[23-4-2 随机数的移动](#)

[23-4-3 隐藏坐标](#)

[23-5 绘制多个图表](#)

[23-5-1 一个程序有多个图表](#)

[23-5-2 含有子图的图表](#)

[23-6 直方图的制作bar\(_\)](#)

[23-7 使用CSV文件绘制图表](#)

[23-7-1 台北2017年1月气象资料](#)

[23-7-2 列出标题数据](#)

[23-7-3 读取最高温与最低温](#)

[23-7-4 绘制最高温](#)

[23-7-5 设定绘图区大小](#)

[23-7-6 日期格式](#)

[23-7-7 在图表增加日期刻度](#)

[23-7-8 日期位置的旋转](#)

[23-7-9 绘制最高温与最低温](#)

[23-7-10 填满最高温与最低温之间的区域](#)

[23-7-11 再谈轴刻度](#)

[第24章 JSON资料](#)

[24-1 认识json数据格式](#)

[24-1-1 对象\(object\)](#)

[24-1-2 数组\(array\)](#)

[24-1-3 json数据存在方式](#)

[24-2 将Python应用在json字符串形式数据](#)

[24-2-1 使用dumps\(_ \)将Python数据转成json格式](#)

[24-2-2 dumps\(_ \)的sort keys参数](#)

[24-2-3 dumps\(_ \)的indent参数](#)

[24-2-4 使用loads\(_ \)将json格式数据转成Python的数据](#)

[24-3 将Python应用在json文件](#)

[24-3-1 使用dump\(_ \)将Python数据转成json文件](#)

[24-3-2 使用load\(_ \)读取json文件](#)

[24-4 简单的json文件应用](#)

[24-5 世界人口数据的json文件](#)

[24-5-1 认识人口统计的json文件](#)

[24-5-2 认识pygal.maps.world的地区代码信息](#)

[第25章 用Python传送手机短信](#)

[25-1 安装twilio模块](#)

[25-2 到Twilio公司注册账号](#)

[25-2-1 申请账号](#)

[25-2-2 获得Account SID](#)

[25-2-3 获得Auth TOKEN](#)

[25-2-4 获得Twilio Number](#)

[25-2-5 设定Twilio使用地区](#)

[25-3 使用Python程序设计发送短信](#)

[第26章 Python与SQLite数据库](#)

[26-1 SQLite基本观念](#)

[26-2 安装SQLite数据库](#)

[26-3 SQLite数据类型](#)

[26-4 建立SQLite数据库表](#)

[26-5 增加SQLite数据库表纪录](#)

[26-6 查询SQLite数据库表](#)

[26-7 更新SQLite数据库表记录](#)

[26-8 删除SQLite数据库表记录](#)

[第27章 用Python处理图像文件](#)

[27-1 认识Pillow模块的RGBA](#)

[27-1-1 __getrgb\(__\)](#)

[27-1-2 __getcolor\(__\)](#)

[27-2 Pillow模块的盒子元组\(Box tuple\)](#)

[27-3 图像的基本操作](#)

[27-3-1 打开图像对象](#)

[27-3-2 图像大小属性](#)

[27-3-3 取得图像对象文件名](#)

[27-3-4 取得图像对象的文件格式](#)

[27-3-5 存储文件](#)

[27-3-6 建立新的图像对象](#)

[27-4 图像的编辑](#)

[27-4-1 更改图像大小](#)

[27-4-2 图像的旋转](#)

[27-4-3 图像的翻转](#)

[27-4-4 图像像素的编辑](#)

[27-5 裁切、复制与图像合成](#)

[27-5-1 裁切图像](#)

[27-5-2 复制图像](#)

[27-5-3 图像合成](#)

[27-5-4 将裁切图片填满图像区间](#)

[27-6 在图像内绘制图案](#)

[27-6-1 绘制点](#)

[27-6-2 绘制线条](#)

[27-6-3 绘制圆或椭圆](#)

[27-6-4 绘制矩形](#)

[27-6-5 绘制多边形](#)

[27-7 在图像内填写文字](#)

[27-8 建立QR code](#)

[第28章 用Python控制鼠标、屏幕与键盘](#)

[28-1 鼠标的控制](#)

[28-1-1 提醒事项](#)

[28-1-2 屏幕坐标](#)

[28-1-3 获得鼠标光标位置](#)

[28-1-4 绝对位置移动鼠标](#)

[28-1-5 相对位置移动鼠标](#)

[28-1-6 键盘Ctrl-C键](#)

[28-1-7 让鼠标位置的输出在固定位置](#)

[28-1-8 单击鼠标click\(_\)](#)

[28-1-9 按住与放开鼠标mouseDown\(_\)和mouseUp\(_\)](#)

[28-1-10 拖曳鼠标](#)

[28-1-11 窗口滚动scroll\(_\)](#)

[28-2 屏幕的处理](#)

[28-2-1 截取屏幕画面](#)

[28-2-2 裁切屏幕图形](#)

[28-2-3 获得图像某位置的像素色彩](#)

[28-2-4 色彩的比对](#)

[28-3 使用Python控制键盘](#)

[28-3-1 基本传送文字](#)

[28-3-2 键盘按键名称](#)

[28-3-3 按下与放开按键](#)

[28-3-4 快速组合键](#)

[28-4 网络窗体的填写](#)

[第29章 文字识别系统](#)

[29-1 安装Tesseract OCR](#)

[29-2 安装pytesseract模块](#)

[29-3 文字识别程序设计](#)

[29-4 识别繁体中文](#)

[29-5 识别简体中文](#)

[第30章 多任务与多线程](#)

[30-1 时间模块datetime](#)

[30-1-1 datetime模块的数据类型datetime](#)

[30-1-2 设定特定时间](#)

[30-1-3 一段时间timedelta](#)

[30-1-4 日期与一段时间相加的应用](#)

[30-1-5 将datetime对象转成字符串](#)

[30-2 多线程](#)

[30-2-1 一个睡眠程序设计](#)

[30-2-2 建立一个简单的多线程](#)

[30-2-3 参数的传送](#)

[30-2-4 线程的命名与取得](#)

- [30-2-5 Daemon线程](#)
- [30-2-6 堵塞主线程join\(_\)](#)
- [30-2-7 检查子线程是否仍在工作isAlive\(_\)](#)
- [30-2-8 了解正在工作的线程](#)
- [30-2-9 自行定义线程和run\(_\)方法](#)
- [30-2-10 资源锁定与解锁Threading.Lock](#)
- [30-2-11 产生锁死](#)
- [30-2-12 资源锁定与解锁Threading.RLock](#)
- [30-2-13 高级锁定threading.Condition](#)
- [30-2-14 queue](#)
- [30-2-15 Semaphore](#)
- [30-2-16 Barrier](#)
- [30-2-17 Event](#)
- [30-3 启动其他应用程序subprocess模块](#)
 - [30-3-1 Popen\(_\)](#)
 - [30-3-2 poll\(_\)](#)
 - [30-3-3 wait\(_\)](#)
 - [30-3-4 Popen\(_\)方法参数的传递](#)
 - [30-3-5 使用默认应用程序打开文件](#)
 - [30-3-6 subprocess.run\(_\)](#)

[第31章 海龟绘图](#)

- [31-1 基本观念与安装模块](#)
- [31-2 绘图初体验](#)
- [31-3 绘图基本练习](#)
- [31-4 控制画笔色彩与线条粗细](#)
- [31-5 绘制圆或弧形](#)
- [31-6 认识与操作海龟图像](#)
 - [31-6-1 隐藏与显示海龟](#)
 - [31-6-2 认识所有的海龟光标](#)
- [31-7 填满颜色](#)
- [31-8 颜色动画的设计](#)

31-9 绘图窗口的相关知识

31-9-1 更改海龟窗口标题与背景颜色

31-9-2 取得/更改窗口宽度与高度

31-9-3 重设世界坐标

31-10 文字的输出

31-11 鼠标与键盘信号

31-11-1 onclick(_)

31-11-2 onkey(_)和listen(_)

第32章 动画与游戏

32-1 建立tkinter对象

32-2 建立按钮

32-3 绘图功能

32-3-1 建立画布

32-3-2 绘线条create_line(_)

32-3-3 绘矩形create_rectangle(_)

32-3-4 绘圆弧create_arc(_)

32-3-5 绘制圆或椭圆create_oval(_)

32-3-6 绘制多边形create_polygon(_)

32-3-7 输出文字create_text(_)

32-3-8 图像的输出create_image(_)

32-3-9 tk窗口标题的设定title(_)

32-3-10 更改画布背景颜色

32-4 滚动条控制画布背景颜色

32-5 动画设计

32-5-1 基本动画

32-5-2 多个球移动的设计

32-5-3 将随机数应用在多个球体的移动

32-5-4 信息绑定

32-6 弹球游戏设计

32-6-1 设计球往下移动

32-6-2 设计让球上下反弹

[32-6-3 设计让球在画布四面反弹](#)

[32-6-4 建立球拍](#)

[32-6-5 设计球拍移动](#)

[32-6-6 球拍与球碰撞的处理](#)

[32-6-7 完整的游戏](#)

[第33章 声音的控制](#)

[33-1 安装与导入](#)

[33-2 一般音效的播放Sound\(\)](#)

[33-3 播放音乐文件music\(\)](#)

[33-4 背景音乐](#)

[33-5 mp3音乐播放器](#)

[第34章 人脸识别系统设计](#)

[34-1 安装OpenCV](#)

[34-1-1 安装OpenCV](#)

[34-1-2 安装Numpy](#)

[34-2 读取和显示图像](#)

[34-2-1 建立OpenCV图像窗口](#)

[34-2-2 读取图像](#)

[34-2-3 使用OpenCV窗口显示图像](#)

[34-2-4 关闭OpenCV窗口](#)

[34-2-5 时间等待](#)

[34-2-6 存储图像](#)

[34-3 OpenCV的绘图功能](#)

[34-4 人脸识别](#)

[34-4-1 下载人脸识别特征文件](#)

[34-4-2 脸部识别](#)

[34-4-3 将脸部存档](#)

[34-4-4 读取摄像头所拍的画面](#)

[34-4-5 脸形比对](#)

[34-5 设计桃园国际机场的出入境人脸识别系统](#)

[附录A 安装Python](#)

[A-1 Windows操作系统的安装Python版](#)

[A-2 Mac OS操作系统的安装Python版](#)

[附录B 安装第三方模块](#)

[B-1 pip工具](#)

[B-1-1 Windows系统Python 3.6.2安装在C:\](#)

[B-1-2 Python 3.6.2安装在硬盘更深层](#)

[B-2 启动DOS与安装模块](#)

[B-2-1 DOS环境](#)

[B-2-2 DOS命令提示字符](#)

[B-3 导入模块安装更新版模块](#)

[B-4 安装更新版模块](#)

[附录C 函数或方法索引表](#)

[\(记录所出现章节\)](#)

[附录D RGB色彩表](#)

[返回总目录](#)

第1章 基本观念

1-1 认识Python

Python是一种直译式 (Interpreted)、面向对象 (Object Oriented) 的程序语言，它拥有完整的函数库，可以协助轻松地完成许多常见的工作。

所谓的直译式语言是指，直译器 (Interpreter) 会将程序代码一句一句直接执行，不需要经过编译 (compile) 动作，将语言先转换成机器码，再予以执行。目前它的直译器是CPython，这是由C语言编写的一个直译程序，与Python一样目前是由Python基金会管理使用。

Python也算是一个动态的高级语言，具有垃圾回收 (garbage collection) 功能，所谓的垃圾回收是指程序执行时，直译程序会主动收回不再需要的动态内存空间，将内存集中管理，这种机制可以减轻程序设计师的负担，当然也就减少了程序设计师犯错的机会。这种垃圾回收功能最早是LISP语言，后来的Java、C#等著名的程序语言都支持这个功能。

由于Python是一个开放的源码 (Open Source)，每个人皆可免费使用或为它贡献，除了它本身有许多内置的套件 (package) 或称模块 (module)，许多单位也为它开发了更多的套件，促使它的功能可以持续扩充，因此Python目前已经是全球最热门的程序语言之一，这也是本书的主题。

1-2 Python的起源

Python的最初设计者是吉多·范罗姆苏 (Guido van Rossum)，他是荷兰人，1956年出生于荷兰哈勒姆，1982年毕业于阿姆斯特丹大学的数学和计算机系，获得硕士学位。

本图片取材自下列网址

https://upload.wikimedia.org/wikipedia/commons/thumb/6/66/Guido_van_Rossum_OSCON_2006.jpg/800px-Guido_van_Rossum_OSCON_2006.jpg

吉多·范罗姆苏 (Guido van Rossum) 在1996年为一本O'Reilly出版社出版作者名为Mark Lutz所著的《Programming Python》的序言表示：“6年前，1989年我想在圣诞节期间思考设计一种程序语言打发时间，当时我正在构思一个新的脚本(script)语言的解释器，它是ABC语言的后代，期待这个程序语言对UNIX C的程序语言设计师会有吸引力。基于我是蒙提派森飞行马戏团 (Monty Python's Flying Circus) 的疯狂爱好者，所以就以 Python为名当作这个程序的标题名称。”



在一些Python的文件或书封面喜欢用蟒蛇代表Python，从吉多·范罗姆苏的上述序言可知，Python灵感的来源是马戏团名称而非蟒蛇。

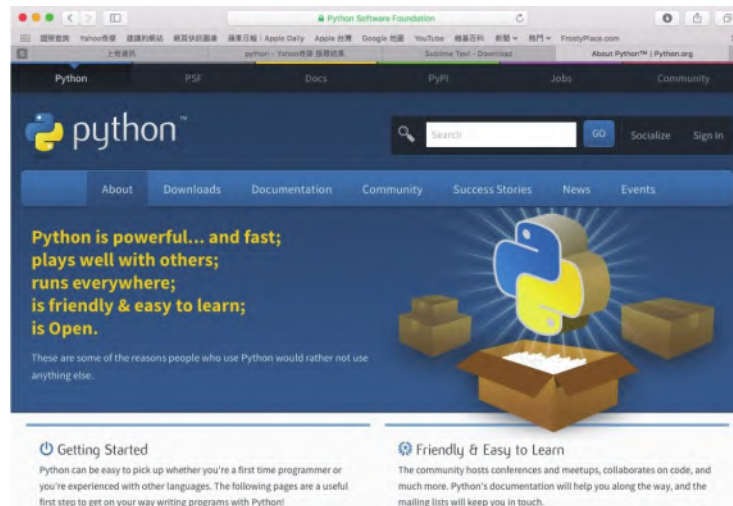
1999年他向美国国防部下的国防高等研究计划署DARPA (Defense Advanced Research Projects Agency) 提出 Computer Programming for Everybody的研发经费申请，他提出了下列Python的目标。

- 这是一个简单直觉式的程序语言，可以和主要程序语言一样强大。
- 这是开放源码 (Open Source)，每个人皆可自由使用与贡献。
- 程序代码像英语一样容易理解与使用。

- 可在短期间内开发一些常用功能。

现在上述目标皆已经实现了，Python已经与C/C++、Java一样成为程序设计师必备的程序语言，然而它却比C/C++和Java更容易学习。

目前Python语言是由Python软件基金会管理，有关新版软件的相关信息可以在这个基金会网址(www.python.org) 上下载浏览。



1-3 Python语言发展史

1991年Python正式诞生，当时的操作系统平台是Mac。尽管吉多·范罗姆苏(Guido van Rossum)坦承Python是构思于ABC语言，但是ABC语言并没有成功，吉多·范罗姆苏本人认为ABC语言并不是一个开放的程序语言，是主要原因。因此，在Python的推广中，他避开了这个错误，将Python推向开放式系统，而获得了很大成功。

Python 2.0发表

2000年10月16日Python 2.0正式发表，主要是增加了垃圾回收的功能，同时支持Unicode。

所谓的Unicode 是一种适合多语系的编码规则，主要方式是使用可变长度字节方式存储字符，以节省内存空间。例如，对于英文字母

而言使用1个字节空间存储即可，对于含有附加符号的希腊文、拉丁文或阿拉伯文等则用2个字节空间存储字符，两岸华人所使用的中文字则是以3个字节空间存储字符，只有极少数的平面辅助文字需要4个字节空间存储字符。也就是说这种编码规则已经包含了全球所有语言的字符了，所以采用这种编码方式设计程序时，其他语系的程序只要支持Unicode编码皆可显示。例如：法国人即使使用法文版的程序，也可以正常显示中文字。

Python 3.0发表

2008年12月3日Python 3.0正式发表。一般程序语言的发展会考虑到兼容特性，但是Python 3在开发时为了不要受到先前2.x版本的束缚，因此没有考虑兼容特性，所以许多早期版本开发的程序是无法在Python 3.x版本上执行。

不过为了解决这个问题，尽管发表了Python 3.x版本，后来陆续将3.x版本的特性移植到Python 2.6/2.7x版本上，所以现在我们进入Python基金会网站时，可以发现2.7x版本和3.6x版本的软件可以下载。

笔者经验提醒：有一些早期开发的冒险游戏软件只支持Python 2.7x版本，目前尚未支持Python 3.6x版本。不过相信这些软件未来也将朝向支持Python 3.6x版本的路迈进。

Python基金会提醒：Python 2.7x已经被确定为最后一个Python 2.x的版本。

笔者在撰写此书时，同时下载2个版本彻底了解了这2个版本的区别，基本上所有程序是以Python 3.x版本作为撰写此书的主要依据。

1-4 Python的应用范围

尽管Python是一个非常适合初学者学习的程序语言，在国外有许多儿童程序语言教学也是以Python为工具，然而它却是一个功能强大的程序语言，下列是它的部分应用。

- 设计动画游戏。
- 支持图形接口(Graphical User Interface, GUI)开发。
- 开发与管理网站。
- 执行大数据分析。
- Google、Yahoo!、YouTube、NASA、Dropbox(文件分享服务)、Reddit(社交网站)在内部皆大量使用Python做开发工具。
- 黑客攻防。

1-5 跨平台的程序语言

Python是一种跨平台的程序语言，几乎主要操作系统，例如，Windows、Mac OS、UNIX/LINUX等，皆可以安装和使用，本书所有程序实例皆在Windows和Mac OS下测试完成。

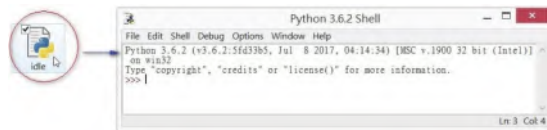
跨平台的程序语言意味，你可以在某一个平台上使用Python设计一个程序，未来这个程序也可以在其他平台上顺利运作。

1-6 系统的安装与执行

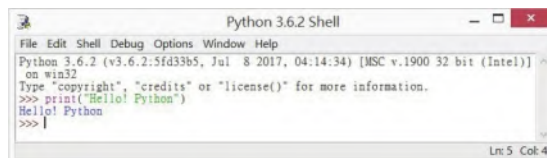
有关安装Python的步骤请参考附录A。下列将以Python 3.6x版本为例做说明。

1-6-1 在Windows启动与执行Python

①请点选在附录A所建、在Windows桌面上的idle 图示，将看到下列Python Shell窗口。



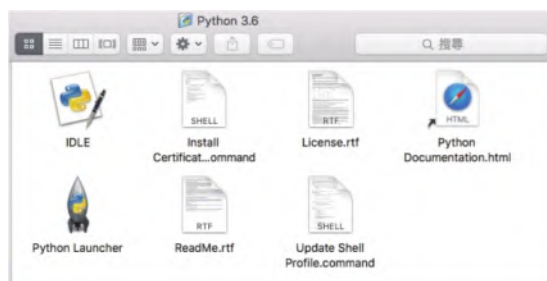
②上述  符号是提示信息，可以在此输入Python指令，下列是一个简单print()函数，目的是输出字符串。



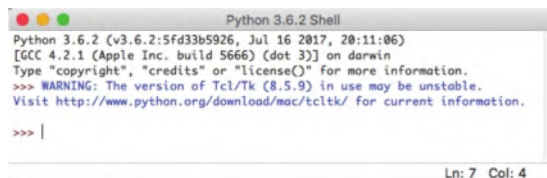
由上图可以确定我们成功执行第一个Python的程序实例了。

1-6-2 在Mac OS启动与执行Python

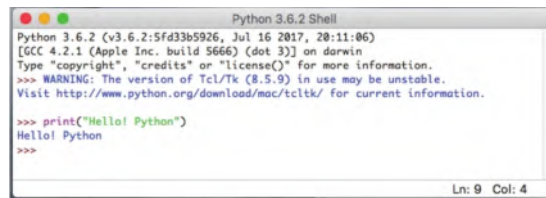
①请点选在应用程序 文件夹的Python 3.6图标，将看到下列窗口。



②请点选IDLE图示，可以看到下列Python Shell窗口。其中可以看到警告WARNING信息“unstable”，读者可以不必理会。



③上述  符号是提示信息，可以在此输入Python指令，下列是一个简单print()函数，目的是输出字符串。

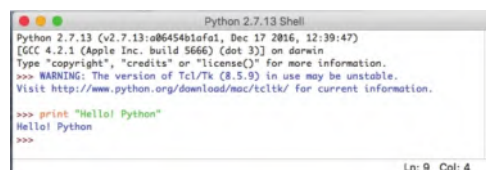


```
Python 3.6.2 Shell
Python 3.6.2 (v3.6.2:5fd33b5926, Jul 16 2017, 20:11:06)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.
>>> print("Hello! Python")
Hello! Python
>>>
```

由上图可以确定我们成功执行第一个Python的程序实例了。

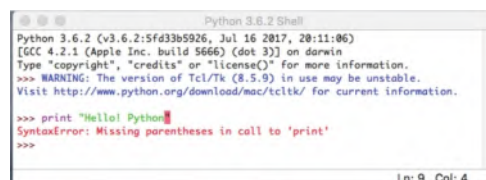
1-7 Python 2与Python 3不相容的验证

下列是早期在Python 2上执行输出字符串的print用法。



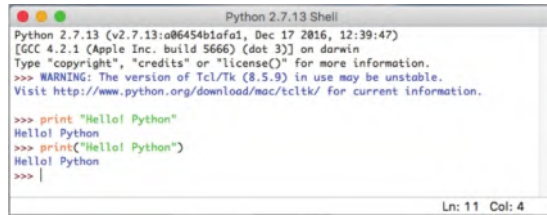
```
Python 2.7.13 Shell
Python 2.7.13 (v2.7.13:a86454b1afaf, Dec 17 2016, 12:39:47)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.
>>> print "Hello! Python"
Hello! Python
>>>
```

如果相同的输出方式应用在Python 3将出现错误。



```
Python 3.6.2 Shell
Python 3.6.2 (v3.6.2:5fd33b5926, Jul 16 2017, 20:11:06)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.
>>> print "Hello! Python"
SyntaxError: Missing parentheses in call to 'print'
>>>
```

会出现错误的原因是在Python 3，print()已经是一个函数。不过笔者在1-3节也提过，Python基金会后来陆续将3.x版本的特性移植到Python 2.6/2.7x版本上，所以如果在Python 2.6/2.7x版本上，使用print()函数，将可以得到正确的输出。



```
Python 2.7.13 Shell
Python 2.7.13 (v2.7.13:006454b10fa1, Dec 17 2016, 12:39:47)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

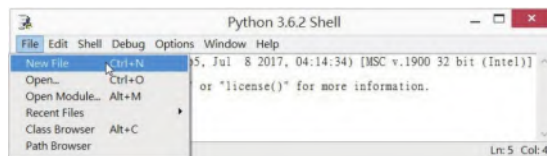
>>> print "Hello! Python"
Hello! Python
>>> print("Hello! Python")
Hello! Python
>>> |
```

1-8 文件的建立、存储、执行与打开

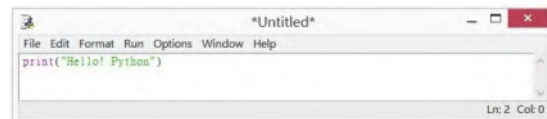
如果设计一个程序每次均要在Python Shell窗口环境重新输入指令的话，这是一件麻烦的事，所以程序设计时，可以将所设计的程序保存在文件内是一件重要的事。

1-8-1 文件的建立

①在Python Shell窗口可以执行File/New File，建立一个空白的Python文件。



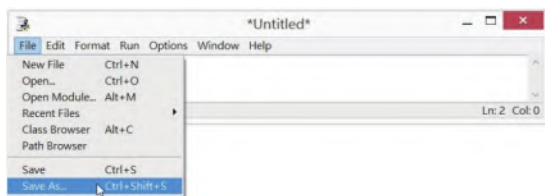
②然后可以建立一个Untitled窗口，窗口内容是空白，下列是笔者在空白文件内输入一道指令的实例。



如果想要执行上述文件，需要先存储上述文件。

1-8-2 文件的存储

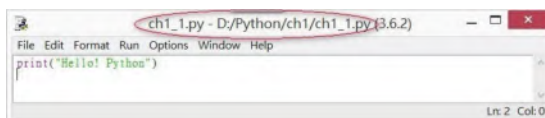
①可以执行File/Save As 存储文件。



②然后将看到另存新文件 对话框，此例笔者将文件存储在 D:/Python/ch1文件夹，档名是ch1_1 (Python的扩展名是py)，可以得到下列结果。



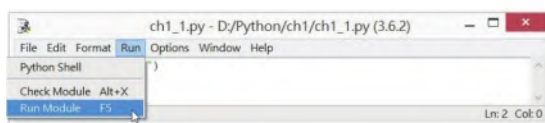
③请单击存档 按钮。



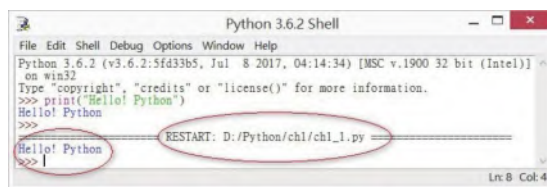
这时原标题Untitled已经改为ch1_1.py文件了。

1-8-3 文件的执行

①可以执行Run/Run Module ，就可以正式执行先前所建的 ch1_1.py文件。



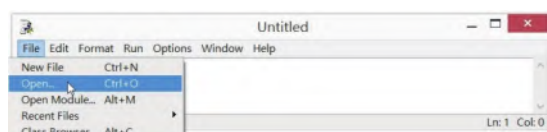
②执行后，在原先的Python Shell窗口可以看到执行结果。



学习到此，恭喜你已经成功地建立一个Python文件，同时执行成功了。

1-8-4 打开文件

①假设已经离开ch1_1.py文件，未来想要打开这个程序文件，可以执行File/Open。



②然后会出现打开旧文件对话框，请选择欲打开的文件即可。

1-9 程序注释

程序注释主要功能是让你所设计的程序可读性更高，更容易了解。在企业工作，一个实用的程序可以很轻易超过几千或上万行，此时你可能需设计好几个月，程序加上注释，可方便你或他人，未来较方便地了解程序内容。

1-9-1 注释符号#

不论是Python Shell直译器或是Python程序文件中，“#”符号右边的文字，皆是称程序注释，Python语言的直译器会忽略此符号右边的文字。可参考下列实例。

实例1： 在Python Shell窗口注释的应用1，注释可以放在程序叙述的右边。

```
>>> print("Python语言 - 王者归来")    # 打印本书名称
Python语言 - 王者归来
>>> |
```

实例2： 在Python Shell窗口注释的应用2，注释可以放在程序叙述的前面。

```
>>> # 打印本书名称
>>> print("Python语言 - 王者归来")
Python语言 - 王者归来
>>> |
```

程序实例ch1_2.py： 重新设计ch1_1.py，为程序增加注释。

```
1 # ch1_2.py
2 print("Hello! Python")    # 打印字符串
```

Python程序左边是没有行号的，这是笔者为了读者阅读方便加上去的。

1-9-2 三个单引号或双引号

如果要进行大段落的注释，可以用三个单引号或双引号将注释文字包夹。

程序实例ch1_3.py： 以三个单引号当作注释。

```
1 '''
2 程序实例ch1_3.py
3 作者:洪锦魁
4 使用三个单引号当作注释
5 '''
6 print("Hello! Python")    # 打印字符串
```

上述前5行是程序注释。

程序实例ch1_4.py： 以三个双引号当作注释。

```
1  """
2  程序实例ch1_4.py
3  作者:洪锦魁
4  使用三个双引号当作注释
5  """
6  print("Hello! Python")  # 打印字符串
```

上述前5行是程序注释。

第2章 认识变量与基本数学运算

本章将从基本数学运算开始，一步一步讲解变量的使用与命名，接着介绍Python的算术运算。

2-1 用Python做计算

假设读者到麦当劳打工，一小时可以获得120元时薪，如果想计算一天工作8小时，可以获得多少工资？我们可以用计算器执行“ 120×8 ”，然后得到执行结果。在Python Shell，可以使用下列方式计算。

```
>>> 120 * 8
960
>>>
```

如果一年实际工作天数是300天，可以用下列方式计算一年所得。

```
>>> 120 * 8 * 300
288000
>>>
>>> |
```

如果读者一个月花费是9000元，可以用下列方式计算一年可以存储多少钱。

```
>>> 9000 * 12
108000
>>> 288000 - 108000
180000
>>>
```

上述笔者先计算一年的花费，再将一年的收入减去一年的花费，可以得到所存储的金额。本章笔者将一步一步推导应如何以程序观念，处理一般的运算问题。

2-2 认识变量

变量是一个暂时存储数据的地方，对于2-1节的内容而言，如果你今天获得了调整时薪，时薪从120元调整到125元，如果想要重新计算一年可以存储多少钱，你将发现所有的计算将要重新开始。为了解决这个问题，我们可以考虑将时薪设为一个变量，未来如果有调整薪资，直接更改变量内容即可。

在Python中可以用“=”等号设定变量的内容，在这个实例中，我们建立了一个变量x，然后用下列方式设定时薪。

```
>>> x = 120
>>>
```

如果想要用Python列出时薪资料可以使用print()函数。

```
>>> print(x)
120
>>>
```

如果今天已经调整薪资，时薪从120元调整到125元，那么我们可以用下列方式表达。

```
>>> x = 125
>>> print(x)
125
>>>
```

注 在Python Shell环境，也可以直接输入变量名称获得执行结果。

```
>>> x = 125
>>> x
125
>>>
```

一个程序是可以使用多个变量的，如果我们想计算一天工作8小时，一年工作300天，可以赚多少钱，假设用变量y存储一年工作所赚的钱，可以用下列方式计算。

```
>>> x = 125
>>> y = x * 8 * 300
>>> print(y)
300000
>>>
```

如果每个月花费是9000元，我们使用变量z存储每个月花费，可以用下列方式计算每年的花费，我们使用a存储每年的花费。

```
>>> z = 9000
>>> a = z * 12
>>> print(a)
108000
>>>
```

如果我们想计算每年可以存储多少钱，我们使用b存储每年所存储的钱，可以使用下列方式计算。

```
>>> x = 125
>>> y = x * 8 * 300
>>> z = 9000
>>> a = z * 12
>>> b = y - a
>>> print(b)
192000
>>>
```

上述我们很顺利地使用Python Shell计算了每年可以存储多少钱，可是上述使用Python Shell做运算潜藏最大的问题是，只要过了一段时间，我们可能忘记当初所有设定的变量是代表什么意义。因此在设计程序时，如果为变量取个有意义的名称，未来看到程序时，可以比较容易记得。下列是笔者重新设计的变量名称：

- 时薪：hourly_salary，用此变量代替x，每小时的薪资。
- 年薪：annual_salary，用此变量代替y，一年工作所赚的钱。
- 月支出：monthly_fee，用此变量代替z，每个月花费。
- 年支出：annual_fee，用此变量代替a，每年的花费。
- 年存储：annual_savings，用此变量代替b，每年所存储的钱。

如果现在使用上述变量重新设计程序，可以得到下列结果。

```
>>> hourly_salary = 125
>>> annual_salary = hourly_salary * 8 * 300
>>> monthly_fee = 9000
>>> annual_fee = monthly_fee * 12
>>> annual_savings = annual_salary - annual_fee
>>> print(annual_savings)
192000
>>>
```

相信经过上述说明，读者应该了解变量的基本意义了。

2-3 认识程序的意义

延续上一节的实例，如果我们时薪改变、工作天数改变或每个月的花费改变，所有输入与运算皆要重新开始，而且每次皆要重新输入程序代码，这是一件很费劲的事，同时很可能会输入错误，为了解决这个问题，我们可以使用Python Shell打开一个档案，将上述运算存储在档案内，这个档案就是所谓的程序。未来有需要时，再打开重新运算即可。

程序实例ch2_1.py： 使用程序计算每年可以存储多少钱，下列是整个程序设计。

```
1 # ch2_1.py
2 hourly_salary = 125
3 annual_salary = hourly_salary * 8 * 300
4 monthly_fee = 9000
5 annual_fee = monthly_fee * 12
6 annual_savings = annual_salary - annual_fee
7 print(annual_savings)
```

执行结果

```
===== RESTART: D:/Python/ch2/ch2_1.py =====
192000
>>>
```

未来我们时薪改变、工作天数改变或每个月的花费改变，只要适度修改变量内容，就可以获得正确的执行结果。

2-4 认识注释的意义

上一节的程序ch2_1.py，尽管我们已经为变量设定了有意义的名称，其实时间一久，常常还是会忘记各个指令的内涵。所以笔者建议，设计程序时，适度地为程序代码加上注释。在1-9节已经讲解注释的方法，下列将直接以实例说明。

程序实例ch2_2.py： 重新设计程序ch2_1.py，为程序代码加上注释。

```

1 # ch2_2.py
2 hourly_salary = 125
3 annual_salary = hourly_salary * 8 * 300
4 monthly_fee = 9000
5 annual_fee = monthly_fee * 12
6 annual_savings = annual_salary - annual_fee
7 print(annual_savings)

```

设定月薪
计算年薪
设定每月花费
计算每年花费
计算每年储存金额
列出每年储存金额

执行结果 与ch2_1.py相同。

相信经过上述注释后，即使再过10年，只要一看到程序也可轻松了解整个程序的意义。

2-5 Python变量与其他程序语言的差异

许多程序语言变量在使用前需要先定义，Python对于变量的使用则是可以在需要时，再直接设定使用。有些程序语言在定义变量时，需要设定变量的数据类型，Python则不需要设定，它会针对变量值的内容自行设定数据类型。

2-6 变量的命名原则

Python对于变量的命名有一些规则要遵守，否则会造成程序错误。

- 必须由英文字母、_(下画线)或中文字开头，建议使用英文字母。
- 变量名称只能由英文字母、数字、_(下画线)或中文字所组成。
- 英文字母大小写是敏感的，例如，Name与name被视为不同变量名称。
- Python系统保留字（或称关键词）或Python内置函数名称不可当作变量名称。

注 虽然变量名称可以用中文字，不过笔者不建议使用中文字，是怕将来也许有兼容性的问题。

下列是不可当作变量名称的Python系统保留字。

and	as	assert	break	class	continue
def	del	elif	else	except	False
finally	for	from	global	if	import
in	is	lambda	none	nonlocal	not
or	pass	raise	return	True	try
while	with	yield			

下列是不可当作变量名称的Python系统内置函数，若是不小心将系统内置函数名称当作变量，程序本身不会错误，但是原先函数功能会丧失。

abs()	all()	any()	apply()	basestring()
bin()	bool()	buffer()	bytearray()	callable()
chr()	classmethod()	cmp()	coerce()	compile()
complex()	delattr()	dict()	dir()	divmod()
enumerate()	eval()	execfile()	file()	filter()
float()	format()	frozenset()	getattr()	globals()
hasattr()	hash()	help()	hex()	id()
input()	int()	intern()	isinstance()	issubclass()
iter()	len()	list()	locals()	long()
map()	max()	memoryview()	min()	next()
object()	oct()	open()	ord()	pow()
print()	property()	range()	raw_input()	reduce()
reload()	repr()	reversed()	round()	set()
setattr()	slice()	sorted()	staticmethod()	str()
sum()	super()	tuple()	type()	unichr()
unicode()	vars()	xrange()	zip()	_import()

实例1： 下列是一些不合法的变量名称。

```
sum, 1 # 变量不可有“,”
3y     # 变量不可由阿拉伯数字开头
x$2    # 变量不可有“$”符号
and     # 这是系统保留字不可当作变量名称
```

实例2： 下列是一些合法的变量名称。

SUM

_fg

x5

总和

实例3： 下列3个代表不同的变量。

SUM

Sum

sum

2-7 基本数学运算

2-7-1 四则运算

四则运算是指加(+)、减(-)、乘(×)和除(/)。

实例1： 下列是加法与减法运算实例。

```
>>> x = 5 + 6          # 将5加6设定给变量x
>>> print(x)
11
>>> y = x - 10         # 将x减10设定给变量y
>>> print(y)
1
>>>
```

实例2： 乘法与除法运算实例。

```
>>> x = 5 * 9          # 将5乘9设定给变量x
>>> print(x)
45
>>> y = 9 / 5          # 将9除以5设定给变量y
>>> print(y)
1.8
>>>
```

2-7-2 余数和整除

余数(mod)所使用的符号是“%”，可计算出除法运算中的余数。整除所使用的符号是“//”，是指除法运算中只保留整数部分。

实例1： 余数和整除运算实例。

```
>>> x = 9 % 5          # 将9除以5的余数设定给变量x
>>> print(x)
4
>>> y = 9 // 2         # 将9除以2的整数结果设定给变量y
>>> print(y)
4
>>>
```

2-7-3 次方

次方的符号是“**”。

实例1： 平方、次方的运算实例。

```
>>> x = 3 ** 2         # 将3的平方设定给变量x
>>> print(x)
9
>>> y = 3 ** 3         # 将3的3次方设定给变量y
>>> print(y)
27
>>>
```

2-7-4 Python语言控制运算的优先级

Python语言碰上计算式同时出现在一个指令内时，除了括号“（”、“）”最优先外，其余计算优先次序如下。

①次方。

②乘法、除法、求余数(%)、求整数(//)，彼此依照出现顺序运算。

③加法、减法，彼此依照出现顺序运算。

实例1： Python语言控制运算的优先级的应用。

```
>>> x = ( 5 + 6 ) * 8 - 2
>>> print(x)
86
>>> y = 5 + 6 * 8 - 2
>>> print(y)
51
>>>
```

2-8 赋值运算符

常见的赋值运算符如下：

运算符	实例	说明
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b
//=	a //= b	a = a // b
**=	a **= b	a = a ** b

实例1： 赋值运算符的实例说明。

```
>>> x = 10
>>> x += 5
>>> print(x)
15
>>> x = 10
>>> x -= 5
>>> print(x)
5
>>> x = 10
>>> x *= 5
>>> print(x)
50
>>> x = 10
>>> x /= 5
>>> print(x)
2.0
>>> x = 10
>>> x %= 5
>>> print(x)
0
>>> x = 10
>>> x //= 5
>>> print(x)
2
>>> x = 10
>>> x **= 5
>>> print(x)
100000
>>>
```

2-9 Python等号的多重指定使用

使用Python时，可以一次设定多个变量等于某一数值。

实例1： 设定多个变量等于某一数值的应用。

```
>>> x = y = z = 10
>>> print(x)
10
>>> print(y)
10
>>> print(z)
10
>>>
```

Python也允许多个变量同时指定不同的数值。

实例2： 设定多个变量，每个变量有不同值。

```
>>> x, y, z = 10, 20, 30
>>> print(x, y, z)
10 20 30
>>>
```

当执行上述多重设定变量值后，甚至可以执行更改变量内容。

实例3： 将2个变量内容交换。

```
>>> x, y = 10, 20
>>> print(x, y)
10 20
>>> x, y = y, x
>>> print(x, y)
20 10
>>>
```

上述原先x, y分别设为10, 20，但是经过多重设定后变为20, 10。

2-10 删除变量

程序设计时，如果某个变量不再需要，可以使用del指令将此变量删除，相当于可以收回原变量所占的内存空间，以节省内存空间。删除变量的格式如下：

del变量名称

实例1： 验证变量名称回收后，将无法再使用。此例，尝试输出已删除的变量，然后程序出现错误信息。

```
>>> x = 10
>>> print(x)
10
>>> del x
>>> print(x)
Traceback (most recent call last):
  File "<pyshell#157>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

设定变量 x

删除变量 x

输出变量 x

由于变量已经删除，所以输出时出现 x 为未定义的错误信息

2-11 Python的断行

2-11-1 一行有多个语句

在Python是允许一行有多个语句，彼此用“;”隔开即可，尽管Python有提供此功能，不过笔者不鼓励如此撰写程序代码。

程序实例ch2_3.py： 一行有多个语句的实例。

```
1 # ch2_3.py
2 x = 10
3 print(x)
4 y = 20; print(y)    # 一行有2个语句, 不过不鼓励这种写法
```

执行结果

```
===== RESTART: D:/Python/ch2/ch2_3.py =====
10
20
>>>
```

2-11-2 将一个语句分成多行

在设计大型程序时，常会碰上一个语句很长，需要分成2行或更多行撰写，此时可以在叙述后面加上“\”符号，Python解释器会将下一行的语句视为这一行的语句。特别注意，在“\”符号右边不可加上任何符号或文字，即使是注释符号也不允许。

另外，也可以在语句内使用小括号，如果使用小括号，就可以在语句右边加上注释符号。

程序实例ch2_4.py： 将一个语句分成多行的应用。

```
1  # ch2_4.py
2  a = b = c = 10
3  x = a + b + c + 12
4  print(x)
5  # 续行方法1
6  y = a + \
7      b + \
8      c + \
9      12
10 print(y)
11 # 续行方法2
12 z = ( a +      # 此处可以加上批注
13      b +
14      c +
15      12 )
16 print(z)
```

执行结果

```
===== RESTART: D:\Python\ch2\ch2_4.py =====
42
42
42
>>>
```

第3章 Python的基本数据类型

Python的基本数据类型有下列几种：

- 数值数据类型：常见的数值数据又可分成整数(int)和浮点数(float)。
- 布尔值(Boolean)数据类型。
- 字符串(string)数据类型。

3-1 type()函数

在正式介绍数据类型前，笔者想介绍一个函数type()，这个函数可以列出变量的数据类型类别。

程序实例ch3_1.py： 列出数值变量的数据类型。

```
1 # ch3_1.py
2 x = 10
3 y = x / 3
4 print(x)
5 print(type(x))
6 print(y)
7 print(type(y))
```

执行结果

```
===== RESTART: D:/Python/ch3/ch3_1.py =====
10
<class 'int'>
3.3333333333333335
<class 'float'>
>>>
```

从上述执行结果可以看到，变量x的内容是10，数据类型是整数(int)。变量y的内容是3.3...5，数据类型是浮点数(float)。下一节会说明，为何是这样。

3-2 数值数据类型

Python在定义变量时可以不用设定这个变量的数据类型，未来如果这个变量内容是整数，这个变量就是整数(int)数据类型，如果这个变量内容是浮点数，这个变量就是浮点数(float)数据类型。整数与浮点数最大的区别是，整数不含小数点，浮点数含小数点。

程序实例ch3_2.py： 测试浮点数。

```
1 # ch3_2.py
2 x = 10.0
3 print(x)
4 print(type(x))
```

执行结果

```
===== RESTART: D:/Python/ch3/ch3_2.py =====
10.0
<class 'float'>
>>>
```

在程序实例ch3_1.py中，x变数的值是“10”，表示x变量是整数变量，在这个实例中，x变量的值是“10.0”，表示x变量是浮点数变量。

3-2-1 整数与浮点数的运算

Python程序设计时不同数据类型也可以执行运算，程序设计时常会发生整数与浮点数之间的数据运算，Python具有简单自动转换能力，在计算时会将整数转换为浮点数再执行运算。

程序实例ch3_3.py： 不同数据类型的运算。

```
1 # ch3_3.py
2 x = 10
3 y = x + 5.5
4 print(x)
5 print(type(x))
6 print(y)
7 print(type(y))
```

执行结果

```
===== RESTART: D:/Python/ch3/ch3_3.py =====
10
<class 'int'>
15.5
<class 'float'>
>>>
```

上述变量y，由于是整数与浮点数的加法，所以结果是浮点数。此外，如果某一个变量是整数，但是最后所存储的值是浮点数，Python也会将此变量转成浮点数。

程序实例ch3_4.py： 整数转换成浮点数的应用。

```
1 # ch3_4.py
2 x = 10
3 print(x)
4 print(type(x))      # 加法前列出x数据类型
5 x = x + 5.5
6 print(x)
7 print(type(x))      # 加法后列出x数据类型
```

执行结果

```
===== RESTART: D:/Python/ch3/ch3_4.py =====
10
<class 'int'>
15.5
<class 'float'>
>>>
```

原先变量x所存储的值是整数，所以列出是整数。后来存储了浮点数，所以列出是浮点数。

3-2-2 2进位整数与函数bin()

我们可以用2进位方式代表整数，Python中定义凡是0b开头的数字，代表这是2进位的整数。

bin()函数可以将一般数字转换为2进位。

程序实例ch3_5.py： 将10进位数值与2进位数值互转的应用。

```

1 # ch3_5.py
2 x = 0b1101      # 这是2进位整数
3 print(x)         # 列出10进位的结果
4 y = 13          # 这是10进位整数
5 print(bin(y))    # 列出转换成2进位的结果

```

执行结果

```

===== RESTART: D:\Python\ch3\ch3_5.py =====
13
0b1101
>>>

```

3-2-3 8进位整数

我们可以用8进位方式代表整数，Python中定义凡是0o开头的数字，代表这是8进位的整数。

oct()函数可以将一般数字转换为8进位。

程序实例ch3_6.py： 将10进位数值与8进位数值互转的应用。

```

1 # ch3_6.py
2 x = 0o57        # 这是8进位整数
3 print(x)         # 列出10进位的结果
4 y = 47          # 这是10进位整数
5 print(oct(y))    # 列出转换成8进位的结果

```

执行结果

```

===== RESTART: D:\Python\ch3\ch3_6.py =====
47
0o57
>>>

```

3-2-4 16进位整数

我们可以用16进位方式代表整数，Python中定义凡是0x开头的数字，代表这是16进位的整数。

hex()函数可以将一般数字转换为16进位。

程序实例ch3_7.py： 将10进位数值与16进位数值互转的应用。

```

1 # ch3_7.py
2 x = 0x5D          # 这是16进位整数
3 print(x)          # 列出10进位的结果
4 y = 93            # 这是10进位整数
5 print(hex(y))     # 列出转换成16进位的结果

```

执行结果

```

===== RESTART: D:\Python\ch3\ch3_7.py =====
93
0x5d
>>>

```

3-2-5 强制数据类型的转换

有时候我们设计程序时，可以自行强制使用下列函数，转换变量的数据类型。

- `int()`：将数据类型强制转换为整数。
- `float()`：将数据类型强制转换为浮点数。

程序实例ch3_8.py： 将浮点数强制转换为整数的运算。

```

1 # ch3_8.py
2 x = 10.5
3 print(x)
4 print(type(x))    # 加法前列出x数据类型
5 y = int(x) + 5
6 print(y)
7 print(type(y))    # 加法后列出y数据类型

```

执行结果

```

===== RESTART: D:\Python\ch3\ch3_8.py =====
10.5
<class 'float'>
15
<class 'int'>
>>>

```

程序实例ch3_9.py： 将整数强制转换为浮点数的运算。

```

1 # ch3_9.py
2 x = 10
3 print(x)
4 print(type(x))    # 加法前列出x数据类型
5 y = float(x) + 10
6 print(y)
7 print(type(y))    # 加法后列出y数据类型

```

执行结果


```
===== RESTART: D:/Python/ch3/ch3_9.py =====
10
<class 'int'>
20.0
<class 'float'>
>>>
```

3-2-6 数值运算常用的函数

下列是数值运算时常用的函数。

- `abs( )` : 计算绝对值。
- `pow(x, y)` : 返回x的y次方。
- `round( )` : 返回五 舍六 入，留意不是四舍五入。

程序实例ch3_10.py: `abs( )`、`pow( )`、`round( )`函数的应用。

```
1 # ch3_10.py
2 x = -10
3 print("以下输出abs( )函数的应用")
4 print(x)          # 输出x变数
5 print(abs(x))      # 输出abs(x)
6 x = 5
7 y = 3
8 print("以下输出pow( )函数的应用")
9 print(pow(x, y))   # 输出pow(x,y)
10 x = 48.4
11 print("以下输出round( )函数的应用")
12 print(x)          # 输出x变数
13 print(round(x))    # 输出round(x)
14 x = 48.5
15 print(x)          # 输出x变数
16 print(round(x))    # 输出round(x)
17 x = 48.6
18 print(x)          # 输出x变数
19 print(round(x))    # 输出round(x)
```

执行结果

```
===== RESTART: D:\Python\ch3\ch3_10.py =====
以下输出abs( )函数的应用
-10
10
以下输出pow( )函数的应用
125
以下输出round( )函数的应用
48.4
48
48.5
48
48.6
49
>>>
```

3-3 布尔值数据类型

Python的布尔值 (Boolean) 数据类型的值有两种，True(真)或False(伪)，它的数据类型代号是bool。这个布尔值一般是应用在程序流程的控制，特别是在条件表达式中，程序可以根据这个布尔值判断应该如何执行工作。

程序实例ch3_11.py: 列出布尔值与布尔值的数据类型。

```
1 # ch3_11.py
2 x = True
3 print(x)
4 print(type(x))      # 列出x数据类型
5 y = False
6 print(y)
7 print(type(y))      # 列出y数据类型
```

执行结果

```
===== RESTART: D:/Python/ch3/ch3_11.py =====
True
<class 'bool'>
False
<class 'bool'>
>>>
```

如果将布尔值数据类型强制转换成整数，当原值是True，将得到1；当原值是False，将得到0。

程序实例ch3_12.py: 将布尔值强制转换为整数，同时列出转换的结果。

```
1 # ch3_12.py
2 x = True
3 print(int(x))
4 print(type(x))      # 列出x数据类型
5 y = False
6 print(int(y))
7 print(type(y))      # 列出y数据类型
```

执行结果

```
===== RESTART: D:/Python/ch3/ch3_12.py =====
1
<class 'bool'>
0
<class 'bool'>
>>>
```

3-4 字符串数据类型

所谓的字符串(string)数据是指两个单引号 (')之间或 是两个双引号 (")之间任意个数字元符号的数据，它的数据类型代号是str。在英文字符串的使用中常会发生某字中间有单引号，其实这是文字的一部分，如下所示：

```
This is James's ball
```

如果我们用单引号去处理上述字符串将产生错误，如下所示：

```
>>> x = 'This is James's ball'
SyntaxError: invalid syntax
>>>
```

碰到这种情况，我们可以用双引号解决，如下所示：

```
>>> x = "This is James's ball"
>>> print(x)
This is James's ball
>>>
```

程序实例ch3_13.py： 使用单引号与双引号设定与输出字符串数据的应用。

```
1 # ch3_13.py
2 x = "DeepStone means Deep Learning" # 双引号设定字符串
3 print(x)
4 print(type(x)) # 列出x字符串数据类型
5 y = '深石数字 - 深度学习滴水穿石' # 单引号设定字符串
6 print(y)
7 print(type(y)) # 列出y字符串数据类型
```

执行结果

```
===== RESTART: D:\Python\ch3\ch3_13.py =====
DeepStone means Deep Learning
<class 'str'>
深石数字 - 深度学习滴水穿石
<class 'str'>
>>>
```

3-4-1 字符串的连接

数学的运算符“+”，可以执行两个字符串相加，产生新的字符串。

程序实例ch3_14.py: 字符串连接的应用。

```
1 # ch3_14.py
2 num1 = 222
3 num2 = 333
4 num3 = num1 + num2
5 print("以下是数值相加")
6 print(num3)
7 numstr1 = "222"
8 numstr2 = "333"
9 numstr3 = numstr1 + numstr2
10 print("以下是由数值组成的字符串相加")
11 print(numstr3)
12 numstr4 = numstr1 + " " + numstr2
13 print("以下是由数值组成的字符串相加，同时中间加上一空格")
14 print(numstr4)
15 str1 = "DeepStone "
16 str2 = "Deep Learning"
17 str3 = str1 + str2
18 print("以下是一般字符串相加")
19 print(str3)
```

执行结果

```
===== RESTART: D:\Python\ch3\ch3_14.py =====
以下是数值相加
555
以下是由数值组成的字符串相加
222333
以下是由数值组成的字符串相加，同时中间加上一空格
222 333
以下是一般字符串相加
DeepStone Deep Learning
>>>
```

3-4-2 处理多于一行的字符串

程序设计时如果字符串长度多于一行，可以使用三个单引号（或是三个双引号）将字符串包夹。

程序实例ch3_15.py: 使用三个单引号处理多于一行的字符串。

```
1 # ch3_15.py
2 str1 = '''Silicon Stone Education is an unbiased organization
3 concentrated on bridging the gap ... '''
4 print(str1)
```

执行结果

```
===== RESTART: D:\Python\ch3\ch3_15.py =====
Silicon Stone Education is an unbiased organization
concentrated on bridging the gap ...
>>>
```

读者可以留意第2行Silicon左边的3个单引号和第3行末端的3个单引号。

3-4-3 逸出字符

在字符串使用中，如果字符串内有一些特殊字符，如单引号、双引号等，必须在此特殊字符前加上“\”（反斜杠），才可正常使用，这种含有“\”符号的字符称**逸出字符**（Escape Character）。下表Hex值是指ASCII值。

逸出字符	Hex 值	意义	逸出字符	Hex 值	意义
\'	27	单引号	\n	0A	换行
\"	22	双引号	\o		八进位表示
\\	5C	反斜杠	\r	0D	光标移至最左位置
\a	07	响铃	\x		十六进位表示
\b	08	BackSpace 键	\t	09	Tab 键效果
\f	0C	换页	\v	0B	垂直定位

字符串使用中特别是碰到字符串含有单引号时，如果是使用单引号定义这个字符串时，必须要使用此**逸出字符**，才可以顺利显示，可参考ch3_16.py的第3行。如果是使用双引号定义字符串则可以不使用**逸出字符**，可参考ch3_16.py的第6行。

程序实例ch3_16.py： 逸出字符的应用，这个程序第9增加“\t”字符，所以“can't”跳到下一个Tab键位置输出。同时有“\n”字符，所以“loving”跳到下一行输出。

```
1 # ch3_16.py
2 #以下输出使用单引号设定的字符串，需使用\'
3 str1 = 'I can\'t stop loving you.'
4 print(str1)
5 #以下输出使用双引号设定的字符串，不需使用\'
6 str2 = "I can't stop loving you."
7 print(str2)
8 #以下输出有\t和\n字符
9 str3 = "I \tcan't stop \nloving you."
10 print(str3)
```

执行结果

```

>>>
===== RESTART: D:/Python/ch3/ch3_16.py =====
I can't stop loving you.
I can't stop loving you.
I      can't stop
loving you.
>>>

```

3-4-4 强制转换为字符串

`str()` 函数可以强制将数值数据转换为字符串数据。

程序实例ch3_17.py: 使用`str()`函数将数值数据强制转换为字符串的应用。

```

1 # ch3_17.py
2 num1 = 222
3 num2 = 333
4 num3 = num1 + num2
5 print("这是数值相加")
6 print(num3)
7 str1 = str(num1) + str(num2)
8 print("强制转换为字符串相加")
9 print(str1)

```

执行结果

```

===== RESTART: D:\Python\ch3\ch3_17.py =====
这是数值相加
555
强制转换为字符串相加
222333
>>>

```

上述字符串相加，读者可以想成是字符串连接执行结果是一个字符串，所以上述执行结果555是数值数据，222333则是一个字符串。

3-4-5 将字符串转换为整数

在未来的程序设计中也会常会发生将字符串转换为整数数据，下列将直接以实例做说明。

程序实例ch3_18.py: 将字符串数据转换为整数数据的应用。

```

1 # ch3_18.py
2 x1 = "22"
3 x2 = "33"
4 x3 = x1 + x2
5 print(x3)           # 打印字符串相加
6 x4 = int(x1) + int(x2)
7 print(x4)           # 打印整数相加

```

执行结果

```

===== RESTART: D:\Python\ch3\ch3_18.py =====
2233
55
>>>

```

上述执行结果55是数值数据，2233则是一个字符串。

3-4-6 字符串数据的转换

如果字符串含一个字符或一个文字时，我们可以使用下列执行数据的转换。

- `chr( x )`：可以返回函数x值的字符，x是ASCII码值。
- `ord( x )`：可以返回函数字符参数的Unicode码值，如果是中文也可传回Unicode码值。如果是英文字符，Unicode码值与ASCII码值是一样的。

程序实例 ch3_19.py： 这个程序首先会将整数97转换成英文字符‘a’，然后将字符‘a’转换成Unicode码值，最后将中文字‘魁’转成Unicode码值。

```

1 # ch3_19.py
2 x1 = 97
3 x2 = chr(x1)
4 print(x2)           # 输出数值97的字符
5 x3 = ord(x2)
6 print(x3)           # 输出字符x3的Unicode码值
7 x4 = '魁'
8 print(ord(x4))       # 输出字符'魁'的Unicode码值

```

执行结果

```

===== RESTART: D:\Python\ch3\ch3_19.py =====
a
97
39745
>>>

```


3-4-7 字符串与整数相乘产生字符串复制效果

在Python可以允许将字符串与整数相乘，结果是字符串将重复该整数的次数。

程序实例ch3_20.py： 字符串与整数相乘的应用。

```
1 # ch3_20.py
2 x1 = "A"
3 x2 = x1 * 10
4 print(x2)          # 打印字符串乘以整数
5 x3 = "ABC"
6 x4 = x3 * 5
7 print(x4)          # 打印字符串乘以整数
```

执行结果

```
===== RESTART: D:/Python/ch3/ch3_20.py =====
AAAAAAAAAA
ABCABCABCABC
>>>
```

3-4-8 聪明地使用字符串加法和换行字符\n

有时设计程序时，想将字符串分行输出，可以使用字符串加法功能，在加法过程中加上换行字符“\n”即可产生字符串分行输出的结果。

程序实例ch3_21.py： 将数据分行输出的应用。

```
1 # ch3_21.py
2 str1 = "洪锦魁著作"
3 str2 = "HTML5+CSS3王者归来"
4 str3 = "Python程序语言王者归来"
5 str4 = str1 + "\n" + str2 + "\n" + str3
6 print(str4)
```

执行结果

```
===== RESTART: D:/Python/ch3/ch3_21.py =====
洪锦魁著作
HTML5+CSS3王者归来
Python程序语言王者归来
>>>
```

3-4-9 字符串前加r

在使用Python时，如果在字符串前加上r，可以防止逸出字符(Escape Character)被转译，可参考3-4-3节的逸出字符表，相当于可以取消逸出字符的功能。

程序实例ch3_22.py： 字符串前加上r的应用。

```
1 # ch3_22.py
2 str1 = "Hello!\nPython"
3 print("不含r字符的输出")
4 print(str1)
5 str2 = r"Hello!\nPython"
6 print("含r字符的输出")
7 print(str2)
```

执行结果



```
RESTART: D:\Python\ch3\ch3_22.py
不含r字符的输出
Hello!
Python
含r字符的输出
Hello!
Python
>>>
```

这个功能在本书第16章正规表达式将会有许多应用。

习题

本书所有程序实作题，叙述不完整部分是由读者自行发挥创意，例如，输入或输出格式、测试数据、验证程序正确的数据笔数等。

1. 请列出下列数值的2进位、8进位、16进位的值。

(a) 100 (b) 55 (c) 299 (d) 399 (e) 86

2. 请将下列数值转成10进位。

(a) 0b11110010 (b) 0o76543 (c) 0xaaabbb

3. 假设a是10，b是18，c是5，请计算下列执行结果，取整数结果。

(a) $s = a + b - c$ (b) $s = 2 * a + 3 - c$ (c) $s = b * c + 20 / b$

(d) $s = a \% c * b + 10$ (e) $s = a ** c - a * b * c$

第4章 基本输入与输出

本章基本上将介绍如何在屏幕上做输入与输出，另外也将讲解使用Python内置的实用功能。

4-1 Python的辅助说明help()

help()函数可以列出某一个Python的指令或函数的使用说明。

实例1： 列出输出函数print()的使用说明。

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file: a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.

>>>
```

当然程序语言是全球化的语言，所有说明是以英文为基础，要有一定的英文能力才可彻底了解，不过，笔者在本书会详尽说明。

4-2 格式化输出数据使用print()

相信读者经过前三章的学习，已经对使用print()函数输出数据非常熟悉了，该是时候完整解说这个输出函数的用法了。

4-2-1 函数print()的基本语法

它的基本语法格式如下：

```
print(value, ... , sep=" ", end="\n",file=sys.stdout,
flush=False)
```

- value表示想要输出的数据，可以一次输出多个数据，各数据间以逗号隔开。
- sep当输出多个数据时，可以插入各个数据的分隔字符，默认是一个空格字符。
- end当数据输出结束时所插入的字符，默认是插入换行字符，所以下一次print()函数的输出会在下一行输出。
- file数据输出位置，默认是sys.stdout，也就是屏幕。
- flush是否清除数据流的缓冲区，预设是不清除。

程序实例ch4_1.py： 重新设计ch3_17.py，其中在第二个print()，2个输出数据的分隔字符是“\$\$\$ ”。

```
1 # ch4_1.py
2 num1 = 222
3 num2 = 333
4 num3 = num1 + num2
5 print("这是数值相加", num3)
6 str1 = str(num1) + str(num2)
7 print("强制转换为字符串相加", str1, sep=" $$$ ")
```

执行结果

```
===== RESTART: D:\Python\ch4\ch4_1.py =====
这是数值相加 $$$
强制转换为字符串相加 $$$ 222333
>>>
```

程序实例ch4_2.py： 重新设计ch4_1.py，将2个数据在同一行输出，彼此之间使用Tab键隔开。

```
1 # ch4_2.py
2 num1 = 222
3 num2 = 333
4 num3 = num1 + num2
5 print("这是数值相加", num3, end="\t") # 以Tab键值位置分隔2个数据输出
6 str1 = str(num1) + str(num2)
7 print("强制转换为字符串相加", str1, sep=" $$$ ")
```

执行结果

```
===== RESTART: D:\Python\ch4\ch4_2.py =====
这是数值相加 $$$      强制转换为字符串相加 $$$ 222333
>>>
```

4-2-2 格式化print()输出

在使用格式化输出时，基本使用格式如下：

```
print(" ...输出格式区... " % ( 变量系列区, ... ))
```

在上述输出格式区 中，可以放置变量系列区 对应的格式化字符，基本意义如下：

- %d ： 格式化整数输出。
- %f ： 格式化浮点数输出。
- %x ： 格式化16进位整数输出。
- %o ： 格式化8进位整数输出。
- %s ： 格式化字符串输出。

程序实例ch4_3.py： 格式化输出的应用。

```
1 # ch4_3.py
2 score = 90
3 str1 = "洪锦魁"
4 count = 1
5 print("%s你的第 %d 次物理考试成绩是 %d" % (str1, count, score))
```

执行结果

```
===== RESTART: D:\Python\ch4\ch4_3.py =====
洪锦魁你的第 1 次物理考试成绩是 90
>>>
```

设计程序时，print()函数内的输出格式区 也可以用一串字符串变量取代。

程序实例ch4_4.py： 重新设计ch4_3.py，在print()内用字符串变量取代字符表列，读者可以参考第5和6行与原先ch4_3.py的第5列作比较。

```
1 # ch4_4.py
2 score = 90
3 str1 = "洪锦魁"
4 count = 1
5 formatstr = "%s你的第 %d 次物理考试成绩是 %d"
6 print(formatstr % (str1, count, score))
```

执行结果 与ch4_3.py相同。

程序实例ch4_5.py: 格式化8进位和16进位输出的应用。

```
1 # ch4_5.py
2 x = 100
3 print("100的16进位 = %x\n100的 8进位 = %o" % (x, x))
```

执行结果

```
===== RESTART: D:\Python\ch4\ch4_5.py =====
100的16进位 = 64
100的 8进位 = 144
>>>
```

程序实例ch4_6.py: 将整数与浮点数分别以%d、%f、%s格式化，同时观察执行结果。特别要注意的是，浮点数以整数%d格式化后，小数数据将被舍去。

```
1 # ch4_6.py
2 x = 10
3 print("整数%d \n浮点数%f \n字符串%s" % (x, x, x))
4 y = 9.9
5 print("整数%d \n浮点数%f \n字符串%s" % (y, y, y))
```

执行结果

```
===== RESTART: D:\Python\ch4\ch4_6.py =====
整数10
浮点数10.000000
字符串10
整数9
浮点数9.900000
字符串9.9
>>>
```

4-2-3 精准控制格式化的输出

在上述程序实例ch4_6.py中，我们发现最大的缺点是无法精确控制浮点数的输出位置，print()函数在格式化过程中，有提供功能可以让我们设定保留多少格的空间让资料做输出，语法如下：

- %(+|-)nd : 格式化整数输出。
- %(+|-)m.nf : 格式化浮点数输出。
- %(+|-)nx : 格式化16进位整数输出。

- `%(+|-)no` : 格式化8进位整数输出。
- `%(-)ns` : 格式化字符串输出。

上述对浮点数而言，`m`代表保留多少格数供输出（包含小数点），`n`则是小数数据保留格数。至于其他的数据格式`n`则是保留多少格数空间，如果保留格数空间不足将完整输出数据，如果保留格数空间太多则数据靠右对齐。

如果格式化数值数据有加上负号(-)，表示保留格数空间有多时，数据将靠左输出。如果格式化数值数据有加上正号(+)，表示输出数据是正值时，将在左边加上正值符号。

程序实例ch4_7.py: 格式化输出的应用。

```
1 # ch4_7.py
2 x = 100
3 print("x=%6d/" % x)
4 y = 10.5
5 print("y=%6.2f/" % y)
6 s = "Deep"
7 print("s=%6s/" % s)
8 print("以下是保留格数空间不足的实例")
9 print("x=%3d/" % x)
10 print("y=%3.2f/" % y)
11 print("s=%3s/" % s)
```

执行结果

```
===== RESTART: D:\Python\ch4\ch4_7.py =====
x=/ 100/
y=/ 10.50/
s=/ Deep/
以下是保留格数空间不足的实例
x=/100/
y=/10.50/
s=/Deep/
>>>
```

程序实例ch4_8.py: 格式化输出，靠左对齐的实例。

```
1 # ch4_8.py
2 x = 100
3 print("x=%-6d/" % x)
4 y = 10.5
5 print("y=%-6.2f/" % y)
6 s = "Deep"
7 print("s=%-6s/" % s)
```

执行结果

```
===== RESTART: D:/Python/ch4/ch4_8.py =====
x=100 /
y=10.50 /
s=Deep /
>>>
```

程序实例ch4_9.py: 格式化输出, 正值数据将出现正号(+).

```
1 # ch4_9.py
2 x = 10
3 print("x=%+6d/" % x)
4 y = 10.5
5 print("y=%+6.2f/" % y)
```

执行结果

```
===== RESTART: D:/Python/ch4/ch4_9.py =====
x=/ +10/
y=/+10.50/
>>>
```

程序实例ch4_10.py: 格式化输出的应用。

```
1 # ch4_10.py
2 print(" 姓名      国文      英文      总分")
3 print("%3s %4d %4d %4d" % ("洪冰儒", 98, 90, 188))
4 print("%3s %4d %4d %4d" % ("洪雨星", 96, 95, 191))
5 print("%3s %4d %4d %4d" % ("洪冰雨", 92, 88, 180))
6 print("%3s %4d %4d %4d" % ("洪星宇", 93, 97, 190))
```

执行结果

```
===== RESTART: D:\Python\ch4\ch4_10.py =====
 姓名      国文      英文      总分
洪冰儒      98      90      188
洪雨星      96      95      191
洪冰雨      92      88      180
洪星宇      93      97      190
>>>
```

4-2-4 format()函数

这是Python增强版的格式化输出功能, 字符串使用format方法做格式化的动作, 它的基本使用格式如下:

```
print(" ...输出格式区... " .format( 变量系列区, ... ))
```

在输出格式区内的字符串变量使用“{ }”表示。

程序实例ch4_11.py: 使用format()函数重新设计ch4_3.py。

```
1 # ch4_11.py
2 score = 90
3 str1 = "洪锦魁"
4 count = 1
5 print("{}你的第 {} 次物理考试成绩是 {}".format(str1, count, score))
```


执行结果 与ch4_3.py相同。

程序实例 ch4_12.py： 以字符串代表输出格式区，重新设计 ch4_11.py。

```
1 # ch4_12.py
2 score = 90
3 str1 = "洪锦魁"
4 count = 1
5 str2 = "{}你的第 {} 次物理考试成绩是 {}"
6 print(str2.format(str1, count, score))
```

执行结果 与ch4_3.py相同。

4-2-5 字符串输出与基本排版的应用

其实适度利用输出格式，也可以产生一封排版的信件，以下程序的前3行会先利用sp字符串变量建立一个含40格的空白格数，然后产生对齐效果。

程序实例ch4_13.py： 有趣排版信件的应用。

```
1 # ch4_13.py
2 sp = " " * 40
3 print("%s 1231 Delta Rd" % sp)
4 print("%s Oxford, Mississippi" % sp)
5 print("%s USA\n\n" % sp)
6 print("Dear Ivan")
7 print("I am pleased to inform you that your application for fall 2020 has")
8 print("been favorably reviewed by the Electrical and Computer Engineering")
9 print("Office.\n\n")
10 print("Best Regards")
11 print("Peter Malong")
```

执行结果

```
===== RESTART: D:/Python/ch4/ch4_13.py =====
                                1231 Delta Rd
                                Oxford, Mississippi
                                USA

Dear Ivan
I am pleased to inform you that your application for fall 2020 has
been favorably reviewed by the Electrical and Computer Engineering
Office.

Best Regards
Peter Malong
>>>
```

4-2-6 一个无聊的操作

程序实例ch4_13.py第2行，利用空格乘以40产生40个空格，功能
是用于排版。如果将某个字符串乘以500，然后用print()输出，可以
在屏幕上建立一个无聊的画面。

实例1： 在屏幕上建立一个无聊的画面。

[illegible]

上述实例是教导读者，活用Python，可以产生许多意外的结果。

4-3 输出数据到文件

在4-2-1节笔者有讲解在`print()`函数中，默认输出是屏幕`sys.stdout`，其实我们可以利用这个特性将输出导向一个文件。

4-3-1 打开一个文件open()

open()函数可以打开一个文件供读取或写入，如果这个函数执行成功，会传回文件对象，这个函数的基本使用格式如下：

```
file_obj = open(file, mode="r") # 左边只列出最常用的2个参数
```



file用字符串列出欲打开的文件。



mode打开文件的模式，如果省略代表是mode=“r”，使用时如果mode=“w”或其他，也可以省略mode=，直接写“w”。也可以同时具有多项模式，例如，“wb”代表以二进制文件打开供写入，可以是下列基本模式。

- “r”：这是预设，打开文件供读取(read)。
- “w”：打开文件供写入，如果原先文件有内容将被覆盖。
- “a”：打开文件供写入，如果原先文件有内容，新写入数据将附加在后面。
- “x”：打开一个新的文件供写入，如果所打开的文件已经存在会产生错误。
- “b”：打开二进制文件模式。
- “t”：打开本文(txt)文件模式，这是默认。
- “+”：打开文件供更新用。



file_obj这是文件对象，读者可以自行给予名称，未来print()函数可以将输出导向此对象，不使用时要关闭“file_obj.close()”，才可以返回操作系统的文件管理器观察执行结果。

4-3-2 使用print()函数输出数据到文件

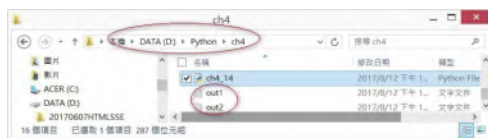
程序实例ch4_14.py： 将数据输出到文件的实例，其中输出到out1.txt采用“w”模式，输出到out2.txt采用“a”模式。

```
1 # ch4_14.py
2 fstream1 = open("d:\python\ch4\out1.txt", mode="w") # 取代先前资料
3 print("Testing for output", file=fstream1)
4 fstream1.close()
5 fstream2 = open("d:\python\ch4\out2.txt", mode="a") # 附加数据后面
6 print("Testing for output", file=fstream2)
7 fstream2.close()
```

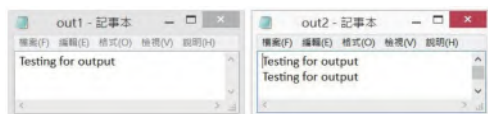
执行结果

```
===== RESTART: D:/Python/ch4/ch4_14.py =====  
>>>
```

如果执行程序一次，可以得到内容相同的out1.txt和out2.txt。但是如果持续执行，out2.txt内容会持续增加，out1.txt内容则保持不变，下列是检查文件夹内容。



下列是执行2次此程序，out1.txt和out2.txt的内容。



4-4 数据输入input()

这个input()函数功能与print()函数功能相反，这个函数会从屏幕读取用户从键盘输入的数据，它的使用格式如下：

```
value = input("prompt: ")
```

value是变量，所输入的数据会存储在此变量内，特别需注意的是所输入的数据不论是字符串或是数值数据返回到value时一律是字符串数据，如果要执行数学运算需要用int()函数转换为整数。

程序实例ch4_15.py： 认识输入数据类型。

```
1 # ch4_15.py  
2 name = input("请输入姓名：")  
3 engh = input("请输入成绩：")  
4 print("name数据类型是", type(name))  
5 print("engh数据类型是", type(engh))
```

执行结果

```
===== RESTART: D:\Python\ch4\ch4_15.py =====
请输入姓名：洪锦魁
请输入成绩：100
name数据类型是 <class 'str'>
engh数据类型是 <class 'str'>
>>>
```

程序实例ch4_16.py： 基本数据输入与运算。

```
1 # ch4_16.py
2 print("欢迎使用成绩输入系统")
3 name = input("请输入姓名：")
4 eng = input("请输入英文成绩：")
5 math = input("请输入数学成绩：")
6 total = int(eng) + int(math)
7 print("%s 你的总分是 %d" % (name, total))
```

执行结果

```
===== RESTART: D:\Python\ch4\ch4_16.py =====
欢迎使用成绩输入系统
请输入姓名：洪锦魁
请输入英文成绩：98
请输入数学成绩：99
洪锦魁 你的总分是 197
>>>
```

接下来的程序主要是处理中文名字与英文名字的技巧，假设要求使用者分别输入姓氏 (lastname) 与名字 (firstname)，在中文要处理成命名，可以使用下列字符串连接方式。

```
fullname = lastname +firstname
```

在英文首先名字在前面，姓氏在后面，同时中间有一个空格，因此处理方式如下：

```
fullname =firstname + " " + lastname
```

程序实例ch4_17.py： 请分别输入中文和英文的姓氏以及名字，本程序将会组合名字并输出问候语。

```
1 # ch4_17.py
2 clastname = input("请输入中文姓氏：")
3 cfirstname = input("请输入中文名字：")
4 cfullname = clastname + cfirstname
5 print("%s 欢迎使用本系统" % cfullname)
6 lastname = input("请输入英文Last Name：")
7 firstname = input("请输入英文First Name：")
8 fullname = firstname + " " + lastname
9 print("%s Welcome to SSE System" % fullname)
```

执行结果

```
RESTART: D:\Python\ch4\ch4_17.py
请输入中文姓名：洪
请输入中文姓名：编辑
洪编辑 欢迎使用本系统
请输入英文Last Name：Hung
请输入英文First Name：JKwei
JKwei Hung Welcome to SSE System
>>>
```

4-5 列出所有内置函数dir()

阅读至此，相信读者已经使用了许多Python内置的函数了，如help()、print()、input()等，读者可能想了解到底Python有提供哪些内置函数可供我们在设计程序时使用，可以使用下列方式列出Python所提供的内置函数。

```
dir(__builtins__) # 列出Python内置函数
```

实例1： 列出Python所有内置函数。

```
>>> dir(__builtins__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
>>>
```

在本书，笔者会依功能分类将常用的内置函数分别融入各章节主题中，如果读者想先了解某一个内置函数的功能，可参考4-1节使用help()函数。

习题

1. 请参考ch4_10.py，使用班上第一次月考成绩，将结果输出至文件。

2. 请参考ch4_13.py, 写一封信给老师, 叙述学习Python的心得。
3. 写一个程序要求用户输入3位数数字, 最后舍去个位数字输出, 例如输入是103输出是100, 输入是776输出是770。
4. 请输入华氏温度, 将结果转成摄氏温度输出。
5. 请输入摄氏温度, 将结果转成华氏温度输出。

第5章 程序的流程控制使用if语句

一个程序如果是按部就班从头到尾，中间没有转折，其实是无法完成太多工作。设计过程难免会需要转折，这个转折在程序设计的术语称**流程控制**，本章将完整讲解有关if语句的流程控制。另外，与程序流程设计有关的关系运算符与逻辑运算符也将在本章做说明，因为这些是if语句流程控制的基础。

5-1 关系运算符

Python语言所使用的关系运算符表：

关系运算符	说明	实例	说明
>	大于	<code>a > b</code>	检查 a 是否大于 b
>=	大于或等于	<code>a >= b</code>	检查 a 是否 大于或等于 b
<	小于	<code>a < b</code>	检查 a 是否小于 b
<=	小于或等于	<code>a <= b</code>	检查 a 是否小于或等于 b
==	等于	<code>a == b</code>	检查 a 是否等于 b
!=	不等于	<code>a != b</code>	检查 a 是否不等于 b

上述运算如果是真 会传回True，如果是伪 会传回False。

实例1： 下列会传回True。


```

>>> x = 10 > 8
>>> print(x)
True
>>> x = 10 >= 10
>>> print(x)
True
>>> x = 10 < 20
>>> print(x)
True
>>> x = 10 <= 10
>>> print(x)
True
>>> x = 10 == 10
>>> print(x)
True
>>> x = 10 != 20
>>> print(x)
True
>>>

```

实例2： 下列会传回False。

```

>>> x = 10 > 20
>>> print(x)
False
>>> x = 10 >= 20
>>> print(x)
False
>>> x = 10 < 5
>>> print(x)
False
>>> x = 10 <= 5
>>> print(x)
False
>>> x = 10 == 5
>>> print(x)
False
>>> x = 10 != 10
>>> print(x)
False
>>>

```

5-2 逻辑运算符

Python所使用的逻辑运算符：

- and ---相当于逻辑符号AND
- or ---相当于逻辑符号OR
- not ---相当于逻辑符号NOT

下列是逻辑运算符and 的图例说明。

and	True	False
True	True	False
False	False	False

下列是逻辑运算符or 的图例说明。

or	True	False
True	True	True
False	True	False

实例1： 下列会传回True。

```
>>> x = (10 > 8) and (20 > 10)
>>> print(x)
True
>>>
```

实例2： 下列会传回False。

```
>>> x = (10 > 8) and (10 > 20)
>>> print(x)
False
>>> x = (10 < 8) and (10 < 20)
>>> print(x)
False
>>> x = (10 < 8) and (10 > 20)
>>> print(x)
False
>>>
```

实例3： 下列会传回True。

```
>>> x = (10 > 8) or (20 > 10)
>>> print(x)
True
>>> x = (10 < 8) or (10 < 20)
>>> print(x)
True
>>> x = (10 > 8) or (10 > 20)
>>> print(x)
True
>>> .
```

实例4： 下列会传回False。

```
>>> x = (10 < 8) or (10 > 20)
>>> print(x)
False
>>> .
```

下列是逻辑运算符not 的图例说明。

not	True	False
	False	True

逻辑运算结果如果是真则传回True，如果是伪则传回False。

实例5： 下列会传回True。

```
>>> x = not(10 < 8)
>>> print(x)
True
>>>
```

实例6： 下列会传回False。

```
>>> x = not(10 > 8)
>>> print(x)
False
>>>
```

5-3 if语句

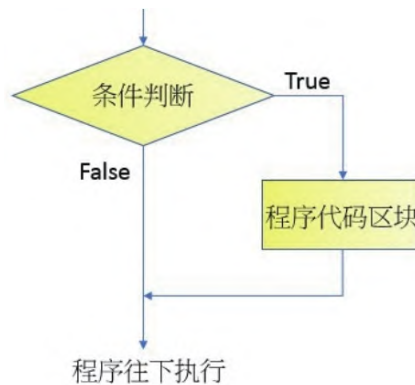
这个if语句的基本语法如下：

```
if (条件判断
):
    程序代码区块
```

上述观念是如果条件判断 是True ，则执行程序代码区块 ，如果条件判断 是False ，则不执行程序代码区块 。如果程序代码区块只有一道指令，可将上述语法写成下列格式。

```
if (条件判断
): 程序代码区块
```

可以用下列流程图说明这个if语句：



如果读者有学习过其他程序语言，例如，Visual Basic、C、JavaScript等，在条件表达式中是使用大括号“{ }”，将if语句的程序代码区块包夹做区隔。如下所示（以C语言为实例）：

```

if (age < 20) {
    printf("你年龄太小");
    printf("需年满20岁才可购买烟酒");
}
  
```

在Python内是使用**内缩方式** 区隔if语句的程序代码区块，编辑程序时可以用**Tab键** 内缩或是直接内缩4个字符空间，表示这是if语句的程序代码区块。相同内容，可以用下列方式处理。

```

If (age < 20): # 程序代码区块1
    print("你年龄太小") # 程序代码区块2
    print("需年满20岁才可购买烟酒") # 程序代码区块2
  
```

在Python中内缩程序代码是有意义的，相同的程序代码区块，必须有相同的内缩，否则会产生错误。

实例1： 正确的if语句程序代码。

```

>>> age = 18
>>> if (age < 20):
>>>     print("你年龄太小")
>>>     print("需年满20岁才可购买烟酒")
      
```

插入点在此时请按Enter键

```

>>> age = 18
>>> if (age < 20):
>>>     print("你年龄太小")
>>>     print("需年满20岁才可购买烟酒")
      
```

你年龄太小
需年满20岁才可购买烟酒

实例2： 不正确的if语句程序代码，下列因为任意内缩造成错误。

任意内缩造成错误

```
>>> age = 18
>>> if (age < 20):
>>>     print("你年龄太小")
>>>     print("需年满20岁才可以购买烟酒")
SyntaxError: unexpected indent
>>>
```

上述笔者讲解if语句是True时需内缩4个字符空间，读者可能会问可不可以内缩5个字符空间，答案是可以的，但是记得相同程序区块必须有相同的内缩空间。不过如果你是使用Python的IDLE编辑环境，当输入if语句后，只要单击Enter键，编辑程序会自动内缩4个字符空间。

程序实例ch5_1.py: if语句的基本应用。

```
1 # ch5_1.py
2 age = input("请输入年龄: ")
3 if (int(age) < 20):
4     print("你年龄太小")
5     print("需年满20岁才可以购买烟酒")
```

执行结果

```
===== RESTART: D:\Python\ch5\ch5_1.py =====
请输入年龄: 18
你年龄太小
需年满20岁才可以购买烟酒
>>>
===== RESTART: D:\Python\ch5\ch5_1.py =====
请输入年龄: 30
>>>
```

程序实例ch5_2.py: 输出绝对值的应用。

```
1 # ch5_2.py
2 print("输出绝对值")
3 num = input("请输入任意整值: ")
4 x = int(num)
5 if (int(x) < 0):
6     x = abs(x)
7 print("绝对值是 %d" % int(x))
```

执行结果

```
===== RESTART: D:\Python\ch5\ch5_2.py =====
输出绝对值
请输入任意整值: 98
绝对值是 98
>>>
===== RESTART: D:\Python\ch5\ch5_2.py =====
输出绝对值
请输入任意整值: -30
绝对值是 30
>>>
```

对于ch5_2.py而言，由于if语句只有一道指令，所以可以将第5行和第6行改写成下列语句。

```
5 if (int(x) < 0): x = abs(x)
```

上述可以得到相同的结果，详情请参考ch5_2_1.py。

5-4 if ... else语句

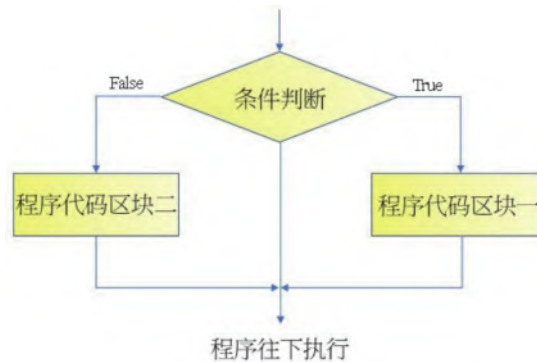
程序设计时更常用的功能是条件判断为True时执行某一个程序代码区块，当条件判断为False时执行另一段程序代码区块，此时可以使用if ... else语句，它的语法格式如下：

```
if (条件判断
):
    # 小括号可以省略

    程序代码区块一

else:
    程序代码区块二
```

上述观念是如果条件判断 是True ，则执行程序代码区块一 ，如果条件判断 是False ，则执行程序代码区块二 。可以用下列流程图说明这个if ... else语句：



程序实例ch5_3.py： 重新设计ch5_1.py，多了年龄满20岁时的输出。

```
1 # ch5_3.py
2 age = input("请输入年龄：")
3 if (int(age) < 20):
4     print("你年龄太小")
5     print("需年满20岁才可以购买烟酒")
6 else:
7     print("欢迎购买烟酒")
```

执行结果

```
===== RESTART: D:\Python\ch5\ch5_3.py =====
请输入年龄: 18
你年龄太小
需年满20岁才可以购买烟酒
>>>
===== RESTART: D:\Python\ch5\ch5_3.py =====
请输入年龄: 30
欢迎购买烟酒
>>>
```

程序实例ch5_4.py： 奇数偶数的判断。

```
1 # ch5_4.py
2 print("奇数偶数判断")
3 num = input("请输入任意整值：")
4 rem = int(num) % 2
5 if (rem == 0):
6     print("%d 是偶数" % int(num))
7 else:
8     print("%d 是奇数" % int(num))
```

执行结果

```
===== RESTART: D:\Python\ch5\ch5_4.py =====
奇数偶数判断
请输入任意整值: 5
5 是奇数
>>>
===== RESTART: D:\Python\ch5\ch5_4.py =====
奇数偶数判断
请输入任意整值: 10
10 是偶数
>>>
```

5-5 if ... elif ...else语句

这是一个多重判断，程序设计时需要多个条件作比较时就比较有用，例如：在美国成绩计分是采取A、B、C、D、F等，通常90-100分是A，80-89分是B，70-79分是C，60-69分是D，低于60分是F。若是使用Python可以用这个语句，很容易就完成这个工作。这个语句的基本语法如下：

```

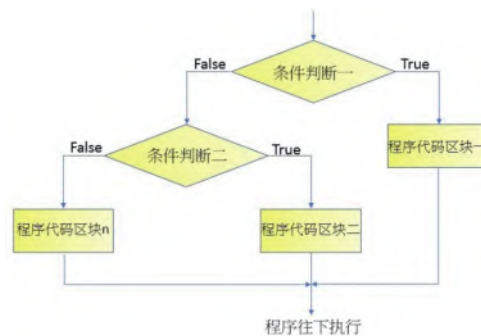
if (条件判断一
):
    程序代码区块一

elif (条件判断二
):
    程序代码区块二

...
else:
    程序代码区块n

```

上述观念是，如果条件判断一 是True则执行程序代码区块一 ，然后离开条件判断。否则检查条件判断二 ，如果是True则执行程序代码区块二 ，然后离开条件判断。如果条件判断 是False则持续进行检查，上述elif的条件判断可以不断扩充，如果所有条件判断是False则执行程序代码n区块 。下列流程图是假设只有2个条件判断说明这个if ... elif ... else语句。



程序实例ch5_5.py: 请输入数字分数，系统将响应A、B、C、D或F等级。

```

1 # ch5_5.py
2 print("计算最终成绩")
3 score = input("请输入分数: ")
4 sc = int(score)
5 if (sc >= 90):
6     print(" A")
7 elif (sc >= 80):
8     print(" B")
9 elif (sc >= 70):
10    print(" C")
11 elif (sc >= 60):
12    print(" D")
13 else:
14    print(" F")

```

执行结果

```

===== RESTART: D:\Python\ch5\ch5_5.py =====
计算最终成绩
请输入分数: 90
A
>>>
===== RESTART: D:\Python\ch5\ch5_5.py =====
计算最终成绩
请输入分数: 83
B
>>>
===== RESTART: D:\Python\ch5\ch5_5.py =====
计算最终成绩
请输入分数: 79
C
>>>
===== RESTART: D:\Python\ch5\ch5_5.py =====
计算最终成绩
请输入分数: 66
D
>>>
===== RESTART: D:\Python\ch5\ch5_5.py =====
计算最终成绩
请输入分数: 55
F
>>>

```

程序实例ch5_6.py: 有一地区的票价收费标准是100元。

- 但是如果小于等于6岁或大于等于80岁，收费是打2折。
- 但是如果是7-12岁或60-79岁，收费是打5折。

请输入岁数，程序会计算票价。

```

1 # ch5_6.py
2 print("计算票价")
3 age = input("请输入年龄: ")
4 age = int(age)
5 ticket = 100
6 if age >= 80 or age <= 6:
7     ticket = ticket * 0.2
8     print("票价是: %d" % ticket)
9 elif age >= 60 or age <= 12:
10    ticket = ticket * 0.5
11    print("票价是: %d" % ticket)
12 else:
13    print("票价是: %d" % ticket)

```

执行结果


```

===== RESTART: D:\Python\ch5\ch5_6.py =====
计算票价
请输入年龄: 81
票价是: 20
>>>
===== RESTART: D:\Python\ch5\ch5_6.py =====
计算票价
请输入年龄: 6
票价是: 20
>>>
===== RESTART: D:\Python\ch5\ch5_6.py =====
计算票价
请输入年龄: 77
票价是: 50
>>>
===== RESTART: D:\Python\ch5\ch5_6.py =====
计算票价
请输入年龄: 12
票价是: 50
>>>
===== RESTART: D:\Python\ch5\ch5_6.py =====
计算票价
请输入年龄: 13
票价是: 100
>>>

```

上述程序的第6行和第9行，如果你对于运算符执行的优先级没有太大的把握，建议可以直接用小括号将条件判断括起来，可参考ch5_6_1.py。

```

6  if (age >= 80) or (age <= 6):
7      ticket = ticket * 0.2
8      print("票价是: %d" % ticket)
9  elif (age >= 60) or (age <= 12):

```

程序实例ch5_7.py： 这个程序会要求输入字符，然后会告知所输入的字符是大写字母、小写字母、阿拉伯数字或特殊字符。

```

1  # ch5_7.py
2  print("判断输入字符类别")
3  ch = input("请输入字符: ")
4  if ord(ch) >= ord("A") and ord(ch) <= ord("Z"):
5      print("这是大写字母")
6  elif ord(ch) >= ord("a") and ord(ch) <= ord("z"):
7      print("这是小写字母")
8  elif ord(ch) >= ord("0") and ord(ch) <= ord("9"):
9      print("这是数字")
10 else:
11     print("这是特殊字符")

```

执行结果

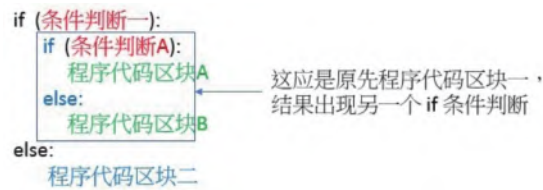
```

===== RESTART: D:\Python\ch5\ch5_7.py =====
判断输入字符类别
请输入字符: K
这是大写字母
>>>
===== RESTART: D:\Python\ch5\ch5_7.py =====
判断输入字符类别
请输入字符: m
这是小写字母
>>>
===== RESTART: D:\Python\ch5\ch5_7.py =====
判断输入字符类别
请输入字符: 9
这是数字
>>>
===== RESTART: D:\Python\ch5\ch5_7.py =====
判断输入字符类别
请输入字符: !
这是特殊字符
>>>

```

5-6 嵌套的if语句

所谓的嵌套的if语句是指在if语句内有其他的if语句，下列是一种情况的实例。



其实Python允许加上许多层，不过层次一多，未来程序维护会变得比较困难。

程序实例ch5_8.py： 测试某一年是否闰年，闰年的条件是首先可以被4整除（相当于没有余数），这个条件成立时，还必须符合，它除以100时余数不为0或是除以400时余数为0，当2个条件皆符合才算闰年。

```
1 # ch5_8.py
2 print("判断输入年份是否闰年")
3 year = input("请输入年份: ")
4 rem4 = int(year) % 4
5 rem100 = int(year) % 100
6 rem400 = int(year) % 400
7 if rem4 == 0:
8     if rem100 != 0 or rem400 == 0:
9         print("%s 是闰年" % year)
10    else:
11        print("%s 不是闰年" % year)
12 else:
13    print("%s 不是闰年" % year)
```

执行结果

The screenshot shows the execution of the program for three different inputs. Each run starts with a 'RESTART' message and the file path 'D:\Python\ch5\ch5_8.py'. The first run shows the input '2018' and the output '2018 不是闰年' (2018 is not a leap year). The second run shows the input '2020' and the output '2020 是闰年' (2020 is a leap year). The third run shows the input '2100' and the output '2100 不是闰年' (2100 is not a leap year). Each run ends with a '>>>' prompt.

5-7 尚未设定的变量值None

有人在程序设计时，喜欢将所有变量一次先予以定义，在尚未用到此变量时先设定这个变量的值是None，如果此时用type()函数了解

它的类别时将显示“NoneType”，如下所示：

```
>>> x = None
>>> print(x)
None
>>> type(x)
<class 'NoneType'>
>>>
```

通常在程序设计时，可使用下列方式自我测试。

程序设计ch5_9.py: if语句与None的应用。

```
1 # ch5_9.py
2 flag = None
3 if flag == None:
4     print("尚未定义flag")
```

执行结果

```
===== RESTART: D:\Python\ch5\ch5_9.py =====
尚未定义flag
>>>
```

习题

1. 请设计一个程序，如果输入是负值则将它改成正值输出，如果输入是正值则将它改成负值输出，如果输入非数字则列出输入错误。
2. 请设计一个程序，此程序可以执行下列3件事：
 - 若输入是大写字符，请改成小写字符输出。
 - 若输入是小写字符，请改成大写字符输出。
 - 若输入是阿拉伯数字，则直接输出。
 - 若输入其他字符，则列出输入错误。
3. 请重新设计第四章实作题第4和5题，用户可以先选择温度转换方式，然后输入一个温度，可以转换成另一种温度。
4. 有一个百货公司庆祝50年周年庆，消费满10万元可打9折，消费满8万元可打95折，消费满5万元，可打98折。如果今年是50岁的消费

者不论消费金额都打95折，请设计这个程序。

5. 假设麦当劳打工薪资如下：

- 小于120小时（月），每小时是120小时工资的80%。
- 等于120小时（月），每小时是150元。
- 介于121至150小时（月），每小时是120小时工资的1.2倍。
- 大于150小时（月），每小时是120小时工资的1.6倍。

请输入工作时数，然后可以计算薪资。

第6章 列表(List)

列表(list)是Python的一种可以更改内容的数据类型，它是由一系列元素所组成的序列。如果现在我们要设计班上同学的成绩表，班上有50位同学，可能需要设计50个变量，这是一件麻烦的事。如果学校单位要设计所有学生的数据库，学生人数有1000人，需要1000个变量，这似乎是不可能的事。Python的列表 数据类型，可以只用一个变量，解决这方面的问题，要存取时用列表名称 加上索引值 即可，这也是本章的主题。

相信阅读至此章节，读者已经对Python有一些基础了解了，这章笔者也将讲解简单的面向对象 (Object Oriented) 观念，同时指导读者学习利用Python所提供的内置资源，未来将一步一步带领读者迈向高手之路。

6-1 认识列表

其实在其他程序语言，相类似的功能是称数组(array) 。不过，Python的列表 功能除了可以存储相同数据类型，例如，整数 、浮点数 、字符串 ，也可以存储不同数据类型，例如，列表 内同时含有整数 、浮点数 和字符串 。甚至一个列表 也可以内含其他列表（将在6-7节解说）或是字典 (dict) (将在9-3节解说)，因此，Python可以工作的能力，将比其他程序语言强大。

6-1-1 列表的基本定义

定义列表的语法格式如下：

```
name_list = [元素1, ... , 元素n] # name_list是假设的列表名称
```

基本上列表的每一个数据称元素，这些元素放在中括号[]内，彼此用逗号“,”隔开。如果要打印列表内容，可以使用print()函数，将列表名称当作变量名称即可。

实例1： NBA球员James前5场比赛得分，分别是23、19、22、31、18，可以用下列方式定义列表。

```
james = [23, 19, 22, 31, 18]
```

实例2： 为所销售的水果，苹果、香蕉、橘子建立列表，可以用下列方式定义列表。

```
fruits = ['apple', 'banana', 'orange']
```

在定义字符串时，元素内容也可以使用中文。

实例3： 为所销售的水果，苹果、香蕉、橘子建立中文元素的列表，可以用下列方式定义列表。

```
fruits = ['苹果', '香蕉', '橘子']
```

实例4： 在实例1的James列表，增加第1个元素，放他的全名。

```
James = ['Lebron James', 23, 19, 22, 31, 18]
```

程序实例ch6_1.py： 定义列表同时打印，最后使用type()列出列表数据类型。

```
1 # ch6_1.py
2 james = [23, 19, 22, 31, 18] # 定义james列表
3 print("打印james列表", james)
4 James = ['Lebron James', 23, 19, 22, 31, 18] # 定义James列表
5 print("打印James列表", James)
6 fruits = ['apple', 'banana', 'orange'] # 定义fruits列表
7 print("打印fruits列表", fruits)
8 cfruits = ['苹果', '香蕉', '橘子'] # 定义cfruits列表
9 print("打印cfruits列表", cfruits)
10 ielts = [5.5, 6.0, 6.5] # 定义IELTS成绩列表
11 print("打印IELTS成绩", ielts)
12 # 列出列表数据类型
13 print("列表james数据类型是：", type(james))
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_1.py =====
打印james列表 [23, 19, 22, 31, 18]
打印James列表 ['Lebron James', 23, 19, 22, 31, 18]
打印fruits列表 ['apple', 'banana', 'orange']
打印cfruits列表 ['苹果', '香蕉', '橘子']
打印IELTS成绩 [5.5, 6.0, 6.5]
列表james数据类型是: <class 'list'>
>>>
```

6-1-2 读取列表元素

我们可以用列表名称与索引读取列表元素的内容，在Python中元素是从索引值0开始配置。所以如果是列表的第一个元素，索引值是0，第二个元素索引值是1，其他依此类推，如下所示：

`name_list[i]` # 读取索引*i*的列表元素

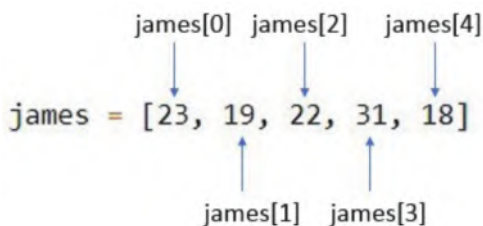
程序实例ch6_2.py： 读取列表元素的应用。

```
1 # ch6_2.py
2 james = [23, 19, 22, 31, 18]          # 定义james列表
3 print("打印james第1场得分", james[0])
4 print("打印james第2场得分", james[1])
5 print("打印james第3场得分", james[2])
6 print("打印james第4场得分", james[3])
7 print("打印james第5场得分", james[4])
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_2.py =====
打印james第1场得分 23
打印james第2场得分 19
打印james第3场得分 22
打印james第4场得分 31
打印james第5场得分 18
>>>
```

上述程序经过第2行的定义后，列表索引值的解释如下：



所以程序第3行至第7行，可以得到上述执行结果。其实我们也可以将2-9节等号多重指定观念应用在列表。

程序实例ch6_2_1.py： 一个传统处理列表元素内容方式，与Python多重指定观念的应用。

```
1 # ch6_2_1.py
2 james = [23, 19, 22, 31, 18]          # 定义james列表
3 # 传统设计方式
4 game1 = james[0]
5 game2 = james[1]
6 game3 = james[2]
7 game4 = james[3]
8 game5 = james[4]
9 print("打印james各场次得分", game1, game2, game3, game4, game5)
10 # Python高手好的设计方式
11 game1, game2, game3, game4, game5 = james
12 print("打印james各场次得分", game1, game2, game3, game4, game5)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_2_1.py =====
打印james各场次得分 23 19 22 31 18
打印james各场次得分 23 19 22 31 18
>>>
```

上述程序第11行让整个Python设计简洁许多，这是Python高手常用的程序设计方式，在上述设计中第11行的多重指定变数的数量需与列表元素的个数相同，否则会有错误产生。

6-1-3 列表切片(list slices)

在设计程序时，常会需要取得列表前几个元素、后几个元素、某区间元素或是依照一定规则排序的元素，所取得的系列元素也可称子列表，这个观念称列表切片 (list slices)，此时可以用下列方法。

```
name_list[start:end]          # 读取从索引 start 到 (end-1) 索引的列表元素
name_list[:n]                 # 取得列表前 n 名
name_list[n:]                 # 取得列表索引 n 到最后
name_list[-n:]                # 取得列表后 n 名
name[:]                       # 取得所有元素，将在 6-8-3 节解说

下列是读取区间，但是用 step 作为每隔多少区间再读取。
name_list[start:end:step]     # 每隔 step，读取从索引 start 到 (end-1)
                              索引的列表元素
```


程序实例ch6_2_2.py: 列出特定区间球员的得分子列表。

```
1 # ch6_2_2.py
2 james = [23, 19, 22, 31, 18]          # 定义james列表
3 print("打印james第1-3场得分", james[0:3])
4 print("打印james第2-4场得分", james[1:4])
5 print("打印james第1,3,5场得分", james[0:6:2])
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_2_2.py =====
打印james第1-3场得分 [23, 19, 22]
打印james第2-4场得分 [19, 22, 31]
打印james第1,3,5场得分 [23, 22, 18]
>>>
```

程序实例ch6_3.py: 列出球队前3名队员、从索引1到最后队员与后3名队员子列表。

```
1 # ch6_3.py
2 warriors = ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
3 first3 = warriors[:3]
4 print("前3名球员", first3)
5 n_to_last = warriors[1:]
6 print("球员索引1到最后", n_to_last)
7 last3 = warriors[-3:]
8 print("后3名球员", last3)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_3.py =====
前3名球员 ['Curry', 'Durant', 'Iquodala']
球员索引1到最后 ['Durant', 'Iquodala', 'Bell', 'Thompson']
后3名球员 ['Iquodala', 'Bell', 'Thompson']
>>>
```

6-1-4 列表索引值是-1

在列表使用中，如果索引值是-1，代表是最后一个列表元素。

程序实例ch6_4.py: 列表索引值是-1的应用，由下列执行结果可以得到各列表的最后一个元素了。

```
1 # ch6_4.py
2 warriors = ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
3 print("最后一名球员", warriors[-1])
4 james = [23, 19, 22, 31, 18]
5 print("最后一场得分", james[-1])
6 mixs = [9, 20.5, 'DeepStone']
7 print("最后一个元素", mixs[-1])
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_4.py =====
最后一名球员 Thompson
最后一场得分 18
最后一个元素 DeepStone
>>>
```

其实在Python中索引-1代表最后1个元素，-2代表最后第2个元素，其他负索引观念可依次类推，可参考下列实例。

程序实例ch6_4_1.py： 使用负索引列出warriors列表内容。

```
1 # ch6_4_1.py
2 warriors = ['Curry', 'Durant', 'Iguodala', 'Bell', 'Thompson']
3 print(warriors[-1],warriors[-2],warriors[-3],warriors[-4],warriors[-5])
```

执行结果

```
===== RESTART: D:/Python/ch6/ch6_4_1.py =====
Thompson Bell Iguodala Durant Curry
>>>
```

6-1-5 列表统计资料、最大值max()、最小值min()、总和sum()

Python有内置一些执行统计运算的函数，如果列表内容全部是数值则可以使用max()函数获得列表的最大值，min()函数可以获得列表的最小值，sum()函数可以获得列表的总和。如果列表内容全部是字符或字符串则可以使用max()函数获得列表的unicode码值的最大值，min()函数可以获得列表的unicode码值最小值。sum()则不可使用在列表元素为非数值情况。

程序实例ch6_5.py： 计算james球员5场的最高得分、最低得分和5场的得分总计。

```
1 # ch6_5.py
2 james = [23, 19, 22, 31, 18] # 定义james的5场比赛得分
3 print("最高得分 = ", max(james))
4 print("最低得分 = ", min(james))
5 print("得分总计 = ", sum(james))
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_5.py =====
最高得分 = 31
最低得分 = 18
得分总计 = 113
>>>
```

上述我们很快获得了统计信息，各位可能会想，当列表内含有字符串，如程序实例ch6_1.py的James列表，这个列表第一个元素是字符串，如果这时仍然直接用max(James)会有错误的。

```
>>> James = ['Lebron James', 23, 19, 22, 31, 18]
>>> x = max(James)
Traceback (most recent call last):
  File "<pyshe11#83>", line 1, in <module>
    x = max(James)
TypeError: '>' not supported between instances of 'int' and 'str'
>>>
```

碰上这类的字符串我们可以使用6-1-2节方式，用切片方式处理，如下所示。

程序实例ch6_6.py： 重新设计ch6_5.py，但是使用James列表。

```
1 # ch6_6.py
2 James = ['Lebron James', 23, 19, 22, 31, 18] # 定义james的5场比赛得分
3 print("最高得分 = ", max(James[1:6]))
4 print("最低得分 = ", min(James[1:6]))
5 print("得分总计 = ", sum(James[1:6]))
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_6.py =====
最高得分 = 31
最低得分 = 18
得分总计 = 113
>>>
```

6-1-6 列表个数len()

程序设计时，可能会增加元素，也有可能会删除元素，时间久了即使是程序设计师也无法得知列表内剩余多少元素，此时可以借用本小节的len()函数，这个函数可以获得列表的元素个数。

程序实例ch6_7.py： 重新设计ch6_5.py，增加场次数据。

```
1 # ch6_7.py
2 James = [23, 19, 22, 31, 18] # 定义james的5场比赛得分
3 games = len(James) # 获得场次数据
4 print("经过 %d 比赛最高得分 = " % games, max(James))
5 print("经过 %d 比赛最低得分 = " % games, min(James))
6 print("经过 %d 比赛得分总计 = " % games, sum(James))
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_7.py =====
经过 5 比赛最高得分 = 31
经过 5 比赛最低得分 = 18
经过 5 比赛得分总计 = 113
>>>
```

6-1-7 更改列表元素的内容

可以使用列表名称和索引值更改列表元素的内容。

程序实例ch6_8.py: 修改james第5场比赛分数。

```
1 # ch6_8.py
2 james = [23, 19, 22, 31, 18] # 定义james的5场比赛得分
3 print("旧的James比赛分数", james)
4 james[4] = 28
5 print("新的James比赛分数", james)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_8.py =====
旧的James比赛分数 [23, 19, 22, 31, 18]
新的James比赛分数 [23, 19, 22, 31, 28]
>>>
```

这个观念可以用在更改整数数据也可以修改字符串数据。

程序实例ch6_9.py: 一家汽车经销商原本可以销售Toyota、Nissan、Honda, 现在Nissan销售权被回收, 改成销售Ford, 可用下列方式设计销售品牌。

```
1 # ch6_9.py
2 cars = ['Toyota', 'Nissan', 'Honda']
3 print("旧汽车销售品牌", cars)
4 cars[1] = 'Ford' # 更改第二个元素内容
5 print("新汽车销售品牌", cars)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_9.py =====
旧汽车销售品牌 ['Toyota', 'Nissan', 'Honda']
新汽车销售品牌 ['Toyota', 'Ford', 'Honda']
>>>
```

6-1-8 列表的相加

Python是允许列表相加的，相当于将列表结合。

程序实例ch6_10.py： 一家汽车经销商原本可以销售Toyota、Nissan、Honda，现在并购一家销售Audi、BMW的经销商，可用下列方式设计销售品牌。

```
1 # ch6_10.py
2 cars1 = ['Toyota', 'Nissan', 'Honda']
3 print("旧汽车销售品牌", cars1)
4 cars2 = ['Audi', 'BMW']
5 cars1 += cars2
6 print("新汽车销售品牌", cars1)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_10.py =====
旧汽车销售品牌 ['Toyota', 'Nissan', 'Honda']
新汽车销售品牌 ['Toyota', 'Nissan', 'Honda', 'Audi', 'BMW']
>>>
```

程序实例ch6_11.py： 整数列表相加的实例。

```
1 # ch6_11.py
2 num1 = [1, 3, 5]
3 num2 = [2, 4, 6]
4 num3 = num1 + num2          # 字符串为主的列表相加
5 print(num3)
```

执行结果

```
===== RESTART: D:/Python/ch6/ch6_11.py =====
[1, 3, 5, 2, 4, 6]
>>>
```

6-1-9 列表乘以一个数字

如果将列表乘以一个数字，这个数字相当于是列表元素重复次数。

程序实例ch6_12.py： 将列表乘以数字的应用。

```
1 # ch6_12.py
2 cars = ['toyota', 'nissan', 'honda']
3 nums = [1, 3, 5]
4 carslist = cars * 3          # 列表乘以数字
5 print(carslist)
6 numslist = nums * 5          # 列表乘以数字
7 print(numslist)
```

执行结果

```
===== RESTART: D:/Python/ch6/ch6_12.py =====
['toyota', 'nissan', 'honda', 'toyota', 'nissan', 'honda', 'toyota', 'nissan',
'honda']
[1, 3, 5, 1, 3, 5, 1, 3, 5, 1, 3, 5, 1, 3, 5]
>>>
```

6-1-10 列表元素的加法运作

既然我们可以读取列表内容，其实就可以使用相同的观念操作列表内的元素数据。

程序实例ch6_13.py: 建立Lebron James和Kevin Love在比赛的得分列表，然后利用列表元素加法运作，列出2个人在第四场比赛的得分总和。

```
1 # ch6_13.py
2 James = ['Lebron James',23, 19, 22, 31, 18] # 定义James列表
3 Love = ['Kevin Love',20, 18, 30, 22, 15] # 定义Love列表
4 game3 = James[4] + Love[4]
5 LKgame = James[0] + ' 和 ' + Love[0] + '第四场总得分 = '
6 print(LKgame, game3)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_13.py =====
Lebron James 和 Kevin Love第四场总得分 = 53
>>>
```

需要注意，由第2行列表定义可知，James[0]是指“Lebron James”，James[1]是第1场得分23，所以James[4]是第4场得分31。第3行Love列表观念相同。

6-1-11 删除列表元素

可以使用下列方式删除指定索引的列表元素：

```
del name_list[i] # 删除索引i的列表元素
```


下列是删除列表区间元素。

```
del name_list[start:end]    # 删除从索引start到(end-1)索引的列表元素
```

下列是删除区间，但是用step作为每隔多少区间再删除。

```
del name_list[start:end:step]    # 每隔step，删除从索引start到(end-1)
```

索引的列表元素

程序实例ch6_14.py： 如果NBA勇士队主将阵容有5名，其中一名队员Bell离队了，可用下列方式设计。

```
1 # ch6_14.py
2 warriors = ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
3 print("2018年初NBA勇士队主将阵容", warriors)
4 del warriors[3]          # 不明原因离队
5 print("2018年末NBA勇士队主将阵容", warriors)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_14.py =====
2018年初NBA勇士队主将阵容 ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
2018年末NBA勇士队主将阵容 ['Curry', 'Durant', 'Iquodala', 'Thompson']
>>>
```

程序实例ch6_15.py： 删除列表元素的应用。

```
1 # ch6_15.py
2 nums1 = [1, 3, 5]
3 print("删除nums1列表索引1元素前  = ", nums1)
4 del nums1[1]
5 print("删除nums1列表索引1元素后  = ", nums1)
6 nums2 = [1, 2, 3, 4, 5, 6]
7 print("删除nums2列表索引[0:2]前  = ", nums2)
8 del nums2[0:2]
9 print("删除nums2列表索引[0:2]后  = ", nums2)
10 nums3 = [1, 2, 3, 4, 5, 6]
11 print("删除nums3列表索引[0:6:2]前 = ", nums3)
12 del nums3[0:6:2]
13 print("删除nums3列表索引[0:6:2]后 = ", nums3)
```

执行结果

```

RESTART: D:\Python\ch6\ch6_15.py
删除nums1列表索引1元素前 = [1, 3, 5]
删除nums1列表索引1元素后 = [1, 5]
删除nums2列表索引[0:2]前 = [1, 2, 3, 4, 5, 6]
删除nums2列表索引[0:2]后 = [3, 4, 5, 6]
删除nums3列表索引[0:6:2]前 = [1, 2, 3, 4, 5, 6]
删除nums3列表索引[0:6:2]后 = [2, 4, 6]
>>>

```

以这种方式删除列表元素最大的缺点是，元素删除后我们无法得知删除的是什么内容。有时我们设计网站时，可能想将某个人从VIP客户降为一般客户，采用上述方式删除元素时，我们就无法再度取得所删除的元素数据，未来笔者会介绍另一种方式删除数据，删除后我们还可善加利用所删除的数据。又或者你设计一个游戏，敌人是放在列表内，采用上述方式删除所杀死的敌人时，我们就无法再度取得所删除的敌人元素数据，如果我们可以取得的话，可以在杀死敌人坐标位置放置庆祝动画等。

6-1-12 列表为空列表的判断

如果想建立一个列表，可是暂时不放置元素，可使用下列方式定义。

```
name_list = [ ] # 这是空的列表
```

程序实例ch6_16.py： 删除列表元素的应用，这个程序基本上会用len()函数判断列表内是否有元素数据，如果有则删除索引为0的元素，如果没有则列出列表内没有元素了。


```

1 # ch6_16.py
2 cars = ['Toyota', 'Nissan', 'Honda']
3 print("cars列表长度是 = %d" % len(cars))
4 if len(cars) != 0:
5     del cars[0]
6     print("删除cars列表元素成功")
7     print("cars列表长度是 = %d" % len(cars))
8 else:
9     print("cars列表内没有元素数据")
10 nums = []
11 print("nums列表长度是 = %d" % len(nums))
12 if len(nums) != 0:
13     del nums[0]
14     print("删除nums列表元素成功")
15 else:
16     print("nums列表内没有元素数据")

```

执行结果

```

===== RESTART: D:\Python\ch6\ch6_16.py
cars列表长度是 = 3
删除cars列表元素成功
cars列表长度是 = 2
nums列表长度是 = 0
nums列表内没有元素数据
>>>

```

6-1-13 删除列表

Python也允许我们删除整个列表，列表一经删除后就无法复原，同时也无法做任何操作了，下列是删除列表的方式：

```
del name_list # 删除列表name_list
```

实例1： 建立列表、打印列表、删除列表，然后尝试再度打印列表结果出现错误信息，因为列表经删除后已经不存在了。

```

>>> x = [1,2,3]
>>> print(x)
[1, 2, 3]
>>> del x
>>> print(x)
Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>

```

6-2 Python简单的面向对象观念

在面向对象的程序设计(Object Oriented Programming)观念里，所有数据皆算是一个对象 (Object)，例如，整数、浮点数、字符串或是本章所提的列表 皆是一个对象。我们可以为所建立的对象设计一些方法 (method)，供这些对象使用，在这里所提的方法 就是函数，其实方法与函数还是有一些差异，后面还会解说。目前Python有 为一些基本对象提供默认的方法，要使用这些方法 可以在对象 后先放小数点，再放方法名称，基本语法格式如下：

对象.方法()

下列将分成几个小节一步一步以实例说明。

6-2-1 字符串的方法

几个字符串操作常用的方法(method)如下：

- lower()：将字符串转成小写字。
- upper()：将字符串转成大写字。
- title()：将字符串转成第一个字母大写，其他是小写。
- rstrip()：删除字符串尾端多余的空白。
- lstrip()：删除字符串开始端多余的空白。
- strip()：删除字符串头尾两边多余的空白。

程序实例ch6_17.py： 将字符串改成小写，与将字符串改成大写，以及将字符串改成第一个字母大写，其他小写。

```
1 # ch6_17.py
2 strN = "DeepStone"
3 strU = strN.upper( )           # 改成大写
4 strL = strN.lower( )          # 改成小写
5 strT = strN.title( )           # 改成第一个字母大写其他小写
6 print("大写输出:",strU,"\\n小写输出:",strL,"\\n第一字母大写:",strT)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_17.py =====
大写输出: DEEPSTONE
小写输出: deepstone
第一字母大写: Deepstone
>>>
```

删除字符串开始或结尾多余空白是一个很好用的方法(method)，特别是系统要求读者输入数据时，一定会有人不小心多输入了一些空格符，此时可以用这个方法删除多余的空白。

程序实例ch6_18.py: 删除开始端与结尾端多余空白的应用。

```
1 # ch6_18.py
2 strN = " DeepStone "
3 strL = strN.lstrip( )      # 删除字符串左边多余空白
4 strR = strN.rstrip( )     # 删除字符串右边多余空白
5 strB = strN.lstrip( )     # 先删除字符串左边多余空白
6 strB = strB.rstrip( )     # 再删除字符串右边多余空白
7 strO = strN.strip( )      # 一次删除头尾端多余空白
8 print("/%s/" % strN)
9 print("/%s/" % strL)
10 print("/%s/" % strR)
11 print("/%s/" % strB)
12 print("/%s/" % strO)
```

执行结果

```
===== RESTART: D:/Python/ch6/ch6_18.py =====
/ DeepStone /
/DeepStone /
/ DeepStone/
/DeepStone/
/DeepStone/
>>>
```

6-2-2 更改字符串大小写

如果列表内的元素字符串数据是小写，例如：输出的车辆名称是“benz”，其实我们可以使用前一小节的title()让开头车辆名称的第一个字母大写，可能会更好。

程序实例ch6_19.py: 将upper()和title()应用在字符串。

```
1 # ch6_19.py
2 cars = ['bmw', 'benz', 'audi']
3 carF = "我开的第一部车是 " + cars[1].title( )
4 carN = "我现在开的车子是 " + cars[0].upper( )
5 print(carF)
6 print(carN)
```

上述第4行是将bmw改为BMW。

执行结果

```
===== RESTART: D:\Python\ch6\ch6_19.py =====
我开的第一部车是 Benz
我现在开的车子是 BMW
>>>
```

6-2-3 dir() 获得系统内部对象的方法

6-2-1节笔者列举了字符串常用的方法 (method)，dir() 函数可以列出对象有哪些内置的方法可以使用。

实例1： 列出字符串对象的方法，处理方式是先设定一个字符串变量，再列出此字符串变量的方法(method)。

```
>>> string = 'abc'
>>> dir(string)
['_add_', '_class_', '_contains_', '_delattr_', '_dir_', '_doc_', '_eq_',
'_format_', '_gc_', '_getattr_', '_getitem_', '_getnewargs_', '_gt_',
'_hash_', '_init_', '_init_subclass_', '_iter_', '_le_', '_len_',
'_lt_', '_mod_', '_mul_', '_ne_', '_new_', '_reduce_', '_reduce_ex_',
'_repr_', '_rmod_', '_rmul_', '_setattr_', '_sizeof_', '_str_', '_subclass_',
'_look_', '_capitalize_', '_casefold_', '_center_', '_count_', '_encode_', '_endswith_', '_expandtab_',
'_find_', '_format_', '_format_map_', '_index_', '_isalnum_', '_isalpha_', '_isdecimal_', '_isdi_',
'_git_', '_isidentifier_', '_islower_', '_isnumeric_', '_isprintable_', '_isspace_', '_istitle_', '_i',
'_supper_', '_join_', '_ljust_', '_lower_', '_rstrip_', '_maketrans_', '_partition_', '_replace_', '_rf',
'_ind_', '_rindex_', '_rjust_', '_rpartition_', '_rsplit_', '_rstrip_', '_split_', '_splitlines_', '_st',
'_artswith_', '_strip_', '_swapcase_', '_title_', '_translate_', '_upper_', '_zfill_']
>>>
```

上述圈起来的，笔者在6-2-1节已有解说。看到上述密密麻麻的方法，不用紧张，也不用想要一次学会，需要时再学即可。如果想要了解上述特定方法可以使用4-1节所介绍的help() 函数，可以用下列方式：

help(对象.方法名称)

实例2： 延续前一个实例，列出对象string，内置的islower的使用说明，同时以string对象为例，测试使用结果。

```
>>> help(string.islower)
Help on built-in function islower:

islower(...) method of builtins.str instance
    S.islower() -> bool

    Return True if all cased characters in S are lowercase and there is
    at least one cased character in S, False otherwise.

>>> x = string.islower( )
>>> print(x)
True
>>>
```

由上述说明可知，`islower()`可以传回对象是否是小写，如果对象全部是小写或至少有一个字符是小写将传回True，否则传回False。在上述实例，由于string对象的内容是“abc”，全部是小写，所以传回True。

上述观念同样可以应用在查询整数对象的方法。

实例3： 列出整数对象的方法，同样可以先设定一个整数变量，再列出此整数变量 的方法(method) 。

```
>>> num = 5
>>> dir(num)
['_abs_', '_add_', '_and_', '_bool_', '_ceil_', '_class_', '_delattr_', '_dir_', '_divmod_', '_doc_', '_eq_', '_float_', '_floor_', '_floordiv_', '_format_', '_ge_', '_getattr_', '_getnewargs_', '_gt_', '_hash_', '_index_', '_init_', '_init_subclass_', '_int_', '_invert_', '_le_', '_lshift_', '_lt_', '_mod_', '_mul_', '_ne_', '_neg_', '_new_', '_or_', '_pos_', '_pow_', '_radd_', '_rand_', '_rdivmod_', '_reduce_', '_reduce_ex_', '_repr_', '_rfloordiv_', '_rshift_', '_rmod_', '_rmul_', '_ror_', '_round_', '_rpow_', '_rrshift_', '_rshift_', '_rsub_', '_rtruediv_', '_rxor_', '_setattr_', '_sizeof_', '_str_', '_sub_', '_subclasshook_', '_truediv_', '_truncate_', '_xor_', '_bit_length_', '_conjugate_', '_denominator_', '_from_bytes_', '_imag_', '_numerator_', '_real_', '_to_bytes_']
>>>
```

上述`bit_length`是可以计算出要多少位以2进位方式存储此变量。

实例4： 列出需要多少位，存储实例3的整数变量num。

```
>>> num = 5
>>> y = num.bit_length()
>>> print(y)
3
>>>
```

6-3 获得列表的方法

这节重点是列表，我们可以使用下列方式获得列表的方法。

实例1： 列出内置列表(list)内字符串(string)元素的方法。

```
>>> string = ["bmw", "benz", "audi"]
>>> dir(string)
['_add_', '_class_', '_contains_', '_delattr_', '_delitem_', '_dir_', '_doc_', '_eq_', '_format_', '_ge_', '_getattr_', '_getitem_', '_gt_', '_hash_', '_iadd_', '_imul_', '_init_', '_init_subclass_', '_iter_', '_le_', '_len_', '_lt_', '_mul_', '_ne_', '_new_', '_reduce_', '_reduce_ex_', '_repr_', '_reversed_', '_rmul_', '_setattr_', '_setitem_', '_sizeof_', '_str_', '_subclasshook_', '_append_', '_clear_', '_copy_', '_count_', '_extend_', '_index_', '_insert_', '_pop_', '_remove_', '_reverse_', '_sort_']
>>>
```

上述实例的重点是我们先建立一个列表string，然后由此列表利用dir()函数可以了解有哪些列表的方法可以使用。

实例2： 列出内置列表(list)内整数(int)元素的方法。

```
>>> numlist = [1, 3, 5]
>>> dir(numlist)
['_add_', '_class_', '_contains_', '_delattr_', '_delitem_', '_dir_', '_doc_', '_eq_', '_format_', '_ge_', '_getattr_', '_getitem_', '_gt_', '_hash_', '_iadd_', '_imul_', '_init_', '_init_subclass_', '_iter_', '_le_', '_len_', '_lt_', '_mul_', '_ne_', '_new_', '_reduce_', '_reduce_ex_', '_repr_', '_reversed_', '_rmul_', '_setattr_', '_setitem_', '_sizeof_', '_str_', '_subclasshook_', '_append_', '_clear_', '_copy_', '_count_', '_extend_', '_index_', '_insert_', '_pop_', '_remove_', '_reverse_', '_sort_']
>>>
```

可以看到实例1与实例2内容完全相同，这表示下一节起讲解操作列表的方法，可以用在字符串元素，也可以用在整数元素。

6-4 增加与删除列表元素

6-4-1 在列表末端增加元素append()

程序设计时常常会发生需要增加列表元素的情况，如果目前元素个数是3个，想要增加第4个元素，读者可能会想可否使用下列传统方式，直接设定新增的值：

```
name_list[3] = value
```

实例1： 使用索引方式，为列表增加元素，但是发生索引值超过列表长度的错误。

```
>>> car = ['Honda', 'Toyata', 'Ford']
>>> print(car)
['Honda', 'Toyata', 'Ford']
>>> car[3] = 'Nissan'
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    car[3] = 'Nissan'
IndexError: list assignment index out of range
>>>
```

读者可能会想可以增加一个新列表，将欲新增的元素放在新列表，然后再将原先列表与新列表相加，就达到增加列表元素的目的

了。这个方法理论是可以，可是太麻烦了。Python为列表内置了新增元素的方法`append()`，这个方法，可以在列表末端直接增加元素。

```
name_list.append('新增元素')
```

程序实例ch6_20.py： 先建立一个空列表，然后分别使用`append()`增加3个元素内容。

```
1 # ch6_20.py
2 cars = []
3 print("目前列表内容 = ",cars)
4 cars.append('Honda')
5 print("目前列表内容 = ",cars)
6 cars.append('Toyota')
7 print("目前列表内容 = ",cars)
8 cars.append('Ford')
9 print("目前列表内容 = ",cars)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_20.py =====
目前列表内容 = []
目前列表内容 = ['Honda']
目前列表内容 = ['Honda', 'Toyota']
目前列表内容 = ['Honda', 'Toyota', 'Ford']
>>>
```

6-4-2 插入列表元素`insert()`

`append()`方法是固定在列表末端插入元素，`insert()`方法则是在任意位置插入元素，它的使用格式如下：

```
insert(索引, 元素内容) # 索引是插入位置，元素内容是插入内容
```

程序实例ch6_21.py： 使用`insert()`插入列表元素的应用。

```

1 # ch6_21.py
2 cars = ['Honda', 'Toyota', 'Ford']
3 print("目前列表内容 = ", cars)
4 print("在索引1位置插入Nissan")
5 cars.insert(1, 'Nissan')
6 print("新的列表内容 = ", cars)
7 print("在索引0位置插入BMW")
8 cars.insert(0, 'BMW')
9 print("最新列表内容 = ", cars)

```

执行结果

```

===== RESTART: D:\Python\ch6\ch6_21.py =====
目前列表内容 = ['Honda', 'Toyota', 'Ford']
在索引1位置插入Nissan
新的列表内容 = ['Honda', 'Nissan', 'Toyota', 'Ford']
在索引0位置插入BMW
最新列表内容 = ['BMW', 'Honda', 'Nissan', 'Toyota', 'Ford']
>>>

```

6-4-3 删除列表元素pop()

6-1-8节笔者有介绍使用del删除列表元素，在该节笔者同时指出最大缺点是，资料删除了就无法取得相关信息。使用pop()方法删除元素最大的优点是，删除后将弹出所删除的值，使用pop()时若是未指明所删除元素的位置，一律删除列表末端的元素。pop()的使用方式如下：

```
value = name_list.pop( )    # 没有索引是删除列表末端元素
```

```
value = name_list.pop(i)    # 是删除指定索引值的列表元素
```

程序实例ch6_22.py： 使用pop()删除列表元素的应用，这个程序第5行未指明删除的索引值，所以删除了列表的最后一个元素。程序第9行则是指明删除索引值为1的元素。

```

1 # ch6_22.py
2 cars = ['Honda', 'Toyota', 'Ford', 'BMW']
3 print("目前列表内容 = ", cars)
4 print("使用pop( )删除列表元素")
5 popped_car = cars.pop( )           # 删除列表末端值
6 print("所删除的列表内容是：", popped_car)
7 print("新的列表内容 = ", cars)
8 print("使用pop(1)删除列表元素")
9 popped_car = cars.pop(1)           # 删除列表索引为1的值
10 print("所删除的列表内容是：", popped_car)
11 print("新的列表内容 = ", cars)

```


执行结果

```
===== RESTART: D:\Python\ch6\ch6_22.py =====
目前列表内容 = ['Honda', 'Toyota', 'Ford', 'BMW']
使用pop()删除列表元素
所删除的列表内容是: BMW
新的列表内容 = ['Honda', 'Toyota', 'Ford']
使用pop()删除列表元素
所删除的列表内容是: Toyota
新的列表内容 = ['Honda', 'Ford']
>>>
```

6-4-4 删除指定的元素remove()

在删除列表元素时，有时可能不知道元素在列表内的位置，此时可以使用remove()方法删除指定的元素，它的使用方式如下：

```
name_list.remove(想删除的元素内容)
```

如果列表内有相同的元素，则只删除第一个出现的元素，如果想要删除所有相同的元素，必须使用循环，下一章将会讲解循环的观念。

程序实例ch6_23.py： 删除列表中第一次出现的元素bmw，这个列表有2个bmw字符串，最后只删除索引为1的bmw字符串。

```
1 # ch6_23.py
2 cars = ['Honda', 'bmw', 'Toyota', 'Ford', 'bmw']
3 print("目前列表内容 = ", cars)
4 print("使用remove()删除列表元素")
5 expensive = 'bmw'
6 cars.remove(expensive)          # 删除第一次出现的元素bmw
7 print("所删除的内容是: " + expensive.upper() + " 因为太贵了")
8 print("新的列表内容", cars)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_23.py =====
目前列表内容 = ['Honda', 'bmw', 'Toyota', 'Ford', 'bmw']
使用remove()删除列表元素
所删除的内容是: BMW 因为太贵了
新的列表内容 ['Honda', 'Toyota', 'Ford', 'bmw']
>>>
```

6-5 列表的排序

6-5-1 颠倒排序reverse()

reverse()可以颠倒排序列表元素，它的使用方式如下：

```
name_list.reverse( )    # 颠倒排序name_list列表元素
```

其实在6-1-3节的切片应用中，也可以用[::-1]方式取得列表颠倒排序。

程序实例ch6_24.py： 使用2种方式执行颠倒排序列表元素。

```
1 # ch6_24.py
2 cars = ['Honda', 'bmw', 'Toyota', 'Ford', 'bmw']
3 print("目前列表内容 = ", cars)
4 # 直接打印cars[::-1]颠倒排序,不更改列表内容
5 print("打印使用[::-1]颠倒排序\n", cars[::-1])
6 # 更改列表内容
7 print("使用reverse( )颠倒排序列表元素")
8 cars.reverse( )          # 颠倒排序列表
9 print("新的列表内容 = ", cars)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_24.py =====
目前列表内容 = ['Honda', 'bmw', 'Toyota', 'Ford', 'bmw']
打印使用[::-1]颠倒排序
['bmw', 'Ford', 'Toyota', 'bmw', 'Honda']
使用reverse( )颠倒排序列表元素
新的列表内容 = ['bmw', 'Ford', 'Toyota', 'bmw', 'Honda']
>>>
```

列表经颠倒排放后，就算永久性更改了，如果要复原，可以再执行一次reverse()方法。

6-5-2 sort()排序

sort()方法可以对列表元素由小到大排序，这个方法同时对纯数值元素与纯英文字符串元素有非常好的效果。需要注意的是，经排序后原列表的元素顺序会被永久更改。它的使用格式如下：

```
name_list.sort( )    # 由小到大排序name_list列表
```

如果是排序英文字符串，建议先将字符串英文字符全部改成小写或全部改成大写。

程序实例ch6_25.py： 数字与英文字符串元素排序的应用。

```
1 # ch6_25.py
2 cars = ['honda', 'bmw', 'toyota', 'ford']
3 print("目前列表内容 = ", cars)
4 print("使用sort()由小排到大")
5 cars.sort()
6 print("排序列表结果 = ", cars)
7 nums = [5, 3, 9, 2]
8 print("目前列表内容 = ", nums)
9 print("使用sort()由小排到大")
10 nums.sort()
11 print("排序列表结果 = ", nums)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_25.py =====
目前列表内容 = ['honda', 'bmw', 'toyota', 'ford']
使用sort()由小排到大
排序列表结果 = ['bmw', 'ford', 'honda', 'toyota']
目前列表内容 = [5, 3, 9, 2]
使用sort()由小排到大
排序列表结果 = [2, 3, 5, 9]
>>>
```

上述内容是由小排到大，sort()方法是允许由大排到小，只要在sort()内增加参数“reverse=True”即可。

程序实例ch6_26.py： 重新设计ch6_25.py，将列表元素由大排到小。

```
1 # ch6_26.py
2 cars = ['honda', 'bmw', 'toyota', 'ford']
3 print("目前列表内容 = ", cars)
4 print("使用sort()由大排到小")
5 cars.sort(reverse=True)
6 print("排序列表结果 = ", cars)
7 nums = [5, 3, 9, 2]
8 print("目前列表内容 = ", nums)
9 print("使用sort()由大排到小")
10 nums.sort(reverse=True)
11 print("排序列表结果 = ", nums)
```

执行结果

```

===== RESTART: D:\Python\ch6\ch6_26.py =====
目前列表内容 = ['honda', 'bmw', 'toyota', 'ford']
使用sort()由大到小
排序列表结果 = ['toyota', 'honda', 'ford', 'bmw']
目前列表内容 = [5, 3, 9, 2]
使用sort()由大到小
排序列表结果 = [9, 5, 3, 2]
>>>

```

6-5-3 sorted() 排序

前一小节的sort() 排序将造成列表元素顺序永久更改，如果你不希望更改列表元素顺序，可以使用另一种排序sorted()，使用这个排序可以获得想要的排序结果，我们可以用新列表存储新的排序列表，同时原先列表的顺序将不更改。它的使用格式如下：

new_list.sorted(name_list) # 用新列表存储排序，原列表序列不更改

程序实例 ch6_27.py： sorted() 排序的应用，这个程序使用car_sorted新列表存储car列表的排序结果，同时使用num_sorted新列表存储num列表的排序结果。

```

1 # ch6_27.py
2 cars = ['honda', 'bmw', 'toyota', 'ford']
3 print("目前串列car内容 = ",cars)
4 print("使用sorted()由小排到大")
5 cars_sorted = sorted(cars)
6 print("排序串列结果 = ",cars_sorted)
7 print("原先串列car内容 = ",cars)
8 nums = [5, 3, 9, 2]
9 print("目前串列num内容 = ",nums)
10 print("使用sorted()由小排到大")
11 nums_sorted = sorted(nums)
12 print("排序串列结果 = ",nums_sorted)
13 print("原先串列num内容 = ",nums)

```

执行结果

```

===== RESTART: D:\Python\ch6\ch6_27.py =====
目前列表car内容 = ['honda', 'bmw', 'toyota', 'ford']
使用sorted()由小排到大
排序列表结果 = ['bmw', 'ford', 'honda', 'toyota']
原先列表car内容 = ['honda', 'bmw', 'toyota', 'ford']
目前列表num内容 = [5, 3, 9, 2]
使用sorted()由小排到大
排序列表结果 = [2, 3, 5, 9]
原先列表num内容 = [5, 3, 9, 2]
>>>

```

如果我们想要从大排到小，可以在sorted()内增加参数“reverse=True”，可参考下列实例第5和11行。

程序实例ch6_28.py： 重新设计ch6_27.py，将列表由大排到小。

```
1 # ch6_28.py
2 cars = ['honda', 'bmw', 'toyota', 'ford']
3 print("目前列表car内容 = ", cars)
4 print("使用sorted( )由大排到小")
5 cars_sorted = sorted(cars, reverse=True)
6 print("排序列表结果 = ", cars_sorted)
7 print("原先列表car内容 = ", cars)
8 nums = [5, 3, 9, 2]
9 print("目前列表num内容 = ", nums)
10 print("使用sorted( )由大排到小")
11 nums_sorted = sorted(nums, reverse=True)
12 print("排序列表结果 = ", nums_sorted)
13 print("原先列表num内容 = ", nums)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_28.py =====
目前列表car内容 = ['honda', 'bmw', 'toyota', 'ford']
使用sorted( )由大排到小
排序列表结果 = ['toyota', 'honda', 'ford', 'bmw']
原先列表car内容 = ['honda', 'bmw', 'toyota', 'ford']
目前列表num内容 = [5, 3, 9, 2]
使用sorted( )由大排到小
排序列表结果 = [9, 5, 3, 2]
原先列表num内容 = [5, 3, 9, 2]
>>>
```

6-6 进阶列表操作

6-6-1 index()

这个方法可以返回特定元素内容第一次出现的索引值，它的使用格式如下：

索引值 = 列表名称.index(搜寻值)

如果搜寻值不在列表会出现错误。

程序实例ch6_29.py： 返回搜寻索引值的应用。

```

1 # ch6_29.py
2 cars = ['toyota', 'nissan', 'honda']
3 search_str = 'nissan'
4 i = cars.index(search_str)
5 print("所搜寻元素 %s 第一次出现位置索引是 %d" % (search_str, i))
6 nums = [7, 12, 30, 12, 30, 9, 8]
7 search_val = 30
8 j = nums.index(search_val)
9 print("所搜寻元素 %s 第一次出现位置索引是 %d" % (search_val, j))

```

执行结果

```

===== RESTART: D:\Python\ch6\ch6_29.py =====
所搜寻元素 nissan 第一次出现位置索引是 1
所搜寻元素 30 第一次出现位置索引是 2
>>>

```

程序实例 ch6_30.py： 使用 ch6_13.py 的列表 James，这个列表有 Lebron James 一系列比赛得分，由此列表请计算他在第几场得最高分，同时列出所得分数。

```

1 # ch6_30.py
2 James = ['Lebron James', 23, 19, 22, 31, 18] # 定义James列表
3 games = len(James) # 求元素数量
4 score_Max = max(James[1:games]) # 最高得分
5 i = James.index(score_Max) # 场次
6 print(James[0], "在第 %d 场得最高分 %d" % (i, score_Max))

```

执行结果

```

===== RESTART: D:\Python\ch6\ch6_30.py =====
Lebron James 在第 4 场得最高分 31
>>>

```

这个实例有一点不完美，因为如果有2场或更多场次得到相同分数的最高分，本程序无法处理，下一章笔者将以实例讲解如何修订此缺点。

6-6-2 count()

这个方法可以返回特定元素内容出现的次数，它的使用格式如下：

次数 = 列表名称.count(搜寻值)

如果搜寻值不在列表会出现错误。

程序实例ch6_31.py: 返回搜寻值出现的次数的应用。

```
1 # ch6_31.py
2 cars = ['toyota', 'nissan', 'honda']
3 search_str = 'nissan'
4 num1 = cars.count(search_str)
5 print("所搜寻元素 %s 出现 %d 次" % (search_str, num1))
6 nums = [7, 12, 30, 12, 30, 9, 8]
7 search_val = 30
8 num2 = nums.count(search_val)
9 print("所搜寻元素 %s 出现 %d 次" % (search_val, num2))
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_31.py =====
所搜寻元素 nissan 出现 1 次
所搜寻元素 30 出现 2 次
>>>
```

6-6-3 列表元素的组合join()

这个方法可以将列表的元素组成一个字符串，它的使用格式如下：

`char.join(seq)` # `seq`表示参数必须是列表、元组等序列数据

至于`char`则是组合后各元素间的分隔字符，可以是单一字符，也可以是字符串。

程序实例ch6_31_1.py: 列表元素组合的应用。

```
1 # ch6_31_1.py
2 char = '-'
3 lst = ['Silicon', 'Stone', 'Education']
4 print(char.join(lst))
5 char = '***'
6 lst = ['Silicon', 'Stone', 'Education']
7 print(char.join(lst))
8 char = '\n' # 换行字符
9 lst = ['Silicon', 'Stone', 'Education']
10 print(char.join(lst))
```

执行结果

```
===== RESTART: D:/Python/ch6/ch6_31_1.py =====
Silicon-Stone-Education
Silicon***Stone***Education
Silicon
Stone
Education
>>>
```

6-7 列表内含列表

列表内含列表的基本格式如下：

```
num = [1, 2, 3, 4, 5, [6, 7, 8]]
```

对上述而言，num是一个列表，在这个列表内有另一个列表[7, 8, 9]，因为内部列表的索引值是5，所以可以用num[5]，获得这个元素列表的内容。

```
>>> num = [1, 2, 3, 4, 5, [6, 7, 8]]
>>> num[5]
[6, 7, 8]
>>>
```

如果想要存取列表内的列表元素，可以使用下列格式：

```
num[索引1][索引2]
```

索引1是元素列表原先索引位置，索引2是元素列表内部的索引。

实例1： 列出列表内的列表元素值。

```
>>> num = [1, 2, 3, 4, 5, [6, 7, 8]]
>>> print(num[5][0])
6
>>> print(num[5][1])
7
>>> print(num[5][2])
8
>>>
```

列表内含列表主要应用是，例如，可以用这个资料格式存储NBA球员Lebron James的数据如下所示：

```
James = [['Lebron James', 'SF', '12/30/1984'], 23, 19, 22, 31, 18]
```


其中第一个元素是列表，用于存储Lebron James个人资料，其他则是存储每场得分数据。

程序实例ch6_32.py： 扩充ch6_30.py；先列出Lebron James个人资料；再计算哪一个场次得到最高分。程序第2行，SF全名是Small Forward(小前锋)。

```
1 # ch6_32.py
2 James = [['Lebron James', 'SF', '12/30/84'], 23, 19, 22, 31, 18] # 定义James列表
3 games = len(James) # 求元素数量
4 score_Max = max(James[1:games]) # 最高得分
5 i = James.index(score_Max) # 场次
6 name = James[0][0]
7 position = James[0][1]
8 born = James[0][2]
9 print("姓名      :", name)
10 print("位置      :", position)
11 print("出生日期  :", born)
12 print("在第 %d 场得最高分 %d" % (i, score_Max))
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_32.py =====
姓名      : Lebron James
位置      : SF
出生日期  : 12/30/84
在第 4 场得最高分 31
>>>
```

6-7-1 再谈append()

在6-4-1节我们有提过可以使用append()方法，将元素插入列表的末端，其实也可以使用append()函数将某一列表插入另一列表的末端，方法与插入元素方式相同，这时就会产生列表中有列表的效果。它的使用格式如下：

列表A.append(列表B) # 列表B将接在列表A末端

程序实例ch6_33.py： 使用append()将列表插入另一列表的末端。

```

1 # ch6_33.py
2 cars1 = ['toyota', 'nissan', 'honda']
3 cars2 = ['ford', 'audi']
4 print("原先cars1列表内容 = ", cars1)
5 print("原先cars2列表内容 = ", cars2)
6 cars1.append(cars2)
7 print("执行append( )后列表cars1内容 = ", cars1)
8 print("执行append( )后列表cars2内容 = ", cars2)

```

执行结果

```

===== RESTART: D:\Python\ch6\ch6_33.py =====
原先cars1列表内容 = ['toyota', 'nissan', 'honda']
原先cars2列表内容 = ['ford', 'audi']
执行append( )后列表cars1内容 = ['toyota', 'nissan', 'honda', ['ford', 'audi']]
执行append( )后列表cars2内容 = ['ford', 'audi']
>>>

```

6-7-2 extend()

这也是2个列表连接的方法，与append()类似，不过这个方法只适用2个列表连接，不能用在一般元素。同时在连接后，extend()会将列表分解成元素，一一插入列表。它的使用格式如下：

列表A.extend(列表B) # 列表B将分解成元素插入列表A末端

程序实例ch6_34.py： 使用extend()方法取代ch6_32.py，并观察执行结果。

```

1 # ch6_34.py
2 cars1 = ['toyota', 'nissan', 'honda']
3 cars2 = ['ford', 'audi']
4 print("原先cars1列表内容 = ", cars1)
5 print("原先cars2列表内容 = ", cars2)
6 cars1.extend(cars2)
7 print("执行extend( )后列表cars1内容 = ", cars1)
8 print("执行extend( )后列表cars2内容 = ", cars2)

```

执行结果

```

===== RESTART: D:\Python\ch6\ch6_34.py =====
原先cars1列表内容 = ['toyota', 'nissan', 'honda']
原先cars2列表内容 = ['ford', 'audi']
执行extend( )后列表cars1内容 = ['toyota', 'nissan', 'honda', 'ford', 'audi']
执行extend( )后列表cars2内容 = ['ford', 'audi']
>>>

```

上述执行后cars1将是含有5个元素的列表，每个元素皆是字符串。

6-8 列表的复制

6-8-1 列表的深复制-deep copy

假设我喜欢的运动是，篮球与棒球，可以用下列方式设定列表：

```
mysports = ['basketball', 'baseball']
```

如果我的朋友也喜欢这2种运动，读者可能会想用下列方式设定列表。

```
friendsports = mysports
```

程序实例ch6_35.py： 列出我和朋友所喜欢的运动。

```
1 # ch6_35.py
2 mysports = ['basketball', 'baseball']
3 friendsports = mysports
4 print("我喜欢的运动 = ", mysports)
5 print("我朋友喜欢的运动 = ", friendsports)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_35.py =====
我喜欢的运动 = ['basketball', 'baseball']
我朋友喜欢的运动 = ['basketball', 'baseball']
>>>
```

初看上述执行结果好像没有任何问题，可是如果我想加入football(美式足球)当作喜欢的运动，我的朋友想加入soccer(传统足球)当作喜欢的运动，这时我喜欢的运动如下：

```
basketball、baseball、football
```

我朋友喜欢的运动如下：

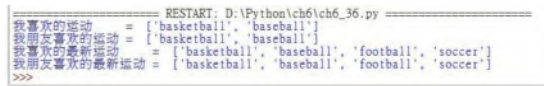
```
basketball、baseball、soccer
```

程序实例ch6_36.py： 继续使用ch6_35.py，加入football(美式足球)当作喜欢的运动，我的朋友想加入soccer(传统足球)当作喜欢的运

动，同时列出执行结果。

```
1 # ch6_36.py
2 mysports = ['basketball', 'baseball']
3 friendsports = mysports
4 print("我喜欢的运动 = ", mysports)
5 print("我朋友喜欢的运动 = ", friendsports)
6 mysports.append('football')
7 friendsports.append('soccer')
8 print("我喜欢的最新运动 = ", mysports)
9 print("我朋友喜欢的最新运动 = ", friendsports)
```

执行结果



```
===== RESTART: D:\Python\ch6\ch6_36.py =====
我喜欢的运动 = ['basketball', 'baseball']
我朋友喜欢的运动 = ['basketball', 'baseball']
我喜欢的最新运动 = ['basketball', 'baseball', 'football', 'soccer']
我朋友喜欢的最新运动 = ['basketball', 'baseball', 'football', 'soccer']
>>>
```

这时获得的结果，不论是我还是我的朋友，喜欢的运动皆相同，football和soccer皆是变成2人共同喜欢的运动。类似这种只要有一个列表更改元素会影响到另一个列表同步更改的复制称深复制(deep copy)。

6-8-2 地址的观念

使用Python可以使用id()函数，获得变量的地址，可参考下列语法。

```
id(x)
```

上述可以获得变量x的地址。对于列表而言，如果使用下列方式设定2个列表变量相等，相当于只是将变量地址复制给另一个变量。

```
friendsports = mysports
```

上述相当于是将mysports变量地址复制给friendsports。所以程序实例ch6_36.py在执行时，2个列表变量所指的地址相同，所以新增运动项目时，皆是将运动项目加在同一变量地址，可参考下列实例。

程序实例ch6_37.py： 重新设计ch6_36.py，增加列出列表变量的地址。

```
1 # ch6_37.py
2 mysports = ['basketball', 'baseball']
3 friendsports = mysports
4 print("列出mysports地址 = ", id(mysports))
5 print("列出friendsports地址 = ", id(friendsports))
6 print("我喜欢的运动 = ", mysports)
7 print("我朋友喜欢的运动 = ", friendsports)
8 mysports.append('football')
9 friendsports.append('soccer')
10 print("-- 新增运动项目后 --")
11 print("列出mysports地址 = ", id(mysports))
12 print("列出friendsports地址 = ", id(friendsports))
13 print("我喜欢的最新运动 = ", mysports)
14 print("我朋友喜欢的最新运动 = ", friendsports)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_37.py =====
列出mysports地址 = 56755464
列出friendsports地址 = 56755464
我喜欢的运动 = ['basketball', 'baseball']
我朋友喜欢的运动 = ['basketball', 'baseball']
-- 新增运动项目后 --
列出mysports地址 = 56755464
列出friendsports地址 = 56755464
我喜欢的最新运动 = ['basketball', 'baseball', 'football', 'soccer']
我朋友喜欢的最新运动 = ['basketball', 'baseball', 'football', 'soccer']
>>>
```

由上述执行结果可以看到，使用程序第3行设定列表变量相等时，实际只是将列表地址复制给另一个列表变量。

6-8-3 列表的浅复制-shallow copy

浅复制(shallow copy) 观念是，执行复制后当一个列表改变后，不会影响另一个列表的内容，这是本小节的重点。方法应该如下：

```
friendsports = mysports[ : ]
```

程序实例ch6_38.py： 使用浅复制方式，重新设计ch6_36.py。下列是与ch6_36.py之间，唯一不同的程序代码。

```
3 friendsports = mysports[:]
```

执行结果

```

===== RESTART: D:\Python\ch6\ch6_38.py =====
列出mysports地址 = 57869576
列出friendsports地址 = 17196016
我喜欢的运动 = ['basketball', 'baseball']
我朋友喜欢的运动 = ['basketball', 'baseball']
-- 新增运动项目后 --
列出mysports地址 = 57869576
列出friendsports地址 = 17196016
我喜欢的最新运动 = ['basketball', 'baseball', 'football']
我朋友喜欢的最新运动 = ['basketball', 'baseball', 'soccer']
>>>

```

由上述执行结果可知，我们已经获得了2个列表彼此是不同的列表地址，同时也得到了想要的结果。

6-9 再谈字符串

3-4节笔者介绍了字符串(string)的观念，在Python的应用中可以将单一字符串当作是一个序列，这个序列是由字符 (character)所组成，可想成字符序列。不过字符串与列表不同的是，字符串内的单一元素内容是不可更改的。

6-9-1 字符串的索引

可以使用索引值的方式取得字符串内容，索引方式则与列表相同。

程序实例ch6_39.py：使用正值与负值的索引列出字符串元素内容。

```

1 # ch6_39.py
2 string = "Python"
3 # 正值索引
4 print(" string[0] = ", string[0],
5       "\n string[1] = ", string[1],
6       "\n string[2] = ", string[2],
7       "\n string[3] = ", string[3],
8       "\n string[4] = ", string[4],
9       "\n string[5] = ", string[5])
10 # 负值索引
11 print(" string[-1] = ", string[-1],
12       "\n string[-2] = ", string[-2],
13       "\n string[-3] = ", string[-3],
14       "\n string[-4] = ", string[-4],
15       "\n string[-5] = ", string[-5],
16       "\n string[-6] = ", string[-6])
17 # 多重指定观念
18 s1, s2, s3, s4, s5, s6 = string
19 print("多重指定观念的输出测试 = ", s1, s2, s3, s4, s5, s6)

```

执行结果

```

===== RESTART: D:\Python\ch6\ch6_39.py =====
string[0] = P
string[1] = y
string[2] = t
string[3] = h
string[4] = o
string[5] = n
string[-1] = n
string[-2] = o
string[-3] = h
string[-4] = t
string[-5] = y
string[-6] = P
多重指定观念的输出测试 = Python
>>>

```

6-9-2 字符串切片

6-1-3节列表切片的观念可以应用在字符串，下列将直接以实例说明。

程序实例ch6_40.py： 字符串切片的应用。

```

1 # ch6_40.py
2 string = "Deep Learning" # 定义字符串
3 print("打印string第1-3元素 = ", string[0:3])
4 print("打印string第2-4元素 = ", string[1:4])
5 print("打印string第2,4,6元素 = ", string[1:6:2])
6 print("打印string第1到最后元素 = ", string[1:])
7 print("打印string前3元素 = ", string[0:3])
8 print("打印string后3元素 = ", string[-3:])

```

执行结果

```

===== RESTART: D:\Python\ch6\ch6_40.py =====
打印string第1-3元素 = Dee
打印string第2-4元素 = eep
打印string第2,4,6元素 = epL
打印string第1到最后元素 = eep Learning
打印string前3元素 = Dee
打印string后3元素 = ing
>>>

```

6-9-3 函数或方法

除了会更改内容的列表函数或方法 不可应用在字符串外，其他则可以用在字符串。

函数	说明
len()	计算字符串长度
max()	最大值
min()	最小值

程序实例ch6_41.py： 将函数len()、max()、min()应用在字符串。

```
1 # ch6_41.py
2 string = "Deep Learning"          # 定义字符串
3 strlen = len(string)
4 print("字符串长度", strlen)
5 maxstr = max(string)
6 print("字符串最大的unicode码值和字符", ord(maxstr), maxstr)
7 minstr = min(string)
8 print("字符串最小的unicode码值和字符", ord(minstr), minstr)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_41.py =====
字符串长度 13
字符串最大的unicode码值和字符 114 r
字符串最小的unicode码值和字符 32
>>>
```

6-9-4 将字符串转成列表

list()函数可以将参数内的对象转成列表，下列是字符串转为列表的实例：

```
>>> x = list('Deep Stone')
>>> print(x)
['D', 'e', 'e', 'p', ' ', 'S', 't', 'o', 'n', 'e']
>>>
```

6-9-5 切片赋值的应用

字符串本身无法用切片方式更改内容，但是将字符串改为列表后，就可以使用切片更改列表内容了，下列是延续6-9-4节的实例。

```
>>> x[5:] = 'Mind'
>>> print(x)
['D', 'e', 'e', 'p', ' ', 'M', 'i', 'n', 'd']
>>>
```

6-9-6 使用split()处理字符串

这个方法(method)，可以将字符串以空格为分隔符，将字符串拆开，变成一个列表。变成列表后我们可以使用len()获得此列表的元

素个数，这相当于可以计算字符串是由多少个英文字母组成，由于中文字之间没有空格，所以本节所述方法只适用 在纯英文文件 。如果我们可以将一篇文章或一本书当做一个字符串变量，可以使用这个方法获得这一篇文章或这一本书的字数。

程序实例ch6_41_1.py： 获得字符串内的字数。

```
1 # ch6_41_1.py
2 str1 = "Silicon Stone Education"
3 str2 = "DeepStone"
4 str3 = "深石数位"
5
6 sList1 = str1.split()          # 字符串转成列表
7 sList2 = str2.split()          # 字符串转成列表
8 sList3 = str3.split()          # 字符串转成列表
9 print(str1, " 列表内容是 ", sList1)    # 打印列表
10 print(str1, " 列表字数是 ", len(sList1)) # 打印字数
11 print(str2, " 列表内容是 ", sList2)    # 打印列表
12 print(str2, " 列表字数是 ", len(sList2)) # 打印字数
13 print(str3, " 列表内容是 ", sList3)    # 打印列表
14 print(str3, " 列表字数是 ", len(sList3)) # 打印字数
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_41_1.py =====
Silicon Stone Education 列表内容是 ['Silicon', 'Stone', 'Education']
Silicon Stone Education 列表字数是 3
DeepStone 列表内容是 ['DeepStone']
DeepStone 列表字数是 1
深石数位 列表内容是 ['深石数位']
深石数位 列表字数是 1
>>>
```

6-10 in和not in表达式

主要是用于判断一个对象是否属于另一个对象，对象可以是字符串(string)、列表(list)、元组(Tuple)（第8章介绍）、字典(Dict)（第9章介绍）。它的语法格式如下：

```
boolean_value = obj1 in obj2          # 对象obj1在对象obj2内会传回True
```

```
boolean_value = obj1 not in obj2      # 对象obj1不在对象obj2内会传回True
```

程序实例ch6_42.py: 请输入字符, 这个程序会判断字符是否在字符串内。

```
1 # ch6_42.py
2 password = 'deepstone'
3 ch = input("请输入字符 = ")
4 print("in表达式")
5 if ch in password:
6     print("输入字符在密码中")
7 else:
8     print("输入字符不在密码中")
9
10 print("not in表达式")
11 if ch not in password:
12     print("输入字符不在密码中")
13 else:
14     print("输入字符在密码中")
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_42.py =====
请输入字符 = d
in表达式
输入字符在密码中
not in表达式
输入字符在密码中
>>>
```

其实这个功能一般更常见是用于检测某个元素是否存在列表中, 如果不存在, 则将它加入列表内, 可参考下列实例。

程序实例ch6_43.py: 这个程序基本上会要求输入一个水果, 如果列表内目前没有这个水果, 就将输入的水果加入列表内。

```
1 # ch6_43.py
2 fruits = ['apple', 'banana', 'watermelon']
3 fruit = input("请输入水果 = ")
4 if fruit in fruits:
5     print("这个水果已经有了")
6 else:
7     fruits.append(fruit)
8     print("谢谢提醒已经加入水果清单: ", fruits)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_43.py =====
请输入水果 = orange
谢谢提醒已经加入水果清单: ['apple', 'banana', 'watermelon', 'orange']
>>>
```

6-11 is或is not表达式

可以用于比较两个对象是否相同，在此所谓相同并不只是内容相同，而是指对象变量指向相同的内存，对象可以是变量、字符串、列表、元组(Tuple) (第8章介绍)、字典(Dict) (第9章介绍)。它的语法格式如下：

```
boolean_value = obj1 is obj2      # 对象obj1等于对象obj2会传回True
```

```
boolean_value = obj1 is not obj2   # 对象obj1不等于对象obj2会传回True
```

6-11-1 整数变量在内存地址的观察

在6-8-2节已经讲解可以使用id()函数获得列表变量地址，其实这个函数也可以获得整数（或浮点数）变量在内存中的地址，当我们在Python程序中设立变量时，如果两个整数（或浮点数）变量内容相同，它们会使用相同的内存地址存储此变量。

程序实例ch6_44.py： 整数变量在内存地址的观察，这个程序比较特别的是，程序执行初，变量x和y值是10，所以可以看到经过id()函数后，彼此有相同的内存位置。变量z和r由于值与x和y不相同，所以有不同的内存地址，经过第9行运算后r的值变为10，最后得到x、y和r不仅值相同，同时也指向相同的内存地址。

```
1 # ch6_44.py
2 x = 10
3 y = 10
4 z = 15
5 r = 20
6 print("x = %d, y = %d, z = %d, r = %d" % (x, y, z, r))
7 print("x地址 = %d, y地址 = %d, z地址 = %d, r地址 = %d"
8     % (id(x), id(y), id(z), id(r)))
9 r = z          # r的值将变为10
10 print("x = %d, y = %d, z = %d, r = %d" % (x, y, z, r))
11 print("x地址 = %d, y地址 = %d, z地址 = %d, r地址 = %d"
12     % (id(x), id(y), id(z), id(r)))
```

执行结果

```

===== RESTART: D:\Python\ch6\ch6_44.py =====
x = 10, y = 10, z = 15, r = 20
x地址 = 1453222160, y地址 = 1453222160, z地址 = 1453222240, r地址 = 1453222320
x = 10, y = 10, z = 15, r = 15
x地址 = 1453222160, y地址 = 1453222160, z地址 = 1453222240, r地址 = 1453222240
>>>

```

当r变量值变为10时，它所指的内存地址与x和y变量相同了。

6-11-2 将is和is not表达式应用在整数变量

程序实例ch6_45.py: is和is not表达式应用在整数变量。

```

1 # ch6_45.py
2 x = 10
3 y = 10
4 z = 15
5 r = z - 5
6 boolean_value = x is y
7 print("x地址 = %d, y地址 = %d" % (id(x), id(y)))
8 print("x = %d, y = %d, " % (x, y), boolean_value)
9
10 boolean_value = x is z
11 print("x地址 = %d, z地址 = %d" % (id(x), id(z)))
12 print("x = %d, z = %d, " % (x, z), boolean_value)
13
14 boolean_value = x is r
15 print("x地址 = %d, r地址 = %d" % (id(x), id(r)))
16 print("x = %d, r = %d, " % (x, r), boolean_value)
17
18 boolean_value = x is not y
19 print("x地址 = %d, y地址 = %d" % (id(x), id(y)))
20 print("x = %d, y = %d, " % (x, y), boolean_value)
21
22 boolean_value = x is not z
23 print("x地址 = %d, z地址 = %d" % (id(x), id(z)))
24 print("x = %d, z = %d, " % (x, z), boolean_value)
25
26 boolean_value = x is not r
27 print("x地址 = %d, r地址 = %d" % (id(x), id(r)))
28 print("x = %d, r = %d, " % (x, r), boolean_value)

```

执行结果

```

===== RESTART: D:\Python\ch6\ch6_45.py =====
x地址 = 1453222160, y地址 = 1453222160
x = 10, y = 10, True
x地址 = 1453222160, z地址 = 1453222240
x = 10, z = 15, False
x地址 = 1453222160, r地址 = 1453222160
x = 10, r = 10, True
x地址 = 1453222160, y地址 = 1453222160
x = 10, y = 10, False
x地址 = 1453222160, z地址 = 1453222240
x = 10, z = 15, True
x地址 = 1453222160, r地址 = 1453222160
x = 10, r = 10, False
>>>

```

6-11-3 将is和is not表达式应用在列表变量

程序实例ch6_46.py： 这个范例所使用的3个列表内容均是相同，但是mysports和sports1所指地址相同所以会被视为相同对象，sports2则指向不同地址所以会被视为不同对象，在使用is指令测试时，不同地址的列表会被视为不同的列表。

```
1 # ch6_46.py
2 mysports = ['basketball', 'baseball']
3 sports1 = mysports          # 复制地址
4 sports2 = mysports[:]       # 复制新串行
5 print("我喜欢的运动 = ", mysports, "地址是 = ", id(mysports))
6 print("运动 1      = ", sports1, "地址是 = ", id(sports1))
7 print("运动 2      = ", sports2, "地址是 = ", id(sports2))
8 boolean_value = mysports is sports1
9 print("我喜欢的运动 is 运动 1      = ", boolean_value)
10
11 boolean_value = mysports is sports2
12 print("我喜欢的运动 is 运动 2      = ", boolean_value)
13
14 boolean_value = mysports is not sports1
15 print("我喜欢的运动 is not 运动 1 = ", boolean_value)
16
17 boolean_value = mysports is not sports2
18 print("我喜欢的运动 is not 运动 2 = ", boolean_value)
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_46.py =====
我喜欢的运动 = ['basketball', 'baseball'] 地址是 = 66061536
运动 1      = ['basketball', 'baseball'] 地址是 = 66061536
运动 2      = ['basketball', 'baseball'] 地址是 = 56910832
我喜欢的运动 is 运动 1      = True
我喜欢的运动 is 运动 2      = False
我喜欢的运动 is not 运动 1 = False
我喜欢的运动 is not 运动 2 = True
>>>
```

6-12 enumerate对象

enumerate()方法可以将iterable类数值的元素 用计数值 与元素 配对方式传回，返回的数据称enumerate对象 。其中iterable类数值可以是列表 (list)、元组 (tuple)(第8章说明)、集合 (set)(第10章说明)等。它的语法格式如下：

```
obj = enumerate(iterable[, start = 0]) # 如果省略start =
设定，默认值是0
```

未来我们可以使用list()将enumerate对象转成列表 ，使用tuple()将enumerate对象转成元组 (第8章说明)。

程序实例ch6_47.py: 将列表数据转成enumerate对象的应用。

```
1 # ch6_47.py
2 drinks = {"coffee", "tea", "wine"}
3 enumerate_drinks = enumerate(drinks)          # 数值初始是0
4 print(enumerate_drinks)                       # 传回enumerate对象所在内存
5 print("下列是输出enumerate对象类型")
6 print(type(enumerate_drinks))                 # 列出对象类型
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_47.py =====
<enumerate object at 0x02E26AA8>
下列是输出enumerate对象类型
<class 'enumerate'>
>>>
```

程序实例 ch6_48.py: 将列表数据转成 enumerate 对象, 再将 enumerate 对象转成列表的实例, start 起始值分别为 0 和 10。

```
1 # ch6_48.py
2 drinks = {"coffee", "tea", "wine"}
3 enumerate_drinks = enumerate(drinks)          # 数值初始是0
4 print("转成列表输出, 初始值是 0 = ", list(enumerate_drinks))
5
6 enumerate_drinks = enumerate(drinks, start = 10) # 数值初始是10
7 print("转成列表输出, 初始值是10 = ", list(enumerate_drinks))
```

执行结果

```
===== RESTART: D:\Python\ch6\ch6_48.py =====
转成列表输出, 初始值是 0 = [(0, 'wine'), (1, 'tea'), (2, 'coffee')]
转成列表输出, 初始值是10 = [(10, 'wine'), (11, 'tea'), (12, 'coffee')]
>>>
```

上述程序第4行的list()函数可以将enumerate对象转成列表, 在7-5节当笔者介绍完循环后, 还将继续使用循环解析enumerate对象。

习题

1. 请用列表同时用英文列出10个心中想去旅游的地方。

(A): 列出这10个地方。

(B): 反向列出这10个地方。

(C): 由小排到大, 同时列出来。

(D)：由大排到小，同时列出来。

(E)：请在第一个位置增加 “Antarctic ”，请在最后位置增加 “Arctic Sea”。

(F)：请在中央位置增加 “Chicago”。

(G)：请分别删除第3和9个元素。

2. 请用中文重新设计上述程序。

3. 请建立一个晚会宴客名单，有3份资料。请做一个选单，每次执行皆会列出目前邀请名单，同时有选单，如果选择1，可以增加一位邀请名单。如果选择2，可以删除一位邀请名单。以目前所学指令，执行程序一次只能调整一次，其他细节可以自行发挥创意。

第7章 循环设计

假设现在笔者要求读者设计一个1加到10的程序，然后打印结果，读者可能用下列方式设计这个程序。

程序实例ch7_1.py: 从1加到10，同时打印结果。

```
1 # ch7_1.py
2 sum = 1+2+3+4+5+6+7+8+9+10
3 print("总和 = ", sum)
```

执行结果

```
===== RESTART: D:\Python\ch7\ch7_1.py =====
总和 = 55
>>>
```

如果现在笔者要求各位从1加到100或1000，此时，若是仍用上述方法设计程序，就显得很不现实。

另一种状况，如果一个数据库列表内含有1000名客户的名字，现在要举办晚宴，所以要打印客户姓名，如果用下列方式设计，将是很不实际的行为。

程序实例ch7_2.py: 一个不完整 且不切实际 的程序。

```
1 # ch7_2.py -- 不完整的程序
2 vipNames = ['James', 'Linda', 'Peter', ... , 'Kevin']
3 print("客户1 = ", vipNames[0])
4 print("客户2 = ", vipNames[1])
5 print("客户3 = ", vipNames[2])
6 ...
7 ...
8 print("客户999 = ", vipNames[999])
```

你的程序可以要写超过1000行，当然碰上这类问题，是不可能用上述方法处理的，不过幸好Python语言提供我们解决这类问题的方式，可以轻松用循环 解决，这也是本章的主题。

7-1 基本for循环

for循环可以让程序将整个对象 内的元素遍历（也可以称迭代），在遍历期间，同时可以纪录或输出每次遍历的状态或称轨迹。for循环基本语法格式如下：

```
for var in 可迭代对象
: # 可迭代对象
```

英文是iterable object
程序代码区块

可迭代对象 (iterable object)可以是列表、元组、字典与集合或range()，在信息科学中迭代 (iteration)可以解释为重复执行，上述语法可以解释为将可迭代对象的元素当作var，重复执行，直到每个元素皆被执行一次，整个循环才会停止。

设计上述程序代码时，必须要留意缩排的问题，可以参考if叙述观念。由于这里笔者介绍列表 (list)，所以读者可以想象这个可迭代对象 (iterable)是列表 (list)，第8章笔者会讲解元组 (Tuple)，第9章会讲解字典 (Dict)，第10章会讲解集合 (Set)。另外，上述for循环的可迭代对象也常是range()函数产生的可迭代对象，将在7-2节说明。

7-1-1 for循环基本运作

例如，一个NBA球队有5位球员，分别是Curry、Jordan、James、Durant、Obama，现在想列出这5位球员，那么使用for循环执行这个工作就很适合。

程序实例ch7_3.py： 列出球员名称。

```
1 # ch7_3.py
2 players = ['Curry', 'Jordan', 'James', 'Durant', 'Obama']
3 for player in players:
4     print(player)
```

执行结果

```
===== RESTART: D:/Python/ch7/ch7_3.py =  
Curry  
Jordan  
James  
Durant  
Obama  
>>>
```

上述程序执行的概念是，当第一次执行下列语句时：

```
for player in players:
```

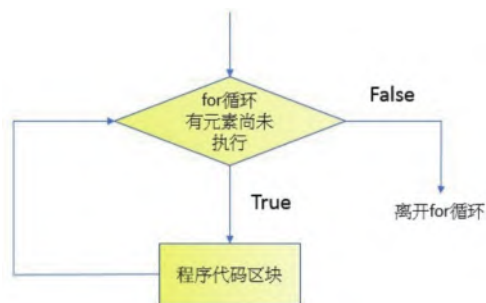
player的内容是‘Curry’，然后执行print(player)，所以会印出‘Curry’。由于列表players内还有其他的元素尚未执行，所以会执行第二次，当执行第二次下列语句时：

```
for player in players:
```

player的内容是‘Jordan’，然后执行print(player)，所以会印出‘Jordan’。由于列表players内还有其他的元素尚未执行，所以会执行第三次，第四次，当执行第五次下列语句时：

```
for player in players:
```

player的内容是‘Obama’，然后执行print(player)，所以会印出‘Obama’。第六次要执行for循环时，由于列表players内所有元素已经执行，所以这个循环就算执行结束。下列是循环的流程示意图。



7-1-2 如果程序代码区块只有一行

使用for循环时，如果程序代码区块只有一行，它的语法格式可以用下列方式表达：

```
for var in 可迭代对象: 程序代码区块
```

程序实例ch7_4.py： 重新设计ch7_3.py。

```
1 # ch7_4.py
2 players = ['Curry', 'Jordan', 'James', 'Durant', 'Obama']
3 for player in players: print(player)
```

执行结果 与ch7_3.py相同。

7-1-3 有多行的程序代码区块

如果for循环的程序代码区块有多行程序时，要留意这些语句同时需要做缩排处理。它的语法格式可以用下列方式表达：

```
for var in 可迭代对象:
    程序代码
    程序代码
    .....
```

程序实例ch7_5.py： 这个程序在设计时，首先笔者将列表的元素英文名字全部改成小写，然后for循环的程序代码区块有2行，这2行（第4和5行）皆需内缩处理，player.title()的title()方法可以处理第一个字母以大写显示。

```
1 # ch7_5.py
2 players = ['curry', 'jordan', 'james', 'durant', 'obama']
3 for player in players:
4     print(player.title() + ", it was a great game.")
5     print("我迫不及待想看下一场比赛，" + player.title())
```

执行结果

```
===== RESTART: D:\Python\ch7\ch7_5.py =
Curry, it was a great game.
我迫不及待想看下一场比赛， Curry
Jordan, it was a great game.
我迫不及待想看下一场比赛， Jordan
James, it was a great game.
我迫不及待想看下一场比赛， James
Durant, it was a great game.
我迫不及待想看下一场比赛， Durant
Obama, it was a great game.
我迫不及待想看下一场比赛， Obama
>>>
```

7-1-4 将for循环应用在列表区间元素

Python也允许将for循环应用在6-1-2节和6-1-3节所截取的区间列表元素上。

程序实例ch7_6.py： 列出列表前3位和后3位的球员名称。

```
1 # ch7_6.py
2 players = ['Curry', 'Jordan', 'James', 'Durant', 'Obama']
3 print("打印前3位球员")
4 for player in players[:3]:
5     print(player)
6 print("打印后3位球员")
7 for player in players[-3:]:
8     print(player)
```

执行结果

```
===== RESTART: D:\Python\ch7\ch7_6.py =====
打印前3位球员
Curry
Jordan
James
打印后3位球员
James
Durant
Obama
>>>
```

这个观念其实很有用，例如，你设计一个学习网站，想要每天列出前3名学生基本数据同时表扬，可以将每个人的学习成果放在列表内，同时用降幂排序方式处理，最后可用本节观念列出前3名学生资料。

注：升幂 是指由小到大排列。降幂 是指由大到小排列。

7-1-5 将for循环应用在数据类别的判断

程序实例ch7_7.py： 有一个files列表内含一系列文件名，请将“.py”的Python程序另外建立到py列表，然后打印。

```
1 # ch7_7.py
2 files = ['da1.c', 'da2.py', 'da3.py', 'da4.java']
3 py = []
4 for file in files:
5     if file.endswith('.py'): # 以.py为扩展名
6         py.append(file)      # 加入列表
7 print(py)
```

执行结果

```
===== RESTART: D:\Python\ch7\ch7_7.py =====
['da2.py', 'da3.py']
>>>
```

程序实例ch7_8.py: 有一个列表names, 元素内容是姓名, 请将姓洪的成员建立在lastname列表内, 然后打印。

```
1 # ch7_8.py
2 names = ['洪锦魁', '洪冰儒', '东霞', '大成']
3 lastname = []
4 for name in names:
5     if name.startswith('洪'): # 是否姓氏洪开头
6         lastname.append(name) # 加入列表
7 print(lastname)
```

执行结果

```
===== RESTART: D:\Python\ch7\ch7_8.py =====
['洪锦魁', '洪冰儒']
>>>
```

7-1-6 删除列表内所有元素

程序实例ch7_9.py: Python没有提供删除整个列表元素的方法, 不过我们可以使用for循环完成此工作。

```
1 # ch7_9.py
2 fruits = ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
3 print("目前fruits列表: ", fruits)
4 i = 1
5 for fruit in fruits[:]:
6     fruits.remove(fruit)
7     print("删除 %s " % fruit)
8     print("目前fruits列表: ", fruits)
```

执行结果

```
===== RESTART: D:\Python\ch7\ch7_9.py =====
目前fruits列表: ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
删除 苹果
目前fruits列表: ['香蕉', '西瓜', '水蜜桃', '百香果']
删除 香蕉
目前fruits列表: ['西瓜', '水蜜桃', '百香果']
删除 西瓜
目前fruits列表: ['水蜜桃', '百香果']
删除 水蜜桃
目前fruits列表: ['百香果']
删除 百香果
目前fruits列表: []
>>>
```

7-2 range()函数

Python可以使用range()函数产生一个等差序列，我们又称这等差序列为可迭代对象(iterable object)，也可以称是range对象。由于range()是产生等差序列，我们可以直接使用，将此等差序列当作循环的计数器。

在前一小节我们使用”for var in 可迭代对象”当作循环，这时会使用可迭代对象元素当作循环指针，如果是要迭代对象内的元素，这是好方法，但是如果只是要执行普通的循环迭代，由于可迭代对象占用内存空间，所以这类循环需要占用较多系统资源。这时我们应该直接使用range()对象，这类迭代只有迭代时的计数指针需要内存，所以可以节省内存空间，range()的用法与列表的切片(slice)类似。

```
range(start  
, stop, step)
```

上述stop是唯一必须的值，等差序列是产生stop的前一个值。例如：如果省略start，所产生等差序列范围是从0至stop-1。step的预设是1，所以预设等差序列是递增1。如果将step设为2，等差序列是递增2。如果将step设为-1，则是产生递减的等差序列。

由range()产生的可迭代等差对象的数据类型是range，可参考下列实例。

```
>>> x = range(3)  
>>> type(x)  
<class 'range'>
```

下列代码用来打印range()对象内容。

```
>>> for x in range(3):  
    print(x)  
  
0  
1  
2
```

```
>>> for x in range(0,3):
    print(x)

0
1
2
```

上述代码执行循环迭代时，即使是执行3个循环，但是系统不用一次预留3个整数空间储存循环计数指针，而是每次循环用1个整数空间储存循环计数指针，所以可以节省系统资源。下列是range()含step参数的应用，第1个是建立1~10的奇数序列，第2个是建立每次递减2的序列。

```
>>> for x in range(1,10,2):
    print(x)

1
3
5
7
9

>>> for x in range(3,-3,-2):
    print(x)

3
1
-1
```

7-2-1 只有一个参数的range()函数的应用

当range(n)函数搭配一个参数时：

range

range(n) # 它将产生0, 1, ..., n-1

的可迭代对象内容

下面来测试range()方法。

程序实例ch7_10.py： 输入数字，本程序会将此数字当作打印星号的数量。

```
1 # ch7_10.py
2 n = int(input("请输入星号数量 : ")) # 定义星号的数量
3 for number in range(n):             # for循环
4     print("*",end="")                # 打印星号
```

执行结果

```
===== RESTART: D:\Python\ch7\ch7_10.py =
请输入星号数量 : 3
***
>>>
===== RESTART: D:\Python\ch7\ch7_10.py =
请输入星号数量 : 10
*****
>>>
```

7-2-2 有2个参数的range()函数

当range()函数搭配2个参数时，它的语法格式如下：

rang

e(start, end)) **# start**是起始值，**end-1**是终止值

上述可以产生start起始值到end-1终止值之间每次递增1的序列，start或end可以是负整数，如果终止值 小于起始值 则产生空序列或称空range对象，可参考下列程序实例。

```
>>> for x in range(10,2):
    print(x)
>>>
```

下列代码是使用负值当作起始值。

```
>>> for x in range(-1,2):
    print(x)

-1
0
1
```

程序实例ch7_11.py： 输入正整数值n，这个程序会计算从0加到n之值。


```

1 # ch7_11.py
2 n = int(input("請輸入n值："))
3 sum = 0
4 for num in range(1,n+1):
5     sum += num
6 print("總和 = ", sum)

```

执行结果

```

===== RESTART: D:/Python/ch7/ch7_11.py ==
請輸入n值：10
總和 = 55
>>>

```

7-2-3 有3个参数的range()函数

当range()函数搭配3个参数时，它的语法格式如下：

rang

e(start, end, step) # start是起始值

, end是终止值

, step是间隔值

然后会从起始值开始产生等差数列，每次间隔step时产生新数值元素，到end-1为止，下列是产生2~11间的偶数。

```

>>> for x in range(2,11,2):
      print(x)

```

```

2
4
6
8
10

```

此外，step值也可以是负值，此时起始值必须大于终止值。

```

>>> for x in range(10,0,-2):
      print(x)

```

```

10
8
6
4
2

```

7-2-4 活用range()应用

程序设计时我们也可以直接应用range()，可以产生程序精简的效果。

程序实例ch7_12.py： 输入一个正整数n，这个程序会输出从1加到n的总和。

```
1 # ch7_12.py
2 n = int(input("请输入整数:"))
3 total = sum(range(n + 1))
4 print("从1到%d的总和是 = " % n, total)
```

执行结果

```
===== RESTART: D:/Python/ch7/ch7_12.py =====
请输入整数:10
从1到10的总和是 = 55
>>>
```

上述程序笔者使用了可迭代对象的内置函数sum执行总和的计算，它的工作原理并不是一次预留储存1, 2, ... 10的内存空间，然后执行运算，而是只有一个内存空间，每次将迭代的指针放在此空间，然后执行sum()运算，可以增加工作效率并节省内存空间。

程序实例ch7_13.py： 建立一个整数平方的列表，为了避免数值太大，若是输入值大于10，此大于10的数值将被设为10。

```
1 # ch7_13.py
2 squares = [] # 建立空列表
3 n = int(input("请输入整数:"))
4 if n > 10 : n = 10 # 最大值是10
5 for num in range(1, n+1):
6     value = num * num # 元素平方
7     squares.append(value) # 加入列表
8 print(squares)
```

执行结果

```
===== RESTART: D:/Python/ch7/ch7_13.py =====
请输入整数:12
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
===== RESTART: D:/Python/ch7/ch7_13.py =====
请输入整数:10
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
===== RESTART: D:/Python/ch7/ch7_13.py =====
请输入整数:5
[1, 4, 9, 16, 25]
>>>
```

对于上述代码而言，下面我们使用“**”代替乘方运算，同时第6和第7行使用更精简的方式。

程序实例ch7_14.py： 用更精简方式设计ch7_13.py。

```
1 # ch7_14.py
2 squares = []           # 建立空列表
3 n = int(input("请输入整数:"))
4 if n > 10 : n = 10     # 最大值是10
5 for num in range(1, n+1):
6     squares.append(num ** 2) # 加入列表
7 print(squares)
```

执行结果 与ch7_13.py相同。

7-2-5 列表生成(list generator)的应用

其实我们还可以进一步简化程序实例ch7_14.py，这个方式是将for循环与加入列表的代码浓缩为一行，对上述程序而言，可以将第5和6行浓缩为一行，在说明实例前先看基本语法。基本语法如下：

新列表

= [表达式

for 项目 in 可迭代对象]

上述语法观念是，将每个可迭代对象套入表达式，每次产生一个列表元素。如果将第5~6行转成上述语法，内容如下：

```
square = [num ** 2 for num in range(1, n+1)]
```

此外，用这种方式设计时，我们可以省略第2行（建立空列表）。

程序实例ch7_15.py： 重新设计ch7_14.py，进阶列表生成的应用。

```
1 # ch7_15.py
2 n = int(input("请输入整数:"))
3 if n > 10 : n = 10     # 最大值是10
4 squares = [num ** 2 for num in range(1, n+1)]
5 print(squares)
```

执行结果 与ch7_14.py相同。

程序实例ch7_16.py： 毕达哥拉斯直角三角形 (A Pythagorean triple)定义，其实这是中学数学的勾股定理，基本观念是直角三角形两边长的平方和等于斜边的平方，如下：

$$a^2 + b^2 = c^2 \quad \# \text{ c是斜边长}$$

这个定理我们可以用 (a, b, c) 方式表达，最著名的实例是 (3, 4, 5)，小括号是数组的表达方式，我们尚未介绍，所以本节使用 [a, b, c] 列表表示。这个程序会生成0~19间符合定义的a、b、c列表值。

```
1 # ch7_16.py
2 x = [[a, b, c] for a in range(1,20) for b in range(a,20) for c in range(b,20)
3     if a**2 + b**2 == c**2]
4 print(x)
```

执行结果

```
===== RESTART: D:/Python/ch7/ch7_16.py =====
[[3, 4, 5], [5, 12, 13], [6, 8, 10], [8, 15, 17], [9, 12, 15]]
>>>
```

程序实例ch7_17.py： 在数学的使用中会碰上下列数学定义。

$$A * B = \{ (a, b) \} : a \text{ 属于A元素, } b \text{ 属于B元素}$$

我们可以用下列程序生成这类的列表。

```
1 # ch7_17.py
2 colors = ["Red", "Green", "Blue"]
3 shapes = ["Circle", "Square", "Line"]
4 result = [[color, shape] for color in colors for shape in shapes]
5 print(result)
```

执行结果

```
===== RESTART: D:/Python/ch7/ch7_17.py =====
[['Red', 'Circle'], ['Red', 'Square'], ['Red', 'Line'], ['Green', 'Circle'], ['Green', 'Square'], ['Green', 'Line'], ['Blue', 'Circle'], ['Blue', 'Square'], ['Blue', 'Line']]
>>>
```

7-2-6 打印含列表元素的列表

这个小节的观念称为list unpacking，这个程序会从每个列表中拉出color和shape的列表元素值。

程序实例ch7_18.py： 简化上一个程序，然后列出列表内每个元素列表值。

```
1 # ch7_18.py
2 colors = ["Red", "Green", "Blue"]
3 shapes = ["Circle", "Square"]
4 result = [[color, shape] for color in colors for shape in shapes]
5 for color, shape in result:
6     print(color, shape)
```

执行结果

```
=====  
Red Circle  
Red Square  
Green Circle  
Green Square  
Blue Circle  
Blue Square  
>>>
```

7-2-7 生成含有条件的列表

这时语法如下：

新列表 = [表达式 for 项目 in 可迭代对象 if 条件式]

下列是用传统方式生成1, 3, ..., 9的列表：

```
>>> for num in range(1,10):
      if num % 2 == 1:
          oddlist.append(num)
```

```
>>> oddlist
[1, 3, 5, 7, 9]
```

下列是使用Python精神，设计含有条件式的列表生成程序。

```
>>> oddlist = [num for num in range(1,10) if num % 2 == 1]
>>> oddlist
[1, 3, 5, 7, 9]
```

7-3 进阶的for循环应用

7-3-1 嵌套for循环

一个循环内有另一个循环，我们称这是**嵌套循环**。如果外循环要执行n次，内循环要执行m次，则整个循环执行的次数是n*m次，设计这类循环时要特别注意下列事项：

外层循环的索引值与内层循环的索引值必须不同。

- 程序代码的内缩一定要小心。
- 下列是嵌套循环基本语法：

```
for 变量 in 对象:           # 外层for循环
```

```
...
    for 变量 in 对象:       # 内层for循环
```

```
...
```

下列将用实例说明。

程序实例ch7_19.py： 打印9*9的乘法表。

```
1 # ch7_19.py
2 for i in range(1, 10):
3     for j in range(1, 10):
4         result = i * j
5         print("%d*d=%-3d" % (i, j, result), end=" ")
6     print()           # 换行输出
```

执行结果

```
===== RESTART: D:/Python/ch7/ch7_19.py =====
1*1=1 1*2=2 1*3=3 1*4=4 1*5=5 1*6=6 1*7=7 1*8=8 1*9=9
2*1=2 2*2=4 2*3=6 2*4=8 2*5=10 2*6=12 2*7=14 2*8=16 2*9=18
3*1=3 3*2=6 3*3=9 3*4=12 3*5=15 3*6=18 3*7=21 3*8=24 3*9=27
4*1=4 4*2=8 4*3=12 4*4=16 4*5=20 4*6=24 4*7=28 4*8=32 4*9=36
5*1=5 5*2=10 5*3=15 5*4=20 5*5=25 5*6=30 5*7=35 5*8=40 5*9=45
6*1=6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36 6*7=42 6*8=48 6*9=54
7*1=7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49 7*8=56 7*9=63
8*1=8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64 8*9=72
9*1=9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
>>>
```

上述程序第5行，%-3d主要是供result使用，表示每一个输出预留3格，同时靠左输出。同一行End=“ ”则是设定，输出完空一格，下次输出不换行输出。当内层循环执行完一次，则执行第6行，这是外层循环叙述，主要是设定下次换行输出，相当于下次再执行内层循环时换行输出。

程序实例ch7_20.py: 绘制直角三角形。

```
1 # ch7_20.py
2 for i in range(1, 10):
3     for j in range(1, 10):
4         if j <= i:
5             print("a", end=" ")
6     print()          # 换行输出
```

执行结果

```
===== RESTART: D:/Python/ch7/ch7_20.py :
a
aa
aaa
aaaa
aaaaa
aaaaaa
aaaaaaa
aaaaaaaa
aaaaaaaaa
aaaaaaaaa
aaaaaaaaa
aaaaaaaaa
aaaaaaaaa
aaaaaaaaa
aaaaaaaaa
aaaaaaaaa
>>>
```

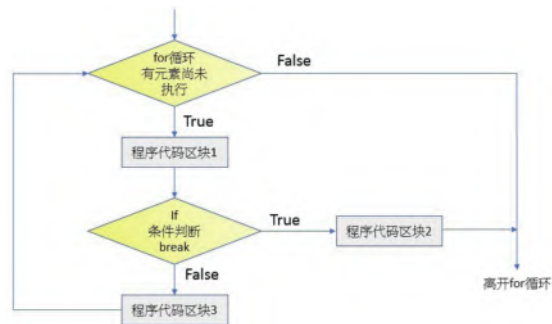
7-3-2 强制离开for循环-break指令

在设计for循环时，如果期待某些条件发生时可以离开循环，可以在循环内执行break指令，即可立即离开循环，这个指令通常是和if语句配合使用。下列是常用的语法格式：

```
for 变量 in 对象:
    程序代码区块 1
    if 条件表达式:          # 判断条件表达式
        程序代码区块 2
        break              # 如果条件表达式是 True 则离开 for 循环
    程序代码区块 3
```

下列是流程图，其中在for循环内的if条件判断，也许前方有程序代码区块1、if条件内有程序代码区块2或是后方有程序代码区块3，只

要if条件判断是True，则执行if条件内的程序代码区块2后，可立即离开循环。



例如，设计一个比赛，可以将参加比赛者的成绩列在列表内，如果想列出前20名参加决赛，可以设定for循环当选取20名后，即离开循环，此时就可以使用break功能。

程序实例ch7_21.py： 输出一系列数字元素，当数字为5时，循环将终止执行。

```
1 # ch7_21.py
2 print("测试1")
3 for digit in range(1, 11):
4     if digit == 5:
5         break
6     print(digit, end=', ')
7 print()
8 print("测试2")
9 for digit in range(0, 11, 2):
10    if digit == 5:
11        break
12    print(digit, end=', ')
```

执行结果

```
===== RESTART: D:\Python\ch7\ch7_21.py =====
测试1
1, 2, 3, 4,
测试2
0, 2, 4, 6, 8, 10,
>>>
```

上述在第一个列表的测试中（第3至6行），当碰到列表元素是5时，循环将终止，所以只有列出“1, 2, 3, 4,”元素。在第二个列表

的测试中（第9至12行），当碰到列表元素是5时，循环将终止，可是这个列表元素中没有5，所以整个循环可以正常执行到结束。

程序实例ch7_22.py： 列出球员名称，列出多少个球员则是由屏幕输入，这个程序同时设定，如果屏幕输入的人数大于列表的球员数时，自动将所输入的人数降为列表的球员数。

```
1 # ch7_22.py
2 players = ['Curry', 'Jordan', 'James', 'Durant', 'Obama', 'Kevin', 'Lin']
3 n = int(input("请输入人数 = "))
4 if n > len(players): n = len(players) # 列出人数不大于列表元素数
5 index = 0 # 索引
6 for player in players:
7     if index == n:
8         break
9     print(player, end=" ")
10    index += 1 # 索引加1
```

执行结果

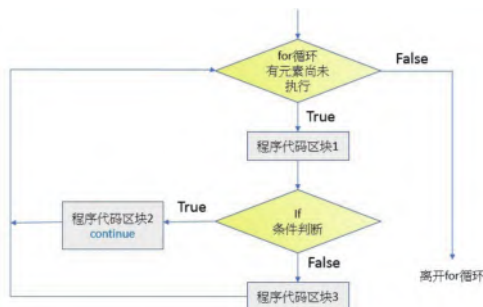
```
===== RESTART: D:\Python\ch7\ch7_22.py =====
请输入人数 = 5
Curry Jordan James Durant Obama
>>>
===== RESTART: D:\Python\ch7\ch7_22.py =====
请输入人数 = 9
Curry Jordan James Durant Obama Kevin Lin
>>>
```

7-3-3 for循环暂时停止不往下执行 - continue指令

在设计for循环时，如果期待某些条件发生时可以不往下执行循环内容，此时可以用continue指令，这个指令通常是和if语句配合使用。下列是常用的语法格式：

```
for 变量 in 对象:
    程序代码区块 1
    if 条件表达式: # 如果条件表达式是 True，则不执行程序代码区块 3
        程序代码区块 2
    continue
    程序代码区块 3
```

下列是流程图，相当于如果发生if条件判断是True时，则不执行程序代码区块3内容。



程序实例ch7_23.py: 有一个列表scores纪录James的比赛得分，设计一个程序可以列出James有多少场次得分大于或等于30分。

```

1 # ch7_23.py
2 scores = [33, 22, 41, 25, 39, 43, 27, 38, 40]
3 games = 0
4 for score in scores:
5     if score < 30:           # 小于30则不往下执行
6         continue
7     games += 1              # 场次加1
8 print("有%d场得分超过30分" % games)

```

执行结果

```

===== RESTART: D:\Python\ch7\ch7_23.py =====
有6场得分超过30分
>>>

```

程序实例ch7_24.py: 有一个列表players，这个列表的元素也是列表，包含球员名字和身高数据，列出所有身高是200(含)公分以上的球员数据。

```

1 # ch7_24.py
2 players = [['James', 202],
3            ['Curry', 193],
4            ['Durant', 205],
5            ['Jordan', 199],
6            ['David', 211]]
7 for player in players:
8     if player[1] < 200:
9         continue
10    print(player)

```

执行结果

```

===== RESTART: D:/Python/ch7/ch7_24.py =====
['James', 202]
['Durant', 205]
['David', 211]
>>>

```

对于上述for循环而言，每次执行第7行时，player的内容是players的一个元素，而这个元素是一个列表，例如：第一次执行时player内容是如下：

```
[ 'James', 202]
```

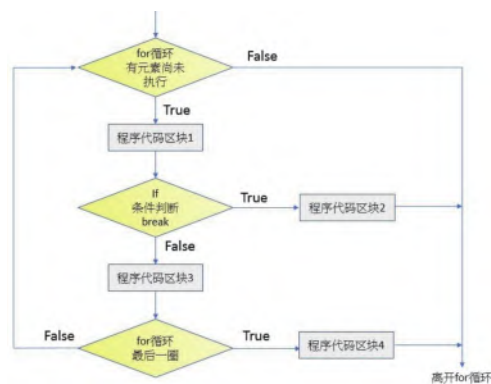
执行第8行时，player[1]的值是202。由于if判断的结果是False，所以会执行第10行的print(player)指令，其他可依次类推。

7-3-4 for ... else循环

在设计for循环时，如果期待所有的if叙述条件是False时，在最后一次循环后，可以执行特定程序区块指令，可使用这个叙述，这个指令通常是和if和break语句配合使用。下列是常用的语法格式：

```
for 变量 in 对象:
    程序代码区块 1
    if 条件表达式:           # 如果条件表达式是 True, 则离开 for 循环
        程序代码区块 2
        break
    程序代码区块 3
else:
    程序代码区块 4           # 最后一次循环条件表达式是 False 则执行
```

下列是流程图，如果最后一次循环if条件表达式仍是False时，才会执行程序代码区块4。



其实这个语法很适合传统数学中测试某一个数字n是否是质数，质数的条件是：

- 2是质数。
- n不可被2至n-1的数字整除。

程序实例ch7_25.py： 质数测试的程序，如果所输入的数字是质数则列出是质数，否则列出不是质数。

```
1 # ch7_25.py
2 num = int(input("请输入大于1的整数做质数测试 = "))
3 if num == 2: # 2是质数所以直接输出
4     print("%d是质数" % num)
5 else:
6     for n in range(2, num): # 用2 .. num-1当除数测试
7         if num % n == 0: # 如果整除则不是质数
8             print("%d不是质数" % num)
9             break # 离开循环
10
11 else: # 否则是质数
12     print("%d是质数" % num)
```

执行结果

```
===== RESTART: D:\Python\ch7\ch7_25.py =====
请输入大于1的整数做质数测试 = 2
2是质数
>>>
===== RESTART: D:\Python\ch7\ch7_25.py =====
请输入大于1的整数做质数测试 = 3
3是质数
>>>
===== RESTART: D:\Python\ch7\ch7_25.py =====
请输入大于1的整数做质数测试 = 12
12不是质数
>>>
===== RESTART: D:\Python\ch7\ch7_25.py =====
请输入大于1的整数做质数测试 = 13
13是质数
>>>
```

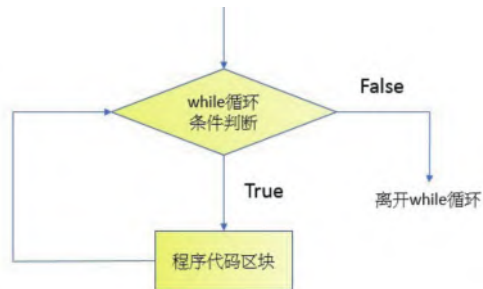
7-4 while循环

这也是一个循环，基本上循环会一直执行直到条件运算为False才会离开循环，所以设计while循环时一定要设计一个条件可以离开循环，相当于让循环结束。程序设计时，如果忘了设计条件可以离开循环，程序造成无限循环状态，此时可以同时按Ctrl+C键，中断程序的执行离开无限循环的陷阱。

一般while循环常应用在不知道循环何时可以结束的状况，for循环在使用时是早已经知道循环即将执行的次数。不过我们也可以透过一些技巧，让while循环也可以应用在已经知道循环即将执行的次数上。它的语法格式如下：

```
while 条件运算:
    程序区块
```

下列是while循环语法流程图。



7-4-1 基本while循环

程序实例ch7_26.py： 这个程序会输出你所输入的内容，当输入q时，程序才会执行结束。

```
1 # ch7_26.py
2 msg1 = '人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!'
3 msg2 = '输入 q 可以结束对话'
4 msg = msg1 + '\n' + msg2 + '\n' + '='
5 input_msg = '' # 默认为空字符串
6 while input_msg != 'q':
7     input_msg = input(msg)
8     print(input_msg)
```

执行结果

```
===== RESTART: D:\Python\ch7\ch7_26.py =====
人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!
输入 q 可以结束对话
= DeepStone深度学习
DeepStone深度学习
人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!
输入 q 可以结束对话
= q
q
>>>
```

上述程序最大的缺点是，当输入q时，程序也将输出q，然后才结束while循环，我们可以使用下列第8行增加if条件判断方式改良。

程序实例ch7_27.py： 改良程序ch7_26.py，当输入q时，不再输出q。

```
1 # ch7_27.py
2 msg1 = '人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!'
3 msg2 = '输入 q 可以结束对话'
4 msg = msg1 + '\n' + msg2 + '\n' + '='
5 input_msg = '' # 默认为空字符串
6 while input_msg != 'q':
7     input_msg = input(msg)
8     if input_msg != 'q': # 如果输入不是q才输出讯息
9         print(input_msg)
```

执行结果

```
===== RESTART: D:\Python\ch7\ch7_27.py =====
人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!
输入 q 可以结束对话
= DeepStone深度学习
DeepStone深度学习
人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!
输入 q 可以结束对话
= q
>>>
```

上述程序尽管可以完成工作，但是当我们在设计大型程序时，如果有更明确的标记记录程序是否继续执行将更佳，下列笔者将用一个布尔变量值active当作标记，如果是True则while循环继续，否则while循环结束。

程序实例ch7_28.py： 改良ch7_27.py程序的可读性，使用标记active记录是否循环继续。

```
1 # ch7_28.py
2 msg1 = '人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!'
3 msg2 = '输入 q 可以结束对话'
4 msg = msg1 + '\n' + msg2 + '\n' + '='
5 active = True
6 while active:          # 循环进行直到active是False
7     input_msg = input(msg)
8     if input_msg != 'q': # 如果输入不是q才输出讯息
9         print(input_msg)
10    else:
11        active = False   # 输入是q所以将active设为False
```

执行结果 与ch7_27.py相同。

程序实例ch7_29.py： 猜数字游戏，程序第2行用变量answer存储欲猜的数字，程序执行时用变量guess存储所猜的数字。

```
1 # ch7_29.py
2 answer = 30          # 正确数字
3 guess = 0            # 设定所猜数字的初始值
4 while guess != answer:
5     guess = int(input("请猜1-100间的数字 = "))
6     if guess > answer:
7         print("请猜小一点")
8     elif guess < answer:
9         print("请猜大一点")
10    else:
11        print("恭喜答对了")
```

执行结果

```
===== RESTART: D:\Python\ch7\ch7_29.py =====
请猜1-100间的数字 = 50
请猜小一点
请猜1-100间的数字 = 25
请猜大一点
请猜1-100间的数字 = 30
恭喜答对了
>>>
```

下列是使用while循环，已经知道要执行多少次循环了的实例。

程序实例ch7_30.py: while循环索引值变化的观察。

```
1 # ch7_30
2 index = 1
3 while index <= 5:
4     print("第 %d 次while循环" % index)
5     index += 1
```

执行结果

```
===== RESTART: D:\Python\ch7\ch7_30.py =====
第 1 次while循环
第 2 次while循环
第 3 次while循环
第 4 次while循环
第 5 次while循环
>>>
```

7-4-2 嵌套while循环

while循环也允许嵌套循环，此时的语法格式如下：

while 条件运算: # 外层while循环

...

 while 条件运算: # 内层while循环

...

程序实例ch7_31.py: 使用while循环重新设计ch7_19.py，打印九九乘法表。

```
1 # ch7_31.py
2 i = 1                   # 设定i初始值
3 while i <= 9:           # 当i大于9跳出外层循环
4     j = 1               # 设定j初始值
5     while j <= 9:       # 当j大于9跳出内层循环
6         result = i * j
7         print("%d*%d=%-3d" % (i, j, result), end=" ")
8         j += 1           # 内层循环加1
9     print()             # 换行输出
10    i += 1               # 外层循环加1
```

执行结果 与ch7_19.py相同。

7-4-3 强制离开while循环-break指令

7-3-2节所介绍的break指令与观念，也可以应用在while循环。在设计while循环时，如果期待某些条件发生时可以离开循环，可以在循环内执行break指令立即离开循环，这个指令通常是和if语句配合使用。下列是常用的语法格式：

```
while 条件表达式 A:
    程序代码区块 1
    if 条件表达式 B:      # 判断条件表达式 B
        程序代码区块 2
        break            # 如果条件表达式 B 是 True, 则离开 while 循环
    程序代码区块 3
```

程序实例ch7_32.py: 这个程序会先建立while无限循环，如果输入q，则可跳出这个while无限循环。程序内容主要是要求输入水果，然后输出此水果。

```
1 # ch7_32.py
2 msg1 = '人机对话专栏,请告诉我妳喜欢吃的水果!'
3 msg2 = '输入 q 可以结束对话'
4 msg = msg1 + '\n' + msg2 + '\n' + '='
5 while True:
6     input_msg = input(msg)
7     if input_msg == 'q':
8         break
9     else:
10        print("我也喜欢吃 %s" % input_msg.title())
```

执行结果

```
===== RESTART: D:\Python\ch7\ch7_32.py
人机对话专栏,请告诉我妳喜欢吃的水果!
输入 q 可以结束对话
= apple
我也喜欢吃 Apple
人机对话专栏,请告诉我妳喜欢吃的水果!
输入 q 可以结束对话
= orange
我也喜欢吃 Orange
人机对话专栏,请告诉我妳喜欢吃的水果!
输入 q 可以结束对话
= q
>>>
```

程序实例ch7_33.py: 使用while循环重新设计ch7_22.py。

```
1 # ch7_33.py
2 players = ['Curry', 'Jordan', 'James', 'Durant', 'Obama', 'Kevin', 'Lin']
3 n = int(input("请输入人数: "))
4 if n > len(players): n = len(players)
5 index = 0
6 while index < len(players):
7     if index == n:
8         break
9     print(players[index], end=" ")
10    index += 1
```


执行结果 与ch7_22.py相同。

上述程序第6行的“index < len(players)”相当于是语法格式的条件表达式A，控制循环是否终止。程序第7行的“index == n”相当于是语法格式的条件表达式B，可以控制是否中途离开while循环。

7-4-4 while循环暂时停止不往下执行-continue指令

在设计while循环时，如果期待某些条件发生时可以不往下执行循环内容，此时可以用continue指令，这个指令通常是和if语句配合使用。下列是常用的语法格式：

```
while 条件运算 A :  
    程序代码区块 1  
    if 条件表达式 B : # 如果条件表达式是 True，则不执行程序代码区块 3  
        程序代码区块 2  
        continue  
    程序代码区块 3
```

程序实例ch7_34.py： 列出1至10之间的偶数。

```
1 # ch7_34.py  
2 index = 0  
3 while index <= 10:  
4     index += 1  
5     if ( index % 2 != 0 ): # 测试是否奇数  
6         continue # 不往下执行  
7     print(index) # 输出偶数
```

执行结果

```
===== RESTART: D:/Python/ch7/ch7_34.py  
2  
4  
6  
8  
10  
>>>
```

7-4-5 while循环条件表达式与对象

while循环的条件表达式也可与对象(列表、元组或字典)配合使用,此时它的语法格式如下:

while 条件表达式: # 与有关的条件表达式

程序区块

程序实例ch7_35.py: 删除列表内的apple字符串,程序第5行,只要在fruits列表内可以找到变量apple,就会传回True,循环将继续。

```
1 # ch7_35.py
2 fruits = ['apple', 'orange', 'apple', 'banana', 'apple']
3 fruit = 'apple'
4 print("删除前的fruits", fruits)
5 while fruit in fruits: # 只要列表内有apple循环就继续
6     fruits.remove(fruit)
7 print("删除后的fruits", fruits)
```

执行结果

```
===== RESTART: D:\Python\ch7\ch7_35.py =====
删除前的fruits ['apple', 'orange', 'apple', 'banana', 'apple']
删除后的fruits ['orange', 'banana']
>>>
```

程序实例ch7_36.py: 有一个列表buyers,此列表内含购买者和消费金额,如果购买金额超过或达到1000元,则归类为VIP买家vipbuyers列表。否则是Gold买家goldbuyers列表。

```
1 # ch7_36.py
2 buyers = [['James', 1030], # 建立买家购买记录
3           ['Curry', 893],
4           ['Durant', 2050],
5           ['Jordan', 990],
6           ['David', 2110]]
7 goldbuyers = [] # Gold买家列表
8 vipbuyers = [] # VIP买家列表
9 while buyers: # 执行买家分类循环分类完成循环才会结束
10     index_buyer = buyers.pop()
11     if index_buyer[1] >= 1000: # 用1000元执行买家分类条件
12         vipbuyers.append(index_buyer) # 加入VIP买家列表
13     else:
14         goldbuyers.append(index_buyer) # 加入Gold买家列表
15 print("VIP 买家资料", vipbuyers)
16 print("Gold买家资料", goldbuyers)
```

执行结果

```
===== RESTART: D:\Python\ch7\ch7_36.py =====
VIP 买家资料 [['David', 2110], ['Durant', 2050], ['James', 1030]]
Gold买家资料 [['Jordan', 990], ['Curry', 893]]
>>>
```

上述程序第9行只要列表不是空列表,while循环就会一直执行。

7-4-6 pass

pass指令是什么事也不做，如果我们想要建立一个无限循环可以使用下列写法。

```
while True:
    pass
```

不过不建议这么做，这会让程序进入无限循环。这个指令有时候会用在设计一个循环或函数（将在第11-8节解说）尚未完成时，先放pass，未来再用完整程序代码取代。

程序实例ch7_37.py: pass应用在循环的实例，这个程序的循环尚未设计完成，所以笔者先用pass处理。

```
1 # ch7_37.py
2 schools = ['明志科大', '台湾科大', '台北科大']
3 for school in schools:
4     pass
```

执行结果 没有任何数据输出。

7-5 enumerate对象使用for循环解析

延续6-11节的enumerate对象可知，这个对象是由计数值 与元素值 配对出现：

计数值 元素值

所以我们可以使用for循环将每一个计数值 与元素值 解析出来。

程序实例ch7_38.py: 继续设计ch6_48.py，将enumerate对象的计数值 与元素值 解析出来。

```

1 # ch7_38.py
2 drinks = {"coffee", "tea", "wine"}
3 # 解析enumerate对象
4 for drink in enumerate(drinks):          # 数值初始是0
5     print(drink)
6 for count, drink in enumerate(drinks):
7     print(count, drink)
8 print("*****")
9 # 解析enumerate对象
10 for drink in enumerate(drinks, 10):     # 数值初始是10
11     print(drink)
12 for count, drink in enumerate(drinks, 10):
13     print(count, drink)

```

执行结果

```

===== RESTART: D:/Python/ch7/ch7_38.py =====
(0, 'tea')
(1, 'coffee')
(2, 'wine')
0 tea
1 coffee
2 wine
*****
(10, 'tea')
(11, 'coffee')
(12, 'wine')
10 tea
11 coffee
12 wine
>>>

```

上述程序第6行观念如下：

第1个变量是计数值
第2个变量是元素值

计数值 元素值

```

for count, drink in enumerate(drinks):
    print(count, drink)

```

由于`enumerate(drinks)`产生的`enumerate`对象是配对存在，可以用2个变量遍历这个对象，只要仍有元素尚未被遍历循环就会继续。

习题

1. 假设你今年体重是50千克，每年可以增加1.2千克，请列出未来10年的体重变化。
2. 请建立一个从1开始到你的年龄的列表，同时打印出来。
3. 有一个水果列表如下：

```
fruits = ['李子', '香蕉', '苹果', '西瓜', '桃子']
```

请用含编号方式列出这些水果。

1. 李子
2. 香蕉
3. 苹果
4. 西瓜
5. 桃子

4. 请重新设计ch7_20.py，但是要得到下列结果。

```
aaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaa
aaaaaaaaa
aaaaaaa
aaaaa
aaa
aa
```

5. 请使用while循环取代for循环，重新设计ch7_24.py。
6. 请重新设计ch7_29.py，增加列出猜多少次才猜对。
7. 请重新设计ch7_30.py，在列出每一次循环时，同时列出累计索引相加的结果。
8. 请重新设计ch7_32.py，输入水果改成输入度假地点，然后输出“我也喜欢这个”度假地点。
9. 请重新设计ch7_36.py，请在buyers列表内自行增加15数据，同时增加若是购买金额达到10000元或以上，归类为infinitebuyers列表。
10. 请分别使用for和while循环执行下列工作，请输入n和m整数值，m值一定大于n值，请列出n加到m的结果。例如，输入n值是1，m值是100，则程序必须列出1加到100的结果是5050。

11. 请建立2个列表分别如下：

```
players = [ 'John' ,    'Peter' ,    'Ivan' ,    'Kevin' ,  
            'Jodan' ]
```

```
teams = [ 'Michael' ,    'Peter' ,    'Curry' ,    'Kevin' ,  
          'Jodan' ]
```

将players列表内元素和teams列表内元素，加入newteam列表，不可有重复名字出现在newteam列表内。

12. 请列出下列数列，其中n值是由屏幕输入。

(a): $1 + 3 + 5 + \cdots + n$ # n请输入奇数

(b): $1 + 2 - 3 + \cdots - (n-1) + n$ # n请输入偶数

(c): $1/n + 2/n + \cdots + n/n$

第8章 元组(Tuple)

在大型的商业或游戏网站设计中，列表(list)是非常重要的数据类型，因为记录各种等级客户、游戏角色等，皆需要使用列表，列表数据可以随时变动更新。Python提供另一种数据类型称元组(tuple)，这种数据类型结构与列表完全相同，但是它与列表最大的差异是，它的元素值与元素个数不可更改，有时又可称不可改变的列表，这也是本章的主题。

8-1 元组的定义

列表在定义时是将元素放在中括号内，元组的定义则是将元素放在小括号“()”内，下列是元组的语法格式。

```
name_tuple = (元素1, ... , 元素n) # name_tuple是假设的元组名称
```

基本上元组的每一个数据称元素，元素可以是整数、字符串或列表等，这些元素放在小括号()内，彼此用逗号“,”隔开。如果要打印元组内容，可以用print()函数，将元组名称当作变量名称即可。

如果元组内的元素只有一个，在定义时需元素右边加上逗号“,”。

```
name_tuple = (元素1,) # 只有一个元素的元组
```

程序实例ch8_1.py：定义与打印元组，最后使用type()列出元组数据类型。

```

1 # ch8_1.py
2 numbers1 = (1, 2, 3, 4, 5)      # 定义元组元素是整数
3 fruits = ('apple', 'orange')   # 定义元组元素是字符串
4 mixed = ('James', 50)          # 定义元组元素是不同类型数据
5 val_tuple = (10,)              # 只有一个元素的元组
6 print(numbers1)
7 print(fruits)
8 print(mixed)
9 print(val_tuple)
10 # 列出元组数据类型
11 print("元组mixed数据类型是: ", type(mixed))

```

执行结果

```

===== RESTART: D:\Python\ch8\ch8_1.py =====
(1, 2, 3, 4, 5)
('apple', 'orange')
('James', 50)
(10,)
元组mixed数据类型是: <class 'tuple'>
>>>

```

8-2 读取元组元素

定义元组时是使用小括号“()”，如果想要读取元组内容，和列表是一样的，用中括号 “[]” 。在Python中元组元素是从索引值0开始配置。所以如果是元组的第一个元素，索引值是0，第二个元素索引值是1，其他依次类推，如下所示：

程序实例ch8_2.py： 读取元组元素的应用。

name_tuple[i] # 读取索引i的元组元素

```

1 # ch8_2.py
2 numbers1 = (1, 2, 3, 4, 5)      # 定义元组元素是整数
3 fruits = ('apple', 'orange')   # 定义元组元素是字符串
4 val_tuple = (10,)              # 只有一个元素的元组
5 print(numbers1[0])              # 以中括号索引值读取元素内容
6 print(numbers1[4])
7 print(fruits[0])
8 print(fruits[1])
9 print(val_tuple[0])

```

执行结果

```

===== RESTART: D:\Python\ch8\ch8_2.py =====
1
5
apple
orange
10
>>>

```

8-3 遍历所有元组元素

在Python可以使用for循环遍历所有元组元素。

程序实例ch8_3.py: 假设元组是由字符串和数值组成的密码，这个程序会列出元组所有元素内容。

```
1 # ch8_3.py
2 keys = ('magic', 'xaab', 9099)      # 定义元组元素是字符串与数字
3 for key in keys:
4     print(key)
```

执行结果

```
===== RESTART: D:/Python/ch8/ch8_3.py =====
magic
xaab
9099
>>>
```

8-4 修改元组内容产生错误的实例

本章前言笔者已经说明元组元素内容是不可更改的，下列是尝试更改元组元素内容的错误实例。

程序实例ch8_4.py: 修改元组内容产生错误的实例。

```
1 # ch8_4.py
2 fruits = ('apple', 'orange')      # 定义元组元素是字符串
3 print(fruits[0])                  # 打印元组fruits[0]
4 fruits[0] = 'watermelon'          # 将元素内容改为watermelon
5 print(fruits[0])                  # 打印元组fruits[0]
```

执行结果 下列是列出错误的画面。

```
===== RESTART: D:\Python\ch8\ch8_4.py =====
apple
Traceback (most recent call last):
  File "D:\Python\ch8\ch8_4.py", line 4, in <module>
    fruits[0] = 'watermelon'      # 将元素内容改为watermelon
TypeError: 'tuple' object does not support item assignment
>>>
```

上述最后一行错误信息TypeError指出tuple对象不支持赋值，相当于不可更改它的元素值。

8-5 可以使用全新定义方式修改元组元素

如果我们想修改元组元素，可以使用重新定义元组方式处理。

程序实例ch8_5.py： 用重新定义方式修改元组元素内容。

```
1 # ch8_5.py
2 fruits = ('apple', 'orange')      # 定义元组元素是水果
3 print("原始fruits元组元素")
4 for fruit in fruits:
5     print(fruit)
6
7 fruits = ('watermelon', 'grape')  # 定义新的元组元素
8 print("\n新的fruits元组元素")
9 for fruit in fruits:
10    print(fruit)
```

执行结果

```
===== RESTART: D:\Python\ch8\ch8_5.py =====
原始fruits元组元素
apple
orange

新的fruits元组元素
watermelon
grape
>>>
```

8-6 元组切片(tuple slices)

元组切片观念与6-1-3节列表切片观念相同，下列将直接用程序实例说明。

程序实例ch8_6.py： 元组切片的应用。

```
1 # ch8_6.py
2 fruits = ('apple', 'orange', 'banana', 'watermelon', 'grape')
3 print(fruits[1:3])
4 print(fruits[:2])
5 print(fruits[1:])
6 print(fruits[-2:])
7 print(fruits[0:5:2])
```

执行结果

```
===== RESTART: D:/Python/ch8/ch8_6.py =====
('orange', 'banana')
('apple', 'orange')
('orange', 'banana', 'watermelon', 'grape')
('watermelon', 'grape')
('apple', 'banana', 'grape')
>>>
```

8-7 方法与函数

应用在列表上的方法 或函数 如果不会更改元组 内容，则可以将它应用在元组，如len()。如果会更改元组内容，则不可以将它应用在元组，如append()、insert()或pop()。

程序实例ch8_7.py： 列出元组元素长度（个数）。

```
1 # ch8_7.py
2 keys = ('magic', 'xaab', 9099)      # 定义元组元素是字符串与数字
3 print("keys元组长度是 %d " % len(keys))
```

执行结果

```
===== RESTART: D:\Python\ch8\ch8_7.py =====
keys元组长度是 3
>>>
```

程序实例ch8_8.py： 误用会减少元组元素的方法pop()，产生错误的实例。

```
1 # ch8_8.py
2 keys = ('magic', 'xaab', 9099)      # 定义元组元素是字符串与数字
3 key = keys.pop()                   # 错误
```

执行结果

```
===== RESTART: D:\Python\ch8\ch8_8.py =====
Traceback (most recent call last):
  File "D:\Python\ch8\ch8_8.py", line 3, in <module>
    key = keys.pop()                  # 错误
AttributeError: 'tuple' object has no attribute 'pop'
>>>
```

上述指出错误是不支持pop()，这是因为pop()将造成元组元素减少。

程序实例ch8_9.py： 误用会增加元组元素的方法append()，产生错误的实例。

```
1 # ch8_9.py
2 keys = ('magic', 'xaab', 9099)      # 定义元组元素是字符串与数字
3 keys.append('secret')               # 错误
```

执行结果

```

===== RESTART: D:\Python\ch8\ch8_9.py =====
Traceback (most recent call last):
  File "D:\Python\ch8\ch8_9.py", line 3, in <module>
    keys.append('secret')      # 错误
AttributeError: 'tuple' object has no attribute 'append'
>>>

```

8-8 列表与元组数据互换

程序设计过程，也许会有需要将列表(list)与元组(tuple)数据类型互换，可以使用下列指令。

`list()`：将元组数据类型改为列表。

`tuple()`：将列表数据类型改为元组。

程序实例ch8_10.py： 重新设计ch8_8.py，将元组改为列表的测试。

```

1 # ch8_10.py
2 keys = ('magic', 'xaab', 9099)      # 定义元组元素是字符串与数字
3 list_keys = list(keys)              # 将元组改为列表
4 list_keys.append('secret')          # 增加元素
5 print("打印元组", keys)
6 print("打印列表", list_keys)

```

执行结果

```

===== RESTART: D:\Python\ch8\ch8_10.py =====
打印元组 ('magic', 'xaab', 9099)
打印列表 ['magic', 'xaab', 9099, 'secret']
>>>

```

上述第4行由于list_keys已经是列表，所以可以使用append()方法。

程序实例ch8_11.py： 将列表改为元组的测试。

```

1 # ch8_11.py
2 keys = ['magic', 'xaab', 9099]      # 定义列表元素是字符串与数字
3 tuple_keys = tuple(keys)            # 将列表改为元组
4 print("打印列表", keys)
5 print("打印元组", tuple_keys)
6 tuple_keys.append('secret')          # 增加元素 --- 错误错误

```

执行结果

```

===== RESTART: D:\Python\ch8\ch8_11.py =====
打印列表 ['magic', 'xaab', 9099]
打印元组 ('magic', 'xaab', 9099)
Traceback (most recent call last):
  File "D:\Python\ch8\ch8_11.py", line 6, in <module>
    tuple_keys.append('secret')      # 增加元素 --- 错误错误
AttributeError: 'tuple' object has no attribute 'append'
>>>

```

上述前5行程序是正确的，所以可以看到有分别打印列表和元组元素，程序第6行的错误是因为tuple_keys是元组，不支持使用append()增加元素。

8-9 其他常用的元组方法

方法	说明
max(tuple)	获得元组内容最大值
min(tuple)	获得元组内容最小值

程序实例ch8_12.py: 元组内建方法max()、min()的应用。

```
1 # ch8_12.py
2 tup = (1, 3, 5, 7, 9)
3 print("tup最大值是", max(tup))
4 print("tup最小值是", min(tup))
```

执行结果

```
===== RESTART: D:/Python/ch8/ch8_12.py =====
tup最大值是 9
tup最小值是 1
>>>
```

8-10 enumerate对象使用在元组

在6-11与7-5节皆已有说明，在此笔者直接以实例解说。

程序实例8_13.py: 将元组转成enumerate对象，再转回元组对象。

```
1 # ch8_13.py
2 drinks = ("coffee", "tea", "wine")
3 enumerate_drinks = enumerate(drinks) # 数值初始是0
4 print("转成元组输出, 初始值是 0 = ", tuple(enumerate_drinks))
5
6 enumerate_drinks = enumerate(drinks, start = 10) # 数值初始是10
7 print("转成元组输出, 初始值是10 = ", tuple(enumerate_drinks))
```

执行结果

```
===== RESTART: D:\Python\ch8\ch8_13.py =====
转成元组输出, 初始值是 0 = ((0, 'coffee'), (1, 'tea'), (2, 'wine'))
转成元组输出, 初始值是10 = ((10, 'coffee'), (11, 'tea'), (12, 'wine'))
>>>
```

程序实例ch8_14.py： 将元组转成enumerate对象，再解析这个enumerate对象。这个程序基本上只是修改ch7_37.py，将列表改为元组。

```
2 drinks = ("coffee", "tea", "wine")
```

执行结果 与ch7_37.py相同。

8-11 zip()

这是一个内置函数，参数内容主要是可迭代(iterable)的对象，如列表等。然后将相对应的元素打包成元组 (tuple)，最后传给zip对象，我们可以使用list()函数将zip对象 转成列表。

程序实例ch8_15.py： zip()的应用。

```
1 # ch8_15.py
2 fields = ['Name', 'Age', 'Hometown']
3 info = ['Peter', '30', 'Chicago']
4 zipData = zip(fields, info) # 执行zip
5 print(type(zipData)) # 打印zip数据类型
6 player = list(zipData) # 将zip数据转成列表
7 print(player) # 打印列表
```

执行结果

```
===== RESTART: D:/Python/ch8/ch8_15.py =====
<class 'zip'>
[('Name', 'Peter'), ('Age', '30'), ('Hometown', 'Chicago')]
>>>
```

如果放在zip()函数的列表 参数长度不相等，由于多出的元素无法匹配，转成列表对象后zip对象 元素数量将是较短的数量。

程序实例ch8_16.py： 重新设计ch8_15.py，fields列表元素数量个数是3个，info列表元素数量个数只有2个，最后zip对象元素数量是2个。

```

1 # ch8_16.py
2 fields = ['Name', 'Age', 'Hometown']
3 info = ['Peter', '30']
4 zipData = zip(fields, info)      # 执行zip
5 print(type(zipData))            # 打印zip数据类型
6 player = list(zipData)          # 将zip数据转成列表
7 print(player)                   # 打印列表

```

执行结果

```

===== RESTART: D:/Python/ch8/ch8_16.py =====
<class 'zip'>
[('Name', 'Peter'), ('Age', '30')]
>>>

```

如果在zip()函数内增加“*”符号，相当于可以unzip()列表。

程序实例ch8_17.py: 扩充设计ch8_15.py，恢复zip前的列表。

```

1 # ch8_17.py
2 fields = ['Name', 'Age', 'Hometown']
3 info = ['Peter', '30', 'Chicago']
4 zipData = zip(fields, info)      # 执行zip
5 print(type(zipData))            # 打印zip数据类型
6 player = list(zipData)          # 将zip数据转成列表
7 print(player)                   # 打印列表
8
9 f, i = zip(*player)              # 执行unzip
10 print("fields = ", f)
11 print("info  = ", i)

```

执行结果

```

===== RESTART: D:/Python/ch8/ch8_17.py =====
<class 'zip'>
[('Name', 'Peter'), ('Age', '30'), ('Hometown', 'Chicago')]
fields = ('Name', 'Age', 'Hometown')
info  = ('Peter', '30', 'Chicago')
>>>

```

8-12 元组的功能

读者也许好奇，元组的数据结构与列表相同，但是元组有不可更改元素内容的限制，为何Python要有类似但功能却受限的数据结构存在？原因是元组有下列优点。



可以更安全地保护数据

程序设计中可能会碰上有些数据是永远不会改变的情况，将它存储在元组(tuple)内，可以安全地被保护。例如，电子邮件的数据结

构，图像处理时对象的长、宽或每一像素的色彩数据，很多都是以元组为数据类型。



增加程序执行速度

元组(tuple)结构比列表(list)简单，占用较少的系统资源，程序执行时速度比较快。

当了解了上述元组的优点后，其实未来设计程序时，如果确定数据可以不更改，就尽量使用元组数据类型吧！

习题

1. 你组织了一个Python的读书小组，这个小组成员有5个人，请将这5个人姓名存储在元组内。
 - (a)：请使用for循环打印这5个人。
 - (b)：请使用重新设定方式，将5个小组成员改为8人。
 - (c)：请尝试修改一个人的名字，然后列出错误所得到的错误信息。

第9章 字典(Dict)

列表(list)与元组(tuple)是依序排列可称是序列 数据结构，只要知道元素的特定位置，即可使用索引 观念取得元素内容。这一章的重点是介绍字典 (dict)，它并不是依序排列的数据结构，通常可称是非序列 数据结构，所以无法使用类似列表的数值(0, 1, ... n)索引观念取得元素内容。

9-1 字典基本操作

9-1-1 定义字典

字典也是一个列表型的数据结构，但是它的元素是用“键-值”方式配对存储，在操作时是用键(key)取得值(value)的内容。定义字典时，是将键-值 放在大括号“{ }”内，字典的语法格式如下：

```
name_dict = { 键1:值1, ... , 键n:值n } # name_dict是字典变量名称
```

字典的值 (value)可以是任何Python的对象，所以可以是数值、字符串、列表等。

程序实例ch9_1.py： 以水果行和面店为例定义一个字典，同时列出字典。下列字典是设定水果一斤的价格、面一碗的价格，最后使用type()列出字典 数据类型。

```

1 # ch9_1.py
2 fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3 noodles = {'牛肉面':100, '肉丝面':80, '阳春面':60}
4 print(fruits)
5 print(noodles)
6 # 列出字典数据类型
7 print("字典fruits数据类型是:", type(fruits))

```

执行结果

```

===== RESTART: D:\Python\ch9\ch9_1.py =====
{'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
{'牛肉面': 100, '肉丝面': 80, '阳春面': 60}
字典fruits数据类型是: <class 'dict'>
>>>

```

在使用Python设计打斗游戏时，玩家通常扮演英雄的角色，敌军可以用字典方式存储，例如，可以用不同颜色的标记设定敌军的小兵，每一个敌军的小兵给予一个分数，这样可以由打死敌军数量再统计游戏得分，可以用下列方式定义字典内容。

程序实例ch9_2.py: 定义soldier0字典tag和score是键，red和3是值。

```

1 # ch9_2.py
2 soldier0 = {'tag':'red', 'score':3}
3 print(soldier0)

```

执行结果

```

===== RESTART: D:/Python/ch9/ch9_2.py =====
{'tag': 'red', 'score': 3}
>>>

```

上述是定义红色(red)小兵，分数是3分，玩家打死红色小兵得3分。

9-1-2 列出字典元素的值

字典的元素是“键-值”配对设定，如果想要取得元素的值，可以将键当作是索引方式处理，因此字典内的元素不可有重复的键，可参考下列实例ch9_3.py的第4行，例如，下列可传回fruits字典 水蜜桃键 的值。

```
fruits['水蜜桃'] # 用字典变量
```

['键']取得值

下列是完整实例。

程序实例ch9_3.py: 分别列出ch9_1.py中水果店水蜜桃一斤的价格和面店牛肉面一碗的价格。

```
1 # ch9_3.py
2 fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3 noodles = {'牛肉面':100, '肉丝面':80, '阳春面':60}
4 print("水蜜桃一斤 = ", fruits['水蜜桃'], "元")
5 print("牛肉面一碗 = ", noodles['牛肉面'], "元")
```

执行结果

```
===== RESTART: D:\Python\ch9\ch9_3.py =====
水蜜桃一斤 = 25 元
牛肉面一碗 = 100 元
>>>
```

程序实例ch9_4.py: 分别列出ch9_2.py小兵字典的tag和score键 的值。

```
1 # ch9_4.py
2 soldier0 = {'tag':'red', 'score':3}
3 print("你刚打死标记 %s 小兵" % soldier0['tag'])
4 print("可以得到 ", soldier0['score'], " 分")
```

执行结果

```
===== RESTART: D:\Python\ch9\ch9_4.py =====
你刚打死标记 red 小兵
可以得到 3 分
>>>
```

9-1-3 增加字典元素

可使用下列语法格式增加字典元素:

```
name_dict[键] = 值 # name_dict是字典变量
```

程序设计ch9_5.py: 为fruits字典增加橘子一斤18元。

```

1 # ch9_5.py
2 fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3 fruits['橘子'] = 18
4 print(fruits)
5 print("橘子一斤 = ", fruits['橘子'], "元")

```

执行结果

```

===== RESTART: D:\Python\ch9\ch9_5.py =====
{'西瓜': 15, '香蕉': 20, '水蜜桃': 25, '橘子': 18}
橘子一斤 = 18 元
>>>

```

在设计打斗游戏时，我们可以使用屏幕坐标标记小兵的位置，下列实例是用xpos/ypos标记小兵的x坐标/y坐标。

程序实例ch9_6.py： 为soldier0字典增加x, y轴坐标 (xpos, ypos) 和移动速度(speed)元素，同时列出结果做验证。

```

1 # ch9_6.py
2 soldier0 = {'tag':'red', 'score':3}
3 soldier0['xpos'] = 100
4 soldier0['ypos'] = 30
5 soldier0['speed'] = 'slow'
6 print("小兵的 x 坐标 = ", soldier0['xpos'])
7 print("小兵的 y 坐标 = ", soldier0['ypos'])
8 print("小兵的移动速度 = ", soldier0['speed'])

```

执行结果

```

===== RESTART: D:\Python\ch9\ch9_6.py =====
小兵的 x 坐标 = 100
小兵的 y 坐标 = 30
小兵的移动速度 = slow
>>>

```

9-1-4 更改字典元素内容

市面上的水果价格是浮动的，如果发生价格异动可以使用本节观念更改。

程序实例ch9_7.py： 将fruits字典的香蕉一斤改成12元。

```

1 # ch9_7.py
2 fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3 print("旧价格香蕉一斤 = ", fruits['香蕉'], "元")
4 fruits['香蕉'] = 12
5 print("新价格香蕉一斤 = ", fruits['香蕉'], "元")

```

执行结果

```
===== RESTART: D:\Python\ch9\ch9_7.py =====
旧价格香蕉一斤 = 20 元
新价格香蕉一斤 = 12 元
>>>
```

在设计打斗游戏时，我们需要时刻移动小兵的位置，此时可以使用本节介绍的方法来更改小兵位置。

程序实例ch9_8.py： 依照soldier字典speed键 的值 移动小兵位置。

```
1 # ch9_8.py
2 soldier0 = {'tag':'red', 'score':3, 'xpos':100,
3             'ypos':30, 'speed':'slow' }
4 print("小兵的 x,y 旧坐标 = ", soldier0['xpos'], ", ", soldier0['ypos'] )
5 if soldier0['speed'] == 'slow':      # 慢
6     x_move = 1
7 elif soldier0['speed'] == 'medium':  # 中
8     x_move = 3
9 else:
10    x_move = 5                        # 快
11 soldier0['xpos'] += x_move
12 print("小兵的 x,y 新坐标 = ", soldier0['xpos'], ", ", soldier0['ypos'] )
```

执行结果

```
===== RESTART: D:\Python\ch9\ch9_8.py =====
小兵的 x,y 旧坐标 = 100 , 100
小兵的 x,y 新坐标 = 101 , 30
>>>
```

上述程序将小兵移动速度分成3个等级，slow是每次xpos移动1单位(5和6行)，medium是每次xpos移动3单位(7和8行)，另一等级则是每次xpos移动5单位(9和10行)。第11行是执行小兵移动，为了简化条件y轴暂不移动。所以可以得到上述小兵x轴位置由100移到101。

9-1-5 删除字典特定元素

如果想要删除字典的特定元素，它的语法格式如下：

```
del name_dict[键] # 可删除特定键
```

的元素

程序实例ch9_9.py： 删除fruits字典的西瓜元素。

```
1 # ch9_9.py
2 fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3 print("旧fruits字典内容:", fruits)
4 del fruits['西瓜']
5 print("新fruits字典内容:", fruits)
```

执行结果

```
===== RESTART: D:\Python\ch9\ch9_9.py =====
旧fruits字典内容: {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
新fruits字典内容: {'香蕉': 20, '水蜜桃': 25}
>>>
```

9-1-6 删除字典所有元素

Python有提供方法clear()可以将字典的所有元素删除，此时字典仍然存在，不过将变成空的字典。

程序实例ch9_10.py： 使用clear()方法删除fruits字典的所有元素。

```
1 # ch9_10.py
2 fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3 print("旧fruits字典内容:", fruits)
4 fruits.clear()
5 print("新fruits字典内容:", fruits)
```

执行结果

```
===== RESTART: D:\Python\ch9\ch9_10.py =====
旧fruits字典内容: {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
新fruits字典内容: {}
>>>
```

9-1-7 删除字典

Python也有提供del指令可以将整个字典删除，字典一经删除就不再存在。它的语法格式如下：

```
del name_dict # 可删除字典name_dict
```

程序实例ch9_11.py： 删除字典的测试，这个程序前4行是没有任何问题，第5行尝试打印已经被删除了的字典，所以产生错误，错误原因是没有定义fruits字典。

```
1 # ch9_11.py
2 fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3 print("旧fruits字典内容:", fruits)
4 del fruits
5 print("新fruits字典内容:", fruits)      # 错误! 错误!
```

执行结果

```
===== RESTART: D:\Python\ch9\ch9_11.py =====
旧fruits字典内容: {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
Traceback (most recent call last):
  File "D:\Python\ch9\ch9_11.py", line 5, in <module>
    print("新fruits字典内容:", fruits)      # 错误! 错误!
NameError: name 'fruits' is not defined
>>>
```

9-1-8 建立一个空字典

在程序设计时，也允许先建立一个空字典，建立空字典的语法如下：

```
name_dict = { }      # name_dict是字典名称
```

上述建立完成后，可以用9-1-3节增加字典元素的方式为空字典建立元素。

程序实例ch9_12.py： 建立一个小兵的空字典，然后为小兵建立元素。

```
1 # ch9_12
2 soldier0 = {}          # 建立空字典
3 print("空小兵字典", soldier0)
4 soldier0['tag'] = 'red'
5 soldier0['score'] = 3
6 print("新小兵字典", soldier0)
```

执行结果

```
===== RESTART: D:\Python\ch9\ch9_12.py =====
空小兵字典 {}
新小兵字典 {'tag': 'red', 'score': 3}
>>>
```

9-1-9 字典的复制

在大型程序开发过程，有时为了要保护原先字典内容，所以常会需要将字典复制，此时可以使用此方法。

```
new_dict = name_dict.copy( ) # name_dict会被复制至new_dict
```

上述所复制的字典是独立存在新地址的字典。

程序实例ch9_13.py: 复制字典的应用，同时列出新字典所在地址，如此可以验证新字典与旧字典是不同的字典。

```
1 # ch9_13.py
2 fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25, '苹果':18}
3 cfruits = fruits.copy( )
4 print("地址 = ", id(fruits), " fruits元素 = ", fruits)
5 print("地址 = ", id(cfruits), " fruits元素 = ", cfruits)
```

执行结果

```
===== RESTART: D:\Python\ch9\ch9_13.py =====
地址 = 58990192 fruits元素 = {'西瓜': 15, '香蕉': 20, '水蜜桃': 25, '苹果': 18}
地址 = 63683680 fruits元素 = {'西瓜': 15, '香蕉': 20, '水蜜桃': 25, '苹果': 18}
>>>
```

9-1-10 取得字典元素数量

在列表(list)或元组(tuple)使用的方法len()也可以应用在字典，它的语法如下：

```
length = len(name_dict) # 将返回name_dict字典的元素数量给length
```


程序实例ch9_14.py: 列出空字典和一般字典的元素数量，本程序第4行由于是建立空字典，所以第7行印出元素数量是0。

```
1 # ch9_14.py
2 fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25, '苹果':18}
3 noodles = {'牛肉面':100, '肉丝面':80, '阳春面':60}
4 empty_dict = {}
5 print("fruits字典元素数量 = ", len(fruits))
6 print("noodles字典元素数量 = ", len(noodles))
7 print("empty_dict字典元素数量 = ", len(empty_dict))
```

执行结果

```
===== RESTART: D:\Python\ch9\ch9_14.py =====
fruits字典元素数量 = 4
noodles字典元素数量 = 3
empty_dict字典元素数量 = 0
>>>
```

9-1-11 验证元素是否存在

可以用下列语法验证元素是否存在。

键 in name_dict # 可验证

键元素是否存在

程序实例ch9_15.py: 这个程序会要求输入键-值，然后判断此元素是否在fruits字典，如果不在此字典则将此键-值 加入字典。

```
1 # ch9_15.py
2 fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3 key = input("请输入键(key) = ")
4 value = input("请输入值(value) = ")
5 if key in fruits:
6     print("%s已经在字典了" % key)
7 else:
8     fruits[key] = value
9     print("新的fruits字典内容 = ", fruits)
```

执行结果

```

===== RESTART: D:\Python\ch9\ch9_15.py =====
请输入键(key) = 西瓜
请输入值(value) = 15
西瓜已经在字典了
>>>

===== RESTART: D:\Python\ch9\ch9_15.py =====
请输入键(key) = 苹果
请输入值(value) = 18
新的fruits字典内容 = {'西瓜': 15, '香蕉': 20, '水蜜桃': 25, '苹果': 18}
>>>

```

9-1-12 设计字典的可读性技巧

设计大型程序时，字典的元素内容很可能是由长字符串所组成，碰上这类情况建议从新的一行开始安置每一个元素，如此可以大大增加字典内容的可读性。例如，有一个players字典，元素是由键（球员名字）-值（球队名称）所组成。如果，我们使用传统方式设计，将让整个字典定义变得很复杂，如下所示：

```

players = {'Stephen Curry': 'Golden State Warriors', 'Kevin Durant': 'Golden State Warriors',
           'Lebron James': 'Cleveland Cavaliers', 'James Harden': 'Houston Rockets', 'Paul Gasol': 'San Antonio Spurs'}

```

碰上这类字典，建议是一行定义一个元素，如下所示：

```

players = {
    'Stephen Curry': 'Golden State Warriors',
    'Kevin Durant': 'Golden State Warriors',
    'Lebron James': 'Cleveland Cavaliers',
    'James Harden': 'Houston Rockets',
    'Paul Gasol': 'San Antonio Spurs'}

```

程序实例ch9_16.py：字典元素是长字符串的应用。

```

1 # ch9_16.py
2 players = {
3     'Stephen Curry': 'Golden State Warriors',
4     'Kevin Durant': 'Golden State Warriors',
5     'Lebron James': 'Cleveland Cavaliers',
6     'James Harden': 'Houston Rockets',
7     'Paul Gasol': 'San Antonio Spurs'}
8 print("Stephen Curry是 %s 的球员" % players['Stephen Curry'])
9 print("Kevin Durant是 %s 的球员" % players['Kevin Durant'])
10 print("Paul Gasol是 %s 的球员" % players['Paul Gasol'])

```

执行结果

```

===== RESTART: D:\Python\ch9\ch9_16.py =====
Stephen Curry是 Golden State Warriors 的球员
Kevin Durant是 Golden State Warriors 的球员
Paul Gasol是 San Antonio Spurs 的球员
>>>

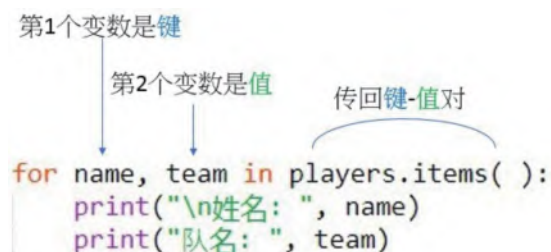
```

9-2 遍历字典

大型程序设计中，字典用久了会产生相当数量的元素，也许是几千个或几十万个或更多。本节将说明如何遍历字典的键-值 对、键或值 。

9-2-1 遍历字典的键-值

Python有提供方法items()，可以让我们取得字典键-值 配对的元素，若是以ch9_16.py的players字典为实例，可以使用for循环加上items()方法，如下所示：



```
for name, team in players.items( ):
    print("\n姓名: ", name)
    print("队名: ", team)
```

上述只要尚未完成遍历字典，for循环将持续进行，如此就可以完成遍历字典，同时传回所有的键-值 。

程序实例ch9_17.py： 列出players字典所有元素，相当于所有球员数据。

```
1 # ch9_17.py
2 players = {'Stephen Curry': 'Golden State Warriors',
3            'Kevin Durant': 'Golden State Warriors',
4            'Lebron James': 'Cleveland Cavaliers',
5            'James Harden': 'Houston Rockets',
6            'Paul Gasol': 'San Antonio Spurs'}
7 for name, team in players.items( ):
8     print("\n姓名: ", name)
9     print("队名: ", team)
```

执行结果

```

===== RESTART: D:\Python\ch9\ch9_17.py =====
姓名: Stephen Curry
队名: Golden State Warriors
姓名: Kevin Durant
队名: Golden State Warriors
姓名: LeBron James
队名: Cleveland Cavaliers
姓名: James Harden
队名: Houston Rockets
姓名: Paul Gasol
队名: San Antonio Spurs
>>>

```

上述实例的执行结果中虽然元素出现顺序与程序第2行到第6行的顺序相同，不过读者须了解Python的直译器并不保证未来一定会保持相同顺序，因为字典(dict)是一个无序的数据结构，Python只会保持键-值，不会关注元素的排列顺序。

9-2-2 遍历字典的键

有时候我们不想要取得字典的值 (value)，只想要键 (keys)，Python有提供方法keys()，可以让我们取得字典的键 内容，若是以ch9_16.py的players字典为实例，可以使用for循环加上keys()方法，如下所示：

```

for name in players.keys():
    print("姓名: ", name)

```

上述for循环会依次将players字典的键 传回。

程序实例ch9_18.py： 列出players字典所有的键 (keys)，此例是所有球员名字。

```

1 # ch9_18.py
2 players = {'Stephen Curry':'Golden State Warriors',
3           'Kevin Durant':'Golden State Warriors',
4           'Lebron James':'Cleveland Cavaliers',
5           'James Harden':'Houston Rockets',
6           'Paul Gasol':'San Antonio Spurs'}
7 for name in players.keys():
8     print("姓名: ", name)

```

执行结果

```
===== RESTART: D:/Python/ch9/ch9_18.py =====
姓名: Stephen Curry
姓名: Kevin Durant
姓名: LeBron James
姓名: James Harden
姓名: Paul Gasol
>>>
```

其实上述实例第7行也可以省略keys()方法，而获得一样的结果，未来各位设计程序是否使用keys()，可自行决定，细节可参考ch9_19.py的第7行。

程序实例ch9_19.py： 重新设计ch9_18.py，此程序省略了keys()方法，但增加一些输出问候语句。

```
1 # ch9_19.py
2 players = {'Stephen Curry':'Golden State Warriors',
3            'Kevin Durant':'Golden State Warriors',
4            'Lebron James':'Cleveland Cavaliers',
5            'James Harden':'Houston Rockets',
6            'Paul Gasol':'San Antonio Spurs'}
7 for name in players:
8     print(name)
9     print("Hi! %s 我喜欢看你在 %s 的表现" % (name, players[name]))
```

执行结果

```
===== RESTART: D:\Python\ch9\ch9_19.py =====
Stephen Curry
Hi! Stephen Curry 我喜欢看你在 Golden State Warriors 的表现
Kevin Durant
Hi! Kevin Durant 我喜欢看你在 Golden State Warriors 的表现
Lebron James
Hi! Lebron James 我喜欢看你在 Cleveland Cavaliers 的表现
James Harden
Hi! James Harden 我喜欢看你在 Houston Rockets 的表现
Paul Gasol
Hi! Paul Gasol 我喜欢看你在 San Antonio Spurs 的表现
>>>
```

9-2-3 排序与遍历字典

Python的字典功能并不会处理排序，如果想要遍历字典同时列出排序结果，可以使用方法sorted()。

程序实例ch9_20.py： 重新设计程序实例ch9_19.py，但是名字将以排序方式列出结果，这个程序的重点是第7行。

```

1 # ch9_20.py
2 players = {'Stephen Curry':'Golden State Warriors',
3            'Kevin Durant':'Golden State Warriors',
4            'Lebron James':'Cleveland Cavaliers',
5            'James Harden':'Houston Rockets',
6            'Paul Gasol':'San Antonio Spurs'}
7 for name in sorted(players.keys( )):
8     print(name)
9     print("Hi! %s 我喜欢看你在 %s 的表现" % (name, players[name]))

```

执行结果

```

===== RESTART: D:\Python\ch9\ch9_20.py =====
James Harden
Hi! James Harden 我喜欢看你在 Houston Rockets 的表现
Kevin Durant
Hi! Kevin Durant 我喜欢看你在 Golden State Warriors 的表现
Lebron James
Hi! Lebron James 我喜欢看你在 Cleveland Cavaliers 的表现
Paul Gasol
Hi! Paul Gasol 我喜欢看你在 San Antonio Spurs 的表现
Stephen Curry
Hi! Stephen Curry 我喜欢看你在 Golden State Warriors 的表现
>>>

```

9-2-4 遍历字典的值

Python有提供方法values()，可以让我们取得字典值 列表，若是以ch9_16.py的players字典为实例，可以使用for循环加上values()方法，如下所示：

程序实例ch9_21.py： 列出players字典的值 列表。

```

1 # ch9_21.py
2 players = {'Stephen Curry':'Golden State Warriors',
3            'Kevin Durant':'Golden State Warriors',
4            'Lebron James':'Cleveland Cavaliers',
5            'James Harden':'Houston Rockets',
6            'Paul Gasol':'San Antonio Spurs'}
7 for team in players.values( ):
8     print(team)

```

执行结果

```

===== RESTART: D:/Python/ch9/ch9_21.py =====
Golden State Warriors
Golden State Warriors
Cleveland Cavaliers
Houston Rockets
San Antonio Spurs
>>>

```

上述Golden State Warriors重复出现，在字典的应用中键 不可有重复，值 是可以重复，如果你希望所列出的值不要重复，可以使用集合(set)观念使用set()函数，例如将第7行改为下列所示即可，这

个实例放在ch9_21_1.py，读者可自行参考。这是下一章的主题，更多细节将在下一章解说。

```
7 for team in set(players.values( )):
```

下列是执行结果，可以发现Golden State Warriors不重复了。

```
===== RESTART: D:/Python/ch9/ch9_21_1.py =====
Houston Rockets
San Antonio Spurs
Golden State Warriors
Cleveland Cavaliers
>>>
```

9-3 建立字典列表

读者可以思考一下程序实例ch9_2.py，我们建立了小兵soldier0字典，在真实的游戏设计中为了让玩家展现雄风，玩家将面对数十、数百或更多个小兵所组成的敌军，为了管理这些小兵，可以将每个小兵当作一个字典，字典内则有小兵的各种信息，然后将这些小兵字典放入列表（list）内。

程序实例ch9_22.py： 建立3个小兵字典，然后将小兵组成列表（list）。

```
1 # ch9_22.py
2 soldier0 = {'tag':'red', 'score':3, 'speed':'slow'}      # 建立小兵
3 soldier1 = {'tag':'blue', 'score':5, 'speed':'medium'}
4 soldier2 = {'tag':'green', 'score':10, 'speed':'fast'}
5 armys = [soldier0, soldier1, soldier2]                  # 小兵组成列表
6 for army in armys:                                       # 打印小兵
7     print(army)
```

执行结果

```
===== RESTART: D:/Python/ch9/ch9_22.py =====
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'blue', 'score': 5, 'speed': 'medium'}
{'tag': 'green', 'score': 10, 'speed': 'fast'}
>>>
```

程序设计中如果每个小兵皆要个别设计这样太没效率了，我们可以使用7-2节的range（）函数处理这类的问题。

程序实例ch9_23.py： 使用range()建立50个小兵， tag是red、score是3、speed是slow。

```
1 # ch9_23.py
2 armys = [] # 建立小兵空列表
3 # 建立50个小兵
4 for soldier_number in range(50):
5     soldier = {'tag': 'red', 'score': 3, 'speed': 'slow'}
6     armys.append(soldier)
7 # 打印前3个小兵
8 for soldier in armys[:3]:
9     print(soldier)
10 # 打印小兵数量
11 print("小兵数量 = ", len(armys))
```

执行结果

```
===== RESTART: D:\Python\ch9\ch9_23.py =====
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
小兵数量 = 50
>>>
```

读者可能会想，上述小兵各种特征皆相同，用处可能不大，其实对Python而言，虽然50个特征相同的小兵放在列表内，但每个小兵皆是独立，可用索引方式存取。通常可以在游戏过程中使用if语句和for循环处理。

程序实例ch9_24.py： 重新设计ch9_23.py，建立50个小兵，但是将编号第36到38名的小兵改成tag是blue、score是5、speed是medium。

```
1 # ch9_24.py
2 armys = [] # 建立小兵空列表
3 # 建立50个小兵
4 for soldier_number in range(50):
5     soldier = {'tag': 'red', 'score': 3, 'speed': 'slow'}
6     armys.append(soldier)
7 # 打印前3个小兵
8 print("前3名小兵资料")
9 for soldier in armys[:3]:
10     print(soldier)
11 # 更改编号36到38的小兵
12 for soldier in armys[35:38]:
13     if soldier['tag'] == 'red':
14         soldier['tag'] = 'blue'
15         soldier['score'] = 5
16         soldier['speed'] = 'medium'
17 # 打印编号35到40的小兵
18 print("打印编号35到40小兵数据")
19 for soldier in armys[34:40]:
20     print(soldier)
```

执行结果


```

===== RESTART: D:\Python\ch9\ch9_24.py
前3名小兵资料
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
打印编号35到40小兵数据
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'blue', 'score': 5, 'speed': 'medium'}
{'tag': 'blue', 'score': 5, 'speed': 'medium'}
{'tag': 'blue', 'score': 5, 'speed': 'medium'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
>>>

```

当然读者可以使用相同方式扩充上述实例，这个将当作习题给读者练习。

9-4 字典内含列表元素

在Python的应用中也允许将列表放在字典内，这时列表 将是字典某键 的值。如果想要遍历这类数据结构，需要使用嵌套循环和字典的方法items()，外层循环 是取得字典的键，内层循环 则是将含列表的值 拆解。下列是定义sports字典的实例：

```

3 sports = {'Curry':['篮球', '美式足球'],
4           'Durant':['棒球'],
5           'James':['美式足球', '棒球', '篮球']}

```

上述sports字典内含3个键-值 配对元素，其中值 的部分皆是列表。程序设计时外层循环配合items()方法，设计如下：

```

7 for name, favorite_sport in sports.items():
8     print("%s 喜欢的运动是：" % name)

```

上述设计后，键 内容会传给name变量，值 内容会传给favorite_sport变量，所以第8行将打印键 内容。内层循环主要是将favorite_sport列表内容拆解，它的设计如下：

```

10 for sport in favorite_sport:
11     print(" ", sport)

```

上述列表内容会随循环传给sport变量，所以第11行可以列出结果。

程序实例ch9_25.py: 字典内含列表元素的应用, 本程序会先定义内含字符串的字典, 然后再拆解打印。

```
1 # ch9_25.py
2 # 建立内含字符串的字典
3 sports = {'Curry':['篮球', '美式足球'],
4           'Durant':['棒球'],
5           'James':['美式足球', '棒球', '篮球']}
6 # 打印key名字 + 字符串'喜欢的运动'
7 for name, favorite_sport in sports.items():
8     print("%s 喜欢的运动是: " % name)
9 # 打印value, 这是列表
10     for sport in favorite_sport:
11         print("    ", sport)
```

执行结果

```
===== RESTART: D:\Python\ch9\ch9_25.py
Curry 喜欢的运动是:
    篮球
    美式足球
Durant 喜欢的运动是:
    棒球
James 喜欢的运动是:
    美式足球
    棒球
    篮球
>>>
```

9-5 字典内含字典

在Python的应用中也允许将字典放在字典内, 这时字典将是字典某键的值。假设微信(wechat_account)账号是用字典存储, 键有2个值是由另外字典组成, 这个内部字典另有3个键, 分别是last_name、first_name和city, 下列是设计实例。

```
3 wechat_account = {'cshung':{
4     'last_name':'洪',
5     'first_name':'锦魁',
6     'city':'台北'},
7     'kevin':{
8     'last_name':'郑',
9     'first_name':'义盟',
10    'city':'北京'}}
```

至于打印方式一样需使用items()函数, 可参考下列实例。

程序实例ch9_26.py: 列出字典内含字典的内容。

```

1 # ch9_26.py
2 # 建立内含字典的字典
3 wechat_account = {'cshung':{
4     'last_name':'洪',
5     'first_name':'锦魁',
6     'city':'台北'},
7     'kevin':{
8     'last_name':'郑',
9     'first_name':'义盟',
10    'city':'北京'}}
11 # 打印内含字典的字典
12 for account, account_info in wechat_account.items():
13     print("使用者账号 = ", account) # 打印键(key)
14     name = account_info['last_name'] + " " + account_info['first_name']
15     print("姓名 = ", name) # 打印值(value)
16     print("城市 = ", account_info['city']) # 打印值(value)

```

执行结果

```

===== RESTART: D:\Python\ch9\ch9_26.py =====
使用者账号 = cshung
姓名 = 洪 锦魁
城市 = 台北
使用者账号 = kevin
姓名 = 郑 义盟
城市 = 北京
>>>

```

9-6 while循环在字典的应用

这一节的内容主要是将while循环应用在字典上。

程序实例ch9_27.py: 这是一个市场梦幻旅游地点调查的实例，此程序会要求输入名字以及梦幻旅游地点，然后存入survey_dict字典，其中键是name，值是travel_location。输入完后程序会询问是否有人要输入，y表示有，n表示没有则程序结束，程序结束前会输出市场调查结果。

```

1 # ch9_27.py
2 survey_dict = {} # 建立市场调查字典
3 market_survey = True # 设定循环布尔值
4
5 # 读取参加市场调查者姓名和梦幻旅游景点
6 while market_survey:
7     name = input("\n请输入姓名: ")
8     travel_location = input("\n梦幻旅游景点: ")
9
10 # 将输入存入survey_dict字典
11     survey_dict[name] = travel_location
12
13 # 可由此决定是否需开市场调查
14     repeat = input("\n是否有人要参加市场调查?(y/n) ")
15     if repeat != 'y': # 不是输入y,则离开while循环
16         market_survey = False
17
18 # 市场调查结束
19 print("\n\n以下是市场调查的结果")
20 for user, location in survey_dict.items():
21     print(user, "梦幻旅游景点: ", location)

```

执行结果

```

===== RESTART: D:\Python\ch9\ch9_27.py =====
请输入姓名 : Peter
梦幻旅游景点: Beijing
是否有人要参加市场调查?(y/n) y

请输入姓名 : Kevin
梦幻旅游景点: Hong Kong
是否有人要参加市场调查?(y/n) n

以下是市场调查的结果
Peter 梦幻旅游景点: Beijing
Kevin 梦幻旅游景点: Hong Kong
>>>

```

有时候设计一个较长的程序时，若是适度空行则整个程序的可读性会更佳，上述笔者分别在第9、12和17行空一行的目的就是如此。

9-7 字典常用的函数和方法

9-7-1 len()

可以列出字典元素的个数。

程序实例ch9_28： 列出字典 以及字典内的字典 元素的个数。

```

1  # ch9_28.py
2  # 建立内含字典的字典
3  wechat_account = {'cshung':{
4      'last_name':'洪',
5      'first_name':'锦魁',
6      'city':'台北'},
7      'kevin':{
8          'last_name':'郑',
9          'first_name':'义盟',
10         'city':'北京'}}
11 # 打印字典元素个数
12 print("wechat_account字典元素个数", len(wechat_account))
13 print("wechat_account['cshung']元素个数", len(wechat_account['cshung']))
14 print("wechat_account['kevin']元素个数", len(wechat_account['kevin']))

```

执行结果

```

===== RESTART: D:\Python\ch9\ch9_28.py =====
wechat_account字典元素个数 2
wechat_account['cshung']元素个数 3
wechat_account['kevin']元素个数 3
>>>

```

9-7-2 fromkeys()

这是建立字典的一个方法，它的语法格式如下：

```
name_dict = dict.fromkeys(seq[, value]) # 使用seq序列建立字典
```

上述会使用seq序列建立字典，序列内容将是字典的键，如果没有设定value则用None当字典键的值。

程序实例ch9_29.py： 分别使用列表和元组建立字典。

```
1 # ch9_29.py
2 # 将列表转成字典
3 seq1 = ['name', 'city'] # 定义列表
4 list_dict1 = dict.fromkeys(seq1)
5 print("字典1 ", list_dict1)
6 list_dict2 = dict.fromkeys(seq1, 'Chicago')
7 print("字典2 ", list_dict2)
8 # 将元组转成字典
9 seq2 = ['name', 'city'] # 定义元组
10 tup_dict1 = dict.fromkeys(seq2)
11 print("字典3 ", tup_dict1)
12 tup_dict2 = dict.fromkeys(seq2, 'New York')
13 print("字典4 ", tup_dict2)
```

执行结果

```
===== RESTART: D:\Python\ch9\ch9_29.py =====
字典1 {'name': None, 'city': None}
字典2 {'name': 'Chicago', 'city': 'Chicago'}
字典3 {'name': None, 'city': None}
字典4 {'name': 'New York', 'city': 'New York'}
>>>
```

9-7-3 get()

搜寻字典的键，如果键存在则传回该键的值，如果不存在则传回默认值。

```
ret_value = dict.get(key[, default=None]) # dict是欲搜寻的字典
```

key是要搜寻的键，如果找不到key则传回default的值（如果没设就传回None）。

程序实例ch9_30.py： get()方法的应用。

```

1 # ch9_30.py
2 fruits = {'Apple':20, 'Orange':25}
3 ret_value1 = fruits.get('Orange')
4 print("Value = ", ret_value1)
5 ret_value2 = fruits.get('Grape')
6 print("Value = ", ret_value2)
7 ret_value3 = fruits.get('Grape', 10)
8 print("Value = ", ret_value3)

```

执行结果

```

===== RESTART: D:\Python\ch9\ch9_30.py =====
Value = 25
Value = None
Value = 10
>>>

```

9-7-4.setdefault()

这个方法基本上与get()相同，不同之处在于get()方法不会改变字典内容。使用setdefault()方法时若所搜寻的键不在，会将键-值加入字典，如果有设定默认值则将键:默认值加入字典，如果没有设定默认值则将键:None加入字典。

ret_value = dict.setdefault(key[, default=None]) # dict是欲搜寻的字典

程序实例ch9_31.py: setdefault()方法，键在字典内的应用。

```

1 # ch9_31.py
2 # key在字典内
3 fruits = {'Apple':20, 'Orange':25}
4 ret_value1 = fruits.setdefault('Orange')
5 print("Value = ", ret_value1)
6 print("fruits字典", fruits)

```

执行结果

```

===== RESTART: D:\Python\ch9\ch9_31.py =====
Value = 25
fruits字典 {'Apple': 20, 'Orange': 25}
>>>

```

程序实例ch9_32.py: setdefault()方法，键不在字典内的应用。


```

1 # ch9_32.py
2 person = {'name': 'John'}
3 print("原先字典内容", person)
4
5 # 'age'键不存在
6 age = person.setdefault('age')
7 print("增加age键 ", person)
8 print("age = ", age)
9
10 # 'sex'键不存在
11 sex = person.setdefault('sex', 'Male')
12 print("增加sex键 ", person)
13 print("sex = ", sex)

```

执行结果

```

===== RESTART: D:\Python\ch9\ch9_32.py =====
原先字典内容 {'name': 'John'}
增加age键 {'name': 'John', 'age': None}
age = None
增加sex键 {'name': 'John', 'age': None, 'sex': 'Male'}
sex = Male
>>>

```

9-7-5 pop()

这个方法可以删除字典元素，它的语法格式如下：

`ret_value = dict.pop(key[, default])` # **dict**是欲删除元素的字典

上述key是要搜寻删除的元素的键，找到时就将该元素从字典内删除，同时将删除键的值回传。当找不到key时则传回default设定的内容，如果没有设定则传回KeyError。

程序实例ch9_33.py： 使用pop()删除元素，同时元素存在的应用。

```

1 # ch9_33.py
2 fruits = {'apple':20, 'banana':15, 'orange':22}
3 ret_value = fruits.pop('orange')
4 print("传回删除元素的值", ret_value)
5 print("删除后的字典内容", fruits)

```

执行结果

```

===== RESTART: D:\Python\ch9\ch9_33.py =====
传回删除元素的值 22
删除后的字典内容 {'apple': 20, 'banana': 15}
>>>

```

程序实例ch9_34.py： 使用pop()删除元素，同时元素不存在的应用。

```
1 # ch9_34.py
2 fruits = {'apple':20, 'banana':15, 'orange':22}
3 ret_value = fruits.pop('grape', 'does not exist')
4 print("传回删除元素的值", ret_value)
5 print("删除后的字典内容", fruits)
```

执行结果

```
===== RESTART: D:\Python\ch9\ch9_34.py =====
传回删除元素的值 does not exist
删除后的字典内容 {'apple': 20, 'banana': 15, 'orange': 22}
>>>
```

习题

1. 将程序实例ch9_4.py的输出结果改成一行。
2. 重新设计ch9_15.py，将程序设计为可以重新输入元素，直到输入是q键程序才结束。
3. 重新设计ch9_24.py，将最后3名小兵改成tag是green、score是10、speed是fast。
4. 请参考ch9_26.py，设计5个旅游地点当键，值 则是由字典组成，内部包含5个键-值，请自行发挥创意，然后打印出来。

第10章 集合(Set)

集合的基本观念是无序且每个元素是唯一的，集合元素的内容是不可变的(immutable)，常见的元素有整数(intger)、浮点数(float)、字符串(string)、元组(tuple)等。至于可变(mutable)内容列表(list)、字典(dict)、集合(set)等不可以是集合元素。但是集合本身是可变的 (mutable)，我们可以增加或删除集合的元素。

10-1 建立集合

Python可以使用大括号“{ }”或set()函数建立集合，下列将分别说明。

10-1-1 使用大括号建立集合

Python允许我们直接使用大括号“{ }”设定集合，例如，集合名称是langs，内容是‘Python’、‘C’、‘Java’。可以使用下列方式设定集合。

程序实例ch10_1.py： 基本集合的建立。

```
1 # ch10_1.py
2 langs = {'Python', 'C', 'Java'}
3 print("打印集合 = ", langs)
4 print("打印类别 = ", type(langs))
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_1.py
打印集合 = {'Python', 'C', 'Java'}
打印类别 = <class 'set'>
>>>
```

集合的特色是元素是唯一的，所以如果设定集合时有重复元素情形，多的部分将被舍去。

程序实例ch10_2.py: 基本集合的建立，建立时部分元素重复，观察执行结果。

```
1 # ch10_2.py
2 langs = {'Python', 'C', 'Java', 'Python', 'C'}
3 print(langs)
```

执行结果

```
===== RESTART: D:/Python/ch10/ch10_2.py =====
{'Python', 'C', 'Java'}
>>>
```

上述‘Python’和‘C’在设定时皆出现2次，但是列出时有重复的元素将只保留1份。集合内容可以是由不同数据类型组成，可参考下列实例。

程序实例ch10_3.py: 使用整数和不同数据类型所建的集合。

```
1 # ch10_3.py
2 # 集合由整数所组成
3 integer_set = {1, 2, 3, 4, 5}
4 print(integer_set)
5 # 集合由不同数据类型所组成
6 mixed_set = {1, 'Python', (2, 5, 10)}
7 print(mixed_set)
8 # 集合的元素是不可变的所以程序第6行所设定的元组元素改成
9 # 第10行列表的写法将会产生错误
10 # mixed_set = { 1, 'Python', [2, 5, 10]}
```

执行结果

```
===== RESTART: D:/Python/ch10/ch10_3.py =====
{1, 2, 3, 4, 5}
{1, (2, 5, 10), 'Python'}
>>>
```

读者可以将第10行的“#”删除，会发现程序有错误产生，原因是[2, 5, 10]是列表，这是可变的元素所以不可以当作集合元素。

读者可能会思考，字典 是用大括号定义，集合 也是用大括号定义，可否直接使用空 的大括号定义空集合？可参考下列实例。

程序实例ch10_4.py： 建立空集合并观察执行结果，发现错误的实例。

```
1 # ch10_4.py
2 x = {} # 这是建立空字典非空集合
3 print("打印 = ", x)
4 print("打印类别 = ", type(x))
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_4.py =====
打印 = {}
打印类别 = <class 'dict'>
>>>
```

结果发现使用空的大括号{ }定义，获得的是空字典，下一小节笔者将会讲解定义空字典的方法。

10-1-2 使用set()函数定义集合

除了以10-1-1节方式建立集合，也可以使用内置的set()函数建立集合，set()函数参数的内容可以是字符串(string)、列表(list)、元组(tuple)等。这时原先字符串(string)、列表(list)、元组(tuple)的元素将被转成集合元素。首先笔者回到建立空集合的主题，如果想建立空集合需使用set()函数。

程序实例ch10_5.py： 重新设计ch10_4.py，使用set()函数建立空集合。

```
1 # ch10_5.py
2 empty_dict = {} # 这是建立空字典
3 print("打印类别 = ", type(empty_dict))
4 empty_set = set() # 这是建立空集合
5 print("打印类别 = ", type(empty_set))
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_5.py =====
打印类别 = <class 'dict'>
打印类别 = <class 'set'>
>>>
```

程序实例ch10_6.py: 使用字符串(string)建立与打印集合, 同时列出集合的数据类型。

```
1 # ch10_6.py
2 x = set('DeepStone mean Deep Learning')
3 print(x)
4 print(type(x))
```

执行结果

```
===== RESTART: D:/Python/ch10/ch10_6.py =====
{'t', ' ', 'S', 'n', 'e', 'a', 'L', 'm', 'D', 'o', 'g', 'p', 'i', 'r'}
<class 'set'>
>>>
```

由于集合元素具有唯一的特性, 所以虽然程序第2行原先字符串有许多字母(例如, e)重复, 经过set()处理后, 所有英文字母将没有重复。

程序实例ch10_7.py: 使用列表(list)建立与打印集合。

```
1 # ch10_7.py
2 # 表达方式1
3 fruits = ['apple', 'orange', 'apple', 'banana', 'orange']
4 x = set(fruits)
5 print(x)
6 # 表达方式2
7 y = set(['apple', 'orange', 'apple', 'banana', 'orange'])
8 print(y)
```

执行结果

```
===== RESTART: D:/Python/ch10/ch10_7.py =====
{'banana', 'orange', 'apple'}
{'banana', 'orange', 'apple'}
>>>
```

读者需留意2种不同的set()函数使用方式, 同时原先列表内容已经变为集合元素内容了。

程序实例ch10_8.py: 使用元组(tuple)建立与打印集合。

```
1 # ch10_8.py
2 cities = set(('Beijing', 'Tokyo', 'Beijing', 'Taipei', 'Tokyo'))
3 print(cities)
```

执行结果

```
===== RESTART: D:/Python/ch10/ch10_8.py =====
{'Beijing', 'Tokyo', 'Taipei'}
>>>
```

10-1-3 大数据与集合的应用

笔者的朋友在某知名企业工作，收集了海量数据使用列表保存，这里面有些数据重复出现，他曾经询问笔者应如何将重复的数据删除，笔者告知如果使用C语言可能需花几小时解决，但是如果了解Python的集合观念，只要花约1分钟就解决了。其实只要将列表数据使用set()函数转为集合数据，再使用list()函数将集合数据转为列表数据就可以了。

程序实例ch10_9.py： 将列表内重复性的数据删除。

```
1 # ch10_9.py
2 fruits1 = ['apple', 'orange', 'apple', 'banana', 'orange']
3 x = set(fruits1)           # 将列表转成集合
4 fruits2 = list(x)          # 将集合转成列表
5 print("原先列表数据fruits1 = ", fruits1)
6 print("新的列表数据fruits2 = ", fruits2)
```

执行结果

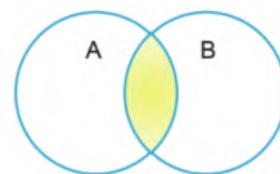
```
===== RESTART: D:\Python\ch10\ch10_9.py =====
原先列表数据fruits1 = ['apple', 'orange', 'apple', 'banana', 'orange']
新的列表数据fruits2 = ['orange', 'banana', 'apple']
>>>
```

10-2 集合的操作

Python 符号	说明
&	交集
	并集
-	差集
^	对称差集
==	等于
!=	不等于
in	是成员
not in	不是成员

10-2-1 交集(intersection)

有A和B两个集合，如果想获得相同的元素，则可以使用交集。例如，有数学(可想成A集合)与物理(可想成B集合)2个夏令营，如果想统计有哪些人同时参加这2个夏令营，则可以使用此功能。



在Python语言的交集符号是“&”，另外，也可以使用intersection()方法完成这个工作。

程序实例ch10_10.py：有数学与物理2个夏令营，这个程序会列出同时参加这2个夏令营的成员。

```
1 # ch10_10.py
2 math = {'Kevin', 'Peter', 'Eric'}      # 设定参加数学夏令营成员
3 physics = {'Peter', 'Nelson', 'Tom'}    # 设定参加物理夏令营成员
4 both = math & physics
5 print("同时参加数学与物理夏令营的成员 ", both)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_10.py =====
同时参加数学与物理夏令营的成员 {'Peter'}
>>>
```

程序实例ch10_11.py：使用intersection()方法完成交集的应用。

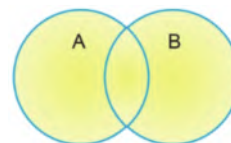
```
1 # ch10_11.py
2 A = {1, 2, 3, 4, 5}      # 定义集合A
3 B = {3, 4, 5, 6, 7}      # 定义集合B
4 # 将intersection()应用在A集合
5 AB = A.intersection(B)   # A和B的交集
6 print("A和B的交集是 ", AB)
7 # 将intersection()应用在B集合
8 BA = B.intersection(A)   # B和A的交集
9 print("B和A的交集是 ", BA)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_11.py =====
A和B的交集是 {3, 4, 5}
B和A的交集是 {3, 4, 5}
>>>
```

10-2-2 并集(union)

有A和B两个集合，如果想获得所有的元素，则可以使用并集。例如，有数学(可想成A集合)与物理(可想成B集合)2个夏令营，如果想统计参加这2个夏令营的全部成员，则可以使用此功能。



在Python语言的并集符号是“|”，另外，也可以使用union()方法完成这个工作。

程序实例ch10_12.py：有数学与物理2个夏令营，这个程序会列出参加这2个夏令营的所有成员。

```
1 # ch10_12.py
2 math = {'Kevin', 'Peter', 'Eric'}      # 设定参加数学夏令营成员
3 physics = {'Peter', 'Nelson', 'Tom'}    # 设定参加物理夏令营成员
4 allmember = math | physics
5 print("参加数学或物理夏令营的全部成员 ",allmember)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_12.py =====
参加数学或物理夏令营的全部成员 {'Nelson', 'Tom', 'Peter', 'Eric', 'Kevin'}
>>>
```

程序实例ch10_13.py：使用union()方法完成并集的应用。

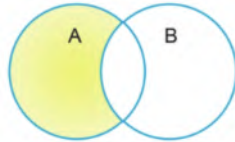
```
1 # ch10_13.py
2 A = {1, 2, 3, 4, 5}                # 定义集合A
3 B = {3, 4, 5, 6, 7}                # 定义集合B
4 # 将union()应用在A集合
5 AorB = A.union(B)                  # A和B的并集
6 print("A和B的并集是 ", AorB)
7 # 将union()应用在B集合
8 BorA = B.union(A)                  # B和A的并集
9 print("B和A的并集是 ", BorA)
```

执行结果

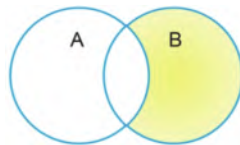
```
===== RESTART: D:\Python\ch10\ch10_13.py =====
A和B的并集是 {1, 2, 3, 4, 5, 6, 7}
B和A的并集是 {1, 2, 3, 4, 5, 6, 7}
>>>
```

10-2-3 差集(difference)

有A和B两个集合，如果想获得属于A集合元素，同时不属于B集合则可以使用差集($A-B$)。如果想获得属于B集合元素，同时不属于A集合则可以使用差集($B-A$)。例如，有数学(可想成A集合)与物理(可想成B集合)2个夏令营，如果想了解参加数学夏令营但是没有参加物理夏令营的成员，则可以使用此功能。



如果想统计参加物理夏令营但是没有参加数学夏令营的成员，也可以使用此功能。



在Python语言的差集符号是“-”，另外，也可以使用difference()方法完成这个工作。

程序实例ch10_14.py：有数学与物理2个夏令营，这个程序会列出参加数学夏令营但是没有参加物理夏令营的所有成员。另外也会列出参加物理夏令营但是没有参加数学夏令营的所有成员。

```
1 # ch10_14.py
2 math = {'Kevin', 'Peter', 'Eric'} # 设定参加数学夏令营成员
3 physics = {'Peter', 'Nelson', 'Tom'} # 设定参加物理夏令营成员
4 math_only = math - physics
5 print("参加数学夏令营同时没有参加物理夏令营的成员 ",math_only)
6 physics_only = physics - math
7 print("参加物理夏令营同时没有参加数学夏令营的成员 ",physics_only)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_14.py =====
参加数学夏令营同时没有参加物理夏令营的成员 {'Kevin', 'Eric'}
参加物理夏令营同时没有参加数学夏令营的成员 {'Nelson', 'Tom'}
>>>
```


程序实例ch10_15.py: 使用difference()方法完成A-B差集与B-A差集的应用。

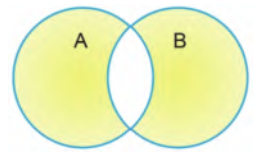
```
1 # ch10_15.py
2 A = {1, 2, 3, 4, 5}      # 定义集合A
3 B = {3, 4, 5, 6, 7}      # 定义集合B
4 # 将difference()应用在A集合
5 A_B = A.difference(B)     # A-B的差集
6 print("A-B的差集是 ", A_B)
7 # 将difference()应用在B集合
8 B_A = B.difference(A)     # B-A的差集
9 print("B-A的差集是 ", B_A)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_15.py
A-B的差集是 {1, 2}
B-A的差集是 {6, 7}
>>>
```

10-2-4 对称差集(symmetric difference)

有A和B两个集合，如果想获得属于A或是B集合元素，但是排除同时属于A和B的元素，则可以使用对称差集。例如，有数学(可想成A集合)与物理(可想成B集合)2个夏令营，如果想统计参加数学夏令营或是参加物理夏令营的成员，但是排除同时参加这2个夏令营的成员，则可以使用此功能。更简单的解释是只参加一个夏令营的成员。



在Python语言的对称差集符号是“^”，另外，也可以使用symmetric_difference()方法完成这个工作。

程序实例ch10_16.py: 有数学与物理2个夏令营，这个程序会列出参加数学夏令营或参加物理夏令营，但是排除同时参加2个夏令营的所有成员。

```
1 # ch10_16.py
2 math = {'Kevin', 'Peter', 'Eric'}      # 设定参加数学夏令营成员
3 physics = {'Peter', 'Nelson', 'Tom'}    # 设定参加物理夏令营成员
4 math_sydi_physics = math ^ physics
5 print("没有同时参加数学和物理夏令营的成员 ", math_sydi_physics)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_16.py =====  
没有同时参加数学和物理夏令营的成员 {'Tom', 'Kevin', 'Eric', 'Nelson'}  
>>>
```

程序实例ch10_17.py: 使用symmetric_difference()方法完成A和B与B和A对称差集 的应用。

```
1 # ch10_17.py  
2 A = {1, 2, 3, 4, 5}           # 定义集合A  
3 B = {3, 4, 5, 6, 7}         # 定义集合B  
4 # 将symmetric_difference( )应用在A集合  
5 A_sydi_B = A.symmetric_difference(B) # A和B的对称差集  
6 print("A和B的对称差集是 ", A_sydi_B)  
7 # 将symmetric_difference( )应用在B集合  
8 B_sydi_A = B.symmetric_difference(A) # B和A的对称差集  
9 print("B和A的对称差集是 ", B_sydi_A)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_17.py =====  
A和B的对称差集是 {1, 2, 6, 7}  
B和A的对称差集是 {6, 7}  
>>>
```

10-2-5 等于

等于的Python符号是“==”，可以得知2个集合是否相等，如果相等传回True，否则传回False。

程序实例ch10_18.py: 测试2个集合是否相等。

```
1 # ch10_18.py  
2 A = {1, 2, 3, 4, 5}           # 定义集合A  
3 B = {3, 4, 5, 6, 7}         # 定义集合B  
4 C = {1, 2, 3, 4, 5}         # 定义集合C  
5 # 列出A与B集合是否相等  
6 print("A与B集合相等", A == B)  
7 # 列出A与C集合是否相等  
8 print("A与C集合相等", A == C)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_18.py =====  
A与B集合相等 False  
A与C集合相等 True  
>>>
```

10-2-6 不等于

不等于的Python符号是“!=”，可以得知2个集合是否不相等，如果不相等传回True，否则传回False。

程序实例ch10_19.py： 测试2个集合是否不相等。

```
1 # ch10_18.py
2 A = {1, 2, 3, 4, 5}           # 定义集合A
3 B = {3, 4, 5, 6, 7}         # 定义集合B
4 C = {1, 2, 3, 4, 5}         # 定义集合C
5 # 列出A与B集合是否相等
6 print("A与B集合相等", A != B)
7 # 列出A与C集合是否相等
8 print("A与C集合相等", A != C)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_19.py
A与B集合相等 False
A与C集合相等 True
>>>
```

10-2-7 是成员in

Python的关键词in可以测试元素是否是集合的元素成员。

程序实例ch10_20.py： 关键词in的应用。

```
1 # ch10_20.py
2 # 方法1
3 fruits = set("orange")
4 print("字符a是否属于fruits集合?", 'a' in fruits)
5 print("字符d是否属于fruits集合?", 'd' in fruits)
6 # 方法2
7 cars = {"Nissan", "Toyota", "Ford"}
8 boolean = "Ford" in cars
9 print("Ford in cars", boolean)
10 boolean = "Audi" in cars
11 print("Audi in cars", boolean)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_20.py
字符a是否属于fruits集合? True
字符d是否属于fruits集合? False
Ford in cars True
Audi in cars False
>>>
```

10-2-8 不是成员not in

Python的关键词not in可以测试元素是否不是集合的元素成员。

程序实例ch10_21.py： 关键词notin的应用。

```
1 # ch10_21.py
2 # 方法1
3 fruits = set("orange")
4 print("字符a是否不属于fruits集合?", 'a' not in fruits)
5 print("字符d是否不属于fruits集合?", 'd' not in fruits)
6 # 方法2
7 cars = {"Nissan", "Toyota", "Ford"}
8 boolean = "Ford" not in cars
9 print("Ford not in cars", boolean)
10 boolean = "Audi" not in cars
11 print("Audi not in cars", boolean)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_21.py
字符a是否不属于fruits集合? False
字符d是否不属于fruits集合? True
Ford not in cars False
Audi not in cars True
>>>
```

10-3 适用集合的方法

方法	说明
add()	加一个元素到集合
clear()	删除集合所有元素
copy()	浅复制 (shallow copy) 方式复制集合
difference_update()	删除集合内与另一集合重复的元素
discard()	如果是集合成员则删除
intersection_update()	可以使用交集更新集合内容
isdisjoint()	如果 2 个集合没有交集返回 True
issubset()	如果另一个集合包含这个集合返回 True
issuperset()	如果这个集合包含另一个集合返回 True
pop()	传回所删除的元素，如果是空集合返回 False
remove()	删除指定元素，如果此元素不存在，程序将返回 KeyError
symmetric_differende_update()	使用对称差集更新集合内容
update()	使用并集更新集合内容

10-3-1 add()

add()可以增加一个元素，它的语法格式如下：

集合A.add(新增元素)

上述会将add()参数的新增元素加到调用此方法的集合A内。

程序实例ch10_22.py： 在集合内新增元素的应用。

```
1 # ch10_22.py
2 cities = { 'Taipei', 'Beijing', 'Tokyo' }
3 # 增加一般元素
4 cities.add('Chicago')
5 print('cities集合内容 ', cities)
6 # 增加已有元素并观察执行结果
7 cities.add('Beijing')
8 print('cities集合内容 ', cities)
9 # 增加元组元素并观察执行结果
10 tup = (1, 2, 3)
11 cities.add(tup)
12 print('cities集合内容 ', cities)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_22.py =====
cities集合内容 {'Tokyo', 'Beijing', 'Chicago', 'Taipei'}
cities集合内容 {'Tokyo', 'Beijing', 'Chicago', 'Taipei'}
cities集合内容 {(1, 2, 3), 'Tokyo', 'Beijing', 'Taipei', 'Chicago'}
>>>
```

上述第7行，由于集合已经有‘Beijing’字符串，将不改变集合cities内容。另外，集合是无序的，你可能获得不同的排列结果。

10-3-2 copy()

集合复制也会像6-8节的列表复制一样，有深复制（deep copy）与浅复制（shallow copy），这个方法不需参数，语法格式如下：

新集合名称 = 旧集合名称.copy()

copy()是浅复制，经过复制后未来一个集合内容改变时，不会影响到另一个集合的内容。

程序实例ch10_23.py： 浅复制与深复制的比较。

```

1 # ch10_23.py
2 # 深复制deep copy
3 numset = {1, 2, 3}
4 deep_numset = numset
5 deep_numset.add(10)
6 print("深复制 - 观察numset", numset)
7 print("深复制 - 观察deep_numset", deep_numset)
8
9 # 浅复制shallow copy
10 shallow_numset = numset.copy()
11 shallow_numset.add(100)
12 print("浅复制 - 观察numset", numset)
13 print("浅复制 - 观察shallow_numset", shallow_numset)

```

执行结果

```

===== RESTART: D:\Python\ch10\ch10_23.py =====
深复制 - 观察numset      {10, 1, 2, 3}
深复制 - 观察deep_numset {10, 1, 2, 3}
浅复制 - 观察numset      {10, 1, 2, 3}
浅复制 - 观察shallow_numset {1, 2, 3, 100, 10}
>>>

```

10-3-3 remove()

如果指定删除的元素存在集合内，remove()可以删除这个集合元素；如果指定删除的元素不存在集合内，将有KeyError产生。它的语法格式如下：

集合A.remove(欲删除的元素)

上述会将集合A内remove()参数指定的元素删除。

程序实例ch10_24.py： 使用remove()删除集合元素成功的应用。

```

1 # ch10_24.py
2 countries = {'Japan', 'China', 'France'}
3 print("删除前的countries集合 ", countries)
4 countries.remove('Japan')
5 print("删除后的countries集合 ", countries)

```

执行结果

```

===== RESTART: D:\Python\ch10\ch10_24.py =====
删除前的countries集合 {'France', 'China', 'Japan'}
删除后的countries集合 {'France', 'China'}
>>>

```

程序实例ch10_25.py： 使用remove()删除集合元素失败 的观察。

```

1 # ch10_25.py
2 animals = {'dog', 'cat', 'bird'}
3 print("删除前的animals集合 ", animals)
4 animals.remove('fish') # 删除不存在的元素产生错误
5 print("删除后的animals集合 ", animals)

```


执行结果

```
===== RESTART: D:\Python\ch10\ch10_25.py =====
删除前的animals集合 {'cat', 'dog', 'bird'}
Traceback (most recent call last):
  File "D:\Python\ch10\ch10_25.py", line 4, in <module>
    animals.remove('fish') # 删除不存在的元素产生错误
KeyError: 'fish'
>>>
```

上述由于fish不存在于animals集合中，所以会产生错误。如果要避免这类错误，可以使用discard()方法。

10-3-4 discard()

discard()可以删除集合的元素，如果元素不存在也不会有错误产生。

```
ret_value = 集合A.discard(欲删除的元素)
```

上述会将集合A内，discard()参数指定的元素删除。不论删除结果为何，这个方法会传回None，这个None在一些程序语言其实是称NULL，本书11-3节会介绍更多函数传回值与传回None的知识。

程序实例ch10_26.py： 使用discard()删除集合元素的应用。

```
1 # ch10_26.py
2 animals = {'dog', 'cat', 'bird'}
3 print("删除前的animals集合 ", animals)
4 # 欲删除元素在集合内
5 animals.discard('cat')
6 print("删除后的animals集合 ", animals)
7 # 欲删除元素没有在集合内
8 animals.discard('pig')
9 print("删除后的animals集合 ", animals)
10 # 打印传回值
11 print("删除数据存在的传回值 ", animals.discard('dog'))
12 print("删除数据不存在的传回值 ", animals.discard('pig'))
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_26.py =====
删除前的animals集合 {'dog', 'bird', 'cat'}
删除后的animals集合 {'dog', 'bird'}
删除后的animals集合 {'dog', 'bird'}
删除数据存在的传回值 None
删除数据不存在的传回值 None
>>>
```

10-3-5 pop()

pop()是用随机方式删除集合元素，所删除的元素将被传回，如果集合是空集合则程序会产生TypeError错误。

```
ret_element = 集合A.pop( )
```

上述会随机删除集合A内的元素，所删除的元素将被传回ret_element。

程序实例ch10_27.py： 使用pop()删除集合元素的应用。

```
1 # ch10_27.py
2 animals = {'dog', 'cat', 'bird'}
3 print("删除前的animals集合 ", animals)
4 ret_element = animals.pop( )
5 print("删除后的animals集合 ", animals)
6 print("所删除的元素是      ", ret_element)
```

执行结果

```
RESTART: D:\Python\ch10\ch10_27.py
删除前的animals集合 {'dog', 'bird', 'cat'}
删除后的animals集合 {'bird', 'cat'}
所删除的元素是      dog
>>>
```

10-3-6 clear()

clear()可以删除集合内的所有元素，传回值是None。

程序实例ch10_28.py： 使用clear()删除集合所有元素的应用，这个程序会列出删除所有集合元素前后的集合内容，同时也列出删除空集合的结果。

```
1 # ch10_28.py
2 states = {'Mississippi', 'Idaho', 'Florida'}
3 print("删除前的states集合 ", states)
4 states.clear( )
5 print("删除前的states集合 ", states)
6
7 # 测试删除空集合
8 empty_set = set( )
9 print("删除前的empty_set集合 ", empty_set)
10 states.clear( )
11 print("删除前的empty_set集合 ", empty_set)
```

执行结果


```

===== RESTART: D:\Python\ch10\ch10_28.py =====
删除前的states集合 {'Florida', 'Idaho', 'Mississippi'}
删除前的states集合 set()
删除前的empty_set集合 set()
删除前的empty_set集合 set()
>>>

```

10-3-7 isdisjoint()

如果2个集合没有共同的元素会传回True，否则传回False。

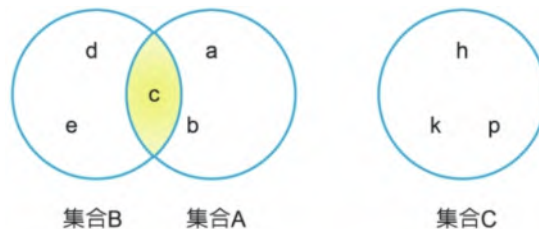
ret_boolean = 集合A.isdisjoint(集合B)

程序实例ch10_29.py： 测试isdisjoint()，下列是集合A、B和C的集合示意图。

```

1 # ch10_29.py
2 A = {'a', 'b', 'c'}
3 B = {'c', 'd', 'e'}
4 C = {'h', 'k', 'p'}
5 # 测试A和B集合
6 boolean = A.isdisjoint(B) # 有共同的元素'c'
7 print("有共同的元素，传回值是 ", boolean)
8
9 # 测试A和C集合
10 boolean = A.isdisjoint(C) # 没有共同的元素
11 print("没有共同的元素，传回值是 ", boolean)

```



执行结果

```

===== RESTART: D:\Python\ch10\ch10_29.py =====
有共同的元素，传回值是 False
没有共同的元素，传回值是 True
>>>

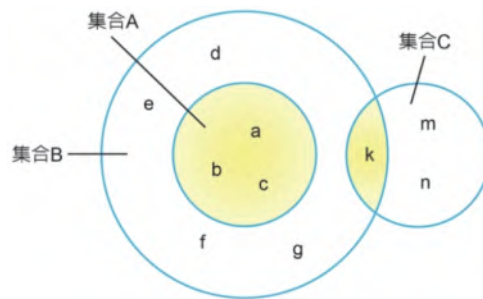
```

10-3-8 issubset()

这个方法可以测试一个函数是否是另一个函数的子集合，例如，A集合所有元素均可在B集合内发现，则A集合是B集合的子集合。如果是则传回True，否则传回False。

程序实例ch10_30.py: 测试issubset(), 下列是A、B和C的集合示意图。

```
1 # ch10_30.py
2 A = {'a', 'b', 'c'}
3 B = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'k'}
4 C = {'k', 'm', 'n'}
5 # 测试A和B集合
6 boolean = A.issubset(B)          # 所有A的元素皆是B的元素
7 print("A集合是B集合的子集, 传回值是 ", boolean)
8
9 # 测试C和B集合
10 boolean = C.issubset(B)          # 有共同的元素k
11 print("C集合是B集合的子集, 传回值是 ", boolean)
```



执行结果

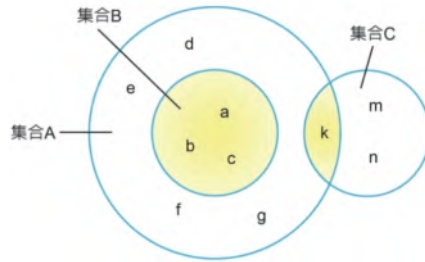
```
===== RESTART: D:\Python\ch10\ch10_30.py =====
A集合是B集合的子集, 传回值是 True
C集合是B集合的子集, 传回值是 False
>>>
```

10-3-9 issuperset()

这个方法可以测试一个函数是否是另一个函数的父集合, 例如, B集合所有元素均可在A集合内发现, 则A集合是B集合的父集合。如果是则传回True, 否则传回False。

程序实例ch10_31.py: 测试issuperset(), 下列是A、B和C的集合示意图。

```
1 # ch10_31.py
2 A = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'k'}
3 B = {'a', 'b', 'c'}
4 C = {'k', 'm', 'n'}
5 # 测试A和B集合
6 boolean = A.issuperset(B)        # 测试
7 print("A集合是B集合的父集合, 传回值是", boolean)
8
9 # 测试A和C集合
10 boolean = A.issuperset(C)        # 测试
11 print("A集合是C集合的父集合, 传回值是", boolean)
```



执行结果

```
===== RESTART: D:\Python\ch10\ch10_31.py =====
A集合是B集合的父集合传回值是 True
A集合是C集合的父集合传回值是 False
>>>
```

10-3-10 intersection update()

这个方法将传回集合的交集，它的语法格式如下：

```
ret_value = A.intersection_update(*B)
```

上述*B代表可以有1到多个集合，如果只有一个集合，例如是B，则执行后A将是A与B的交集。如果*B代表(B, C)，则执行后A将是A、B与C的交集。

上述传回值是None，此值将设定给ret_value，接下来几个小节的方法皆会传回None，将不再叙述。

程序实例ch10_32.py: intersection_update()的应用。

```
1 # ch10_32.py
2 A = {'a', 'b', 'c', 'd'}
3 B = {'a', 'k', 'c'}
4 C = {'c', 'f', 'w'}
5 # A将是A和B的交集
6 ret_value = A.intersection_update(B)
7 print(ret_value)
8 print("A集合 = ", A)
9 print("B集合 = ", B)
10
11 # A将是A、B和C的交集
12 ret_value = A.intersection_update(B, C)
13 print(ret_value)
14 print("A集合 = ", A)
15 print("B集合 = ", B)
16 print("C集合 = ", C)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_32.py =====
None
A集合 = {'a', 'c'}
B集合 = {'a', 'c', 'k'}
None
A集合 = {'c'}
B集合 = {'a', 'c', 'k'}
C集合 = {'c', 'w', 'f'}
>>>
```

10-3-11 update()

可以将一个集合的元素加到调用此方法的集合内，它的语法格式如下：

集合A.update(集合B)

上述是将集合B的元素加到集合A内。

程序实例ch10_33.py： update()的应用。

```
1 # ch10_33.py
2 cars1 = {'Audi', 'Ford', 'Toyota'}
3 cars2 = {'Nissan', 'Toyota'}
4 print("执行update( )前列出cars1和cars2内容")
5 print("cars1 = ", cars1)
6 print("cars2 = ", cars2)
7 cars1.update(cars2)
8 print("执行update( )后列出cars1和cars2内容")
9 print("cars1 = ", cars1)
10 print("cars2 = ", cars2)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_33.py =====
执行update( )前列出cars1和cars2内容
cars1 = {'Ford', 'Audi', 'Toyota'}
cars2 = {'Nissan', 'Toyota'}
执行update( )后列出cars1和cars2内容
cars1 = {'Ford', 'Audi', 'Toyota', 'Nissan'}
cars2 = {'Nissan', 'Toyota'}
>>>
```

10-3-12 difference update()

可以删除集合内与另一集合重复的元素，它的语法格式如下：

集合A.difference_update(集合B)

上述是将集合A内与集合B重复的元素删除，结果存在A集合。

程序实例ch10_34.py: `difference_update()`的应用，执行这个程序后，在集合A内与集合B重复的元素Toyota将被删除。

```
1 # ch10_34.py
2 cars1 = {'Audi', 'Ford', 'Toyota'}
3 cars2 = {'Nissan', 'Toyota'}
4 print("执行difference_update()前列出cars1和cars2内容")
5 print("cars1 = ", cars1)
6 print("cars2 = ", cars2)
7 cars1.difference_update(cars2)
8 print("执行difference_update()后列出cars1和cars2内容")
9 print("cars1 = ", cars1)
10 print("cars2 = ", cars2)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_34.py =====
执行difference_update()前列出cars1和cars2内容
cars1 = {'Ford', 'Toyota', 'Audi'}
cars2 = {'Nissan', 'Toyota'}
执行difference_update()后列出cars1和cars2内容
cars1 = {'Ford', 'Audi'}
cars2 = {'Nissan', 'Toyota'}
>>>
```

10-3-13 `symmetric difference update()`

与10-2-4小节的对称差集观念一样，但是只更改调用此方法的集合。

集合A.`symmetric_difference_update`(集合B)

程序实例ch10_35.py: `symmetric_difference_update()`的基本应用。

```
1 # ch10_35.py
2 cars1 = {'Audi', 'Ford', 'Toyota'}
3 cars2 = {'Nissan', 'Toyota'}
4 print("执行symmetric_difference_update()前列出cars1和cars2内容")
5 print("cars1 = ", cars1)
6 print("cars2 = ", cars2)
7 cars1.symmetric_difference_update(cars2)
8 print("执行symmetric_difference_update()后列出cars1和cars2内容")
9 print("cars1 = ", cars1)
10 print("cars2 = ", cars2)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_35.py =====
执行symmetric_difference_update()前列出cars1和cars2内容
cars1 = {'Ford', 'Audi', 'Toyota'}
cars2 = {'Nissan', 'Toyota'}
执行symmetric_difference_update()后列出cars1和cars2内容
cars1 = {'Ford', 'Audi', 'Nissan'}
cars2 = {'Nissan', 'Toyota'}
>>>
```

10-4 适用集合的基本函数操作

函数名称	说明
enumerate()	传回连续整数配对的 enumerate 对象
len()	元素数量
max()	最大值
min()	最小值
sorted()	传回已经排序的列表，集合本身则不改变
sum()	总合

10-4-1 max()/min()/sum()

如果元素内容是数值，可以使用max()列出最大值、min()列出最小值和sum()列出总合。如果元素内容是字符或字符串，可以使用max()列出unicode码的最大值、min()列出unicode码的最小值，sum()则不可用在字符或字符串元素。

程序实例ch10_36.py： max()、min()和sum()的基本应用。

```
1 # ch10_36.py
2 numList = {100, 99, 64, 101, 55}
3 chrList = {'a', 'x', 'R', 'z', 'b'}
4 # 数值所组成的集合运算
5 print("列出数值集合的最大值 = ", max(numList))
6 print("列出数值集合的最小值 = ", min(numList))
7 print("列出数值集合的总和 = ", sum(numList))
8
9 # 字符所组成的集合运算
10 print("列出字符集合的最大值 = ", max(chrList))
11 print("列出字符集合的最小值 = ", min(chrList))
```

执行结果

```

===== RESTART: D:\Python\ch10\ch10_36.py =====
列出数值集合的最大值 = 101
列出数值集合的最小值 = 55
列出数值集合的总和 = 419
列出字符集合的最大值 = z
列出字符集合的最小值 = R
>>>

```

10-4-2 len()

可以列出集合元素的数量。

程序实例ch10_37.py: 列出不同元素组成集合的长度。

```

1 # ch10_37.py
2 # 集合由整数所组成
3 integer_set = {1, 2, 3, 4, 5}
4 # 集合由不同数据类型所组成
5 mixed_set = {1, 'Python', (2, 5, 10)}
6 print("integer_set长度 = ", len(integer_set))
7 print("mixed_set 长度 = ", len(mixed_set))

```

执行结果

```

===== RESTART: D:\Python\ch10\ch10_37.py =====
integer_set长度 = 5
mixed_set 长度 = 3
>>>

```

10-4-3 sorted()

6-5-3小节已经说明过sorted()执行列表排序，我们可以将它应用在集合排序，将排序结果存在新列表对象，不过集合本身是不会更改的。

程序实例ch10_38.py: 使用sorted()将集合内容以排序方式传回。

```

1 # ch10_38.py
2 cars = {"Nissan", "Toyota", "Ford"}
3 carslist = sorted(cars)
4 carslist_reverse = sorted(cars, reverse = True)
5 print("由小到大排列 = ", carslist)
6 print("由大到小排列 = ", carslist_reverse)

```

执行结果

```

===== RESTART: D:\Python\ch10\ch10_38.py =====
由小到大排列 = ['Ford', 'Nissan', 'Toyota']
由大到小排列 = ['Toyota', 'Nissan', 'Ford']
>>>

```


10-4-4 enumerate()

可以传回连续整数配对的enumerate对象，在本书6-11节与7-4节对于enumerate对象已有做完整说明，本节将直接用实例说明应如何将enumerate对象应用在集合。

程序实例ch10_39.py： 将集合转成enumerate对象并打印，然后再转成列表再打印，最后再将enumerate对象解析和打印。

```
1 # ch10_39.py
2 drinks = {"coffee", "tea", "wine"}
3 enumerate_drinks = enumerate(drinks)          # 数值初始是0
4 print(enumerate_drinks)                       # 传回enumerate对象所在内存
5 print("下列是输出enumerate对象类型")
6 print(type(enumerate_drinks))                 # 列出对象类型
7 print("下列是转成列表输出")
8 print(list(enumerate_drinks))                  # 转成列表再输出列表
9 print("下列是循环输出")
10 for item in enumerate(drinks):                 # 循环输出
11     print(item)
12
13 print("\n")
14 for count, item in enumerate(drinks):          # 将counter和元素内容分开输出
15     print(count, item)
16
17 print("\n")
18 for count, item in enumerate(drinks, 10):      # 将counter起始值设为10输出
19     print(count, item)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_39.py =====
<enumerate object at 0x030C4198>
下列是输出enumerate对象类型
<class 'enumerate'>
下列是转成列表输出
[(0, 'coffee'), (1, 'tea'), (2, 'wine')]
下列是循环输出
(0, 'coffee')
(1, 'tea')
(2, 'wine')

0 coffee
1 tea
2 wine

10 coffee
11 tea
12 wine
>>>
```

10-5 冻结集合frozenset

set是可变集合，frozenset是不可变集合也可直译为冻结集合，这是一个新的类别(class)，只要设定元素后，这个冻结集合就不能

再更改了。如果将元组(tuple)想成不可变列表 (immutable list)，冻结集合就是不可变集合 (immutable set)。

冻结集合的不可变特性优点是可以用它作字典的键(key)，也可以作为其他集合的元素。冻结集合的建立方式是使用frozenset()函数，冻结集合建立完成后，不可使用add()或remove()更改冻结集合的内容。但是可以执行intersection()、union()、difference()、symmetric_difference()、copy()、issubset()、issuperset()、isdisjoint()等方法。

程序实例ch10_40.py： 建立冻结集合与操作。

```
1 # ch10_40
2 X = frozenset([1, 3, 5])
3 Y = frozenset([5, 7, 9])
4 print(X)
5 print(Y)
6 print("交集 = ", X & Y)
7 print("并集 = ", X | Y)
8 A = X & Y
9 print("交集A = ", A)
10 A = X.intersection(Y)
11 print("交集A = ", A)
```

执行结果

```
===== RESTART: D:\Python\ch10\ch10_40.py =====
frozenset({1, 3, 5})
frozenset({5, 7, 9})
交集 = frozenset({5})
并集 = frozenset({1, 3, 5, 7, 9})
交集A = frozenset({5})
交集A = frozenset({5})
>>>
```

习题

1. 请建立2个列表：

A: 1, 3, 5, ..., 99

B: 0, 5, 10, ..., 100

然后求上述的交集，并集， $A - B$ ， $B - A$ ， $A - B$ 对称差集， $B - A$ 对称差集。

2. 请建立2个列表：

A: 1, 3, 5, $\dots$, 99

B: 1至100的质数

然后求上述的交集，并集， $A - B$ ， $B - A$ ， $A - B$ 对称差集， $B - A$ 对称差集。

3. 有一段英文段落如下：

Silicon Stone Education is an unbiased organization, concentrated on bridging the gap between academic and the working world in order to benefit society as a whole. We have carefully crafted our online certification system and test content databases. The content for each topic is created by experts and is all carefully designed with a comprehensive knowledge to greatly benefit all candidates who participate.

请将上述文章处理成没有重复字符的字符列表。

4. 有3个夏令营集合分别如下：

Math: Peter, Norton, Kevin, Mary, John, Ford, Nelson, Damon, Ivan, Tom

Computer: Curry, James, Mary, Turisa, Tracy, Judy, Lee, Jarmul, Damon, Ivan

Physics: Eric, Lee, Kevin, Mary, Christy, Josh, Nelson, Kazil, Linda, Tom

请分别列出下列资料：

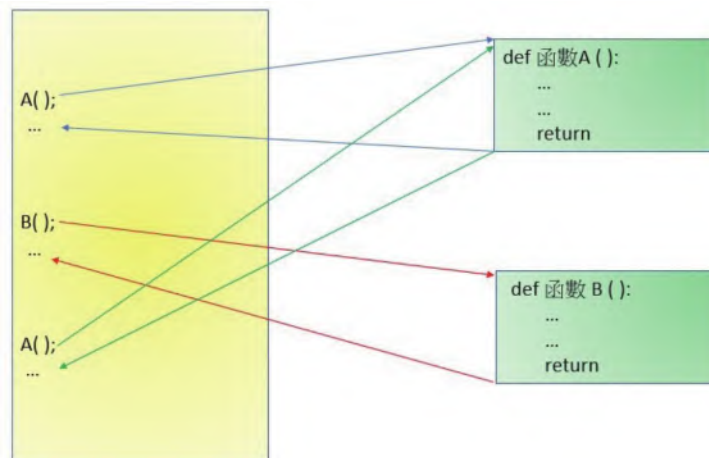
- a: 同时参加3个夏令营的名单。
- b: 同时参加Math和Computer的夏令营的名单。
- c: 同时参加Math和Physics的夏令营的名单。
- d: 同时参加Pyhsics和Computer的夏令营的名单。

第11章 函数设计

函数设计所谓的函数(function)其实就是一系列指令语句所组成，它的目的有两个。

1. 当我们在设计一个大型程序时，若是能将这个程序依功能，将其分割成较小的功能，然后依这些较小功能要求撰写函数程序，如此，不仅使程序简单化，最后程序侦错也变得容易。另外，撰写大型程序时应该是团队合作，每一个人负责一个小功能，可以缩短程序开发的时间。
2. 在一个程序中，也许会发生某些指令被重复书写在许多不同的地方，若是我们能将这些重复的指令撰写成一个函数，需要用时再加以调用，如此，不仅减少编辑程序的时间，更可使程序精简、清晰、明了。

下列是调用函数的基本流程图。



当一个程序在调用函数时，Python会自动跳到被调用的函数上执行工作，执行完后，会回到原先程序执行位置，然后继续执行下一道指令。

11-1 Python函数基本观念

从前面的学习相信读者已经熟悉使用Python内置的函数了，例如，`len()`、`add()`、`remove()`等。有了这些函数，我们可以随时调用使用，让程序设计变得很简洁，这一章主题将是如何设计这类的函数。

11-1-1 函数的定义

函数的语法格式如下：

```
def 函数名称 (参数值 1[, 参数值 2, ... ]):  
    """ 函数批注 (docstring) """  
    程序代码区块                                # 需要内缩  
    return [返回值 1, 返回值 2 , ... ]
```

-  **函数名称** 名称必须是唯一的，程序未来可以调用引用。
-  **参数值** 这是可有可无的，完全视函数设计需要，可以接收调用函数传来的变量，各参数值之间是用逗号“,”隔开。
-  **函数批注** 这是可有可无的，不过如果是参与大型程序设计计划，当负责一个小程序时，建议所设计的函数需要加上批注，除了自己需要也是方便他人阅读。主要是注明此函数的功能，由于可能有多行批注所以可以用3个双引号（或单引号）包夹。许多英文Python资料称此为docstring(document string的缩写)。
-  **return [返回值1, 返回值2 , ...]**不论是return或接续右边的返回值皆是可有可无，如果有返回多个数据彼此需以逗号“,”隔开。

11-1-2 无参数无返回值的函数

程序实例ch11_1.py: 第一次设计Python函数。

```
1 # ch11_1.py
2 def greeting( ):
3     """我的第一个Python函数设计"""
4     print("Python欢迎你")
5     print("祝福学习顺利")
6     print("谢谢")
7
8 # 以下的程序代码也可称主程序
9 greeting( )
10 greeting( )
11 greeting( )
12 greeting( )
13 greeting( )
```

执行结果



在程序设计的观念中，有时候我们也可以将第8行以后的程序代码称主程序。读者可以想想看，如果没有函数功能我们的程序设计将如下所示：

程序实例ch11_2.py: 重新设计ch11_1.py，但是不使用函数设计。

```
1 # ch11_2.py
2 print("Python欢迎你")
3 print("祝福学习顺利")
4 print("谢谢")
5 print("Python欢迎你")
6 print("祝福学习顺利")
7 print("谢谢")
8 print("Python欢迎你")
9 print("祝福学习顺利")
10 print("谢谢")
11 print("Python欢迎你")
12 print("祝福学习顺利")
13 print("谢谢")
14 print("Python欢迎你")
15 print("祝福学习顺利")
16 print("谢谢")
```

执行结果 与ch11_1.py相同。

上述程序虽然也可以完成工作，但是可以发现重复的语句太多了，这不是一个好的设计。同时如果要将“Python欢迎你”改成


```

1 # ch11_3.py
2 def greeting(name):
3     """Python函数需传递名字name"""
4     print("Hi, ", name, "Good Morning!")
5     greeting('Nelson')

```

执行结果

```

===== RESTART: D:/Python/ch11/ch11_3.py =====
Hi, Nelson Good Morning!
>>>

```

上述执行时，第5行调用函数greeting()时，所放的参数是Nelson，这个字符串将传给函数括号内的name参数，所以程序第4行会将Nelson字符串透过name参数打印出来。

在Python应用中，有时候也常会将第4行写成下列语法，可参考ch11_3_1.py，执行结果是相同的。

```

4     print("Hi, " + name + " Good Morning!")

```

特别留意由于我们可以在Python Shell环境调用函数，所以在设计与使用者(user)交流的程序时，也可以先省略第5行的调用，让调用留到Python提示信息(Python prompt)环境。

程序实例ch11_4.py： 程序设计时不做调用，在Python提示信息环境调用。

```

1 # ch11_4.py
2 def greeting(name):
3     """Python函数需传递名字name"""
4     print("Hi, " + name + " Good Morning!")

```

执行结果

```

===== RESTART: D:/Python/ch11/ch11_4.py =====
>>> greeting('Nelson')
Hi, Nelson Good Morning!
>>> greeting('Tina')
Hi, Tina Good Morning!
>>>

```

上述程序最大的特色是greeting('Nelson')与greeting('Tina')，皆是从Python提示信息环境做输入。

11-2-2 多个参数传递

当所设计的函数需要传递多个参数，调用此函数时就需要特别留意传递参数的位置需要正确，最后才可以获得正确的结果。最常见的传递参数是数值或字符串数据，有时也会需要传递列表、元组或字典。

程序实例ch11_5.py： 设计减法的函数subtract()，第一个参数会减去第二个参数，然后列出执行结果。

```
1 # ch11_5.py
2 def subtract(x1, x2):
3     """ 减法设计 """
4     result = x1 - x2
5     print(result)          # 输出减法结果
6 print("本程序会执行 a - b 的运算")
7 a = int(input("a = "))
8 b = int(input("b = "))
9 print("a - b = ", end="")  # 输出a-b字符串,接下来输出不跳行
10 subtract(a, b)
```

执行结果

```
===== RESTART: D:\Python\ch11\ch11_5.py =====
本程序会执行 a - b 的运算
a = 10
b = 5
a - b = 5
>>>
```

上述函数功能是减法运算，所以需要传递2个参数，然后执行第一个数值减去第2个数值。调用这类的函数时，就必须留意参数的位置，否则会有错误信息产生。对于上述程序而言，变量a和b皆是从屏幕输入，执行第10行调用subtract()函数时，a将传给x1，b将传给x2。

程序实例ch11_6.py： 这也是一个需传递2个参数的实例，第一个是兴趣 (interest)，第二个是主题 (subject)。

```
1 # ch11_6.py
2 def interest(interest_type, subject):
3     """ 显示兴趣和主题 """
4     print("我的兴趣是 " + interest_type )
5     print("在 " + interest_type + " 中, 最喜欢的是 " + subject)
6     print( )
7
8 interest('旅游', '敦煌')
9 interest('程序设计', 'Python')
```

执行结果

```
===== RESTART: D:\Python\ch11\ch11_6.py =====
我的兴趣是 旅游
在 旅游 中, 最喜欢的是 敦煌

我的兴趣是 程序设计
在 程序设计 中, 最喜欢的是 Python

>>>
```

上述程序第8行调用 `interest()` 时, ‘旅游’ 会传给 `interest_type` 、 ‘敦煌’ 会传给 `subject`。第9行调用 `interest()` 时, ‘程序设计’ 会传给 `interest_type` 、 ‘Python’ 会传给 `subject`。对于上述的实例, 相信读者应该了解调用需要传递多个参数的函数时, 所传递参数的位置很重要否则会有不可预期的错误。如下列所示:

```
===== RESTART: D:\Python\ch11\ch11_6.py =====
我的兴趣是 旅游
在 旅游 中, 最喜欢的是 敦煌

我的兴趣是 程序设计
在 程序设计 中, 最喜欢的是 Python

>>> interest('敦煌', '旅游')
我的兴趣是 敦煌
在 敦煌 中, 最喜欢的是 旅游

>>>
```

11-2-3 关键词参数 参数名称=值

所谓的关键词参数 (keyword arguments) 是指调用函数时, 参数是用 **参数名称=值** 配对方式呈现。Python 也允许在调用需传递多个参数的函数时, 直接将 **参数名称=值** 用配对方式传送, 这个时候参数的位置就不重要了。

程序实例 ch11_7.py: 这个程序基本上是重新设计 `ch11_6.py`, 但是传递参数时, 其中一个参数直接用 **参数名称=值** 配对方式传送。

```
1 # ch11_7.py
2 def interest(interest_type, subject):
3     """ 显示兴趣和主题 """
4     print("我的兴趣是 " + interest_type)
5     print("在 " + interest_type + " 中, 最喜欢的是 " + subject)
6     print()
7
8 interest(interest_type = '旅游', subject = '敦煌') # 位置正确
9 interest(subject = '敦煌', interest_type = '旅游') # 位置更正
```

执行结果

```
===== RESTART: D:\Python\ch11\ch11_7.py =====
我的兴趣是 旅游
在 旅游 中, 最喜欢的是 敦煌

我的兴趣是 旅游
在 旅游 中, 最喜欢的是 敦煌

>>>
```

读者可以留意程序第8行和第9行的“`interest_type = ‘旅游’`”，当调用函数用配对方式传送参数时，即使参数位置错误，程序执行结果也会相同，因为在调用时已经明确指出所传递的值是要给哪一个参数了。

11-2-4 参数默认值的处理

在设计函数时也可以给参数默认值，如果调用的这个函数没有给参数值，函数的默认值将派上用场。特别需留意：函数设计时含有默认值的参数，必须放置在参数列的最右边。请参考下列程序第2行，如果将“`subject = ‘敦煌’`”与“`interest_type`”位置对调，程序会有错误产生。

程序实例ch11_8.py：重新设计ch11_7.py，这个程序会将subject的默认值设为“敦煌”。程序将用不同方式调用，读者可以从中体会程序参数默认值的意义。

```
1 # ch11_8.py
2 def interest(interest_type, subject = '敦煌'):
3     """ 显示兴趣和主题 """
4     print("我的兴趣是 " + interest_type )
5     print("在 " + interest_type + " 中, 最喜欢的是 " + subject)
6     print()
7
8 interest('旅游') # 传递一个参数
9 interest(interest_type = '旅游') # 传递一个参数
10 interest('旅游', '张家界') # 传递二个参数
11 interest(interest_type = '旅游', subject = '张家界') # 传递二个参数
12 interest(subject = '张家界', interest_type = '旅游') # 传递二个参数
13 interest('阅读', '旅游类') # 传递二个参数,不同的主题
```

执行结果

```
===== RESTART: D:\Python\ch11\ch11_8.py =====
我的兴趣是 旅游
在 旅游 中, 最喜欢的是 敦煌

我的兴趣是 旅游
在 旅游 中, 最喜欢的是 敦煌

我的兴趣是 旅游
在 旅游 中, 最喜欢的是 张家界

我的兴趣是 旅游
在 旅游 中, 最喜欢的是 张家界

我的兴趣是 旅游
在 旅游 中, 最喜欢的是 张家界

我的兴趣是 阅读
在 阅读 中, 最喜欢的是 旅游类

>>>
```

上述程序第8行和9行只传递一个参数，所以subject就会使用默认值“敦煌”，第10、11和12行传送了2个参数，其中第11和12行笔者用参数名称=值 用配对方式调用传送，可以获得一样的结果。第13行主要说明使用不同类的参数一样可以获得正确语意的结果。

11-3 函数返回值

在前面的章节实例我们有执行调用许多内置的函数，有时会返回一些有意义的数据，例如：`len()`返回元素数量。有些没有返回值，此时Python会自动返回None，例如：`clear()`。为何会如此？本节会完整解说函数返回值的知识。

11-3-1 返回None

前2个小节所设计的函数全部没有“`return [返回值]`”，Python在直译时会自动返回处理成“`return None`”，相当于返回None。在一些程序语言，例如，C语言这个None就是NULL，None在Python中独立成为一个数据类型NoneType，下列是实例观察。

程序实例ch11_9.py： 重新设计ch11_3.py，这个程序并没有做返回值设计，不过笔者将列出Python返回`greeting()`函数的数据是否是None，同时列出返回值的数据类型。

```

1 # ch11_9.py
2 def greeting(name):
3     """Python函数需传递名字name"""
4     print("Hi, ", name, " Good Morning!")
5 ret_value = greeting('Nelson')
6 print("greeting( )返回值 = ", ret_value)
7 print(ret_value, " 的 type = ", type(ret_value))

```

执行结果

```

===== RESTART: D:\Python\ch11\ch11_9.py =====
Hi, Nelson Good Morning!
greeting( )返回值 = None
None 的 type = <class 'NoneType'>
>>>

```

上述函数greeting()没有return, Python将自动处理成return None。其实即使函数设计时有return但是没有返回值, Python也将自动处理成return None, 可参考下列实例第5行。

程序实例ch11_10.py: 重新设计ch11_9.py, 函数末端增加return。

```

1 # ch11_10.py
2 def greeting(name):
3     """Python函数需传递名字name"""
4     print("Hi, ", name, " Good Morning!")
5     return # Python将自动回传None
6 ret_value = greeting('Nelson')
7 print("greeting( )返回值 = ", ret_value)
8 print(ret_value, " 的 type = ", type(ret_value))

```

执行结果 与ch11_9.py相同。

11-3-2 简单返回数值数据

参数具有返回值功能, 将可以大大增加程序的可读性, 返回的基本方式可参考下列程序第5行:

```

    return result

# result

```

就是返回的值

程序实例ch11_11.py: 利用函数的返回值, 重新设计ch11_5.py减法的运算。

```

1 # ch11_11.py
2 def subtract(x1, x2):
3     """ 减法设计 """
4     result = x1 - x2
5     return result # 返回减法结果
6 print("本程序会执行 a - b 的运算")
7 a = int(input("a = "))
8 b = int(input("b = "))
9 print("a - b = ", subtract(a, b)) # 输出a-b字符串和结果

```

执行结果

```

===== RESTART: D:\Python\ch11\ch11_11.py =====
本程序会执行 a - b 的运算
a = 10
b = 5
a - b = 5
>>>

```

一个程序常常是由许多函数所组成，下列是程序含2个函数的应用。

程序实例ch11_12.py： 设计加法和减法器。

```

1 # ch11_12.py
2 def subtract(x1, x2):
3     """ 减法设计 """
4     return x1 - x2 # 返回减法结果
5 def addition(x1, x2):
6     """ 加法设计 """
7     return x1 + x2 # 返回加法结果
8
9 # 使用者输入
10 print("请输入运算")
11 print("1:加法")
12 print("2:减法")
13 op = int(input("输入1/2: "))
14 a = int(input("a = "))
15 b = int(input("b = "))
16
17 # 程序运算
18 if op == 1:
19     print("a + b = ", addition(a, b)) # 输出a-b字符串和结果
20 elif op == 2:
21     print("a - b = ", subtract(a, b)) # 输出a-b字符串和结果
22 else:
23     print("运算方法输入错误")

```

执行结果


```
===== RESTART: D:\Python\ch11\ch11_12.py =====
请输入运算
1:加法
2:减法
输入1/2: 1
a = 5
b = 3
a + b = 8
>>>

===== RESTART: D:\Python\ch11\ch11_12.py =====
请输入运算
1:加法
2:减法
输入1/2: 2
a = 5
b = 3
a - b = 2
>>>
```

11-3-3 返回多个数据的应用

使用return返回函数数据时，也允许返回多个数据，各个数据间只要以逗号隔开即可，读者可参考下列实例第8行。

程序实例ch11_13.py： 请输入2个数据，此函数将返回加法、减法、乘法、除法的执行结果。

```
1 # ch11_13.py
2 def mutifunction(x1, x2):
3     """ 加，减，乘，除四则运算 """
4     addresult = x1 + x2
5     subresult = x1 - x2
6     mulresult = x1 * x2
7     divresult = x1 / x2
8     return addresult, subresult, mulresult, divresult
9
10 x1 = x2 = 10
11 add, sub, mul, div = mutifunction(x1, x2)
12 print("加法结果 = ", add)
13 print("减法结果 = ", sub)
14 print("乘法结果 = ", mul)
15 print("除法结果 = ", div)
```

执行结果

```
===== RESTART: D:\Python\ch11\ch11_13.py
加法结果 = 20
减法结果 = 0
乘法结果 = 100
除法结果 = 1.0
>>>
```

11-3-4 简单返回字符串数据

返回字符串的方法与11-3-2节返回数值的方法相同。

程序实例ch11_14.py： 一般中文姓名是3个字，笔者将中文姓名拆解为第一个字是姓 lastname，第二个字是中间名 middlename，第三个

字是名 `firstname`。这个程序内有一个函数 `guest_info()`，参数意义分别是名 `firstname`、中间名 `middlename` 和姓 `lastname`，以及性别 `gender` 组织起来，同时加上问候语 返回。

```
1 # ch11_14.py
2 def guest_info(firstname, middlename, lastname, gender):
3     """ 整合客户名字数据 """
4     if gender == "M":
5         welcome = lastname + middlename + firstname + '先生欢迎你'
6     else:
7         welcome = lastname + middlename + firstname + '小姐欢迎您'
8     return welcome
9
10 info1 = guest_info('宇', '星', '洪', 'M')
11 info2 = guest_info('雨', '冰', '洪', 'F')
12 print(info1)
13 print(info2)
```

执行结果

```
===== RESTART: D:\Python\ch11\ch11_14.py =====
洪星宇先生欢迎你
洪冰雨小姐欢迎您
>>>
```

如果读者是处理外国人的名字，则需在 `lastname`、`middlename` 和 `firstname` 之间加上空格，同时外国人名字处理方式顺序是 `firstname middlename lastname`，这将是各位的习题。

11-3-5 再谈参数默认值

虽然大多数国人的名字是由3个字所组成，但是偶尔也会遇上2个字的状况，例如，著名影星刘涛。其实外国人的名字中，有些人也是只有2个字，因为没有中间名 `middlename`。如果能让 `ch11_14.py` 更完美，可以在函数设计时将 `middlename` 默认为空字符串，这样就可以处理没有中间名的问题，参考 `ch11_8.py` 可知，设计时必须将默认为空字符串的参数放函数参数列的最右边。

程序实例 `ch11_15.py`： 重新设计 `ch11_14.py`，这个程序会将 `middlename` 默认为空字符串，这样就可以处理没有中间名 `middlename`

的问题，请留意函数设计时需将此参数预设放在最右边，可以参考第2行。

```
1 # ch11_15.py
2 def guest_info(firstname, lastname, gender, middlename = ''):
3     """ 整合客户名字数据 """
4     if gender == "M":
5         welcome = lastname + middlename + firstname + '先生欢迎你'
6     else:
7         welcome = lastname + middlename + firstname + '小姐欢迎你'
8     return welcome
9
10 info1 = guest_info('涛', '刘', 'M')
11 info2 = guest_info('雨', '洪', 'F', '冰')
12 print(info1)
13 print(info2)
```

执行结果

```
===== RESTART: D:\Python\ch11\ch11_15.py =====
刘涛先生欢迎你
洪冰雨小姐欢迎你
>>>
```

上述第10行调用`guest_info()`函数时只有3个参数，`middlename`就会使用默认的空字符串。第11行调用`guest_info()`函数时有4个参数，`middlename`就会使用调用函数时所设的字符串‘冰’。

11-3-6 函数返回字典数据

函数除了可以返回数值或字符串数据外，也可以返回比较复杂的数据，例如，字典或列表等。

程序实例ch11_16.py： 这个程序会调用`build_vip`函数，在调用时会输入VIP_ID编号和Name姓名数据，函数将返回所建立的字典数据。

```
1 # ch11_16.py
2 def build_vip(id, name):
3     """ 建立VIP信息 """
4     vip_dict = {'VIP_ID':id, 'Name':name}
5     return vip_dict
6
7 member = build_vip('101', 'Nelson')
8 print(member)
```

执行结果

```
===== RESTART: D:\Python\ch11\ch11_16.py =====
{'VIP_ID': '101', 'Name': 'Nelson'}
>>>
```

上述字典数据只是一个简单的应用，在真正的企业建立VIP数据的案例中，可能还需要性别、电话号码、年龄、电子邮件、地址等信息。在建立VIP数据过程，也许有些人会乐意提供手机号码，有些人不愿意提供，函数设计时我们也可以将Tel电话号码默认为空字符串，但是如果有提供电话号码时，程序也可以将它纳入字典内容。

程序实例ch11_17.py： 扩充ch11_16.py，增加电话号码，调用时若没有提供电话号码则字典不含此字段，调用时若有提供电话号码则字典含此字段。

```
1 # ch11_17.py
2 def build_vip(id, name, tel = ''):
3     """ 建立VIP信息 """
4     vip_dict = {'VIP_ID':id, 'Name':name}
5     if tel:
6         vip_dict['Tel'] = tel
7     return vip_dict
8
9 member1 = build_vip('101', 'Nelson')
10 member2 = build_vip('102', 'Henry', '0952222333')
11 print(member1)
12 print(member2)
```

执行结果

```
===== RESTART: D:/Python/ch11/ch11_17.py =====
{'VIP_ID': '101', 'Name': 'Nelson'}
{'VIP_ID': '102', 'Name': 'Henry', 'Tel': '0952222333'}
>>>
```

程序第10行调用build_vip()函数时，由于有提供电话号码字段，所以上述程序第5行会得到if叙述的tel是True，所以在第6行会将此字段增加到字典中。

11-3-7 将循环应用在建立VIP会员字典

我们可以将循环的观念应用在VIP会员字典的建立。

程序实例ch11_18.py： 这个程序在执行时基本上是用无限循环的观念，但是当数据建立完成时，会询问是否继续，如果输入

非 ‘y’ 字符，程序将执行结束。

```
1 # ch11_18.py
2 def build_vip(id, name, tel = ''):
3     """ 建立VIP信息 """
4     vip_dict = {'VIP_ID':id, 'Name':name}
5     if tel:
6         vip_dict['Tel'] = tel
7     return vip_dict
8
9 while True:
10    print("建立VIP信息系统")
11    idnum = input("请输入ID: ")
12    name = input("请输入姓名: ")
13    tel = input("请输入电话号码: ") # 如果直接按Enter可不建立此字段
14    member = build_vip(idnum, name, tel) # 建立字典
15    print(member, "\n")
16    repeat = input("是否继续(y/n)? 输入非y字符可结束系统: ")
17    if repeat != 'y':
18        break
19
20 print("欢迎下次再使用")
```

执行结果

```
===== RESTART: D:\Python\ch11\ch11_18.py =====
建立VIP信息系统
请输入ID: 100
请输入姓名: James
请输入电话号码: 0911223344
{'VIP_ID': '100', 'Name': 'James', 'Tel': '0911223344'}
是否继续(y/n)? 输入非y字符可结束系统: y
建立VIP信息系统
请输入ID: 101
请输入姓名: Kevin
请输入电话号码:
{'VIP_ID': '101', 'Name': 'Kevin'}
是否继续(y/n)? 输入非y字符可结束系统: n
欢迎下次再使用
>>>
```

笔者在上述输入第2个数据时，在电话号码字段没有输入直接单击Enter键，这个动作相当于不做输入，此时将造成可以省略此字段。

11-4 调用函数时参数是列表

11-4-1 基本传递列表参数的应用

在调用函数时，也可以将列表（此列表可以由数值、字符串或字典所组成）当参数传递给函数，然后函数可以遍历列表内容，然后执行更进一步的运作。

程序实例ch11_19： 传递列表给product_msg()函数，函数会遍历列表，然后列出一封产品发表会的信件。

```

1 # ch11_19
2 def product_msg(customers):
3     str1 = '亲爱的:'
4     str2 = '本公司将在2020年12月20日北京举行产品发表会'
5     str3 = '总经理:深石敬上'
6     for customer in customers:
7         msg = str1 + customer + '\n' + str2 + '\n' + str3
8         print(msg, '\n')
9
10 members = ['Damon', 'Peter', 'Mary']
11 product_msg(members)

```

执行结果

```

===== RESTART: D:\Python\ch11\ch11_19.py =====
亲爱的: Damon
本公司将在2020年12月20日北京举行产品发表会
总经理:深石敬上

亲爱的: Peter
本公司将在2020年12月20日北京举行产品发表会
总经理:深石敬上

亲爱的: Mary
本公司将在2020年12月20日北京举行产品发表会
总经理:深石敬上
>>>

```

11-4-2 在函数内修订列表的内容

Python允许在函数内直接修订列表的内容，同时列表经过修正后，主程序的列表也将随之永久性更改结果。

程序实例ch11_20.py： 设计一个麦当劳的点餐系统，顾客在麦当劳点餐时，可以将所点的餐点放入unserved列表，服务完成后将已服务餐点放入served列表。

```

1 # ch11_20.py
2 def kitchen(unserved, served):
3     """ 将未服务的餐点转为已经服务 """
4     print("厨房处理顾客所点的餐点")
5     while unserved:
6         current_meal = unserved.pop( )
7         # 模拟出餐点过程
8         print("菜单: ", current_meal)
9         # 将已出餐点转入已经服务列表
10        served.append(current_meal)
11
12 def show_unserved_meal(unserved):
13     """ 显示尚未服务的餐点 """
14     print("=== 下列是尚未服务的餐点 ===")
15     if not unserved:
16         print("*** 没有餐点 ***", "\n")
17     for unserved_meal in unserved:
18         print(unserved_meal)
19
20 def show_served_meal(served):
21     """ 显示已经服务的餐点 """

```

```

22     print("=== 下列是已经服务的餐点 ===")
23     if not served:
24         print("*** 没有餐点 ***", "\n")
25     for served_meal in served:
26         print(served_meal)
27
28     unserved = ['大麦克', '劲辣鸡腿堡', '麦克鸡块']
29     served = []
30
31     # 列出餐厅处理前的点餐内容
32     show_unserved_meal(unserved)
33     show_served_meal(served)
34
35     # 餐厅服务过程
36     kitchen(unserved, served)
37     print("\n", "=== 厨房处理结束 ===", "\n")
38
39     # 列出餐厅处理后的点餐内容
40     show_unserved_meal(unserved)
41     show_served_meal(served)

```

执行结果

```

===== RESTART: D:\Python\ch11\ch11_20.py =====
=== 下列是尚未服务的餐点 ===
大麦克
劲辣鸡腿堡
麦克鸡块
=== 下列是已经服务的餐点 ===
*** 没有餐点 ***

厨房处理顾客所点的餐点
菜单: 麦克鸡块
菜单: 劲辣鸡腿堡
菜单: 大麦克

=== 厨房处理结束 ===

=== 下列是尚未服务的餐点 ===
*** 没有餐点 ***

=== 下列是已经服务的餐点 ===
麦克鸡块
劲辣鸡腿堡
大麦克
>>>

```

这个程序的主程序从第28行开始，基本上将所点的餐点放unserved列表，第29行将已经处理的餐点放在served列表，程序刚开始是设定空列表。为了了解所做的设定，所以第32和33行是列出尚未服务的餐点和已经服务的餐点。

程序第36行是调用kitchen()函数，这个程序主要是列出餐点，同时将已经处理的餐点从尚未服务列表unserved，转入已经服务的列表served。

程序第40和41行执行一次列出尚未服务餐点和已经服务餐点，以便验证整个执行过程。

对于上述程序而言，读者可能会好奇，主程序部分与函数部分是使用相同的列表变量served与unserved，所以经过第36行调用kitchen()后造成列表内容的改变，是否设计这类欲更改列表内容的程序，函数与主程序的变量名称一定要相同？答案是否定的。其实这牵涉到全局变量（global variable）与局部变量（local variable）的观念，将在11-7节说明。

程序实例ch11_21.py： 重新设计ch11_20.py，但是主程序的尚未服务列表改为order_list，已经服务列表改为served_list，下列只列出主程序内容。

```
28 order_list = ['大麦克', '劲辣鸡腿堡', '麦克鸡块'] # 所点餐点
29 served_list = [] # 已服务餐点
30
31 # 列出餐厅处理前的点餐内容
32 show_unserved_meal(order_list) # 列出未服务餐点
33 show_served_meal(served_list) # 列出已服务餐点
34
35 # 餐厅服务过程
36 kitchen(order_list, served_list) # 餐厅处理过程
37 print("\n", "=== 厨房处理结束 ===", "\n")
38
39 # 列出餐厅处理后的点餐内容
40 show_unserved_meal(order_list) # 列出未服务餐点
41 show_served_meal(served_list) # 列出已服务餐点
```

执行结果 与ch11_20.py相同。

上述结果最主要原因是，当传递列表给函数时，即使函数内的列表与主程序列表是不同的名称，但是函数列表unserved/served与主程序列表order_list/served_list是指向相同的内存位置，所以在函数更改列表内容时主程序列表内容也随着更改。

11-4-3 使用副本传递列表

有时候设计餐厅系统时，可能想要保存餐点内容，但是经过先前程序设计可以发现order_list列表已经变为空列表了，为了避免这样的情形发生，可以在调用kitchen()函数时传递副本列表，处理方式如下：


```
kitchen(order_list[:], served_list) # 传递副本列表
```

程序实例 ch11_22.py： 重新设计 ch11_21.py，但是保留原 order_list 的内容，整个程序主要是在第36行，笔者使用副本传递列表，其他只是程序语意批注有一些小调整，例如，原先函数 show_unserved_meal() 改名为 show_order_meal()。

```
1 # ch11_22.py
2 def kitchen(unserved, served):
3     """ 将所点的餐点转为已经服务 """
4     print("厨房处理顾客所点的餐点")
5     while unserved:
6         current_meal = unserved.pop()
7         # 模拟出餐过程
8         print("菜单:", current_meal)
9         # 将已出餐点转入已经服务列表
10        served.append(current_meal)
11
12 def show_order_meal(unserved):
13     """ 显示所点的餐点 """
14     print("=== 下列是所点的餐点 ===")
15     if not unserved:
16         print("*** 没有餐点 ***", "\n")
17     for unserved_meal in unserved:
18         print(unserved_meal)
19
20 def show_served_meal(served):
21     """ 显示已经服务的餐点 """
22     print("=== 下列是已经服务的餐点 ===")
23     if not served:
24         print("*** 没有餐点 ***", "\n")
25     for served_meal in served:
26         print(served_meal)
27
28 order_list = ['大麦克', '劲辣鸡腿堡', '麦克鸡块'] # 所点餐点
29 served_list = [] # 已服务餐点
30
31 # 列出餐厅处理前的点餐内容
32 show_order_meal(order_list) # 列出所点的餐点
33 show_served_meal(served_list) # 列出已服务餐点
34
35 # 餐厅服务过程
36 kitchen(order_list[:], served_list) # 餐厅处理过程
37 print("\n", "=== 厨房处理结束 ===", "\n")
38
39 # 列出餐厅处理后的点餐内容
40 show_order_meal(order_list) # 列出所点的餐点
41 show_served_meal(served_list) # 列出已服务餐点
```

执行结果

```

===== RESTART: D:\Python\ch11\ch11_22.py =====
== 下列是所点的餐点 ==
大麦克
劲辣鸡腿堡
麦克鸡块
== 下列是已经服务的餐点 ==
*** 没有餐点 ***

厨房处理顾客所点的餐点
菜单: 麦克鸡块
菜单: 劲辣鸡腿堡
菜单: 大麦克

== 厨房处理结束 ==

== 下列是所点的餐点 ==
大麦克
劲辣鸡腿堡
麦克鸡块
== 下列是已经服务的餐点 ==
大麦克
劲辣鸡腿堡
麦克鸡块
>>>

```

由上述执行结果可以发现，原先存储点餐的order_list列表经过kitchen()函数后，此列表的内容没有改变。

11-5 传递任意数量的参数

11-5-1 基本传递处理任意数量的参数

在设计Python的函数时，有时候可能会碰上不知道会有多少个参数会传递到这个函数，此时可以用下列方式设计。

程序实例ch11_23.py: 建立一个冰淇淋的配料程序，一般冰淇淋可以在上面加上配料，这个程序在调用制作冰淇淋函数make_icecream()时，可以传递0到多个配料，然后make_icecream()函数会将配料结果的冰淇淋列出来。

```

1 # ch11_23.py
2 def make_icecream(*toppings):
3     # 列出制作冰淇淋的配料
4     print("这个冰淇淋所加配料如下")
5     for topping in toppings:
6         print("--- ", topping)
7
8 make_icecream('草莓酱')
9 make_icecream('草莓酱', '葡萄干', '巧克力碎片')

```

执行结果


```

===== RESTART: D:\Python\ch11\ch11_23.py =====
这个冰淇淋所加配料如下
--- 草莓酱
这个冰淇淋所加配料如下
--- 草莓酱
--- 葡萄干
--- 巧克力碎片
>>>

```

上述程序最关键的是第2行 `make_icecream()` 函数的参数“*toppings”，这个加上“*”符号的参数代表可以有1到多个参数将传递到这个函数内。

11-5-2 设计含有一般参数与任意数量参数的函数

程序设计时有时会遇上需要传递一般参数与任意数量参数，碰上这类状况，任意数量的参数必须放在最右边。

程序实例ch11_24.py： 重新设计ch11_23.py，传递参数时第一个参数是冰淇淋的种类，然后才是不同数量的冰淇淋的配料。

```

1 # ch11_24.py
2 def make_icecream(icecream_type, *toppings):
3     # 列出制作冰淇淋的配料
4     print("这个 ", icecream_type, " 冰淇淋所加配料如下")
5     for topping in toppings:
6         print("--- ", topping)
7
8 make_icecream('香草', '草莓酱')
9 make_icecream('芒果', '草莓酱', '葡萄干', '巧克力碎片')

```

执行结果

```

===== RESTART: D:\Python\ch11\ch11_24.py =====
这个 香草 冰淇淋所加配料如下
--- 草莓酱
这个 芒果 冰淇淋所加配料如下
--- 草莓酱
--- 葡萄干
--- 巧克力碎片
>>>

```

11-5-3 设计含有一般参数与任意数量的关键词参数

在11-2-3节笔者有介绍函数的参数是关键词参数，其实我们也可以设计含任意数量关键词参数的函数。

程序实例ch11_25.py： 这个程序基本上是用build_dict()函数建立一个球员的字典数据，主程序会传入一般参数与任意数量的关键词参数，最后可以列出执行结果。

```
1 # ch11_25.py
2 def build_dict(name, age, **players):
3     # 建立NBA球员的字典数据
4     info = {} # 建立空字典
5     info['Name'] = name
6     info['Age'] = age
7     for key, value in players.items():
8         info[key] = value
9     return info # 返回所建的字典
10
11 player_dict = build_dict('James', '32',
12                           City = 'Cleveland',
13                           State = 'Ohio')
14
15 print(player_dict) # 打印所建字典
```

执行结果

```
===== RESTART: D:/Python/ch11/ch11_25.py =====
{'Name': 'James', 'Age': '32', 'City': 'Cleveland', 'State': 'Ohio'}
>>>
```

上述最关键的是第2行 build_dict() 函数内的参数 “**player”，这表示可以接受任意数量关键词参数。

11-6 递归式函数设计recursive

一个函数可以调用其他函数也可以调用自己，其中调用本身的动作称递归式 (recursive)调用，递归式调用有下列特色：

- 每次调用自己时，都会使范围越来越小。
- 必须要有一个终止的条件来结束递归函数。

递归函数可以使程序变得很简洁，但是设计这类程序一不小心就很容易掉入无限循环的陷阱，所以使用这类函数时一定要特别小心。递归函数最常见的应用是处理正整数的阶乘(factorial)，一个正整数的阶乘是所有小于以及等于该数的正整数的积，同时如果正整数是0则

阶乘为1，依照观念正整数是1时阶乘也是1。此阶乘数字的表示法为n!。

实例1： n是3，下列是阶乘数的计算方式。

$n! = 1 * 2 * 3$
结果是6。

实例2： n是5，下列是阶乘数的计算方式。

$n! = 1 * 2 * 3 * 4 * 5$
结果是120。

阶乘数观念是由法国数学家克里斯蒂安·克兰普 (Christian Kramp, 1760-1826)所发表，他学医但同时数学感兴趣，发表许多数学文章。

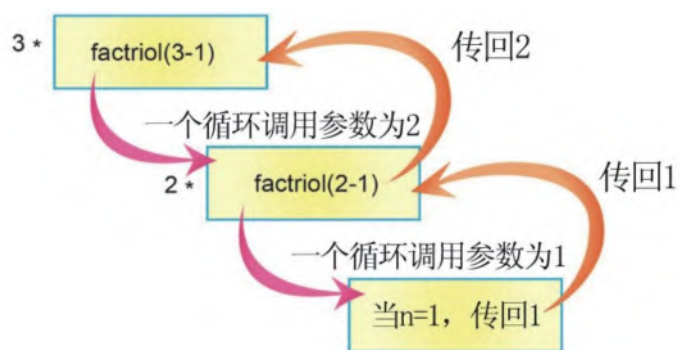
程序实例ch11_26.py： 使用递归函数执行阶乘(factorial)运算。

```
1 # ch11_26.py
2 def factorial(n):
3     # 计算n的阶乘, n 必须是正整数
4     if n == 1:
5         return 1
6     else:
7         return (n * factorial(n-1))
8
9 value = 3
10 print(value, " 的阶乘结果是 =", factorial(value))
11 value = 5
12 print(value, " 的阶乘结果是 =", factorial(value))
```

执行结果

```
===== RESTART: D:\Python\ch11\ch11_26.py =====
3 的阶乘结果是 = 6
5 的阶乘结果是 = 120
>>>
```

上述factorial()函数的终止条件是参数值为1的情况，由第4行判断然后返回1，下列是正整数为3时递归函数的情况解说。



11-7 局部变量与全局变量

在设计函数时，另一个重点是适当地使用变量名称，某个变量只有在该函数内使用，影响范围限定在这个函数内，这个变量称**局部变量** (local variable)。如果某个变量的影响范围是在整个程序，则这个变量称**全局变量** (global variable)。

Python程序在调用函数时会建立一个内存工作区间，在这个内存工作区间可以处理属于这个函数的变量，当函数工作结束，返回原先调用程序时，这个内存工作区间就被收回，原先存在的变量也将被销毁，这也是为何**局部变量** 的影响范围只限定在所属的函数内。

对于**全局变量** 而言，一般是在主程序内建立，程序在执行时，不仅主程序可以引用，所有属于这个程序的函数也可以引用，所以它的影响范围是整个程序。

11-7-1 全局变量可以在所有函数使用

一般在主程序内建立的变量称全局变量，这个变量程序内与本程序的所有函数皆可以引用。

程序实例ch11_27.py: 这个程序会设定一个全局变量，然后函数也可以调用引用。

```

1 # ch11_27.py
2 def printmsg( ):
3     # 函数本身没有定义变量，只有执行打印全局变量功能
4     print("函数打印: ", msg)    # 打印全局变量
5
6 msg = 'Global Variable'        # 设定全局变量
7 print("主程序行印: ", msg)    # 打印全局变量
8 printmsg( )                   # 呼叫函数

```

执行结果

```

===== RESTART: D:\Python\ch11\ch11_27.py =====
主程序行印: Global Variable
函数打印: Global Variable
>>>

```

11-7-2 局部变量与全局变量使用相同的名称

在程序设计时建议全局变量与函数内的局部变量不要使用相同的名称，因为很容易造成混淆。如果全局变量与函数内的局部变量使用相同的名称，Python会将相同名称的区域 与全局 变量视为不同的变量，在局部变量所在的函数是使用局部变量内容，其他区域则是使用全局变量的内容。

程序实例ch11_28.py： 局部变量与全局变量定义了相同的变量msg，但是内容不相同。然后执行打印，可以发现在函数与主程序所打印的内容有不同的结果。

```

1 # ch11_28.py
2 def printmsg( ):
3     # 函数本身有定义变量，将执行打印局部变量功能
4     msg = 'Local Variable'    # 设定局部变量
5     print("函数打印: ", msg)  # 打印局部变量
6
7 msg = 'Global Variable'      # 这是全局变量
8 print("主程序行印: ", msg)   # 打印全局变量
9 printmsg( )                 # 呼叫函数

```

执行结果

```

===== RESTART: D:\Python\ch11\ch11_28.py =====
主程序行印: Global Variable
函数打印: Local Variable
>>>

```

11-7-3 程序设计需注意事项

一般程序设计时有关使用局部变量需注意下列事项，否则程序会有错误产生。

- 局部变量内容无法在其他函数引用。
- 局部变量内容无法在主程序引用。

程序实例ch11_29.py： 局部变量在其他函数引用，造成程序错误的

应用。

```
1 # ch11_29.py
2 def defmsg( ):
3     msg = 'pringmsg variable'
4
5 def printmsg( ):
6     print(msg)      # 打印defmsg( )函数定义的局部变量
7
8 printmsg( )        # 呼叫printmsg( )
```

执行结果

```
===== RESTART: D:\Python\ch11\ch11_29.py =====
Traceback (most recent call last):
  File "D:\Python\ch11\ch11_29.py", line 8, in <module>
    printmsg( )      # 呼叫printmsg( )
  File "D:\Python\ch11\ch11_29.py", line 6, in printmsg
    print(msg)      # 打印defmsg( )函数定义的局部变量
NameError: name 'msg' is not defined
>>>
```

上述程序的错误原因主要是defmsg()函数内没有定义msg变量，所以产生程序错误。

程序实例ch11_30.py： 局部变量在主程序引用产生错误的实例。

```
1 # ch11_30.py
2 def defmsg( ):
3     msg = 'pringmsg variable'
4
5 print(msg)          # 主程序打印局部变量产生错误
```

执行结果

```
===== RESTART: D:\Python\ch11\ch11_30.py =====
Traceback (most recent call last):
  File "D:\Python\ch11\ch11_30.py", line 5, in <module>
    print(msg)      # 主程序打印局部变量产生错误
NameError: name 'msg' is not defined
>>>
```

上述程序的错误原因主要是主程序内没有定义msg变量，所以产生程序错误。

11-8 匿名函数lambda

所谓的匿名函数 (anonymous function)是指一个没有名称的函数，Python是使用def定义一般函数，匿名函数则是使用lambda 来定义，有的人称之为lambda表达式，也可以将匿名函数称lambda函数。通常会将匿名函数与Python的内置函数filter()、map()等共同使用，此时匿名函数将只是这些函数的参数，笔者未来将以实例做解说。

11-8-1 匿名函数lambda的语法

匿名函数最大特色是可以有许多的参数，但是只能有一个程序码表达式，然后将执行结果返回。

lambda arg1[, arg2, ..., argn]:expression # arg1是参数，可以有多个参数

程序实例ch11_31.py: 这是单一参数的匿名函数应用，可以返回平方值。

```
1 # ch11_31.py
2 # 定义lambda函数
3 square = lambda x: x ** 2
4
5 # 输出平方值
6 print(square(10))
```

执行结果

```
===== RESTART: D:/Python/ch11/ch11_31.py
100
>>>
```

读者可以留意第6行调用匿名函数方式，其实上述匿名函数可以用一般函数取代。

程序实例ch11_32.py： 使用一般函数取代匿名函数，重新设计ch11_31.py。

```
1 # ch11_32.py
2 # 定义lambda函数
3 product = lambda x, y: x * y
4
5 # 输出变数的积
6 print(product(5, 10))
```

执行结果 与ch11_31.py相同。

下列是匿名函数含有多个参数的应用。

程序实例ch11_33.py： 含2个参数的匿名函数应用，可以返回参数的积（相乘的结果）。

```
1 # ch11_33.py
2 # 定义lambda函数
3 product = lambda x, y: x * y
4
5 # 输出相乘结果
6 print(product(5, 10))
```

执行结果



```
===== RESTART: D:/Python/ch11/ch11_33.py
50
>>>
```

11-8-2 匿名函数使用与filter()

匿名函数一般是用在不需要函数名称的场合，例如，一些高阶函数(higher-order function)的参数可能是函数，这时就很适合使用匿名函数，同时让程序变得更简洁。有一个内置函数filter()，它的语法格式如下：

```
filter(function, iterable)
```

上述函数将依次对iterable(可以重复执行，例如，字符串string、列表list或元组tuple)的元素(item)放入function(item)

内，然后将function()函数执行结果是True的元素(item)组成新的筛选对象 (filter object)返回。

程序实例ch11_34.py: 使用传统函数定义方式将列表元素内容是奇数的元素筛选出来。

```
1 # ch11_34.py
2 def oddfn(x):
3     return x if (x % 2 == 1) else None
4
5 mylist = [5, 10, 15, 20, 25, 30]
6 filter_object = filter(oddfn, mylist) # 传回filter object
7
8 # 输出奇数列表
9 print("奇数列表: ",[item for item in filter_object])
```

执行结果

```
===== RESTART: D:\Python\ch11\ch11_34.py =====
奇数列表: [5, 15, 25]
>>>
```

上述第9行笔者使用item for item in filter_object, 这是可以取得filter object元素的方式, 这个操作方式与下列for循环类似。

```
for item in filter_object:
    print(item)
```

若是想要获得列表结果, 可以使用下列方式:

```
oddlist = [item for item in filter_object]
```

程序实例ch11_35.py: 重新设计ch11_34.py, 将filter object转为列表, 下列只列出与ch11_34.py不同的程序代码。

```
7 oddlist = [item for item in filter_object]
8 # 输出奇数列表
9 print("奇数列表: ",oddlist)
```

执行结果 与ch11_34.py相同。

匿名函数的最大优点是可以让程序变得更简洁, 可参考下列程序实例。

程序实例ch11_36.py: 使用匿名函数重新设计ch11_35.py。

```
1 # ch11_36.py
2 mylist = [5, 10, 15, 20, 25, 30]
3
4 oddlist = list(filter(lambda x: (x % 2 == 1), mylist))
5
6 # 输出奇数列表
7 print("奇数列表: ",oddlist)
```

执行结果 与ch11_35.py相同。

上述程序第4行笔者直接使用list()函数将返回的filter object转成列表了。

11-8-3 匿名函数使用与map()

有一个内置函数map()，它的语法格式如下：

```
map(function, iterable)
```

上述函数将依次对iterable(可以重复执行，例如，字符串string、列表list或元组tuple)的元素(item)放入function(item)内，然后将function()函数执行结果组成新的筛选对象 (filter object)返回。

程序实例ch11_37.py: 使用匿名函数对列表元素执行计算平方运算。

```
1 # ch11_37.py
2 mylist = [5, 10, 15, 20, 25, 30]
3
4 squarelist = list(map(lambda x: x ** 2, mylist))
5
6 # 输出列表元素的平方值
7 print("列表的平方值: ",squarelist)
```

执行结果

```
===== RESTART: D:\Python\ch11\ch11_37.py =====
列表的平方值: [25, 100, 225, 400, 625, 900]
>>>
```

11-9 pass与函数

在7-4-6节已经有对pass指令做介绍，其实当我们在设计大型程序时，可能会先规划各个函数的功能，然后逐一完成各个函数设计，但是在程序完成前我们可以先将尚未完成的函数内容放上pass。

程序实例ch11_38.py： 将pass应用在函数设计。

```
1 # ch11_38.py
2 def fun(arg):
3     pass
```

执行结果 程序没有执行结果。

11-10 type关键词应用在函数

在结束本章前笔者列出函数的数据类型，读者可以参考。

程序实例ch11_39.py： 输出函数与匿名函数的数据类型。

```
1 # ch11_39.py
2 def fun(arg):
3     pass
4
5 print("列出fun的type类型: ", type(fun))
6 print("列出lambda的type类型: ", type(lambda x:x))
7 print("列出内置函数abs的type类型: ", type(abs))
```

执行结果

```
===== RESTART: D:\Python\ch11\ch11_39.py =====
列出fun的type类型: <class 'function'>
列出lambda的type类型: <class 'function'>
列出内置函数abs的type类型: <class 'builtin_function_or_method'>
>>>
```

习题

1. 请设计一个绝对值absolute()函数，如果输入-5返回5，如果输入5返回5。
2. 请将ch11_12.py扩充为可以执行加法、减法、乘法、除法运算的小型计算器。

3. 请将ch11_12.py扩充为可以重复执行，直到输入运算是q时，程序才执行结束。
4. 请将习题3的观念应用在习题2上。
5. 请重新设计ch11_14.py，将程序处理为适合外国人姓名的使用环境。
6. 请重新设计ch11_28.py，新增一个printing()函数，这个函数不定义msg变量，但是执行打印msg，请观察执行结果。
7. 请用递归函数设计费式数列Fibonacci，费式数列的规则如下：

```
F0 = 0
F1 = 1
Fn = Fn-1 + Fn-2 ( n >= 2)
```

最后值应该是0, 1, 1, 2, 3, 5, 8, 13, 21, 34,

8. 请重新设计ch11_24.py，将程序改为制作pizza，第一个参数改为pizza的尺寸，然后请至pizza店实际选择5种配料。
9. 美国NBA球员Lin的前10场得分资料如下：

25, 18, 12, 22, 31, 17, 26, 19, 18, 10

请使用匿名函数和filter()函数，列出得分超过20分（含）的列表。

第12章 类-面向对象的程序设计

Python 其实是一种面向对象的编程 (Object Oriented Programming)，在Python中其实所有的数据类型皆是对象，Python也允许程序设计师自创数据类型，这种自创的数据类型就是本章的主题类 (class)。

设计程序时可以将世间万物分组归类，然后使用类 (class) 定义你的分类，笔者在本章将举一系列不同的类，扩展读者的思维。

12-1 类的定义与使用

类的语法定义如下：

```
class          Classname ( )    # 类名称第一个字母必须大写

    statement1
    ...
    statementn
```

本节将以银行为例，说明最基本的类观念。

12-1-1 定义类

程序实例ch12_1.py: Banks的类定义。

```
1  # ch12_1.py
2  class Banks():
3      # 定义银行类别
4      title = 'Taipei Bank'      # 定义属性
5      def motto(self):           # 定义方法
6          return "以客为尊"
```

执行结果 这个程序没有输出结果。

对上述程序而言，Banks是类名称，在这个类中笔者定义了一个属性 (attribute)title与一个方法 (method)motto。

在类内定义方法(method)的方式与第11章定义函数的方式相同，但是不可以称之为函数(function)，必须称之为方法 (method)，在程序设计时我们可以随时调用函数，但是只有属于该类的对象(object)才可调用相关的方法。

12-1-2 操作类的属性与方法

若是想操作类的属性与方法首先需定义该类的对象(object)变量，可以简称对象，然后使用下列方式操作。

object.类的属性
object.类的方法()

程序实例ch12_2.py： 扩充ch12_1.py，列出银行名称与服务宗旨。

```
1 # ch12_2.py
2 class Banks():
3     # 定义银行类别
4     title = 'Taipei Bank'      # 定义属性
5     def motto(self):           # 定义方法
6         return "以客为尊"
7
8 userbank = Banks()             # 定义对象userbank
9 print("目前服务银行是 ", userbank.title)
10 print("银行服务理念是 ", userbank.motto())
```

执行结果

```
===== RESTART: D:\Python\ch12\ch12_2.py =====
目前服务银行是 Taipei Bank
银行服务理念是 以客为尊
>>>
```

从上述执行结果可以发现我们成功地存取了Banks类内的属性与方法了。上述程序观念是，程序第8行定义了userbank当作是Banks类的对象，然后使用userbank对象读取了Banks类内的title属性与motto()方法。这个程序主要是列出title属性值与motto()方法传回的内容。

12-1-3 类的构造函数

建立类很重要的一个工作是初始化整个类，所谓的初始化类是在类内建立一个初始化方法（method），这是一个特殊方法，当在程序内定义这个类的对象时将自动执行这个方法。初始化方法有一个固定名称是“__init__()”，写法是init左右皆是2个底线字符，init其实是initialization的缩写，通常又将这类初始化的方法称构造函数（constructor）。在这初始化的方法内可以执行一些属性变量设定，下列笔者先用一个实例做解说。

程序实例ch12_3.py：重新设计ch12_2.py，设定初始化方法，同时存第一笔开户的钱100元入银行，然后列出存款金额。

```
1 # ch12_3.py
2 class Banks():
3     # 定义银行类
4     title = 'Taipei Bank'           # 定义属性
5     def __init__(self, uname, money): # 初始化方法
6         self.name = uname           # 设定存款者名字
7         self.balance = money        # 设定所存的钱
8
9     def get_balance(self):           # 获得存款余额
10        return self.balance
11
12 hungbank = Banks('hung', 100)      # 定义对象hungbank
13 print(hungbank.name, title(), " 存款余额是 ", hungbank.get_balance())
```

执行结果

```
===== RESTART: D:\Python\ch12\ch12_3.py =====
Hung 存款余额是 100
>>>
```

上述在程序12行定义Banks类的hungbank对象时，Banks类会自动启动__init__()初始化函数，在这个定义中self是必需的，同时需放在所有参数的最前面（相当于最左边），Python在初始化时会自动传入这个参数self，代表的是类本身的对象，未来在类内想要参照各属性与函数执行运算皆要使用self，可参考第6、7和10行。

在这个Banks类的__init__(self, uname, money)方法中，有另外2个参数uname和money，未来我们在定义Banks类的对象时（第12

行) 需要传递2个参数, 分别给uname和money。至于程序第6和7行内容如下:

```
self.name

= uname      ; name是Banks类的属性
self.balance

= money      ; balance是Banks类的属性
```

读者可能会思考, 既然__init__这么重要, 为何ch12_2.py没有这个初始化函数仍可运行, 其实对ch12_2.py而言是使用预设没有参数的__init__()方法。

在程序第9行另外有一个get_balance(self)方法, 在这个方法内只有一个参数self, 所以调用时可以不用任何参数, 可以参考第13行。这个方法目的是传回存款余额。

程序实例ch12_4.py: 扩充ch12_3.py, 主要是增加执行存款与提款功能, 同时在类内可以直接列出目前余额。

```
1 # ch12_4.py
2 class Banks():
3     # 定义银行类
4     title = 'Taipei Bank'           # 定义属性
5     def __init__(self, uname, money): # 初始化方法
6         self.name = uname           # 设定存款者名字
7         self.balance = money        # 设定所存的钱
8
9     def save_money(self, money):     # 设计存款方法
10        self.balance += money        # 执行存款
11        print("存款 ", money, " 完成") # 打印存款完成
12
13    def withdraw_money(self, money):  # 设计提款方法
14        self.balance -= money        # 执行提款
15        print("提款 ", money, " 完成") # 打印提款完成
16
17    def get_balance(self):            # 获得存款余额
18        print(self.name.title(), " 目前余额: ", self.balance)
19
20 hungbank = Banks('hung', 100)      # 定义对象hungbank
21 hungbank.get_balance()               # 获得存款余额
22 hungbank.save_money(300)            # 存款300元
23 hungbank.get_balance()              # 获得存款余额
24 hungbank.withdraw_money(200)        # 提款200元
25 hungbank.get_balance()              # 获得存款余额
```

执行结果


```

=====RESTART: D:\Python\ch12\ch12_4.py=====
Hung 目前余额: 100
存款 300 完成
Hung 目前余额: 400
取款 200 完成
Hung 目前余额: 200
>>>

```

其实类建立完成后，我们随时可以使用多个对象引用这个类的属性与函数，可参考下列实例。

程序实例ch12_5.py： 使用与ch12_4.py相同的Banks类，然后定义2个对象使用操作这个类。下列是与ch12_4.py不同的程序代码内容。

```

20 hungbank = Banks('hung', 100)          # 定义对象hungbank
21 johnbank = Banks('john', 300)          # 定义对象johnbank
22 hungbank.get_balance()                  # 获得hung存款余额
23 johnbank.get_balance()                  # 获得john存款余额
24 hungbank.save_money(100)                # hung存款100
25 johnbank.withdraw_money(150)            # john取款150
26 hungbank.get_balance()                  # 获得hung存款余额
27 johnbank.get_balance()                  # 获得john存款余额

```

执行结果

```

=====RESTART: D:\Python\ch12\ch12_5.py=====
Hung 目前余额: 100
John 目前余额: 300
存款 100 完成
取款 150 完成
Hung 目前余额: 200
John 目前余额: 150
>>>

```

12-1-4 属性初始值的设定

在先前程序的Banks类中第4行title是设为“Taipei Bank”，其实这是初始值的设定，通常Python在设初始值时是将初始值设在__init__()方法内，下列这个程序在定义Banks类对象时，省略开户金额，相当于定义Banks类对象时只要2个参数。

程序实例ch12_6.py： 设定开户（定义Banks类对象）只要姓名，同时设定开户金额是0元，读者可留意第7和8行的设定。

```

1  # ch12_6.py
2  class Banks():
3      # 定义银行类
4
5      def __init__(self, uname):          # 初始化方法
6          self.name = uname              # 设定存款者名字
7          self.balance = 0               # 设定开户金额是0
8          self.title = "Taipei Bank"     # 设定银行名称
9
10     def save_money(self, money):         # 设计存款方法
11         self.balance += money           # 执行存款
12         print("存款 ", money, " 完成")  # 打印存款完成
13
14     def withdraw_money(self, money):     # 设计提款方法
15         self.balance -= money           # 执行提款
16         print("提款 ", money, " 完成")  # 打印提款完成
17
18     def get_balance(self):               # 获得存款余额
19         print(self.name.title(), " 目前余额: ", self.balance)
20
21 hungbank = Banks('hung')               # 定义对象hungbank
22 print("目前开户银行 ", hungbank.title)  # 列出目前开户银行
23 hungbank.get_balance()                  # 获得hung存款余额
24 hungbank.save_money(100)               # hung存款100
25 hungbank.get_balance()                 # 获得hung存款余额

```

执行结果

```

===== RESTART: D:\Python\ch12\ch12_6.py =====
目前开户银行 Taipei Bank
Hung 目前余额: 0
存款 100 完成
Hung 目前余额: 100
>>>

```

12-2 类的访问权限——封装(encapsulation)

学习类至今可以看到我们可以从程序直接引用类内的属性（可参考ch12_6.py的第22行）与方法（可参考ch12_6.py的第23行），像这种类内的属性可以让外部引用的称公有(public)属性，而可以让外部引用的方法称公有方法。其实前面所使用的Banks类内的属性与方法皆是公有属性与方法。但是程序设计时可以发现，外部直接引用时也代表可以直接修改类内的属性值，这将造成类数据不安全。

因此Python也提供一个私有属性与方法的概念，这个观念的主要精神是类外无法直接更改类内的私有属性，类外也无法直接调用私有方法，这个观念又称封装(encapsulation)。

12-2-1 私有属性

为了确保类内的属性的安全，其实有必要限制外部无法直接获取类内的属性值。

程序实例ch12_7.py: 外部直接获取属性值，造成存款余额不安全的实例。

```
21 hungbank = Banks('hung')          # 定义对象hungbank
22 hungbank.get_balance()
23 hungbank.balance = 10000           # 类外直接篡改存款余额
24 hungbank.get_balance()
```

执行结果

```
===== RESTART: D:\Python\ch12\ch12_7.py =====
Hung 目前余额: 0
Hung 目前余额: 10000
>>>
```

上述程序第23行笔者直接在类外就更改了存款余额了，当第24行列出存款余额时，可以发现在没有经过Banks类内的save_money()方法存钱动作，整个余额就从0元增至10000元。为了避免这种现象产生，Python对于类内的属性增加了私有属性 (private attribute) 的观念，应用方式是定义时在属性名称前面增加__(2个底线)，定义为私有属性 后，类外的程序就无法引用了。

程序实例ch12_8.py: 重新设计ch12_7.py，主要是将Banks类的属性定义为私有属性，这样就无法由外部程序修改了。

```
1 # ch12_8.py
2 class Banks():
3     # 定义银行类
4
5     def __init__(self, uname):          # 初始化方法
6         self.__name = uname            # 设定私有存款者名字
7         self.__balance = 0             # 设定私有开户金额是0
8         self.__title = "Taipei Bank"   # 设定私有银行名称
9
10    def save_money(self, money):          # 设计存款方法
11        self.__balance += money          # 执行存款
12        print("存款 ", money, " 完成")   # 打印存款完成
13
14    def withdraw_money(self, money):      # 设计提款方法
15        self.__balance -= money          # 执行提款
16        print("提款 ", money, " 完成")   # 打印提款完成
17
18    def get_balance(self):                # 获得存款余额
19        print(self.__name.title(), " 目前余额: ", self.__balance)
20
21 hungbank = Banks('hung')               # 定义对象hungbank
22 hungbank.get_balance()
23 hungbank.balance = 10000               # 类外直接篡改存款余额
24 hungbank.get_balance()
```

执行结果

```
===== RESTART: D:\Python\ch12\ch12_8.py =====
Hung 目前余额: 0
Hung 目前余额: 0
>>>
```

请读者留意第6、7和8行笔者设定私有属性的方式，上述程序第23行笔者尝试修改存款余额，但从输出结果可以知道修改失败，因为执行结果的存款余额是0。对上述程序而言，存款余额只会在以存款(`save_money()`)和提款(`withdraw_money()`)方法被触发时，依参数金额更改。

12-2-2 私有方法

既然类有私有的属性，那么也有私有方法 (private method)，它的观念与私有属性类似，类外的程序无法调用。至于定义方式与私有属性相同，只要在方法前面加上`__`(2个底线)符号即可。若是延续上述程序实例，我们可能会遇上换汇的问题，通常银行在换汇时会针对客户对银行的贡献订出不同的汇率与手续费，这个部分是客户无法得知的，碰上这类的应用就很适合以私有方法处理换汇程序，为了简化问题，下列是在初始化类时，先设定美金与台币的汇率以及换汇的手续费，其中汇率(`__rate`)与手续费率(`__service_charge`)皆是私有属性。

```
9      self.__rate = 30          # 预设美金与台币换汇比例
10     self.__service_charge = 0.01 # 换汇的服务费
```

下列是使用者可以调用的公有方法，在这里只能输入换汇的金额。

```
23     def usa_to_taiwan(self, usa_d):      # 美金兑换台币方法
24         self.result = self.__cal_rate(usa_d)
25         return self.result
```

在上述公有方法中调用了__cal_rate(usa_d)，这是私有方法，类外无法调用使用，下列是此私有方法的内容。

```
27     def __cal_rate(self,usa_d):          # 定义换汇是私有方法
28         return int(usa_d * self.__rate * (1 - self.__service_charge))
```

在上述私有方法中可以看到内部包含比较敏感且不适合外部人参与的数据。

程序实例ch12_9.py： 下列是私有方法应用的完整程序代码实例。

```
1  # ch12_9.py
2  class Banks():
3      # 定义银行类
4
5      def __init__(self, uname):          # 初始化方法
6          self.__name = uname            # 设定私有存款者名字
7          self.__balance = 0             # 设定私有开户金额是0
8          self.__title = "Taipei Bank"   # 设定私有银行名称
9          self.__rate = 30               # 预设美金与台币换汇比例
10         self.__service_charge = 0.01    # 换汇的服务费
11
12     def save_money(self, money):          # 设计存款方法
13         self.__balance += money          # 执行存款
14         print("存款 ", money, " 完成")   # 打印存款完成
15
16     def withdraw_money(self, money):      # 设计提款方法
17         self.__balance -= money          # 执行提款
18         print("提款 ", money, " 完成")   # 打印提款完成
19
20     def get_balance(self):                # 获得存款余额
21         print(self.__name.title(), " 目前余额: ", self.__balance)
22
23     def usa_to_taiwan(self, usa_d):       # 美金兑换台币方法
24         self.result = self.__cal_rate(usa_d)
25         return self.result
26
27     def __cal_rate(self,usa_d):           # 定义换汇是私有方法
28         return int(usa_d * self.__rate * (1 - self.__service_charge))
29
30     hungbank = Banks('hung')             # 定义对象hungbank
31     usdallor = 50
32     print(usdallor, " 美金可以兑换 ", hungbank.usa_to_taiwan(usdallor), " 台币")
```

执行结果

```
===== RESTART: D:\Python\ch12\ch12_9.py =====
50 美金可以兑换 1485 台币
>>>
```

12-3 类的继承

在面向对象程序设计中类是可以继承的，其中被继承的类称父类 (parent class) 或基类 (base class)，继承的类称子类 (child

class)或衍生类 (derived class)。类继承的最大优点是许多父类的公有方法 或属性 ，在子类中不用重新设计，可以直接引用。



在程序设计时，基类(base class)必须在衍生类(derived class)前面，整个程序代码结构如下：

```
class BaseClassName( ):                # 先定义基类
    Base Class 的内容
class DerivedClassName(BaseClassName):  # 再定义衍生类
    Derived Class 的内容
```

衍生类继承了基类的公有属性与方法，同时也可以有自己的属性与方法。

12-3-1 衍生类继承基类的实例应用

程序实例ch12_10.py： 延续Banks类建立一个分行Shilin_Banks，这个衍生类没有任何数据，直接引用基类的公有函数，执行银行的存款作业。下列是与ch12_9.py不同的程序代码。

```
30 class Shilin_Banks(Banks):
31     # 定义士林分行
32     pass
33
34 hungbank = Shilin_Banks('hung')    # 定义对象hungbank
35 hungbank.save_money(500)
36 hungbank.get_balance()
```

执行结果

```
===== RESTART: D:\Python\ch12\ch12_10.py =====
存款 500 完成
Hung 目前余额: 500
>>>
```

上述第35和36行所引用的方法就是基类Banks的公有方法。

12-3-2 如何取得基类的私有属性

基于保护原因，类外是无法直接取得类内的私有属性，即使是它的衍生类也无法直接读取，如果真要取得可以使用return方式，传回私有属性内容。

程序实例ch12_11.py： 衍生类对象取得基类的银行名称title的属性。

```
30 def bank_title(self):           # 获得银行名称
31     return self.__title
32
33 class Shilin_Banks(Banks):
34     # 定义士林分行
35     pass
36
37 hungbank = Shilin_Banks('hung') # 定义对象hungbank
38 print("我的存款银行是：", hungbank.bank_title())
```

执行结果

```
===== RESTART: D:\Python\ch12\ch12_11.py =====
我的存款银行是： Taipei Bank
>>>
```

12-3-3 衍生类与基类有相同名称的属性

程序设计时，衍生类也可以有自己的初始化__init__()方法，同时也有可能衍生类的属性与方法名称和基类重复，碰上这个状况Python会先找寻衍生类是否有这个名称，如果有则先使用，如果没有则使用基类的名称内容。

程序实例ch12_12.py： 这个程序主要是将Banks类的title属性改为公有属性，但是在衍生类中则有自己的初始化方法，主要是基类与衍生类均有title属性，不同类对象将呈现不同的结果。下列是第8行的内容。

```
8 self.title = "Taipei Bank"      # 设定公有银行名称
```

下列是修改部分程序代码内容。

```
33 class Shilin_Banks(Banks):
34     # 定义士林分行
35     def __init__(self, uname):
36         self.title = "Taipei Bank - Shilin Branch" # 定义分行名称
37
38 jamesbank = Banks('James') # 定义Banks类对象
39 print("James's banks = ", jamesbank.title) # 打印银行名称
40 hungbank = Shilin_Banks('Hung') # 定义Shilin_Banks类对象
41 print("Hung's banks = ", hungbank.title) # 打印银行名称
```

执行结果

```
===== RESTART: D:/Python/ch12/ch12_12.py =====
James's banks = Taipei Bank
Hung's banks = Taipei Bank - Shilin Branch
>>>
```

从上述可知Banks类对象James所使用的title属性是Taipei Bank，Shilin_Banks类对象Hung所使用的title属性是Taipei Bank - Shilin Branch。

12-3-4 衍生类与基类有相同名称的方法

程序设计时，衍生类也可以有自己的方法，同时也有可能衍生类的方法名称和基类方法名称重复，碰上这个状况Python会先找寻衍生类是否有这个名称，如果有则先使用，如果没有则使用基类的名称内容。

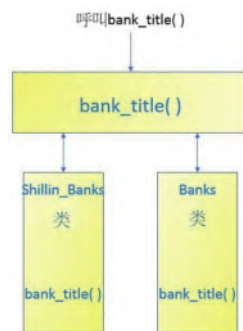
程序实例ch12_13.py： 衍生类与基类名称重复的实例，这个程序的基类与衍生类均有bank_title()函数，Python会触发bank_title()方法的对象去判别应使用哪一个方法执行。

```
30     def bank_title(self): # 获得银行名称
31         return self.__title
32
33 class Shilin_Banks(Banks):
34     # 定义士林分行
35     def __init__(self, uname):
36         self.title = "Taipei Bank - Shilin Branch" # 定义分行名称
37     def bank_title(self): # 获得银行名称
38         return self.title
39
40 jamesbank = Banks('James') # 定义Banks类对象
41 print("James's banks = ", jamesbank.bank_title()) # 打印银行名称
42 hungbank = Shilin_Banks('Hung') # 定义Shilin_Banks类对象
43 print("Hung's banks = ", hungbank.bank_title()) # 打印银行名称
```


执行结果

```
===== RESTART: D:/Python/ch12/ch12_13.py =====  
James's banks = Taipei Bank  
Hung's banks = Taipei Bank - Shilin Branch  
>>>
```

上述程序的概念如下：



上述第30行的`bank_title()`属于`Banks`类，第37行的`bank_title()`属于`Shilin_Banks`类。第40行是`Banks`对象，所以第41行会触发第30行的`bank_title()`方法。第42行是`Shilin_Banks`对象，所以第43行会触发第37行的`bank_title()`方法。其实上述方法就是面向对象的多型（polymorphism），但是多型不一定需要是有父子关系的类。读者可以将以上想成方法多功能化，相同的函数名称，放入不同类型的对象可以产生不同的结果。至于使用者不需要知道是如何设计的，隐藏在内部的设计细节交由程序设计师负责。12-4节笔者还会举实例说明。

12-3-5 衍生类引用基类的方法

衍生类引用基类的方法时需使用`super()`，下列将使用另一类的类了解这个观念。

程序实例ch12_14.py： 这是一个衍生类调用基类方法的实例，笔者首先建立一个`Animals`类，然后建立这个类的衍生类`Dogs`，`Dogs`类在初

始化中会使用super()调用Animals类的初始化方法，可参考第14行，经过初始化处理后，mydog.name将由“lily”变为“My pet lily”。

```
1 # ch12_14.py
2 class Animals():
3     """Animals类，这是基类"""
4     def __init__(self, animal_name, animal_age):
5         self.name = animal_name # 记录动物名称
6         self.age = animal_age # 记录动物年龄
7
8     def run(self):
9         # 输出动物 is running
10        print(self.name.title(), " is running")
11
12 class Dogs(Animals):
13     """Dogs类，这是Animal的衍生类"""
14     def __init__(self, dog_name, dog_age):
15         super().__init__('My pet ' + dog_name.title(), dog_age)
16
17 mycat = Animals('lucy', 5) # 建立Animals对象以及测试
18 print(mycat.name.title(), ' is ', mycat.age, " years old.")
19 mycat.run()
20
21 mydog = Dogs('lily', 6) # 建立Dogs对象以及测试
22 print(mydog.name.title(), ' is ', mydog.age, " years old.")
23 mydog.run()
```

执行结果

```
===== RESTART: D:\Python\ch12\ch12_14.py =====
Lucy is 5 years old.
Lucy is running
My Pet Lily is 6 years old.
My Pet Lily is running
>>>
```

12-3-6 三代同堂的类与取得基类的属性super()

在继承观念里，我们也可以使用Python的super()方法取得基类的属性，这对于设计三代同堂的类是很重要的。

下列是一个三代同堂的程序，在这个程序中有祖父(Grandfather)类，它的子类是父亲(Father)类，父亲类的子类是Ivan类。其实Ivan要取得父亲类的属性很容易，可是要取得祖父类的属性时就会碰上困难，解决方式是在Father类与Ivan类的__init__()方法中增加下列设定：

```
super( ).__init__( ) # 将父类的属性复制
```

这样就可以使Ivan取得祖父(Grandfather)类的属性了。

程序实例ch12_15.py： 这个程序会建立一个Ivan类的对象ivan，然后分别调用Father类和Grandfather类的方法打印信息，接着分别取得Father类和Grandfather类的属性。

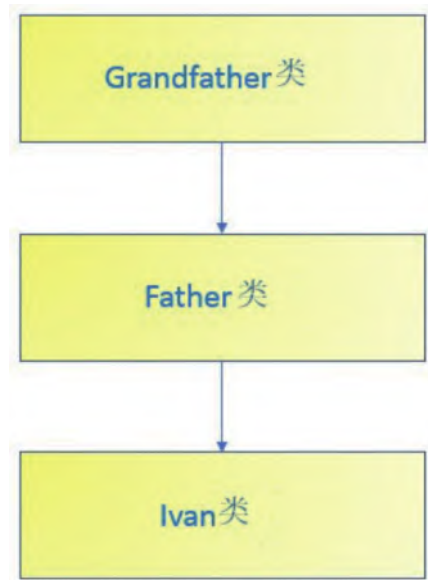
```
1 # ch12_15
2 class Grandfather():
3     """ 定义祖父的资产 """
4     def __init__(self):
5         self.grandfathermoney = 10000
6     def get_info1(self):
7         print("Grandfather's information")
8
9 class Father(Grandfather):    # 父类是Grandfather
10    """ 定义父亲的资产 """
11    def __init__(self):
12        self.fathermoney = 8000
13        super().__init__()
14    def get_info2(self):
15        print("Father's information")
16
17 class Ivan(Father):          # 父类是Father
18    """ 定义Ivan的资产 """
19    def __init__(self):
20        self.ivanmoney = 3000
21        super().__init__()
22    def get_info3(self):
23        print("Ivan's information")
24    def get_money(self):        # 取得资产明细
25        print("\nIvan资产: ", self.ivanmoney,
26              "\n父亲资产: ", self.fathermoney,
27              "\n祖父资产: ", self.grandfathermoney)
28
29 ivan = Ivan()
30 ivan.get_info3()              # 从Ivan中获得
31 ivan.get_info2()             # 流程 Ivan -> Father
32 ivan.get_info1()             # 流程 Ivan -> Father -> Grandfather
33 ivan.get_money()             # 取得资产明细
```

执行结果

```
===== RESTART: D:\Python\ch12\ch12_15.py =====
Ivan's information
Father's information
Grandfather's information

Ivan资产: 3000
父亲资产: 8000
祖父资产: 10000
>>>
```

上述程序各类的相关图形如下：



12-3-7 兄弟类属性的取得

假设有一个父亲(Father)类，这个父亲类有2个儿子，分别是Ivan类和Ira类，如果Ivan类想取得Ira类的属性iramoney，可以使用下列方法。

```
Ira( ).iramoney # Ivan取得Ira的属性iramoney
```

程序实例ch12_16.py： 设计3个类，Father类是Ivan和Ira类的父类，所以Ivan和Ira算是兄弟类，这个程序可以从Ivan类分别读取Father和Ira类的资产属性。这个程序最重要的是第21行，请留意取得Ira属性的写法。

```

1 # ch12_16
2 class Father():
3     """ 定义父亲的资产 """
4     def __init__(self):
5         self.fathermoney = 10000
6
7 class Ira(Father):
8     """ 定义Ira的资产 """
9     def __init__(self):
10         self.iramoney = 8000
11         super().__init__()
12
13 class Ivan(Father):
14     """ 定义Ivan的资产 """
15     def __init__(self):
16         self.ivanmoney = 3000
17         super().__init__()
18     def get_money(self):
19         print("Ivan资产: ", self.ivanmoney,
20               "\n父亲资产: ", self.fathermoney,
21               "\nIra资产: ", Ira().iramoney)
22
23 ivan = Ivan()
24 ivan.get_money()

```

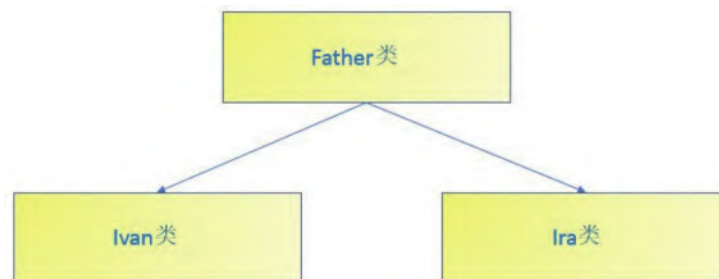
执行结果

```

===== RESTART: D:\Python\ch12\ch12_16.py =====
Ivan资产: 3000
父亲资产: 10000
Ira资产: 8000
>>>

```

上述程序各类的相关图形如下：



12-4 多型(polymorphism)

在12-3-4节笔者已经有说明基类与衍生类有相同方法名称的实例，其实那就是本节欲说明的**多型** (polymorphism)的基本观念，但是**多型** (polymorphism)的观念是不局限在必须有父子关系的类中的。

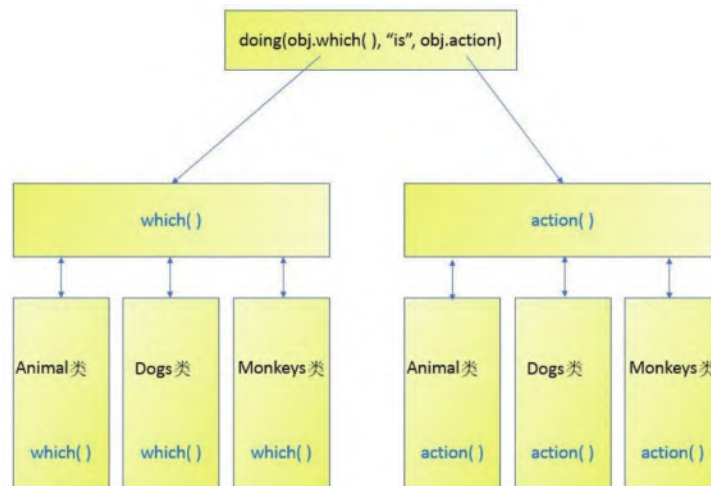
程序实例ch12_17.py: 这个程序有3个类, Animals类是基类, Dogs类是Animals类的衍生类, 基于继承的特性所以2个类皆有which()和action()方法, 另外设计了一个与上述无关的类Monkeys, 这个类也有which()和action()方法, 然后程序分别调用which()和action()方法, 程序会由对象类判断应该使用哪一个方法响应程序。

```
1 # ch12_17.py
2 class Animals():
3     """Animals类, 这是基类"""
4     def __init__(self, animal_name):
5         self.name = animal_name # 纪录动物名称
6     def which(self):
7         return 'My pet ' + self.name.title() # 回传动物名称
8     def action(self):
9         return ' sleeping' # 动物的行为
10
11 class Dogs(Animals):
12     """Dogs类, 这是Animal的衍生类"""
13     def __init__(self, dog_name):
14         super().__init__(dog_name.title()) # 纪录动物名称
15     def action(self):
16         return ' running in the street' # 动物的行为
17
18 class Monkeys():
19     """猴子类, 这是其他类"""
20     def __init__(self, monkey_name):
21         self.name = 'My monkey ' + monkey_name.title() # 纪录动物名称
22     def which(self):
23         return self.name # 回传动物名称
24     def action(self):
25         return ' running in the forest' # 动物的行为
26
27 def doing(obj):
28     print(obj.which(), "is", obj.action()) # 列出动物的行为
29
30 my_cat = Animals('lucy') # Animals物件
31 doing(my_cat)
32 my_dog = Dogs('gimi') # Dogs物件
33 doing(my_dog)
34 my_monkey = Monkeys('taylor') # Monkeys物件
35 doing(my_monkey)
```

执行结果

```
===== RESTART: D:/Python/ch12/ch12_17.py =====
My pet Lucy is sleeping
My pet Gimi is running in the street
My monkey Taylor is running in the forest
>>>
```

上述程序各类的相关图形如下:



对上述程序而言，第30行的my_cat是Animal类对象，所以在31行此对象会触发Animal类的which()和action()方法。第32行的my_dog是Dogs类对象，所以在33行此对象会触发Dogs类的which()和action()方法。第34行的my_monkey是Monkeys类对象，所以在35行此对象会触发Monkeys类的which()和action()方法。

12-5 多重继承

在面向对象的程序设计中，也常会发生一个类继承多个类的应用，此时子类也同时继承了多个类的方法。在这个时候，读者应该了解当多个父类拥有相同名称的方法时，应该先执行哪一个父类的方法。在程序中可用下列语法代表继承多个类。

```
class 类名称(父类1, 父类2, ... , 父类n):
    类内容
```

程序实例ch12_18.py： 这个程序Ivan类继承了Father和Uncle类，Grandfather类则是Father和Uncle类的父类。在这个程序中笔者只设定一个Ivan类的对象ivan，然后由这个类分别调用action3()、action2()和action1()，其中Father和Uncle类同时拥有action2()方法，读者可以观察最后是执行哪一个action2()方法。


```

1 # ch12_18.py
2 class Grandfather():
3     """ 定义祖父类 """
4     def action1(self):
5         print("Grandfather")
6
7 class Father(Grandfather):
8     """ 定义父亲类 """
9     def action2(self):
10         print("Father")
11
12 class Uncle(Grandfather):
13     """ 定义叔父类 """
14     def action2(self):
15         print("Uncle")
16
17 class Ivan(Father, Uncle):
18     """ 定义Ivan类 """
19     def action3(self):
20         print("Ivan")
21
22 ivan = Ivan()
23 ivan.action3()
24 ivan.action2()
25 ivan.action1()

```

顺序 Ivan
顺序 Ivan -> Father
顺序 Ivan -> Father -> Grandfather

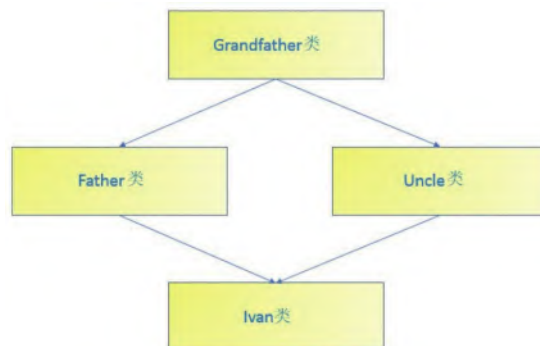
执行结果

```

===== RESTART: D:\Python\ch12\ch12_18.py =====
Ivan
Father
Grandfather
>>>

```

上述程序各类的相关图形如下：



程序实例ch12_19.py： 这个程序基本上是重新设计ch12_18.py，主要是Father和Uncle类的方法名称不一样，Father类是action3()，Uncle类是action2()，这个程序在建立Ivan类的ivan对象后，会分别启动各类的actionX()方法。


```

1 # ch12_19.py
2 class Grandfather():
3     """ 定义祖父类 """
4     def action1(self):
5         print("Grandfather")
6
7 class Father(Grandfather):
8     """ 定义父亲类 """
9     def action3(self): # 定义action3()
10        print("Father")
11
12 class Uncle(Grandfather):
13     """ 定义叔叔类 """
14     def action2(self): # 定义action2()
15        print("Uncle")
16
17 class Ivan(Father, Uncle):
18     """ 定义Ivan类 """
19     def action4(self):
20        print("Ivan")
21
22 ivan = Ivan()
23 ivan.action4() # 顺序 Ivan
24 ivan.action3() # 顺序 Ivan -> Father
25 ivan.action2() # 顺序 Ivan -> Father -> Uncle
26 ivan.action1() # 顺序 Ivan -> Father -> Uncle -> Grandfather

```

执行结果

```

===== RESTART: D:/Python/ch12/ch12_19.py =====
Ivan
Father
Uncle
Grandfather
>>>

```

12-6 type与instance

一个大型程序设计可能是由许多人合作设计，有时我们想了解某个对象变量的数据类型，或是所属类关系，可以使用本节所述的方法。

12-6-1 type()

这个函数先前已经使用许多次了，可以使用type()函数得到某一对象变量的类。

程序实例ch12_20.py： 列出类对象与对象内方法的数据类型。

```

1 # ch12_20.py
2 class Grandfather():
3     """ 定义祖父类 """
4     pass
5
6 class Father(Grandfather):
7     """ 定义父亲类 """
8     pass
9
10 class Ivan(Father):
11     """ 定义Ivan类 """
12     def fn(self):
13         pass
14
15 grandfather = Grandfather()
16 father = Father()
17 ivan = Ivan()
18 print("grandfather对象类型: ", type(grandfather))
19 print("father对象类型      : ", type(father))
20 print("ivan对象类型        : ", type(ivan))
21 print("ivan对象fn方法类型 : ", type(ivan.fn))

```

执行结果

```

===== RESTART: D:\Python\ch12\ch12_20.py =====
grandfather对象类型: <class '__main__.Grandfather'>
father对象类型      : <class '__main__.Father'>
ivan对象类型        : <class '__main__.Ivan'>
ivan对象fn方法类型 : <class 'method'>
>>>

```

由上图可以得到类的对象类型是class，同时会列出“__main__.
类的名称”。如果是类内的方法同时也列出“method”方法。

12-6-2 isinstance()

isinstance()函数可以传回对象的类是否属于某一类，它包含2个参数，它的语法如下：

isinstance(对象, 类) # 可传回True或False

如果对象的类是属于第2个参数类 或属于第2个参数的子类， 则传回True，否则传回False。

程序实例ch12_21.py： 一系列isinstance()函数的测试。

```

1 # ch12_21.py
2 class Grandfather():
3     """ 定义祖父类 """
4     pass
5
6 class Father(Grandfather):
7     """ 定义父亲类 """
8     pass
9
10 class Ivan(Father):
11     """ 定义Ivan类 """
12     def fn(self):
13         pass
14
15 grandfa = Grandfather()
16 father = Father()
17 ivan = Ivan()
18 print("ivan属于Ivan类: ", isinstance(ivan, Ivan))
19 print("ivan属于Father类: ", isinstance(ivan, Father))
20 print("ivan属于GrandFather类: ", isinstance(ivan, Grandfather))
21 print("father属于Ivan类: ", isinstance(father, Ivan))
22 print("father属于Father类: ", isinstance(father, Father))
23 print("father属于Grandfather类: ", isinstance(father, Grandfather))
24 print("grandfa属于Ivan类: ", isinstance(grandfa, Ivan))
25 print("grandfa属于Father类: ", isinstance(grandfa, Father))
26 print("grandfa属于Grandfather类: ", isinstance(grandfa, Grandfather)))

```

执行结果

```

===== RESTART: D:\Python\ch12\ch12_21.py =====
ivan属于Ivan类: True
ivan属于Father类: True
ivan属于GrandFather类: True
father属于Ivan类: False
father属于Father类: True
father属于Grandfather类: True
grandfa属于Ivan类: False
grandfa属于Father类: False
grandfa属于Grandfather类: True
>>>

```

12-7 特殊属性

当设计或是看到别人设计的Python程序时，若是遇到__xx__类的字符串就要特别留意了，这些大多数是特殊属性或方法，笔者将简要说明几个重要常见的属性。

12-7-1 文档字符串_doc__

文档字符串的英文原意是文档字符串(docstring)，Python鼓励程序设计师在设计函数或类时，尽量为函数或类增加文档的批注，未来可以使用__doc__特殊属性列出此文档批注。

程序实例ch12_22.doc: 将文档批注应用在函数。

```

1 # ch12_22.py
2 def getMax(x, y):
3     '''文档字符串实例
4     建议x, y是整数
5     这个函数将传回较大值'''
6     if int(x) > int(y):
7         return x
8     else:
9         return y
10
11 print(getMax(2, 3))          # 打印较大值
12 print(getMax.__doc__)       # 打印文档字符串docstring

```

执行结果

```

===== RESTART: D:\Python\ch12\ch12_22.py =====
3
文档字符串实例
建议x, y是整数
这个函数将传回较大值
>>>

```

程序实例ch12_23. doc: 将文档批注应用在类与类内的方法。

```

1 # ch12_23.py
2 class Myclass:
3     '''文档字符串实例
4     Myclass类别的应用'''
5     def __init__(self, x):
6         self.x = x
7     def printMe(self):
8         '''文档字符串实例
9         Myclass类内printMe方法的应用'''
10        print("Hi", self.x)
11
12 data = Myclass(100)
13 data.printMe()
14 print(data.__doc__)          # 打印Myclass文档字符串docstring
15 print(data.printMe.__doc__)  # 打印printMe文档字符串docstring

```

执行结果

```

===== RESTART: D:\Python\ch12\ch12_23.py =====
Hi 100
文档字符串实例
Myclass类别的应用
文档字符串实例
Myclass类内printMe方法的应用
>>>

```

了解以上观念后，如果读者看到有一个程序代码如下：

```

>>> x = 'abc'
>>> print(x.__doc__)
str(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or
errors is specified, then the object must expose a data buffer
that will be decoded using the given encoding and error handler.
Otherwise, returns the result of object.__str__() (if defined)
or repr(object).
encoding defaults to sys.getdefaultencoding().
errors defaults to 'strict'.
>>>

```

以上只是列出Python系统内部有关字符串的docstring。

12-7-2 __name__属性

如果你是Python程序设计师，常在网络上看到别人写的程序，一定会经常在程序末端看到下列叙述：

```
if __name__ == '__main__':  
    doSomething( )
```

初学Python时，笔者照上述撰写，程序一定可以执行，当时不晓得意义，现在觉得应该要告诉读者。如果上述程序是自己执行，那么__name__就一定是__main__。

程序实例ch12_24.py： 一个程序只有一行，就是打印__name__。

```
1 # ch12_24.py  
2 print('ch12_24.py module name = ', __name__)
```

执行结果

```
===== RESTART: D:/Python/ch12/ch12_24.py =====  
ch12_24.py module name = __main__  
>>>
```

经过上述实例，我们知道，如果程序是自己执行，__name__就是__main__。所以下列程序实例可以列出结果。

程序实例ch12_25.py： __name__ == __main__的应用。

```
1 # ch12_25.py  
2 def myFun():  
3     print("__name__ == __main__")  
4 if __name__ == '__main__':  
5     myFun()
```

执行结果

```
===== RESTART: D:/Python/ch12/ch12_25.py =====  
__name__ == __main__  
>>>
```

如果ch12_24.py是被import到另一个程序，则__name__是本身的文件名。

程序实例ch12_26.py： 这个程序import导入ch12_24.py，结果__name__变成了ch12_24。

```
1 # ch12_26.py
2 import ch12_24
```

执行结果

```
===== RESTART: D:/Python/ch12/ch12_26.py
ch12_24.py module name = ch12_24
>>>
```

程序实例ch12_27.py： 这个程序import导入ch12_25.py，由于__name__已经不再是__main__，所以程序没有任何输出。

```
1 # ch12_27.py
2 import ch12_25
```

执行结果

```
===== RESTART: D:/Python/ch12/ch12_27.py
>>>
```

所以总结就是__name__可以判别这个程序是自己执行或是被其他程序import导入当成模块使用。

12-8 类的特殊方法

12-8-1 __str__()方法

这是类的特殊方法，可以协助返回易读取的字符串。

程序实例ch12_28.py： 在没有定义__str__()方法下，列出类的对象。

```

1 # ch12_28.py
2 class Name:
3     def __init__(self, name):
4         self.name = name
5
6 a = Name('Hung')
7 print(a)

```

执行结果

```

===== RESTART: D:/Python/ch12/ch12_28.py ==
<_main__.Name object at 0x0291AC70>
>>>

```

上述在没有定义__str__()方法下，我们获得了一个不太容易阅读的结果。

程序实例ch12_29.py： 在定义__str__()方法下，重新设计上一个程序。

```

1 # ch12_29.py
2 class Name:
3     def __init__(self, name):
4         self.name = name
5     def __str__(self):
6         return '%s' % self.name
7
8 a = Name('Hung')
9 print(a)

```

执行结果

```

===== RESTART: D:/Python/ch12/ch12_29.py
Hung
>>>

```

上述定义了__str__()方法后，就得到一个适合阅读的结果了。对于程序ch12_29.py而言，如果我们在Python Shell窗口输入a，将一样获得不容易阅读的结果。

```

===== RESTART: D:/Python/ch12/ch12_29.py =====
Hung
>>> a
<_main__.Name object at 0x013EAC70>
>>>

```

12-8-2 __repr__()方法

上述原因是，如果只是在Python Shell窗口输入类变量a，系统是调用__repr__()方法做响应，为了要获得容易阅读的结果，我们也需要定义此方法。

程序实例ch12_30.py： 定义__repr__()方法，其实此方法内容与__str__()相同，所以可以用等号取代。

```
1 # ch12_30.py
2 class Name:
3     def __init__(self, name):
4         self.name = name
5     def __str__(self):
6         return '%s' % self.name
7     __repr__ = __str__
8
9 a = Name('Hung')
10 print(a)
```

执行结果

```
===== RESTART: D:/Python/ch12/ch12_30.py
Hung
>>> a
Hung
>>> |
```

12-8-3 __iter__()方法

建立类的时候也可以将类定义成一个迭代对象，类似list或tuple，供for ... in循环内使用，这时类需设计next()方法，取得下一个值，直到达到结束条件，可以使用raise StopIteration (第15章会解说，raise)终止继续。

程序实例ch12_31.py： Fib序列数的设计。


```

1 # ch12_31.py
2 class Fib():
3     def __init__(self, max):
4         self.max = max
5
6     def __iter__(self):
7         self.a = 0
8         self.b = 1
9         return self
10
11    def __next__(self):
12        fib = self.a
13        if fib > self.max:
14            raise StopIteration
15        self.a, self.b = self.b, self.a + self.b
16        return fib
17 for i in Fib(100):
18     print(i)

```

执行结果

```

===== RESTART: D:/Python/ch12/ch12_31.py
0
1
1
2
3
5
8
13
21
34
55
89
>>>

```

习题

1. 设计一个类Myschool，这个类包含属性name和score，这个类也有一个msg()方法，程序设定Myschool对象时需传递2个参数，下列是示范设定方式：

```
hung = Myschool('kevin', 80)
```

这个类的方法，主要是可以输出问候语和成绩，下列是示范输出。

Hi! Kevin, 你的成绩是80分。

请注意英文名字第一个输出字母是大写。

2. 请扩充习题1，增加初始化title属性，title内容是‘Python School’，请设计msg()方法输出第一行是title，第二行才是原

先的输出。

3. 请利用ch12_9.py的类，同时修改部分内容，在程序部分执行下列工作：

A: 存款5000元

B: 提款3000元

C: 存款1500元

D: 购买美金外币100美金

E: 列出剩余金额请列出上述每次的执行结果账单。

4. 请扩充ch12_13.py，增加Banks子类北投(Beitou)分行，北投分行内容可以参照士林分行，程序末端增加北投分行类对象（可参考43行），然后打印银行名称（可参考44行）。

5. 请扩充ch12_14.py，为Animals类增加Birds子类，这个子类有自己的run()方法，输出方式可以比照第9行，但是字符串是“ is flying.”。请为这个程序增加类似20到22行的工作，但是将对象类设为Birds。

6. 请扩充ch12_15.py，增加Grandmother类，这是Father类的父类，它的资产是20000，请参考Grandfather类建立get_info4()方法，同时在程序中扩充输出Grandmother的资产。

7. 请适度修订ch12_16.py，将第23行语句改为：

```
ira = Ira( )
```

第24行也需修改，在Ira类内增加可以调用Ivan类的get_money()方法，然后输出结果。

8. 请扩充ch12_18.py，增加Grandfather类的子类Aunt类，这个类也是Ivan类的父类。请参考第14行建立action2()方法但是列出

“Aunt”。在第17行Ivan类内的参数如下：

```
Father, Uncle, Aunt
```

请再设计2个程序参数分别如下：

```
Uncle, Aunt, Father  
Aunt, Father, Uncle
```

同时列出结果。

第13章 设计与应用模块

第11章笔者介绍了函数(function)，第12章笔者介绍了类(class)，其实在大型计划的程序设计中，每个人可能只是负责一小功能的函数或类设计，为了可以让团队的其他人可以互相分享设计成果，最后每个人所负责的功能函数或类将存储在**模块** (module)中，然后供团队其他成员使用。在网络上或国外的技术文件常可以看到有的文章将**模块** (module)称为**套件** (package)，意义是一样的。

本章笔者将讲解如何将自己所设计的函数或类存储成模块然后加以引用，最后也将讲解Python常用的内置模块。Python最大的优势是免费资源，因此有大量公司使用它开发了大量功能强大的模块，笔者将在第二篇迈向Python高手之路，详细说明使用外部模块执行更多有意义的工作。

13-1 将自建的函数存储在模块中

一个大型程序一定是由许多的函数或类所组成，为了让程序的工作可以分工以及增加程序的可读性，我们可以将所建的函数或类存储成模块(module)形式的独立文件，未来再加以调用引用。

13-1-1 先前准备工作

假设有一个程序内容是用于建立冰淇淋(ice cream)与饮料(drink)，如下所示：

程序实例ch13_1.py： 这个程序基本上是扩充ch11_23.py，再增加建立饮料的函数。

```

1 # ch13_1.py
2 def make_icecream(*toppings):
3     # 列出制作冰淇淋的配料
4     print("这个冰淇淋所加配料如下")
5     for topping in toppings:
6         print("--- ", topping)
7
8 def make_drink(size, drink):
9     # 输入饮料规格与种类, 然后输出饮料
10    print("所点饮料如下")
11    print("--- ", size.title())
12    print("--- ", drink.title())
13
14    make_icecream('草莓酱')
15    make_icecream('草莓酱', '葡萄干', '巧克力碎片')
16    make_drink('large', 'coke')

```

执行结果

```

===== RESTART: D:\Python\ch13\ch13_1.py
这个冰淇淋所加配料如下
--- 草莓酱
这个冰淇淋所加配料如下
--- 草莓酱
--- 葡萄干
--- 巧克力碎片
所点饮料如下
--- Large
--- Coke
>>>

```

假设我们会常常需要在其他程序调用 `make_icecream()` 和 `make_drink()`，此时可以考虑将这2个函数建立成模块（module），未来可以供其他程序调用使用。

13-1-2 建立函数内容的模块

模块的扩展名与Python程序文件一样，是py，对于程序实例 `ch13_1.py` 而言，我们可以只保留 `make_icecream()` 和 `make_drink()`。

程序实例makefood.py： 使用 `ch13_1.py` 建立一个模块，此模块名称是 `makefood.py`。

```

1 # makefood.py
2 # 这是一个包含2个函数的模块(module)
3 def make_icecream(*toppings):
4     # 列出制作冰淇淋的配料
5     print("这个冰淇淋所加配料如下")
6     for topping in toppings:
7         print("--- ", topping)
8
9 def make_drink(size, drink):
10    # 输入饮料规格与种类,然后输出饮料
11    print("所点饮料如下")
12    print("--- ", size.title())
13    print("--- ", drink.title())

```

执行结果 由于这不是程序所以没有任何执行结果。

现在我们已经成功地建立模块makefood.py了。

13-2 应用自己建立的函数模块

有几种方法可以应用函数模块，下列将分成6小节说明。

13-2-1 import模块名称

要导入13-1-2节所建的模块，只要在程序内加上下列简单的语法即可：

```
import 模块名称 # 导入模块
```

若以13-1-2节的实例，只要在程序内加上下列简单的语法即可：

```
import makefood
```

程序中要引用模块的函数语法如下：

```
模块名称.函数名称 # 模块名称与函数名称间有小数点“.”
```

程序实例ch13_2.py： 实际导入模块makefood.py的应用。

```

1 # ch13_2.py
2 import makefood          # 导入模块makefood.py
3
4 makefood.make_icecream('草莓酱')
5 makefood.make_icecream('草莓酱', '葡萄干', '巧克力碎片')
6 makefood.make_drink('large', 'coke')

```

执行结果 与ch13_1.py相同。

13-2-2 导入模块内特定单一函数

如果只想导入模块内单一特定的函数，可以使用下列语法：

`from 模块名称 import 函数名称`

未来程序引用所导入的函数时可以省略模块名称。

程序实例 ch13_3.py： 这个程序只导入 makefood.py 模块的 make_icecream()函数，所以程序第4和5行执行没有问题，但是执行程序第6行时就会产生错误。

```

1 # ch13_3.py
2 from makefood import make_icecream # 导入模块makefood.py的函数make_icecream
3
4 make_icecream('草莓酱')
5 make_icecream('草莓酱', '葡萄干', '巧克力碎片')
6 make_drink('large', 'coke')      # 因为没有导入此函数所以会产生错误

```

执行结果

```

===== RESTART: D:\Python\ch13\ch13_3.py =====
这个冰淇淋所加配料如下
--- 草莓酱
这个冰淇淋所加配料如下
--- 草莓酱
--- 葡萄干
--- 巧克力碎片
Traceback (most recent call last):
  File "D:\Python\ch13\ch13_3.py", line 6, in <module>
    make_drink('large', 'coke')      # 因为没有导入此函数所以会产生错误
NameError: name 'make_drink' is not defined
>>>

```

13-2-3 导入模块内多个函数

`from 模块名称 import 函数名称1, 函数名称2, ... , 函数名称n`

如果想导入模块内多个函数，函数名称间需以逗号隔开，语法如下：

程序实例ch13_4.py： 重新设计ch13_3.py，增加导入makedrink()函数。

```
1 # ch13_4.py
2 # 导入模块makefood.py的make_icecream和make_drink函数
3 from makefood import make_icecream, make_drink
4
5 make_icecream('草莓酱')
6 make_icecream('草莓酱', '葡萄干', '巧克力碎片')
7 make_drink('large', 'coke')
```

执行结果 与ch13_1.py相同。

13-2-4 导入模块所有函数

如果想导入模块内所有函数，语法如下：

```
from 模块名称 import *
```

程序实例ch13_5.py： 导入模块所有函数的应用。

```
1 # ch13_5.py
2 from makefood import *      # 导入模块makefood.py所有函数
3
4 make_icecream('草莓酱')
5 make_icecream('草莓酱', '葡萄干', '巧克力碎片')
6 make_drink('large', 'coke')
```

执行结果 与ch13_1.py相同。

13-2-5 使用as给函数指定替代名称

有时候会碰上所设计程序的函数名称与模块内的函数名称相同，或是感觉模块的函数名称太长，此时可以自行给模块的函数名称一个替代名称，未来可以使用这个替代名称代替原先模块的名称。语法格式如下：

```
from 模块名称 import 函数名称 as 替代名称
```


程序实例ch13_6.py： 使用替代名称icecream代替make_icecream，重新设计ch13_3.py。

```
1 # ch13_6.py
2 # 使用icecream替代make_icecream函数名称
3 from makefood import make_icecream as icecream
4
5 icecream('草莓酱')
6 icecream('草莓酱', '葡萄干', '巧克力碎片')
```

执行结果

```
===== RESTART: D:\Python\ch13\ch13_6.py =====
这个冰淇淋所加配料如下
--- 草莓酱
这个冰淇淋所加配料如下
--- 草莓酱
--- 葡萄干
--- 巧克力碎片
>>>
```

13-2-6 使用as给模块指定替代名称

Python也允许给模块替代名称，未来可以使用此替代名称导入模块，其语法格式如下：

```
import 模块名称 as 替代名称
```

程序实例ch13_7.py： 使用m当作模块替代名称，重新设计ch13_2.py。

```
1 # ch13_7.py
2 import makefood as m          # 导入模块makefood.py的替代名称m
3
4 m.make_icecream('草莓酱')
5 m.make_icecream('草莓酱', '葡萄干', '巧克力碎片')
6 m.make_drink('large', 'coke')
```

执行结果 与ch13_1.py相同。

13-3 将自建的类存储在模块内

第12章笔者介绍了类，当程序设计越来越复杂时，可能我们也会建立许多类，Python也允许我们将所建立的类储存在模块内，这将是

本节的重点。

13-3-1 先前准备工作

笔者将使用第12章的程序实例，说明将类储存在模块的方式。

程序实例ch13_8.py： 笔者修改了ch12_13.py，简化了Banks类，同时让程序有2个类，至于程序内容读者应该可以轻易了解。

```
1 # ch13_8.py
2 class Banks():
3     # 定义银行类
4
5     def __init__(self, uname):          # 初始化方法
6         self.__name = uname            # 设定私有存款者名字
7         self.__balance = 0             # 设定私有开户金额是0
8         self.__title = "Taipei Bank"   # 设定私有银行名称
9
10    def save_money(self, money):         # 设计存款方法
11        self.__balance += money         # 执行存款
12        print("存款 ", money, " 完成")  # 打印存款完成
13
14    def withdraw_money(self, money):     # 设计提款方法
15        self.__balance -= money         # 执行提款
16        print("提款 ", money, " 完成")  # 打印提款完成
17
18    def get_balance(self):               # 获得存款余额
19        print(self.__name.title(), " 目前余额: ", self.__balance)
20
21    def bank_title(self):                # 获得银行名称
22        return self.__title
23
24    class Shilin_Banks(Banks):
25        # 定义士林分行
26        def __init__(self, uname):
27            self.title = "Taipei Bank - Shilin Branch" # 定义分行名称
28        def bank_title(self):
29            return self.title
30
31    jamesbank = Banks('James')          # 定义Banks类对象
32    print("James's banks = ", jamesbank.bank_title()) # 打印银行名称
33    jamesbank.save_money(500)            # 存钱
34    jamesbank.get_balance()              # 列出存款金额
35    hungbank = Shilin_Banks('Hung')      # 定义Shilin_Banks类对象
36    print("Hung's banks = ", hungbank.bank_title()) # 打印银行名称
```

执行结果

```
===== RESTART: D:\Python\ch13\ch13_8.py =====
James's banks = Taipei Bank
存款 500 完成
James 目前余额: 500
Hung's banks = Taipei Bank - Shilin Branch
>>>
```

13-3-2 建立类内容的模块

模块的扩展名与Python程序文件一样，是py，对于程序实例ch13_8.py而言，我们可以只保留Banks类和Shilin_Banks类。

程序实例banks.py： 使用ch13_8.py建立一个模块，此模块名称是banks.py。

```
1 # banks.py
2 # 这是一个包含2个类的模块(module)
3 class Banks():
4     # 定义银行类
5     def __init__(self, uname):          # 初始化方法
6         self.__name = uname            # 设定私有存款者名字
7         self.__balance = 0             # 设定私有开户金额是0
8         self.__title = "Taipei Bank"   # 设定私有银行名称
9
10    def save_money(self, money):         # 设计存款方法
11        self.__balance += money         # 执行存款
12        print("存款 ", money, " 完成")  # 打印存款完成
13
14    def withdraw_money(self, money):     # 设计取款方法
15        self.__balance -= money         # 执行取款
16        print("取款 ", money, " 完成")  # 打印取款完成
17
18    def get_balance(self):               # 获得存款余额
19        print(self.__name.title(), " 目前余额: ", self.__balance)
20
21    def bank_title(self):                # 获得银行名称
22        return self.__title
23
24    class Shilin_Banks(Banks):
25        # 定义士林分行
26        def __init__(self, uname):
27            self.title = "Taipei Bank - Shilin Branch" # 定义分行名称
28        def bank_title(self):
29            return self.title
```

执行结果 由于这不是程序所以没有任何执行结果。

现在我们已经成功地建立模块banks.py了。

13-4 应用自己建立的类模块

其实导入模块内的类与导入模块内的函数观念是一致的，下列将分成各小节说明。

13-4-1 导入模块的单一类

观念与13-2-2节相同，它的语法格式如下：

```
from 模块名称 import 类名称
```

程序实例ch13_9.py： 使用导入模块方式，重新设计ch13_8.py。由于这个程序只导入Banks类，所以此程序不执行原先35和36行。

```

1 # ch13_9.py
2 from banks import Banks          # 导入banks模块的Banks类
3
4 jamesbank = Banks('James')      # 定义Banks类对象
5 print("James's banks = ", jamesbank.bank_title()) # 打印银行名称
6 jamesbank.save_money(500)        # 存钱
7 jamesbank.get_balance()          # 列出存款金额

```

执行结果

```

===== RESTART: D:\Python\ch13\ch13_9.py =====
James's banks = Taipei Bank
存款 500 完成
James 目前余额: 500
>>>

```

由执行结果读者应该体会到，整个程序变得非常简洁了。

13-4-2 导入模块的多个类

观念与13-2-3节相同，如果模块内有多个类别，我们也可以使用下列方式导入多个类别，所导入的类别名称间需以逗号隔开。

from 模块名称 import 类别名称1, 类别名称2, ... , 类别名称n

程序实例ch13_10.py：以同时导入Banks类别和Shilin_Banks类别的方式，重新设计ch13_8.py。

```

1 # ch13_10.py
2 # 导入banks模块的Banks和Shilin_Banks类别
3 from banks import Banks, Shilin_Banks
4
5 jamesbank = Banks('James')      # 定义Banks类别对象
6 print("James's banks = ", jamesbank.bank_title()) # 打印银行名称
7 jamesbank.save_money(500)        # 存钱
8 jamesbank.get_balance()          # 列出存款金额
9 hungbank = Shilin_Banks('Hung')  # 定义Shilin_Banks类别对象
10 print("Hung's banks = ", hungbank.bank_title()) # 打印银行名称

```

执行结果 与ch13_8.py相同。

13-4-3 导入模块内所有类

观念与13-2-4节相同，如果想导入模块内所有类别，语法如下：

from 模块名称 import *

程序实例 ch13_11.py： 使用导入模块所有类别的方式重新设计 ch13_8.py。

```
1 # ch13_11.py
2 from banks import *                # 导入banks模块所有类别
3
4 jamesbank = Banks('James')        # 定义Banks类别对象
5 print("James's banks = ", jamesbank.bank_title()) # 打印银行名称
6 jamesbank.save_money(500)          # 存钱
7 jamesbank.get_balance()            # 列出存款金额
8 hungbank = Shilin_Banks('Hung')    # 定义Shilin_Banks类别对象
9 print("Hung's banks = ", hungbank.bank_title()) # 打印银行名称
```

执行结果 与ch13_8.py相同。

13-4-4 import模块名称

观念与13-2-1节相同，要导入13-3-2节所建的模块，只要在程序内加上下列简单的语法即可：

```
import 模块名称 # 导入模块
```

若以13-3-2节的实例，只要在程序内加上下列简单的语法即可：

```
import banks
```

程序中要引用模块的类别，语法如下：

```
模块名称.类别名称 # 模块名称与类别名称间有小数点“.”
```

程序实例 ch13_12.py： 使用 import 模块名称方式，重新设计 ch13_8.py，读者应该留意第2、4和8行的设计方式。

```
1 # ch13_12.py
2 import banks                # 导入banks模块
3
4 jamesbank = banks.Banks('James') # 定义Banks类别对象
5 print("James's banks = ", jamesbank.bank_title()) # 打印银行名称
6 jamesbank.save_money(500)        # 存钱
7 jamesbank.get_balance()          # 列出存款金额
8 hungbank = banks.Shilin_Banks('Hung') # 定义Shilin_Banks类别对象
9 print("Hung's banks = ", hungbank.bank_title()) # 打印银行名称
```

执行结果 与ch13_8.py相同。

13-4-5 模块内导入另一个模块的类

有时候可能一个模块内有太多类别了，此时可以考虑将一系列的类别分成2个或更多个模块储存。如果拆成类别的模块彼此有衍生关系，则子类别也需将父类别导入，执行时才不会有错误产生。下列是将Banks模块拆成2个模块的内容。

程序实例banks1.py: 这个模块含父类别Banks的内容。

```
1 # banks1.py
2 # 这是一个包含Banks类别的模块(module)
3 class Banks():
4     # 定义银行类别
5     def __init__(self, uname):          # 初始化方法
6         self.__name = uname            # 设定私有存款者名字
7         self.__balance = 0             # 设定私有开户金额是0
8         self.__title = "Taipei Bank"   # 设定私有银行名称
9
10    def save_money(self, money):         # 设计存款方法
11        self.__balance += money         # 执行存款
12        print("存款 ", money, " 完成")  # 打印存款完成
13
14    def withdraw_money(self, money):     # 设计提款方法
15        self.__balance -= money         # 执行提款
16        print("提款 ", money, " 完成")  # 打印提款完成
17
18    def get_balance(self):               # 获得存款余额
19        print(self.__name.title(), " 目前余额: ", self.__balance)
20
21    def bank_title(self):                # 获得银行名称
22        return self.__title
```

程序实例shilin_banks.py: 这个模块含子类别Shilin_Banks的内容，读者应留意第3行，笔者在这的模块内导入了banks1.py模块的Banks类别。

```
1 # shilin_banks.py
2 # 这是一个包含Shilin_Banks类别的模块(module)
3 from banks1 import Banks              # 导入Banks类别
4
5 class Shilin_Banks(Banks):
6     # 定义士林分行
7     def __init__(self, uname):
8         self.title = "Taipei Bank - Shilin Branch" # 定义分行名称
9     def bank_title(self):                  # 获得银行名称
10        return self.title
```


程序实例ch13_13.py： 在这个程序中，笔者在第2和3行分别导入2个模块，至于整个程序的执行内容，则与ch13_8.py相同。

```
1 # ch13_13.py
2 from banks1 import Banks          # 导入banks模块的Banks类别
3 from shilin_Banks import Shilin_Banks  # 导入Shilin_Banks模块的Shilin_Banks类别
4
5 jamesbank = Banks('James')      # 定义Banks类别对象
6 print("James's banks = ", jamesbank.bank_title()) # 打印银行名称
7 jamesbank.save_money(500)        # 存钱
8 jamesbank.get_balance()          # 列出存款金额
9 hungbank = Shilin_Banks('Hung')  # 定义Shilin_Banks类别对象
10 print("Hung's banks = ", hungbank.bank_title()) # 打印银行名称
```

执行结果 与ch13_8.py相同。

13-5 随机数random模块

所谓的随机数是指平均散布在某区间的数字，随机数其实用途很广，最常见的应用是设计游戏时可以控制输出结果，赌场的吃角子老虎机器就是靠它赚钱。这节笔者将介绍random模块中最有用的3个方法，同时也会分析赌场赚钱的利器。

13-5-1 randint()

这个方法可以随机产生指定区间的整数，它的语法如下：

`randint(min, max)` # 可以产生min与max之间的整数值

程序实例ch13_14.py： 建立一个程序分别产生3组在1-100、500-1000、2000-3000的数字。

```
1 # ch13_14.py
2 import random          # 导入模块random
3
4 n = 3
5 for i in range(n):
6     print("1-100 : ", random.randint(1, 100))
7
8 for i in range(n):
9     print("500-1000 : ", random.randint(500, 1000))
10
11 for i in range(n):
12     print("2000-3000 : ", random.randint(2000, 3000))
```

执行结果

```
===== RESTART: D:/Python/ch13/ch13_14.py =====
1-100 : 85
1-100 : 77
1-100 : 23
500-1000 : 981
500-1000 : 845
500-1000 : 826
2000-3000 : 2138
2000-3000 : 2784
2000-3000 : 2727
>>>
```

程序实例ch13_15.py： 猜数字游戏，这个程序首先会用randint()方法产生一个1到10之间的数字，然后如果猜的数值太小会要求猜大一些，如果猜的数值太大会要求猜小一些，最后列出猜了几次才答对。

```
1 # ch13_15.py
2 import random          # 导入模块random
3
4 min, max = 1, 10
5 ans = random.randint(min, max) # 随机数产生答案
6 while True:
7     yourNum = int(input("请猜1-10之间数字: "))
8     if yourNum == ans:
9         print("恭喜!答对了")
10        break
11    elif yourNum < ans:
12        print("请猜大一些")
13    else:
14        print("请猜小一些")
```

执行结果

```
===== RESTART: D:\Python\ch13\ch13_15.py =====
请猜1-10之间数字: 5
请猜大一些
请猜1-10之间数字: 8
请猜大一些
请猜1-10之间数字: 9
恭喜!答对了
>>>
```

一般赌场的机器其实可以用随机数控制输赢，例如：某个猜大小机器，一般人以为猜对率是50%，但是只要控制随机数，赌场可以直接控制输赢比例。

程序实例ch13_16.py： 这是一个猜大小的游戏，程序执行初可以设定庄家的输赢比例，程序会立即回应是否猜对。


```

1 # ch13_16.py
2 import random # 导入模块random
3
4 min, max = 1, 100 # 随机数最小与最大值设定
5 winPercent = int(input("请输入庄家赢的比率(0-100)之间 :"))
6
7 while True:
8     print("猜大小游戏: L或l表示大, S或s表示小, Q或q则程序结束")
9     customerNum = input("= ") # 读取玩家输入
10    if customerNum == 'Q' or customerNum == 'q': # 若输入Q或q
11        break # 程序结束
12    num = random.randint(min, max) # 产生是否让玩家答对的随机数
13    if num > winPercent: # 随机数在81-100间回应玩家猜对
14        print("恭喜!答对了\n")
15    else: # 随机数在1-80间回应玩家猜错
16        print("答错了!请再试一次\n")

```

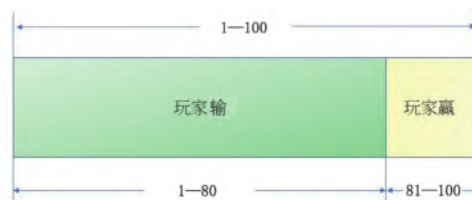
执行结果

```

===== RESTART: D:\Python\ch13\ch13_16.py =====
请输入庄家赢的比率(0-100)之间 :80
猜大小游戏: L或l表示大, S或s表示小, Q或q则程序结束
= l
答错了!请再试一次
猜大小游戏: L或l表示大, S或s表示小, Q或q则程序结束
= l
答错了!请再试一次
猜大小游戏: L或l表示大, S或s表示小, Q或q则程序结束
= s
答错了!请再试一次
猜大小游戏: L或l表示大, S或s表示小, Q或q则程序结束
= q
>>>

```

这个程序的关键点1是程序第5行，庄家可以在程序启动时先设定赢的比率。第2个关键点是程序第12行产生的随机数，由1~100的随机数决定玩家是赢或输，猜大小只是幌子。例如，庄家刚开始设定赢的机率是80%，相当于如果随机数是在81~100的算玩家赢，如果随机数是1~80算玩家输。



13-5-2 choice()

这个方法可以让我们在一个列表(list)中随机传回一个元素。

程序实例ch13_17.py: 有一个水果列表, 使用choice()方法随机选取一个水果。

```
1 # ch13_17.py
2 import random          # 导入模块random
3
4 fruits = ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
5 print(random.choice(fruits))
```

执行结果 下列是程序执行2次的执行结果。

```
===== RESTART: D:\Python\ch13\ch13_17.py =====
香蕉
>>>
===== RESTART: D:\Python\ch13\ch13_17.py =====
西瓜
>>>
```

13-5-3 shuffle()

这个方法可以将列表元素重新排列, 如果你欲设计扑克牌(Poker)游戏, 在发牌前可以使用这个方法将牌打乱重新排列。

程序实例ch13_18.py: 将列表内的扑克牌次序打乱, 然后重新排列。

```
1 # ch13_18.py
2 import random          # 导入模块random
3
4 poker = ['2', '3', '4', '5', '6', '7', '8',
5          '9', '10', 'J', 'Q', 'K', 'A']
6 random.shuffle(poker)  # 将次序打乱重新排列
7 print(poker)
```

执行结果

```
===== RESTART: D:/Python/ch13/ch13_18.py =====
['2', '3', '10', 'J', '4', 'A', '8', '7', '6', 'K', 'Q', '9', '5']
>>>
===== RESTART: D:/Python/ch13/ch13_18.py =====
['A', '7', 'Q', '5', '4', '6', '8', '10', '3', '2', '9', 'K', 'J']
>>>
```

将列表元素打乱, 很适合老师出防止作弊的考题, 例如, 有50位学生, 为了避免学生偷窥邻座的考卷, 建议可以将出好的题目处理成列表, 然后使用for循环执行50次shuffle(), 这样就可以得到50份考题相同但是次序不同的考卷。笔者将这个观念当作是习题。

13-6 时间time模块

13-6-1 time()

time()方法可以传回自1970年1月1日00:00:00AM以来的秒数，初看好像用处不大，但如果想要掌握某段工作所花时间则很有用，例如，若应用在程序实例ch13_15.py，你可以用它计算猜数字所花时间。

程序实例ch13_19.py： 计算自1970年1月1日00:00:00AM以来的秒数。

```
1 # ch13_19.py
2 import time                # 导入模块time
3
4 print("计算1970年1月1日00:00:00至今的秒数 = ", int(time.time()))
```

执行结果

```
===== RESTART: D:\Python\ch13\ch13_19.py =====
计算1970年1月1日00:00:00至今的秒数 = 1513061845
>>>
```

读者的执行结果将和笔者不同，因为我们是在不同的时间点执行这个程序。

程序实例ch13_20.py： 扩充ch13_15.py的功能，主要是增加计算花多少时间猜对数字。

```
1 # ch13_20.py
2 import random              # 导入模块random
3 import time                # 导入模块time
4
5 min, max = 1, 10
6 ans = random.randint(min, max) # 随机数产生答案
7 yourNum = int(input("请猜1-10之间数字: "))
8 starttime = int(time.time()) # 起始秒数
9 while True:
10     if yourNum == ans:
11         print("恭喜!答对了")
12         endtime = int(time.time()) # 结束秒数
13         print("所花时间: ", endtime - starttime, " 秒")
14         break
15     elif yourNum < ans:
16         print("请猜大一些")
17     else:
18         print("请猜小一些")
19     yourNum = int(input("请猜1-10之间数字: "))
```

执行结果

```
===== RESTART: D:\Python\ch13\ch13_20.py =====
请猜1-10之间数字: 5
请猜大一些
请猜1-10之间数字: 8
请猜小一些
请猜1-10之间数字: 6
恭喜!答对了
所花时间: 5 秒
>>>
```

13-6-2 sleep()

sleep()方法可以让工作暂停，这个方法的参数单位是秒。这个方法对于设计动画非常有帮助，未来我们还会介绍这个方法更多的应用。

程序实例ch13_21.py： 每秒打印一次列表的内容。

```
1 # ch13_21.py
2 import time                # 导入模块time
3
4 fruits = ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
5 for fruit in fruits:
6     print(fruit)
7     time.sleep(1)          # 暂停1秒
```

执行结果

```
===== RESTART: D:\Python\ch13\ch13_21.py =====
苹果
香蕉
西瓜
水蜜桃
百香果
>>>
```

13-6-3 asctime()

这个方法会以可以阅读方式列出目前系统时间。

程序实例ch13_22.py： 列出目前系统时间。

```
1 # ch13_22.py
2 import time                # 导入模块time
3
4 print(time.asctime())      # 列出目前系统时间
```

执行结果

```
===== RESTART: D:/Python/ch13/ch13_22.py =====
Tue Sep 26 15:44:47 2017
>>>
```

13-6-4 localtime()

这个方法可以返回目前时间的结构数据，所返回的结构可以用索引方式获得个别内容。

程序实例ch13_23.py： 使用`localtime( )`方法列出目前时间的结构数据，同时使用索引列出个别内容。

```
1 # ch13_23.py
2 import time                # 导入模块time
3
4 xtime = time.localtime()
5 print(xtime)               # 列出目前系统时间
6 print("年 ", xtime[0])
7 print("月 ", xtime[1])
8 print("日 ", xtime[2])
9 print("时 ", xtime[3])
10 print("分 ", xtime[4])
11 print("秒 ", xtime[5])
12 print("星期几 ", xtime[6])
13 print("第几天 ", xtime[7])
14 print("夏令时间 ", xtime[8])
```

执行结果

```
===== RESTART: D:\Python\ch13\ch13_23.py =====
time.struct_time(tm_year=2017, tm_mon=12, tm_mday=12, tm_hour=15, tm_min=26, tm_
sec=56, tm_wday=1, tm_yday=346, tm_isdst=0)
年 2017
月 12
日 12
时 15
分 26
秒 56
星期几 1
第几天 346
夏令时间 0
>>>
```

上述索引第12行[6]是代表星期几的设定，0代表星期一，1代表星期二。上述第13行索引[7]是第几天的设定，代表这是一年中的第几天。上述第14行索引[8]是夏令时间的设定，0代表不是，1代表是。

13-7 系统sys模块

这个模块可以控制Python Shell窗口信息。

13-7-1 version属性

这个属性可以列出目前所使用Python的版本信息。

程序实例ch13_24.py： 列出目前所使用Python的版本信息。

```
1 # ch13_24.py
2 import sys
3
4 print("目前Python版本是：", sys.version)
```

执行结果

```
===== RESTART: D:\Python\ch13\ch13_24.py =====
目前Python版本是： 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:14:34) [MSC v.1900 32
bit (Intel)]
>>>
```

13-7-2 stdin对象

这是一个对象，stdin是standard input的缩写，是指从屏幕输入（可想成Python Shell窗口），这个对象可以搭配readline()方法，然后可以读取屏幕输入直到按下Enter键的字符串。

程序实例ch13_25.py： 读取屏幕输入。

```
1 # ch13_25.py
2 import sys
3 print("请输入字符串，输入完按Enter = ", end = "")
4 msg = sys.stdin.readline()
5 print(msg)
```

执行结果

```
===== RESTART: D:\Python\ch13\ch13_25.py =====
请输入字符串，输入完按Enter = Python王者归来
Python王者归来
>>>
```

在 readline() 方法内可以加上正整数参数，例如：readline(n)，这个n代表所读取的字符数，其中一个中文字或空格也算一个字符数。

程序实例ch13_26.py: 从屏幕读取8个字符数的应用。

```
1 # ch13_26.py
2 import sys
3 print("请输入字符串, 输入完按Enter = ", end = "")
4 msg = sys.stdin.readline(8)      # 读8个字
5 print(msg)
```

执行结果

```
===== RESTART: D:\Python\ch13\ch13_26.py =====
请输入字符串, 输入完按Enter = Python王者归来
Python王者
>>>
===== RESTART: D:\Python\ch13\ch13_26.py =====
请输入字符串, 输入完按Enter = I like Python
I like P
>>>
```

13-7-3 stdout对象

这是一个对象, stdout是standard output的缩写, 是指从屏幕输出(可想成Python Shell窗口), 这个对象可以搭配write()方法, 然后可以从屏幕输出数据。

程序实例ch13_27.py: 使用stdout对象输出数据。

```
1 # ch13_27.py
2 import sys
3
4 sys.stdout.write("I like Python")
```

执行结果

```
===== RESTART: D:/Python/ch13/ch13_27.py =====
I like Python
>>>
```

其实这个对象若是使用Python Shell窗口, 最后会同时列出输出的字符数。

```
>>> import sys
>>> sys.stdout.write("I like Python")
I like Python13
>>>
```

13-8 keyword模块

这个模块有一些Python关键词的功能。

13-8-1 kwlist属性

这个属性含所有Python的关键词。

程序实例ch13_28.py： 列出所有Python关键词。

```
1 # ch13_28.py
2 import keyword
3
4 print(keyword.kwlist)
```

执行结果

```
===== RESTART: D:/Python/ch13/ch13_28.py =====
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue',
def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if',
'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'retu
rn', 'try', 'while', 'with', 'yield']
>>>
```

13-8-2 iskeyword()

这个方法可以传回参数的字符串是否是关键词，如果是 传回True，如果否 传回False。

程序实例ch13_29.py： 检查列表内的字是否是关键词。

```
1 # ch13_29.py
2 import keyword          # 导入keyword模块
3
4 keywordLists = ['as', 'while', 'break', 'sse', 'Python']
5 for x in keywordLists:
6     print("%8s " % x, keyword.iskeyword(x))
```

执行结果

```
===== RESTART: D:/Python/ch13/ch13_29.py =====
      as  True
     while True
      break True
       sse False
    Python False
>>>
```


习题

1. 请扩充makefood模块，增加make_noodle()函数，这个函数的第一个参数是面的种类，例如，牛肉面、肉丝面等。第2到多个参数则是自选配料，然后参考ch13_2.py调用方式，产生结果。
2. 请建立一个模块，这个模块含4个运算的类别，分别是加法、减法、乘法和除法，运算完成后需回传结果。基本上每个方法皆含2个参数，运算原则是：

参数1 op 参数2

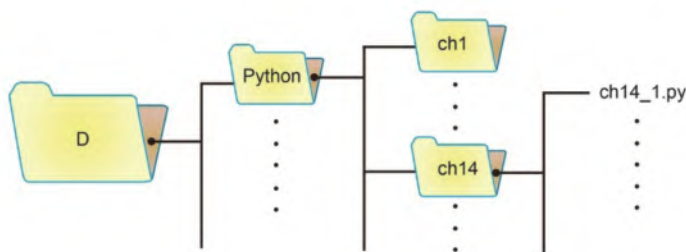
请分别用2组数字测试这个模块。
3. 请重新设计ch13_15.py，将所猜数值改为0—100间，增加猜几次才答对，若是输入Q或q，程序可直接结束。
4. 请测试randint(1, 100)随机数方法，请执行1000次，然后输出1—10、2—20、…、91—100各出现多少次。
5. 扩充设计ch13_16.py，程序开始玩家有500元筹码，玩一次100元，如果答对得100元，如果答错扣100元。当玩家筹码是0元时，程序也执行结束。
6. 请重新设计ch13_17.py，每执行一次即将输出的水果从列表内删除，直到fruits列表元素为无。
7. 将本章的是非题处理成列表，每一题当作是一个元素，请处理20份次序打乱的题目。

第14章 文件的读取与写入

本章笔者将讲解使用Python处理Windows操作系统内文件的完整相关知识，例如，文件路径的管理、文件的读取与写入、目录的管理、文件压缩与解压缩、认识编码规则与剪贴板的相关应用。

14-1 文件夹与文件路径

有一个文件路径图形如下：



对于ch14_1.py而言，它的文件路径名称是：

D:\Python\ch14\ch14_1.py

对于ch14_1.py而言，它的当前工作目录（也可称文件夹）名称是：

D:\Python\ch14

14-1-1 绝对路径与相对路径

在操作系统中可以使用2种方式表达文件路径，下列是以ch14_1.py为例：

- ① **绝对路径**：路径从根目录开始表达，以14-1节的文件路径图为例，它的绝对路径是：

```
D:\Python\ch14\ch14_1.py
```

- ② **相对路径**：是指相对于当前工作目录的路径，以14-1节的文件路径图为例，若是当前工作目录是D:\Python\ch14，它的相对路径是：

```
ch14_1.py
```

另外，在操作系统处理文件夹的观念中会使用2个特殊符号“.”和“..”，“.”指的是当前文件夹，“..”指的是上一层文件夹。但是在使用上，当指当前文件夹时也可以省略“.\”。所以使用“.\ch14_1.py”与“ch14_1.py”意义相同。

14-1-2 os模块与os.path模块

在Python内有关文件路径的模块是os，所以在本节实例最前面均需导入此模块。

```
import os # 导入os模块
```

在os模块内有另一个常用模块os.path，14-1节主要是使用这2个模块的方法，讲解与文件路径有关的文件夹知识，由于os.path是在os模块内，所以导入os模块后不用再导入os.path模块。

14-1-3 取得当前工作目录os.getcwd()

os模块内的getcwd()可以取得当前工作目录。

程序实例ch14_1.py： 列出当前工作目录。

```

1 # ch14_1.py
2 import os
3
4 print(os.getcwd())          # 列出目前工作目录

```

执行结果

```

===== RESTART: D:/Python/ch14/ch14_1.py =====
D:\Python\ch14
>>>

```

14-1-4 取得绝对路径os.path.abspath

os.path模块的abspath(path)会传回path的绝对路径，通常我们可以使用这个方法将文件或文件夹的相对路径转成绝对路径。

程序实例ch14_2.py： 取得绝对路径的应用。

```

1 # ch14_2.py
2 import os
3
4 print(os.path.abspath('.'))          # 列出目前工作目录的绝对路径
5 print(os.path.abspath('../'))       # 列出上一层工作目录的绝对路径
6 print(os.path.abspath('ch14_2.py')) # 列出目前文件的绝对路径

```

执行结果

```

===== RESTART: D:/Python/ch14/ch14_2.py =====
D:\Python\ch14
D:\Python
D:\Python\ch14\ch14_2.py
>>>

```

14-1-5 传回特定路段相对路径os.path.relpath()

os.path模块的relpath(path, start)会传回从start到path的相对路径，如果省略start，则传回当前工作目录至path的相对路径。

程序实例ch14_3.py： 传回特定路段相对路径的应用。

```

1 # ch14_3.py
2 import os
3
4 print(os.path.relpath('D:\\'))          # 列出目前工作目录至D:\\的相对路径
5 print(os.path.relpath('D:\\Python\\ch13')) # 列出目前工作目录至特定path的相对路径
6 print(os.path.relpath('D:\\', 'ch14_3.py')) # 列出目前文件至D:\\的相对路径

```

执行结果

```
===== RESTART: D:/Python/ch14/ch14_3.py =====
>>>
..\..
..\ch13
..
>>>
```

14-1-6 检查路径方法exist/isabs/isdir/isfile

下列是常用的os.path模块方法。

exist(path): 如果path的文件或文件夹存在传回True, 否则传回False。

isabs(path): 如果path的文件或文件夹是绝对路径传回True, 否则传回False。

isdir(path): 如果path是文件夹传回True, 否则传回False。

isfile(path): 如果path是文件传回True, 否则传回False。

程序实例ch14_4.py: 检查路径方法的应用。

```
1 # ch14_4.py
2 import os
3
4 print("文件或文件夹存在 = ", os.path.exists('ch14'))
5 print("文件或文件夹存在 = ", os.path.exists('D:\\Python\\ch14'))
6 print("文件或文件夹存在 = ", os.path.exists('ch14_4.py'))
7 print(" --- ")
8
9 print("是绝对路径 = ", os.path.isabs('ch14_4.py'))
10 print("是绝对路径 = ", os.path.isabs('D:\\Python\\ch14\\ch14_4.py'))
11 print(" --- ")
12
13 print("是文件夹 = ", os.path.isdir('D:\\Python\\ch14\\ch14_4.py'))
14 print("是文件夹 = ", os.path.isdir('D:\\Python\\ch14'))
15 print(" --- ")
16
17 print("是文件 = ", os.path.isfile('D:\\Python\\ch14\\ch14_4.py'))
18 print("是文件 = ", os.path.isfile('D:\\Python\\ch14'))
```

执行结果

```

===== RESTART: D:\Python\ch14\ch14_4.py =====
文件或文件夹存在 = False
文件或文件夹存在 = True
文件或文件夹存在 = True
---
是绝对路径 = False
是绝对路径 = True
---
是文件夹 = False
是文件夹 = True
---
是文件 = True
是文件 = False
>>>

```

14-1-7 文件与目录的操作 mkdir/rmdir/remove/chdir

这几个方法是在os模块内，建议执行下列操作前先用os.path.exists()检查是否存在。

mkdir(path)：建立path目录。

rmdir(path)：删除path目录，限制只能是空的目录。如果要删除底下有文件的目录需参考14-4-7小节。

remove(path)：删除path文件。

chdir(path)：将当前工作文件夹改至path。

程序实例ch14_5.py： 使用mkdir建立文件夹的应用。

```

1 # ch14_5.py
2 import os
3
4 mydir = 'testch14'
5 # 如果mydir不存在就建立此文件夹
6 if os.path.exists(mydir):
7     print("已经存在 %s " % mydir)
8 else:
9     os.mkdir(mydir)
10    print("建立 %s 文件夹成功" % mydir)

```

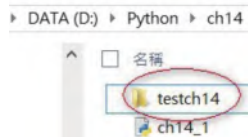
执行结果

```

===== RESTART: D:\Python\ch14\ch14_5.py =====
建立 testch14 文件夹成功
>>>

```

下列是验证testch14建立成功的画面。



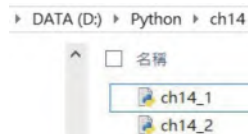
程序实例ch14_6.py: 使用rmdir删除文件夹的应用。

```
1 # ch14_6.py
2 import os
3
4 mydir = 'testch14'
5 # 如果mydir存在就删除此文件夹
6 if os.path.exists(mydir):
7     os.rmdir(mydir)
8     print("删除 %s 文件夹成功" % mydir)
9 else:
10    print("%s 文件夹不存在" % mydir)
```

执行结果

```
===== RESTART: D:\Python\ch14\ch14_6.py
testch14 文件夹不存在
>>>
```

下列是验证testch14已经删除的画面。



程序实例ch14_7.py: 删除指定path文件的应用。

```
1 # ch14_7.py
2 import os
3
4 myfile = 'test.py'
5 # 如果myfile存在就删除此文件
6 if os.path.exists(myfile):
7     os.remove(myfile)
8     print("删除 %s 文件成功" % myfile)
9 else:
10    print("%s 文件不存在" % myfile)
```

执行结果 下列分别是删除文件不存在（左边）或存在（右边）的执行结果画面。

```
===== RESTART: D:\Python\ch14\ch14_7.py
test.py 文件不存在
>>>
```

```
===== RESTART: D:/Python/ch14/test.py
删除 test.py 文件成功
>>>
```

程序实例ch14_8.py: 更改当前工作文件夹, 然后再返回原先工作文件夹。

```
1 # ch14_8.py
2 import os
3
4 newdir = 'D:\\Python'
5 currentdir = os.getcwd()
6 print("列出目前工作文件夹 ", currentdir)
7
8 # 如果newdir不存在就建立此文件夹
9 if os.path.exists(newdir):
10     print("已经存在 %s " % newdir)
11 else:
12     os.mkdir(newdir)
13     print("建立 %s 文件夹成功" % newdir)
14
15 # 将目前工作文件夹改至newdir
16 os.chdir(newdir)
17 print("列出最新工作文件夹 ", os.getcwd())
18
19 # 将目前工作文件夹返回
20 os.chdir(currentdir)
21 print("列出返回工作文件夹 ", currentdir)
```

执行结果

```
===== RESTART: D:\Python\ch14\ch14_8.py
列出目前工作文件夹 D:\Python\ch14
已经存在 D:\Python
列出最新工作文件夹 D:\Python
列出返回工作文件夹 D:\Python\ch14
>>>
```

14-1-8 传回文件路径os.path.join()

这个方法可以将os.path.join()参数内的字符串结合为一个文件路径, 参数可以有2个到多个。

程序实例ch14_9.py: os.path.join()方法的应用, 这个程序会用2、3、4个参数测试这个方法。

```
1 # ch14_9.py
2 import os
3
4 print(os.path.join('D:\\', 'Python', 'ch14', 'ch14_9.py')) # 4个参数
5 print(os.path.join('D:\\Python', 'ch14', 'ch14_9.py')) # 3个参数
6 print(os.path.join('D:\\Python\\ch14', 'ch14_9.py')) # 2个参数
```


执行结果

```
===== RESTART: D:/Python/ch14/ch14_9.py =====
D:\Python\ch14\ch14_9.py
D:\Python\ch14\ch14_9.py
D:\Python\ch14\ch14_9.py
>>>
```

程序实例ch14_10.py: 使用for循环将一个列表内的文件与一个路径结合。

```
1 # ch14_10.py
2 import os
3
4 files = ['ch14_1.py', 'ch14_2.py', 'ch14_3.py']
5 for file in files:
6     print(os.path.join('D:\\Python\\ch14', file))
```

执行结果

```
===== RESTART: D:/Python/ch14/ch14_10.py =====
D:\Python\ch14\ch14_1.py
D:\Python\ch14\ch14_2.py
D:\Python\ch14\ch14_3.py
>>>
```

14-1-9 获得特定文件的大小os.path.getsize()

这个方法可以获得特定文件的大小。

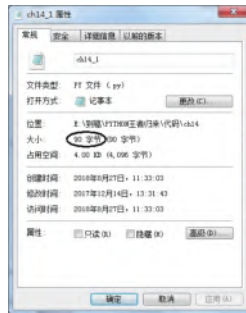
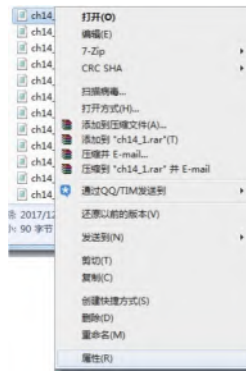
程序实例ch14_11.py: 获得ch14_1.py的文件大小, 从执行结果可以知道是90字节。

```
1 # ch14_11.py
2 import os
3
4 # 如果文件在目前工作目录下可以省略路径
5 print(os.path.getsize("ch14_1.py"))
6 print(os.path.getsize("D:\\Python\\ch14\\ch14_1.py"))
```

执行结果

```
===== RESTART: D:/Python/ch14/ch14_11.py =====
90
90
>>>
```

下列是验证结果。



14-1-10 获得特定工作目录的内容os.listdir()

这个方法将以列表方式列出特定工作目录的内容。

程序实例ch14_12.py: 以两种方式列出D:\Python\ch14的工作目录内容。

```
1 # ch14_12.py
2 import os
3
4 print(os.listdir("D:\\Python\\ch14"))
5 print(os.listdir(".")) # 这代表目前工作目录
```

执行结果

```
===== RESTART: D:/Python/ch14/ch14_12.py =====
['ch14_1.py', 'ch14_10.py', 'ch14_11.py', 'ch14_12.py', 'ch14_2.py', 'ch14_3.py',
'ch14_4.py', 'ch14_5.py', 'ch14_6.py', 'ch14_7.py', 'ch14_8.py', 'ch14_9.py',
'testch14']
['ch14_1.py', 'ch14_10.py', 'ch14_11.py', 'ch14_12.py', 'ch14_2.py', 'ch14_3.py',
'ch14_4.py', 'ch14_5.py', 'ch14_6.py', 'ch14_7.py', 'ch14_8.py', 'ch14_9.py',
'testch14']
>>>
```

程序实例ch14_13.py: 列出特定工作目录所有文件的大小。

```

1 # ch14_13.py
2 import os
3
4 totalsizes = 0
5 print("列出D:\\Python\\ch14工作目录的所有文件")
6 for file in os.listdir('D:\\Python\\ch14'):
7     print(file)
8     totalsizes += os.path.getsize(os.path.join('D:\\Python\\ch14', file))
9
10 print("全部文件大小是 = ", totalsizes)

```

执行结果



```

===== RESTART: D:\Python\ch14\ch14_13.py =====
列出D:\Python\ch14工作目录的所有文件
ch14_1.py
ch14_10.py
ch14_11.py
ch14_12.py
ch14_13.py
ch14_2.py
ch14_3.py
ch14_4.py
ch14_5.py
ch14_6.py
ch14_7.py
ch14_8.py
ch14_9.py
全部文件大小是 = 3631
>>>

```

14-1-11 获得特定工作目录内容glob

Python内还有一个模块可用于列出特定工作目录内容glob，当导入这个模块后可以使用glob方法获得特定工作目录的内容，这个方法最大特色是可以使用通配符“*”，例如，可用“*.txt”获得所有txt扩展名的文件，更多应用可参考下列实例。

程序实例ch14_14.py： 方法1是列出所有工作目录的文件，方法2是列出ch14_1开头的扩展名是py的文件，方法3是列出ch14_2开头的文件。

```

1 # ch14_14.py
2 import glob
3
4 print("方法1:列出\\Python\\ch14工作目录的所有文件")
5 for file in glob.glob('D:\\Python\\ch14\\*.\\*'):
6     print(file)
7
8 print("方法2:列出目前工作目录的特定文件")
9 for file in glob.glob('ch14_1*.py'):
10     print(file)
11
12 print("方法3:列出目前工作目录的特定文件")
13 for file in glob.glob('ch14_2*.\\*'):
14     print(file)

```

执行结果

```
RESTART: D:\Python\ch14\ch14_14.py
方法1:列出\Python\ch14工作目录的所有文件
D:\Python\ch14\ch14_1.py
D:\Python\ch14\ch14_10.py
D:\Python\ch14\ch14_11.py
D:\Python\ch14\ch14_12.py
D:\Python\ch14\ch14_13.py
D:\Python\ch14\ch14_14.py
D:\Python\ch14\ch14_2.py
D:\Python\ch14\ch14_3.py
D:\Python\ch14\ch14_4.py
D:\Python\ch14\ch14_5.py
D:\Python\ch14\ch14_6.py
D:\Python\ch14\ch14_7.py
D:\Python\ch14\ch14_8.py
D:\Python\ch14\ch14_9.py
方法2:列出当前工作目录的特定文件
ch14_1.py
ch14_10.py
ch14_11.py
ch14_12.py
ch14_13.py
ch14_14.py
方法3:列出当前工作目录的特定文件
ch14_2.py
>>>
```

14-1-12 遍历目录树os.walk()

在os模块内有提供一个os.walk()方法可以让我们遍历目录树，这个方法每次执行循环时将传回3个值：

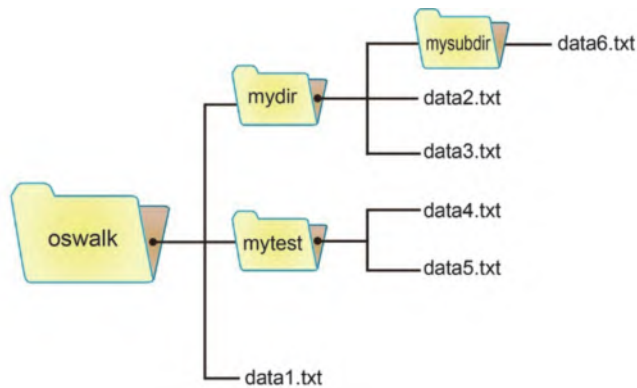
- ①当前工作目录名称(dirName)。
- ②当前工作目录底下的子目录列表(sub_dirNames)。
- ③当前工作目录底下的文件列表(fileNames)。

下列是语法格式：

```
for dirName, sub_dirNames, fileNames in
os.walk(目录路径
):
    程序区块
```

上述dirName , sub_dirNames , fileNames 名称可以自行命名，顺序则不可以更改，至于目录路径 可以使用绝对地址或相对地址，如果不注明则代表当前工作目录的子目录。

程序实例ch14_14_1.py： 在笔者范例D:\Python\ch14目录下列有一个oswalk目录，此目录内容如下：



本程序将遍历此oswalk目录，同时列出内容。

```
1 # ch14_14_1.py
2 import os
3
4 for dirName, sub_dirNames, fileNames in os.walk('oswalk'):
5     print("目前工作目录名称: ", dirName)
6     print("目前子目录名称列表: ", sub_dirNames)
7     print("目前文件名列表: ", fileNames, "\n")
```

执行结果

```
===== RESTART: D:\Python\ch14\ch14_14_1.py =====
目前工作目录名称:  oswalk
目前子目录名称列表:  ['mydir', 'mytest']
目前文件名列表:    ['data1.txt']

目前工作目录名称:  oswalk\mydir
目前子目录名称列表:  ['mysubdir']
目前文件名列表:    ['data2.txt', 'data3.txt']

目前工作目录名称:  oswalk\mydir\mysubdir
目前子目录名称列表:  []
目前文件名列表:    ['data6.txt']

目前工作目录名称:  oswalk\mytest
目前子目录名称列表:  []
目前文件名列表:    ['data4.txt', 'data5.txt']

>>>
```

从上述执行结果可以看到，`os.walk()`将遍历指定目录底下的子目录同时传回子目录列表和文件列表，如果所传回的子目录列表是`[]`代表底下没有子目录。

14-2 读取文件

Python处理读取或写入文件首先需将文件打开，然后可以一次读取所有文件或是一行一行读取文件内容。Python可以使用`open()`

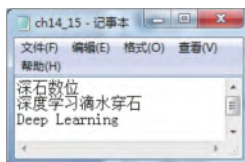
函数打开文件，文件打开后会传回文件对象，未来可用读取此文件对象方式读取文件内容，更多有关open()函数可参考4-3-1小节。

14-2-1 读取整个文件read()

文件打开后，可以使用read()读取所打开的文件，使用read()读取时，所有的文件内容将以一个字符串方式被读取然后存入字符串变量内，未来只要打印此字符串变量相当于可以打印整个文件内容。

在本书ch14文件夹有ch14_15.txt文件。

程序实例ch14_15.py： 读取ch14_15.txt文件然后输出，请读者留意程序第5行，笔者使用打印一般变量方式就打印了整个文件了。



```
1 # ch14_15.py
2
3 fn = 'ch14_15.txt'          # 设定欲开启的文件
4 file_Obj = open(fn)         # 用预设mode=r开启文件，传回调案对象file_Obj
5 data = file_Obj.read()      # 读取文件到变量data
6 file_Obj.close()            # 关闭文件对象
7 print(data)                  # 输出变量data相当于输出文件
```

执行结果

```
===== RESTART: D:/Python/ch14/ch14_15.py =====
深石数位
深度学习滴水穿石
Deep Learning
>>>
```

上述使用open()打开文件时，建议使用close()将文件关闭可参考第6行，若是没有关闭也许未来文件内容会有不可预期的损害。

另外，上述程序第3和4行所打开的文件ch14_15.txt没有文件路径，这表示这个文件需与程序文件在相同的工作目录，否则会有找不

到这个文件的情况发生。当然程序设计时，也可以在第3行直接配置文件案的绝对路径，如下所示：

```
D:\Python\ch14\ch14_15.txt
```

如果这样，就不必担心数据文件 ch14_15.txt 与程序文件 ch14_15.py 是否在相同目录了。

14-2-2 with 关键词

其实Python提供一个关键词with应用在打开文件 与建立文件对象 时，使用方式如下：

```
with open(欲打开的文件
) as 文件对象
:
    相关系列指令
```

使用这种方式打开文件，最大特色是可以不必在程序中关闭文件，with指令会在结束不需要此文件时自动将它关闭，文件经“with open() as文件对象”打开后会有有一个文件对象，就可以使用前一节的read()读取此文件对象的内容。

程序实例ch14_16.py： 使用with关键词重新设计ch14_15.py。

```
1 # ch14_16.py
2
3 fn = 'ch14_15.txt'          # 设定欲开启的文件
4 with open(fn) as file_Obj:  # 用默认mode=r开启文件,传回文件对象file_Obj
5     data = file_Obj.read()   # 读取文件到变量data
6     print(data)              # 输出变量data相当于输出文件
```

执行结果 与ch14_15.py相同。

由于整个文件是以字符串方式被读取与储存，所以打印字符串时最后一行的空白行也将显示出来，不过我们可以使用`rstrip()`将`data`字符串变量（文件）末端的空格符删除。

程序实例`ch14_17.py`： 重新设计`ch14_16.py`，但是删除文件末端的空白。

```
1 # ch14_17.py
2
3 fn = 'ch14_15.txt'          # 设定欲开启的文件
4 with open(fn) as file_Obj:  # 用默认mode=r开启文件，传回文件对象file_Obj
5     data = file_Obj.read()   # 读取文件到变量data
6     print(data.rstrip())     # 输出变量data相当于输出文件，同时删除末端字符
```

执行结果



```
===== RESTART: D:\Python\ch14\ch14_17.py =====
深石数位
深度学习滴水穿石
Deep Learning
>>>
```

由执行结果可以看到文件末端不再有空白行了。

14-2-3 逐行读取文件内容

在Python若想逐行读取文件内容，可以使用下列循环：

```
for line
in file_Obj: # line
```

和 `file_Obj`

可以自行取名， `file_Obj`

是文件对象

循环相关系列指令

程序实例`ch14_18.py`： 逐行读取和输出文件。


```

1 # ch14_18.py
2
3 fn = 'ch14_15.txt'      # 设定欲开启的文件
4 with open(fn) as file_Obj: # 用默认mode=r开启文件,传回文件对象file_Obj
5     for line in file_Obj: # 逐行读取文件到变量line
6         print(line)      # 输出变量line相当于输出一行

```

执行结果

```

===== RESTART: D:/Python/ch14/ch14_18.py =====
深石数位
深度学习滴水穿石
Deep Learning
>>>

```

因为以记事本编辑的ch14_15.txt文本文件每行末端有换行符号，同时print()在输出时也有一个换行输出的符号，所以才会得到上述每行输出后有空一行的结果。

程序实例ch14_19.py：重新设计ch14_18.py，但是删除每行末端的换行符号。

```

1 # ch14_19.py
2
3 fn = 'ch14_15.txt'      # 设定欲开启的文件
4 with open(fn) as file_Obj: # 用默认mode=r开启文件,传回文件对象file_Obj
5     for line in file_Obj: # 逐行读取文件到变量line
6         print(line.rstrip()) # 输出变量line相当于输出一行,同时删除末尾字符

```

执行结果

```

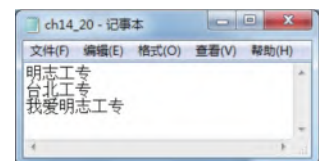
===== RESTART: D:/Python/ch14/ch14_19.py =====
深石数位
深度学习滴水穿石
Deep Learning
>>>

```

14-2-4 逐行读取使用readlines()

使用with关键词配合open()时，所打开的文件对象当前只在with区块内使用，特别是想要遍历此文件对象时。Python另外有一个方法readlines()可以逐行读取，同时以列表方式储存，另一个特色是读取时每行的换行字符皆会储存在列表内。当然更重要的是我们可以在with区块外遍历原先文件对象内容。

在ch14文件夹有下列ch14_20.txt文件。



程序实例ch14_20.py： 使用readlines()逐行读取ch14_20.txt，存入列表，然后打印此列表的结果。

```
1 # ch14_20.py
2
3 fn = 'ch14_20.txt'          # 设定欲开启的文件
4 with open(fn) as file_Obj:  # 用默认mode=r开启文件,传回文件对象file_Obj
5     obj_list = file_Obj.readlines() # 每次读一行
6
7 print(obj_list)             # 打印列表
```

执行结果

```
===== RESTART: D:/Python/ch14/ch14_20.py =====
['明志高专\n', '台北高专\n', '我爱明志高专\n']
>>>
```

由上述执行结果可以看到在txt文件的换行字符也出现在列表元素内。

程序实例ch14_21.py： 逐行输出ch14_20.py所保存的列表内容。

```
1 # ch14_21.py
2
3 fn = 'ch14_20.txt'          # 设定欲开启的文件
4 with open(fn) as file_Obj:  # 用默认mode=r开启文件,传回文件对象file_Obj
5     obj_list = file_Obj.readlines() # 每次读一行
6
7 for line in obj_list:
8     print(line.rstrip())      # 打印列表
```

执行结果

```
===== RESTART: D:/Python/ch14/ch14_21.py =====
明志高专
台北高专
我爱明志高专
>>>
```

14-2-5 数据组合

Python的多功能用途，可以让我们很轻松地组合数据，例如，我们可以将原先分成3行显示的数据，以隔一个空格方式显示。

程序实例ch14_22.py： 重新设计ch14_21.py，将分成3行显示的数据用1行显示。

```

1 # ch14_22.py
2
3 fn = 'ch14_20.txt' # 设定欲开启的文件
4 with open(fn) as file_Obj: # 用默认mode=r开启文件,传回文件对象file_Obj
5     obj_list = file_Obj.readlines() # 每次读一行
6
7 str_Obj = '' # 先设为空字符串
8 for line in obj_list: # 将各行字符串存入
9     str_Obj += line.rstrip()
10
11 print(str_Obj) # 打印文件字符串

```

执行结果

```

===== RESTART: D:/Python/ch14/ch14_22.py =====
明志工专台北工专我爱明志工专
>>>

```

14-2-6 字符串的替换

使用Word处理时常常会使用寻找/取代 功能，Python也有这个方法可以使新字符串取代旧字符串。

字符串对象.replace(旧字符串, 新字符串) # 在字符串对象内，新字符串将取代旧字符串

程序实例ch14_23.py: 重新设计ch14_21.py，但是将“工专”改为“科大”。

```

1 # ch14_23.py
2
3 fn = 'ch14_20.txt' # 设定欲开启的文件
4 with open(fn) as file_Obj: # 传回文件物件file_Obj
5     data = file_Obj.read() # 读取文件到变量data
6     new_data = data.replace('工专', '科大') # 新变量储存
7     print(new_data.rstrip()) # 输出文件

```

执行结果

```

===== RESTART: D:/Python/ch14/ch14_23.py =====
明志科大
台北科大
我爱明志科大
>>>

```

14-2-7 数据的搜寻

使用Word软件时也常会有寻找功能，使用Python这类工作变得相对简单。在ch14文件夹找到ch14_20.txt文件。

程序实例ch14_24.py： 数据搜寻的应用，这个程序会读取sse.txt文件，然后要求输入欲搜寻的字符串，最后会响应此字符串是否在sse.txt文件中。



```
1 # ch14_24.py
2
3 fn = 'sse.txt' # 设定欲开启的文件
4 with open(fn) as file_Obj: # 用默认mode=r开启文件,传回文件对象file_Obj
5     obj_list = file_Obj.readlines() # 每次读一行
6
7 str_Obj = '' # 先设为空字符串
8 for line in obj_list: # 将各行字符串存入
9     str_Obj += line.rstrip()
10
11 findstr = input("请输入欲搜寻字符串 = ")
12 if findstr in str_Obj: # 搜寻文件是否有欲寻找字符串
13     print("搜寻 %s 字符串存在 %s 文件中" % (findstr, fn))
14 else:
15     print("搜寻 %s 字符串不存在 %s 文件中" % (findstr, fn))
```

执行结果

```
===== RESTART: D:\Python\ch14\ch14_24.py =====
请输入欲搜寻字符串 = Stone
搜寻 Stone 字符串存在 sse.txt 文件中
>>>
===== RESTART: D:\Python\ch14\ch14_24.py =====
请输入欲搜寻字符串 = Deep
搜寻 Deep 字符串不存在 sse.txt 文件中
>>>
```

14-2-8 数据搜寻使用find()

对于字符串的使用，Python提供一个方法find()，这个方法除了可以执行数据搜寻以外，如果搜寻到数据还会传回数据的索引位置，如果没有找到则传回-1。

index = S.find(sub[, start[, end]]) # s代表被搜寻的字符串,sub是欲搜寻字符串

index是如果搜寻到时传回的索引值，start和end代表可以被搜寻字符串的区间，若是省略表示全部搜寻，如果没有找到则传回-1给index。

程序实例ch14_25.py：重新设计ch14_24.py，当搜寻到字符串时同时列出字符串所在索引的位置。

```
1 # ch14_25.py
2
3 fn = 'sse.txt'          # 设定欲开启的文件
4 with open(fn) as file_Obj: # 用默认mode=r开启文件,传回文件对象file_Obj
5     obj_list = file_Obj.readlines() # 每次读一行
6
7 str_Obj = ''           # 先设为空字符串
8 for line in obj_list:  # 将各行字符串存入
9     str_Obj += line.rstrip()
10
11 findstr = input("请输入欲搜寻字符串 = ")
12 index = str_Obj.find(findstr) # 搜寻findstr字符串是否存在
13 if index >= 0:             # 搜寻文件是否有欲寻找字符串
14     print("搜寻 %s 字符串存在 %s 文件中" % (findstr, fn))
15     print("在索引 %s 位置出现" % index)
16 else:
17     print("搜寻 %s 字符串不存在 %s 文件中" % (findstr, fn))
```

执行结果

```
===== RESTART: D:\Python\ch14\ch14_25.py =====
请输入欲搜寻字符串 = sse
搜寻 sse 字符串不存在 sse.txt 文件中
>>>
===== RESTART: D:\Python\ch14\ch14_25.py =====
请输入欲搜寻字符串 = Stone
搜寻 Stone 字符串存在 sse.txt 文件中
在索引 8 位置出现
>>>
```

14-3 写入文件

程序设计时一定会碰上要求将执行结果保存起来的情况，此时就可以将执行结果存入文件内。

14-3-1 将执行结果写入空的文件内

打开文件open()函数使用时默认是mode= 'r' 读取文件模式，因此如果打开文件是供读取可以省略mode= 'r'。若是要供写入，那么就要设定写入模式mode= 'w'，程序设计时可以省略mode

，直接在open()函数内输入‘w’。如果所打开的文件需要读取和写入可以使用‘r+’。如果所打开的文件不存在open()会建立该文件对象，如果所打开的文件已经存在，原文件内容将被清空。

至于输出到文件可以使用write()方法，语法格式如下：

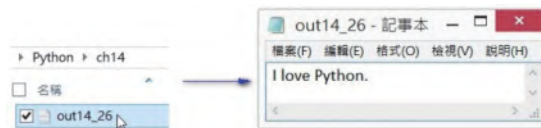
程序实例ch14_26.py： 输出数据到文件的应用。

文件对象

.write(欲输出数据) # 可将数据输出到文件对象

```
1 # ch14_26.py
2 fn = 'out14_26.txt'
3 string = 'I love Python.'
4
5 with open(fn, 'w') as file_Obj:
6     file_Obj.write(string)
```

执行结果 这个程序执行时在Python Shell窗口看不到结果，必须至ch14工作目录查看所建的out14_26.txt文件，同时打开可以得到下列结果。



14-3-2 写入数值资料

write()输出时无法输出数值数据，可参考下列错误范例。

程序实例ch14_27.py： 使用write()输出数值数据产生错误的实例。

```
1 # ch14_27.py
2 fn = 'out14_27.txt'
3 x = 100
4
5 with open(fn, 'w') as file_Obj:
6     file_Obj.write(x) # 直接输出数值x产生错误
```


执行结果

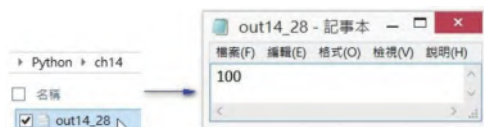
```
===== RESTART: D:\Python\ch14\ch14_27.py =====
Traceback (most recent call last):
  File "D:\Python\ch14\ch14_27.py", line 6, in <module>
    file_Obj.write(x)      # 直接输出数值x产生错误
TypeError: write() argument must be str, not int
>>>
```

如果想要使用write()将数值数据输出，必须使用str()将数值数据转成字符串数据。

```
1 # ch14_28.py
2 fn = 'out14_28.txt'
3 x = 100
4
5 with open(fn, 'w') as file_Obj:
6     file_Obj.write(str(x))    # 使用str(x)输出
```

程序实例ch14_28.py： 将数值数据转成字符串数据输出的实例。这个程序执行时在Python Shell窗口看不到结果，必须至ch14工作目录查看所建的out14_28.txt文件，同时打开可以得到下列结果。

执行结果



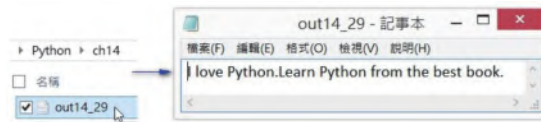
14-3-3 输出多行数据的实例

如果多行数据输出到文件，设计程序时需留意各行间的换行符号问题，write()不会主动在行的末端加上换行符号，如果有需要需自己处理。

程序实例ch14_29.py： 使用write()输出多行数据的实例。

```
1 # ch14_29.py
2 fn = 'out14_29.txt'
3 str1 = 'I love Python.'
4 str2 = 'Learn Python from the best book.'
5
6 with open(fn, 'w') as file_Obj:
7     file_Obj.write(str1)
8     file_Obj.write(str2)
```

执行结果 这个程序执行时在Python Shell窗口看不到结果，必须至ch14工作目录查看所建的out14_29.txt文件，同时打开可以得到下列结果。



其实输出至文件时我们可以使用空格或换行符号，以便获得想要的输出结果。

程序实例ch14_30.py： 增加换行符号方式重新设计ch14_30.py。

```
1 # ch14_30.py
2 fn = 'out14_30.txt'
3 str1 = 'I love Python.'
4 str2 = 'Learn Python from the best book.'
5
6 with open(fn, 'w') as file_Obj:
7     file_Obj.write(str1 + '\n')
8     file_Obj.write(str2 + '\n')
```

执行结果 这个程序执行时在Python Shell窗口看不到结果，必须至ch14工作目录查看所建的out14_30.txt文件，同时打开可以得到下列结果。



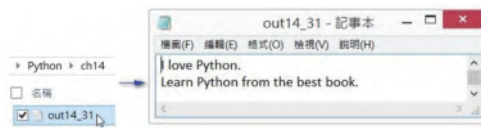
14-3-4 建立附加文件

建立附加文件主要是可以将文件输出到所打开的文件末端，当以open()打开时，需增加参数mode= 'a' 或是用 'a'，其实a是append的缩写。用open()打开文件使用'a'参数时，如果所打开的文件不存在，Python会打开文件供写入；如果所打开的文件存在，Python在执行写入时不会清空原先的文件内容。

程序实例ch14_31.py： 建立附加文件的应用。

```
1 # ch14_31.py
2 fn = 'out14_31.txt'
3 str1 = 'I love Python.'
4 str2 = 'Learn Python from the best book.'
5
6 with open(fn, 'a') as file_Obj:
7     file_Obj.write(str1 + '\n')
8     file_Obj.write(str2 + '\n')
```

执行结果 本书ch14工作目录没有out14_31.txt文件，所以执行第一次时，可以建立out14_31.txt文件，然后得到下列结果。



执行第二次时可以得到下列结果。



上述只要持续执行，输出数据将持续累积。

14-4 shutil模块

这个模块有提供一些方法可以让我们在Python程序内执行文件或目录的复制、删除、更改位置和更改名称。当然在使用前须加上下列加载模块指令。

```
import shutil # 加载模块指令
```

14-4-1 文件的复制copy()

在shutil模块可以使用copy()执行文件的复制，语法格式如下：

```
shutil.copy(source, destination)
```

上述可将source文件复制到destination目的位置，执行前source文件一定要存在否则会产生错误。

程序实例ch14_32.py： 执行文件复制的应用。

```
1 # ch14_32.py
2 import shutil
3
4 shutil.copy('source.txt', 'dest.txt')      # 当前工作目录文件复制
5 shutil.copy('source.txt', 'D:\\Python')    # 当前工作目录文件复制至D:\\Python
6 shutil.copy('D:\\Python\\source.txt', 'D:\\dest.txt') # 不同工作目录文件复制
```

执行结果 这个程序没有列出任何数据，它的说明如下：

第4行，当前工作目录source.txt复制一份在当前工作目录，文件名是dest.txt。

第5行，当前工作目录source.txt使用相同名称复制一份在D:\\Python。

第6行，D:\\Python目录source.txt复制一份在D:\\，名称是dest.txt。

14-4-2 目录的复制copytree()

copytree()的语法格式与copy()相同，只不过这是复制目录，复制时目录底下的子目录或文件也将被复制，此外，执行前目录一定要存在否则会产生错误。

程序实例ch14_33.py： 目录复制的应用。

```
1 # ch14_33.py
2 import shutil
3
4 shutil.copytree('old14', 'new14')          # 当前工作目录的目录复制
5 shutil.copytree('D:\\Python\\old14', 'D:\\new14') # 不同工作目录的目录复制
```

执行结果 这个程序没有列出任何数据，它的说明如下：

第4行，当前工作目录old14复制一份在当前工作目录，名称是new14。

第5行，D:\Python复制old14目录至D:\，名称是new14。

14-4-3 文件的移动move()

在shutil模块可以使用move()执行文件的移动，语法格式如下：

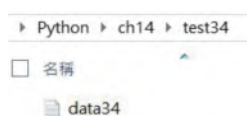
```
shutil.move(source, destination)
```

上述可将source文件移动到destination目的位置，执行前source文件一定要存在否则会产生错误，执行后source文件将不再存在。

程序实例ch14_34.py： 将当前目录的data34.txt移至当前目录的test34子目录。

```
1 # ch14_34.py
2 import shutil
3
4 shutil.move('data34.txt', './test34') # 移动当前工作目录data34.txt
```

执行结果 执行前当前目录下需有test34子目录，然后可以得到下列结果。



14-4-4 文件名的更改move()

在移动过程如果destination路径含有文件名，则可以达到更改名称的效果。

```
1 # ch14_35.py
2 import shutil
3
4 shutil.move('data35.txt', 'out35.txt') # 更改文件名
```

程序实例ch14_35.py： 在同目录下更改文件名。上述程序会将data35.txt改名为out35.txt。

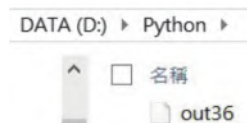
在文件移动过程中若是destination的目录不存在，也将造成文件名的更改。

程序实例ch14_36.py： 文件名更改的另一种状况。

执行结果

```
1 # ch14_36.py
2 import shutil
3
4 shutil.move('data36.txt', 'D:\\Python\\out36') # out36不存在
```

执行结果 下列是验证结果。



上述执行前D:\Python\out36不存在，将以D:\Python\out36.txt存储此文件。

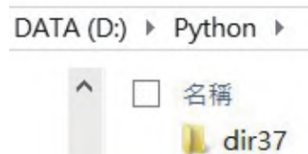
14-4-5 目录的移动move()

这个move()也可以执行目录的移动，在移动时子目录也将随着移动。

程序实例 ch14_37.py： 将当前工作目录的子目录dir37移至D:\Python目录下。

```
1 # ch14_37.py
2 import shutil
3
4 shutil.move('dir37', 'D:\\Python')
```

执行结果 下列是验证结果。



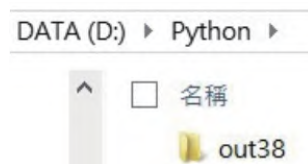
14-4-6 目录的更改名称move()

如果在移动过程destination的目录不存在，此时就可以达到目录更改名称的目的了，甚至路径名称也可能更改。

程序实例ch14_38.py： 将当前子目录dir38移动至D:\Python同时改名为out38。

```
1 # ch14_38.py
2 import shutil
3
4 shutil.move('dir38', 'D:\\Python\\out38')
```

执行结果



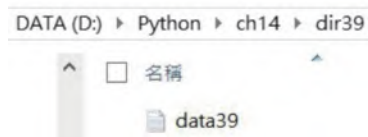
14-4-7 删除底下有数据的目录rmtree()

os模块的rmdir()只能删除空的目录，如果要删除含数据文件的目录须使用本节所述的rmtree()。

程序实例ch14_39.py： 删除dir39目录，这个目录下有数据文件data39.txt。执行后下列D:\Python\ch14\dir39将被删除。

```
1 # ch14_39.py
2 import shutil
3
4 shutil.rmtree('dir39')
```

执行结果



14-4-8 安全删除文件或目录send2trash()

Python内置的shutil模块在删除文件后就无法复原了，当前有一个第三方的模块send2trash，执行删除文件或文件夹后是将被删除的文件放在回收站，如果后悔可以救回。不过在使用此模块前须先下载这个外部模块。可以进入安装Python的文件夹，然后在DOS环境安装此模块，安装指令如下：

```
pip install send2trash
```

有关安装第3方模块的方法可参考附录B。安装完成后就可以使用下列方式删除文件或目录了。

```
import send2trash # 导入send2trash模块

send2trash.send2trash(文件或文件夹) # 语法格式
```

程序实例ch14_40.py： 删除文件data40.txt，未来可以在回收站找到此文件。

```
1 # ch14_40.py
2 import send2trash
3
4 send2trash.send2trash('data40.txt')
```

执行结果 通过回收站可以找到data40.txt。

14-5 文件压缩与解压缩zipFile

Windows操作系统有提供功能将一般文件或目录压缩，压缩后的扩展名是zip，Python内有zipFile模块也可以将文件或目录压缩以及解压缩。当然程序开头需要加上下列指令导入此模块。

```
import zipfile
```

14-5-1 执行文件或目录的压缩

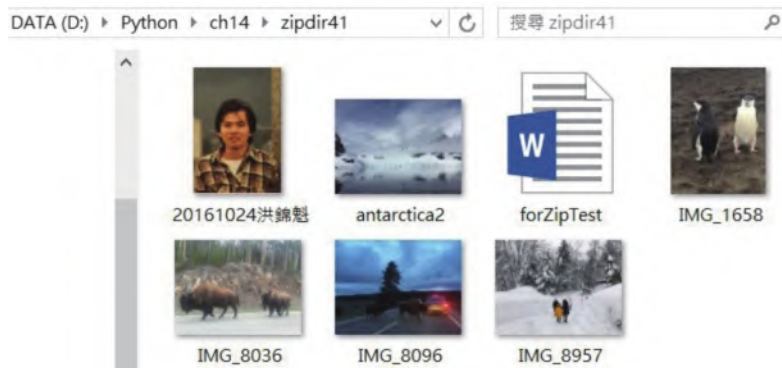
执行文件压缩前首先要使用ZipFile()方法建立一份压缩后的档名，在这个方法中另外要加上‘w’参数，注明未来是供write()方法写入。

```
fileZip  
  
= zipfile.ZipFile('out.zip'  
, 'w') # out.zip
```

是未来储存压缩结果

上述fileZip和out.zip皆可以自由设定名称，fileZip是压缩文件对象 代表的是out.zip，未来将被压缩的文件数据写入此对象，就可以将结果存入out.zip。虽然ZipFile()无法执行整个目录的压缩，不过可用循环方式将目录底下的文件压缩，即可达到压缩整个目录的目的。

程序实例ch14_41.py： 这个程序会将当前工作目录底下的zipdir41目录压缩，压缩结果储存在out41.zip内。这个程序执行前的zipdir41内容如下：



下列是程序内容。

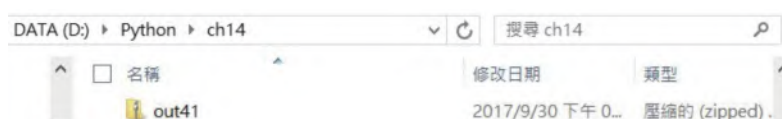
```

1 # ch14_41.py
2 import zipfile
3 import glob, os
4
5 fileZip = zipfile.ZipFile('out41.zip', 'w')
6 for name in glob.glob('zipdir41/*'): # 遍历zipdir41目录
7     fileZip.write(name, os.path.basename(name), zipfile.ZIP_DEFLATED)
8
9 fileZip.close()

```

↑
说明压缩方式

执行结果 可以在相同目录得到下列压缩文件out41。



14-5-2 读取zip文件

ZipFile对象有`namelist()`方法可以传回zip文件内所有被压缩的文件或目录名称，同时以列表方式传回此对象。这个传回的对象可以使用`infolist()`方法传回各元素的属性，如文件名`filename`、文件大小`file_size`、压缩结果大小`compress_size`、文件时间`data_time`。

程序实例ch14_42.py: 将ch14_41.py所建的zip文件解析，列出所有被压缩的文件，以及文件名、文件大小和压缩结果大小。


```

1 # ch14_42.py
2 import zipfile
3
4 listZipInfo = zipfile.ZipFile('out41.zip', 'r')
5 print(listZipInfo.namelist())      # 以列表列出所有压缩文件案
6 print("\n")
7 for info in listZipInfo.infolist():
8     print(info.filename, info.file_size, info.compress_size)

```

执行结果

```

===== RESTART: D:\Python\ch14\ch14_42.py =====
['20161024洪錦魁.jpg', 'antarctica2.jpg', 'forZipTest.docx', 'IMG_1658.jpg', 'IMG_8036.jpg', 'IMG_8096.jpg', 'IMG_8957.JPG']

20161024洪錦魁.jpg 166763 166531
antarctica2.jpg 1440258 1430105
forZipTest.docx 1266045 1252488
IMG_1658.jpg 1478242 1475740
IMG_8036.jpg 2885322 2877251
IMG_8096.jpg 1473764 1471145
IMG_8957.JPG 129424 126337
>>>

```

14-5-3 解压缩zip文件

解压缩zip文件可以使用extractall()方法。

程序实例ch14_43.py： 将程序实例ch14_41.py所建的out41.zip解压缩，同时将解压缩结果存入out43目录。

```

1 # ch14_43.py
2 import zipfile
3
4 fileUnZip = zipfile.ZipFile('out41.zip')
5 fileUnZip.extractall('out43')
6 fileUnZip.close()

```

执行结果



14-6 认识编码格式encode

当前为止所谈到的文本文件(.txt)的文件打开有关文件编码部分皆是使用Windows操作系统默认方式，文本模式下常用的编码方式有

utf-8和cp950。使用open()打开文件时，可以增加另一个常用的参数encoding，整个open()的语法将如下所示：

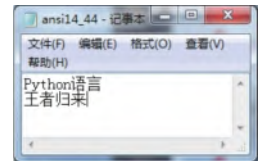
```
file_Obj = open(file, mode="r", encoding="utf-8")
```

14-6-1 中文Windows操作系统记事本默认的编码

请打开中文Windows操作系统的记事本建立下列文件。

请执行“文件 /另存为” 命令。

上述默认编码是ANSI，在这个编码格式下，在Python的open()内我们可以使用预设的encoding=“cp950”编码，因为这是Python预设所以我们可以省略此参数。请将上述文件使用默认的ANSI编码存至ansil4_44.txt。



程序实例ch14_44.py： 使用encoding=“950”打开ansil4_44.txt，然后输出。

```
1 # ch14_44.py
2
3 fn = 'ansil4_44.txt'          # 设定欲开启的文件
4 file_Obj = open(fn, encoding='cp950') # 用预设encoding='cp950'开启文件
5 data = file_Obj.read()       # 读取文件到变量data
6 file_Obj.close()             # 关闭文件对象
7 print(data)                  # 输出变量data相当于输出文件
```

执行结果

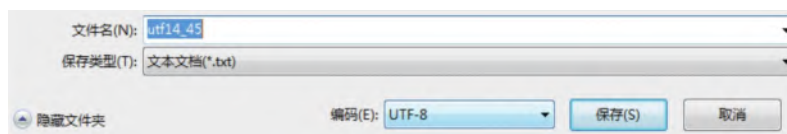
```
===== RESTART: D:\Python\ch14\ch14_44.py =====
Python语言
王者归来
>>>
```

14-6-2 utf-8编码

utf-8英文全名是8-bit Unicode Transformation Format，这是一种适合多语系的编码规则，主要方法是使用可变长度字节方式储存字符，以节省内存空间。例如，对于英文字母而言是使用1个字节空间储存即可，对于含有附加符号的希腊文、拉丁文或阿拉伯文等则用2个字节空间储存字符，两岸华人所使用的中文字则是以3个字节空间储存字符，只有极少数的平面辅助文字需要4个字节空间储存字符。也就是说这种编码规则已经包含了全球所有语言的字符了，所以采用这种编码方式设计网页时，其他国家的浏览器只要支持utf-8编码皆可显示。例如，美国人即使使用英文版的Internet Explorer浏览器，也可以正常显示中文字。

另外，有时我们在网络世界浏览其他国家的网页时，会发生显示乱码情况，主要原因就是对方网页设计师并没有将此属性设为“utf-8”。例如，早期最常见的是，中国大陆简体中文的编码是“gb2312”，这种编码方式是以2个字符组储存一个简体中文字，由于这种编码方式不适用多语系，无法在繁体中文Windows环境中使用，所以如果中国大陆的网页设计师采用此编码，将造成港、澳或台湾繁体中文Windows的用户在繁体中文窗口环境浏览此网页时出现乱码。

其实utf-8是国际通用的编码，如果你使用Linux或Mac OS，一般也是用国际编码，所以如果打开文件发生错误，请先检查文件的编码格式。请打开ansi14_44.txt文件，然后执行另存新文件，此时编码规则请选utf-8编码，将文件存入utf14_45.txt，如下所示：



程序实例 ch14_45.py：重新设计 ch14_44.py，使用 encoding=‘950’ 打开文件发生错误的实例。

```

1 # ch14_45.py
2
3 fn = 'utf14_45.txt'          # 设定欲开启的文件
4 file_Obj = open(fn, encoding='cp950') # 用预设encoding='cp950'开启文件
5 data = file_Obj.read()      # 读取文件到变量data
6 file_Obj.close()           # 关闭文件对象
7 print(data)                 # 输出变量data相当于输出文件

```

执行结果

```

===== RESTART: D:\Python\ch14\ch14_45.py =====
Traceback (most recent call last):
  File "D:\Python\ch14\ch14_45.py", line 5, in <module>
    data = file_Obj.read() # 读取文件到变量data
UnicodeDecodeError: 'cp950' codec can't decode byte 0x9e in position 11: illegal
multibyte sequence
>>>

```

上述很明显指出是decode错误。

程序实例ch14_46.py: 重新设计ch14_45.py, 使用encoding= ‘utf-8’ 。

```

1 # ch14_46.py
2
3 fn = 'utf14_45.txt'          # 设定欲开启的文件
4 file_Obj = open(fn, encoding='utf-8') # 用encoding='utf-8'开启文件
5 data = file_Obj.read()      # 读取文件到变量data
6 file_Obj.close()           # 关闭文件对象
7 print(data)                 # 输出变量data相当于输出文件

```

执行结果

```

===== RESTART: D:\Python\ch14\ch14_46.py =====
Python语言
王者归来
>>>

```

14-6-3 认识utf-8编码的BOM

使用中文Windows操作系统的记事本以utf-8执行编码时，操作系统会在文件前端增加字节顺序记号 (Byte Order Mark, BOM)，俗称文件前端代码，主要功能是判断文字以Unicode表示时，字节的排序方式。

程序实例ch14_47.py: 重新设计ch14_20.py, 使用逐行读取方式读取utf-8编码格式的utf14_45.txt文件，验证BOM的存在。

```

1 # ch14_47.py
2
3 fn = 'utf14_45.txt' # 设定欲开启的文件
4 with open(fn, encoding='utf-8') as file_Obj: # 开启utf-8文件
5     obj_list = file_Obj.readlines() # 每次读一行
6
7 print(obj_list) # 打印串行

```

执行结果

```

===== RESTART: D:\Python\ch14\ch14_47.py =====
['\ufeffPython语言\n', '王者归来\n']
>>>

```

从上述执行结果可以看到\ufeff 字符，其实u代表这是Unicode编码格式，fe和ff是16进位的编码格式，这是代表编码格式。在utf-8的编码中有2种编码格式主张，有一派主张数值较大的byte要放在前面，这种方式称Big Endian(BE) 系统。另一派主张数值较小的byte要放在前面，这种方式称Little Endian(LE) 系统。以笔者名字的“魁”为例，Unicode的码值是0x9B41，若是使用BE系统，码值编法是0x9B 0x41 。若是使用LE系统，码值编法是0x41 0x9B 。当前Windows系统的编法是LE系统，它的BOM内容是\ufeff ，由于当前没有所谓的\ufffe 内容，所以一般就用BOM内容是\ufeff 代表这是LE的编码系统。这2个字符在Unicode中不占空间，所以许多时候是感觉不到它们的存在的。

open()函数使用时，也可以很明确地使用encoding= ‘utf-8-sig’ 格式，这时即使是逐行读取也可以将BOM去除。

程序实例ch14_48.py： 重新设计ch14_47.py，使用encoding= ‘utf-8-sig’ 格式。

```

1 # ch14_48.py
2
3 fn = 'utf14_45.txt' # 设定欲开启的文件
4 with open(fn, encoding='utf-8-sig') as file_Obj: # 开启utf-8文件
5     obj_list = file_Obj.readlines() # 每次读一行
6
7 print(obj_list) # 打印列表

```

执行结果 从执行结果可以看到\ufeff字符没有了。

```

===== RESTART: D:\Python\ch14\ch14_48.py =====
['Python语言\n', '王者归来\n']
>>>

```

程序实例 ch14_49.py：重新设计 ch14_47.py，这次改读取 utf14_49.txt，并观察执行结果，可以看到\ufeff字符不见了。

```
1 # ch14_49.py
2
3 fn = 'utf14_49.txt'          # 设定欲开启的文件
4 with open(fn, encoding='utf-8') as file_Obj:  # 开启utf-8文件
5     obj_list = file_Obj.readlines()          # 每次读一行
6
7 print(obj_list)              # 打印列表
```

执行结果

```
===== RESTART: D:\Python\ch14\ch14_49.py =====
['Python语言\n', '王者归来\n']
>>>
```

14-7 剪贴板的应用

剪贴板的功能是属第三方pyperclip模块内，使用前需使用下列方式安装此模块，更多知识可参考附录B：

```
pip install pyperclip
```

然后程序前面加上下列导入pyperclip模块功能。

```
import pyperclip
```

安装完成后就可以使用下列两个方法：

- copy()：可将列表数据复制至剪贴板。
- paste()：将剪贴板数据复制回字符串变量。

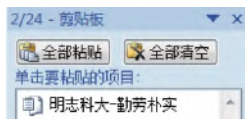
程序实例ch14_50.py：将数据复制至剪贴板，再将剪贴板数据复制回字符串变量string，同时打印string字符串变量。

```
1 # ch14_50.py
2 import pyperclip
3
4 pyperclip.copy('明志科大-勤劳朴实')          # 将字符串复制至剪贴板
5 string = pyperclip.paste()                      # 从剪贴板复制回string
6 print(string)                                   # 打印
```

执行结果


```
===== RESTART: D:\Python\ch14\ch14_50.py =====  
明志科大-勤劳朴实  
>>>
```

其实上述执行第4行后，如果你打开剪贴板（可打开Word再进入剪贴板功能）可以看到“明志科大-勤劳朴实”字符串已经出现在剪贴板。程序第5行则是将剪贴板数据复制至string字符串变量，第6行则是打印string字符串变量。



习题

1. 请更改设计ch14_22.py，让各行字符串间空一格，下列是执行结果。

明志工专 台北工专 我爱明志工专

2. 本章讲解了读取文件的知识，也讲解了写入文件的知识，请设计一个copy程序，将一个文件写入另一个文件内。程序执行时会先要求输入原始档的档名，然后要求输入目的文件的文件名，程序会将原始文件的内容写入目的档内。
3. 有5个字符列表内容如下：

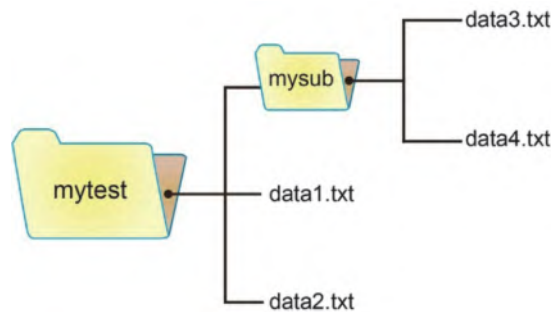
```
str1 = 'Python入门到高手之路'  
str2 = '作者：洪锦魁'  
str3 = '深石数字科技'  
str4 = 'DeepStone Corporation'  
str5 = 'Deep Learning'
```

请依上述字符串执行下列工作：

- A：分5行输出，将执行结果存入ex3_1.txt。
- B：同一行输出，彼此不空格，将执行结果存入ex3_2.txt。

C: 同一行输出，彼此空一格，将执行结果存入ex3_3.txt。

4. 请一次读取ex3_1.txt，然后输出到屏幕。
5. 请一次一行读取ex3_1.txt，然后输出到屏幕。
6. 请一次一行读取ex3_1.txt，然后处理成一行且彼此不空格，然后输出到屏幕。
7. 请使用ex3_1.txt的内容进行Python程序设计，将字符串‘高手’改为‘专家’，将结果存入ex3_4.txt。
8. 请在当前工作目录建立下列目录mytest：



请设计程序将上述目录移至C:\mytest目录下。

9. 请参考ch14_41.py，使用程序设计列出zipdir41内最大的文件与最小的文件。
10. 请将习题8的目录mytest使用Python设计程序压缩，压缩的文件名是自己的名字，然后Email给老师。
11. 请设计程序将习题8的目录名称改为自己的名字，然后使用send2trash安全删除，最后在资源回收桶救回。请使用Word的插入 / 图例 / 屏幕快照 功能截取所删除目录在资源回收桶的画面，然后交此画面给老师。

第15章 程序除错与异常处理

15-1 程序异常

有时也可以将程序错误 (error) 称作程序异常 (exception)，相信每一位写程序的人一定会常常碰上程序错误，过去碰上这类情况程序将终止执行，同时出现错误信息，错误信息内容通常是显示 Traceback，然后列出异常报告。Python 提供功能可以让我们捕捉异常和撰写异常处理程序，当发生异常被我们捕捉时会去执行异常处理程序，然后程序可以继续执行。

15-1-1 一个除数为0的错误

本节将以一个除数为0的错误开始说明。

程序实例 ch15_1.py：建立一个除法运算的函数，这个函数将接受2个参数，然后执行第一个参数除以第二个参数。

```
1 # ch15_1.py
2 def division(x, y):
3     return x / y
4
5 print(division(10, 2))    # 列出10/2
6 print(division(5, 0))    # 列出5/0
7 print(division(6, 3))    # 列出6/3
```

执行结果

```
===== RESTART: D:/Python/ch15/ch15_1.py =====
5.0
Traceback (most recent call last):
  File "D:/Python/ch15/ch15_1.py", line 6, in <module>
    print(division(5, 0))    # 列出5/0
  File "D:/Python/ch15/ch15_1.py", line 3, in division
    return x / y
ZeroDivisionError: division by zero
>>>
```

上述程序在执行第5行时，一切还是正常。但是到了执行第6行时，因为第2个参数是0，导致发生ZeroDivisionError : division by zero的错误，所以整个程序就执行终止了。其实对于上述程序而言，若是程序可以执行第7行，是可以正常得到执行结果的，可是程序第6行已经造成程序终止了，所以无法执行第7行。

15-1-2 撰写异常处理程序try - except

这一小节笔者将讲解如何捕捉异常与设计异常处理程序，发生异常被捕捉时程序会执行异常处理程序，然后跳开异常位置，再继续往下执行。这时要使用try - except指令，它的语法格式如下：

```
try:
    指令                # 预先设想可能引发错误异常的指令
except 异常对象:        # 若以 ch15_1.py 而言，异常对象就是指 ZeroDivisionError
    异常处理程序        # 通常是指出异常原因，方便修正
```

上述会执行try:下面的指令，如果正常则跳离except部分，如果指令有错误异常，则检查此异常是否是异常对象 所指的错误，如果是代表异常被捕捉了，则执行此异常对象 下面的异常处理程序。

程序实例ch15_2.py: 重新设计ch15_1.py，增加异常处理程序。

```
1 # ch15_2.py
2 def division(x, y):
3     try:                # try - except指令
4         return x / y
5     except ZeroDivisionError: # 除数为0时执行
6         print("除数不可为0")
7
8 print(division(10, 2))    # 列出10/2
9 print(division(5, 0))    # 列出5/0
10 print(division(6, 3))   # 列出6/3
```

执行结果

```
===== RESTART: D:\Python\ch15\ch15_2.py
5.0
除数不可为0
None
2.0
>>>
```

上述程序执行第8行时，会将参数(10, 2)带入division()函数，由于执行try的指令的“x / y”没有问题，所以可以执行“return x / y”，这时Python将跳过except的指令。当程序执行第9行时，会将参数(5, 0)带入division()函数，由于执行try的指令的“x / y”产生了除数为0的ZeroDivisionError 异常，这时Python会找寻是否有处理这类异常的except ZeroDivisionError 存在，如果有就表示此异常被捕捉，就去执行相关的错误处理程序，此例是执行第6行，印出“除数不可为0”的错误。函数返回然后印出结果None，None是一个对象表示结果不存在，最后返回程序第10行，继续执行相关指令。

从上述可以看到，程序增加了try - except后，若是异常被except捕捉，出现的异常信息比较友善了，同时不会有程序中断的情况发生。

特别需留意的是在try - except的使用中，如果在try:后面的指令产生异常时，这个异常不是我们设计的except 异常对象，表示异常没被捕捉到，这时程序依旧会像ch15_1.py一样，直接出现错误信息，然后程序终止。

程序实例ch15_2_1.py： 重新设计ch12_2.py，但是程序第9行使用字符 调用除法运算，造成程序异常。

```
1 # ch15_2_1.py
2 def division(x, y):
3     try:                                # try - except指令
4         return x / y
5     except ZeroDivisionError:          # 除数为0时执行
6         print("除数不可为0")
7
8 print(division(10, 2))                 # 列出10/2
9 print(division('a', 'b'))             # 列出'a' / 'b'
10 print(division(6, 3))                 # 列出6/3
```

执行结果

```
===== RESTART: D:/Python/ch15/ch15_2_1.py =====
5.0
Traceback (most recent call last):
  File "D:/Python/ch15/ch15_2_1.py", line 9, in <module>
    print(division('a', 'b')) # 列出'a' / 'b'
  File "D:/Python/ch15/ch15_2_1.py", line 4, in division
    return x / y
TypeError: unsupported operand type(s) for /: 'str' and 'str'
>>>
```

由上述执行结果可以看到异常原因是TypeError，由于我们在程序中没有设计except TypeError的异常处理程序，所以程序会终止执行。更多相关处理将在15-2节说明。

15-1-3 try - except - else

Python在try - except中又增加了else指令，这个指令存放的主要目的是try内的指令正确时，可以执行else内的指令区块，我们可以将这部分指令区块称正确处理程序，这样可以增加程序的可读性。此时语法格式如下：

程序实例 ch15_3.py： 使用 try - except - else 重新设计 ch15_2.py。

```
try:
    指令          # 预先设想可能引发异常的指令
except 异常对象:  # 若以 ch15_1.py 而言，异常对象就是指 ZeroDivisionError
    异常处理程序 # 通常是指出异常原因，方便修正
else:
    正确处理程序 # 如果指令正确执行此区块指令

1 # ch15_3.py
2 def division(x, y):
3     try:          # try - except指令
4         ans = x / y
5     except ZeroDivisionError: # 除数为0时执行
6         print("除数不可为0")
7     else:
8         return ans      # 传回正确的执行结果
9
10 print(division(10, 2))  # 列出10/2
11 print(division(5, 0))  # 列出5/0
12 print(division(6, 3))  # 列出6/3
```

执行结果：与ch15_2.py相同。

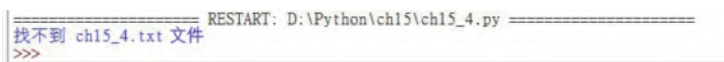
15-1-4 找不到文件的错误FileNotFoundError

程序设计时另一个常常发生的异常是打开文件时找不到文件，这时会产生FileNotFoundError异常。

程序实例ch15_4.py: 打开一个不存在的文件ch15_4.txt产生异常的实例, 这个程序会有一个异常处理程序, 列出文件不存在。如果文件存在则打印文件内容。

```
1 # ch15_4.py
2
3 fn = 'ch15_4.txt'          # 设定欲开启的文件
4 try:
5     with open(fn) as file_Obj: # 用默认mode=r开启文件, 传回文件对象file_Obj
6         data = file_Obj.read() # 读取文件到变量data
7 except FileNotFoundError:
8     print("找不到 %s 文件" % fn)
9 else:
10    print(data)              # 输出变量data相当于输出文件
```

执行结果

A screenshot of a Python interpreter window. The title bar reads "RESTART: D:\Python\ch15\ch15_4.py". The main text area displays the error message "找不到 ch15_4.txt 文件" (File not found: ch15_4.txt) in red, followed by the prompt ">>>" in green.

本文件夹ch15内有ch15_5.txt, 相同的程序只是第4行打开的文件不同, 将可以获得打印出ch15_5.txt。

程序实例ch15_5.py: 与ch15_4.py内容基本上相同, 只是打开的文件不同。

```
3 fn = 'ch15_5.txt'          # 设定欲开启的文件
```

执行结果

A screenshot of a Python interpreter window. The title bar reads "RESTART: D:/Python/ch15/ch15_5.py". The main text area displays the content of the file "ch15_5.txt" in blue: "深石数位科技", "深度学习滴水穿石", and "Deep Learning". The prompt ">>>" is visible at the bottom.

15-1-5 分析单一文件的字数

有时候在读一篇文章时, 可能会想知道这篇文章的字数, 这时我们可以采用下列方式分析。在正式分析前, 可以先来看一个简单的程序应用。如果忘记split()方法, 可重新温习6-9-6节。

程序实例ch15_6.py: 分析一个文件内有多少个单字。

```

1 # ch15_6.py
2
3 fn = 'ch15_6.txt'          # 设定欲开启的文件
4 try:
5     with open(fn) as file_Obj: # 用默认mode=r开启文件,传回文件对象file_Obj
6         data = file_Obj.read() # 读取文件到变量data
7 except FileNotFoundError:
8     print("找不到 %s 文件" % fn)
9 else:
10    wordList = data.split()    # 将文章转成列表
11    print(fn, " 文章的字数是 ", len(wordList)) # 打印文章字数

```

执行结果

```

===== RESTART: D:\Python\ch15\ch15_6.py =====
ch15_6.txt 文章的字数是 43
>>>

```

如果程序设计时常常有需要计算某篇文章的字数，可以考虑将上述计算文章的字数处理成一个函数，这个函数的参数是文章的文件名，然后函数直接打印出文章的字数。

程序实例ch15_7.py： 设计一个计算文章字数的函数wordsNum，只要传递文章文件名，就可以获得此篇文章的字数。

```

1 # ch15_7.py
2 def wordsNum(fn):
3     """适用英文文件，输入文章的文件名,可以计算此文章的字数"""
4     try:
5         with open(fn) as file_Obj: # 用默认"r"传回文件对象 file_Obj
6             data = file_Obj.read() # 读取文件到变量data
7     except FileNotFoundError:
8         print("找不到 %s 文件" % fn)
9     else:
10        wordList = data.split()    # 将文章转成列表
11        print(fn, " 文章的字数是 ", len(wordList)) # 打印文章字数
12
13 file = 'ch15_6.txt'
14 wordsNum(file)

```

执行结果 与ch15_6.py相同。

15-1-6 分析多个文件的字数

程序设计时你可能需设计读取许多文件做分析，部分文件可能存在，部分文件可能不存在，这时就可以使用本节的观念做设计了。在接下来的程序实例分析中，笔者将欲读取的文件名放在列表内，然后使用循环将文件分次传给程序实例ch15_7.py建立的wordsNum函数，如果文件存在将打印出字数，如果文件不存在将列出找不到此文件。

程序实例ch15_8.py： 分析data1.txt、data2.txt、data3.txt这3个文件的字数，同时笔者在ch15文件夹没有放置data2.txt，所以程序遇到分析此文件时，将列出找不到此文件。

```
1 # ch15_8.py
2 def wordsNum(fn):
3     """适用英文文件，输入文章的文件名，可以计算此文章的字数"""
4     try:
5         with open(fn) as file_Obj: # 用默认"r"传回文件对象file_Obj
6             data = file_Obj.read() # 读取文件到变量data
7     except FileNotFoundError:
8         print("找不到 %s 文件" % fn)
9     else:
10        wordList = data.split() # 将文章转成列表
11        print(fn, " 文章的字数是 ", len(wordList)) # 打印文章字数
12
13 files = ['data1.txt', 'data2.txt', 'data3.txt'] # 文件列表
14 for file in files:
15     wordsNum(file)
```

执行结果

```
===== RESTART: D:\Python\ch15\ch15_8.py =====
data1.txt 文章的字数是 43
找不到 data2.txt 文件
data3.txt 文章的字数是 39
>>>
```

15-2 设计多组异常处理程序

在程序实例ch15_1.py、ch15_2.py和ch15_2_1.py的实例中，我们清楚地了解了程序设计中有太多各种不可预期的异常发生，所以我们知道设计程序时可能需要同时设计多个异常处理程序。

15-2-1 常见的异常对象

异常对象名称	说明
AttributeError	通常是指对象没有这个属性
Exception	一般错误皆可使用
FileNotFoundError	找不到 <code>open()</code> 打开的文件
IOError	在输入或输出时发生错误
IndexError	索引超出范围区间
KeyError	在映射中没有这个键
MemoryError	需求内存空间超出范围
NameError	对象名称未声明
SyntaxError	语法错误
SystemError	直译器的系统错误
TypeError	数据类型错误
ValueError	传入无效参数
ZeroDivisionError	除数为 0

在ch15_2_1.py的程序应用中可以发现，异常发生时如果except设定的异常对象不是发生的异常，相当于except没有捕捉到异常，所设计的异常处理程序变成无效的异常处理程序。Python提供了一个通用型的异常对象Exception，它可以捕捉各式的基础异常。

程序实例 ch15_9.py： 重新设计 ch15_2_1.py，异常对象设为Exception。

```

1  # ch15_9.py
2  def division(x, y):
3      try:
4          return x / y
5      except Exception:
6          print("通用错误发生")
7
8  print(division(10, 2))
9  print(division(5, 0))
10 print(division('a', 'b'))
11 print(division(6, 3))

```

try - except指令
通用错误使用
列出10/2
列出5/0
列出'a' / 'b'
列出6/3

执行结果

```

===== RESTART: D:\Python\ch15\ch15_9.py
5.0
通用错误发生
None
通用错误发生
None
2.0
>>>

```


从上述可以看到第9行除数为0 或是第10行字符相除 所产生的异常皆可以使用except Exception予以捕捉，然后执行异常处理程序。甚至这个通用型的异常对象也可以应用在取代FileNotFoundError异常对象。

程序实例ch15_10.py： 使用Exception取代FileNotFoundError，重新设计ch15_8.py。

```
7  except Exception:
8      print("Exception找不到 %s 文件" % fn)
```

执行结果

```
===== RESTART: D:\Python\ch15\ch15_10.py =====
data1.txt 文章的字数是 43
Exception找不到 data2.txt 文件
data3.txt 文章的字数是 39
>>>
```

15-2-2 设计捕捉多个异常

在try: - except的使用中，可以设计多个except捕捉多种异常，此时语法如下：

```
try:
    指令                                # 预先设想可能引发错误异常的指令
except 异常对象 1:                      # 如果指令发生异常对象 1 执行
    异常处理程序 1
except 异常对象 2:                      # 如果指令发生异常对象 2 执行
    异常处理程序 2
```

当然也可以视情况设计更多异常处理程序。

程序实例ch15_11.py： 重新设计ch15_9.py设计捕捉2个异常对象，可参考第5行和第7行。

```

1 # ch15_11.py
2 def division(x, y):
3     try:                                # try - except指令
4         return x / y
5     except ZeroDivisionError:           # 除数为0使用
6         print("除数为0发生")
7     except TypeError:                   # 数据类型错误
8         print("使用字符做除法运算异常")
9
10 print(division(10, 2))                  # 列出10/2
11 print(division(5, 0))                  # 列出5/0
12 print(division('a', 'b'))              # 列出'a' / 'b'
13 print(division(6, 3))                  # 列出6/3

```

执行结果 与ch15_9.py相同。

15-2-3 使用一个except捕捉多个异常

Python也允许设计一个except，捕捉多个异常，此时语法如下：

```

try:
    指令                                # 预先设想可能引发错误异常的指令
except (异常对象1, 异常对象2, ...):    # 指令发生其中所列异常对象执行
    异常处理程序

```

程序实例ch15_12.py： 重新设计ch15_11.py，用一个except捕捉2个异常对象，下列程序读者需留意第5行的except的写法。

```

1 # ch15_12.py
2 def division(x, y):
3     try:                                # try - except指令
4         return x / y
5     except (ZeroDivisionError, TypeError): # 2个异常
6         print("除数为0发生 或 使用字符做除法运算异常")
7
8 print(division(10, 2))                  # 列出10/2
9 print(division(5, 0))                  # 列出5/0
10 print(division('a', 'b'))              # 列出'a' / 'b'
11 print(division(6, 3))                  # 列出6/3

```

执行结果

```

===== RESTART: D:\Python\ch15\ch15_12.py =====
5.0
除数为0发生 或 使用字符做除法运算异常
None
除数为0发生 或 使用字符做除法运算异常
None
2.0
>>>

```

15-2-4 处理异常但是使用Python内置的错误信息

在先前所有实例，发生异常同时被捕捉皆是使用我们自建的异常处理程序，Python也支持发生异常时使用系统内置的异常处理信息。此时语法格式如下：

```
try:
    指令                                     # 预先设想可能引发错误异常的指令
except 异常对象 as e:                       # 使用 as e
    print(e)                               # 输出 e
```

上述e是系统内置 的异常处理信息，e可以是任意字符，笔者此处使用e 是因为代表e rror的内涵。当然上述except语法也接受同时处理多个异常对象，可参考下列程序实例第5行。

程序实例ch15_13.py： 重新设计ch15_12.py，使用Python内置的错误信息。

```
1 # ch15_13.py
2 def division(x, y):
3     try:                                     # try - except指令
4         return x / y
5     except (ZeroDivisionError, TypeError) as e: # 2个异常
6         print(e)
7
8 print(division(10, 2))    # 列出10/2
9 print(division(5, 0))    # 列出5/0
10 print(division('a', 'b')) # 列出'a' / 'b'
11 print(division(6, 3))    # 列出6/3
```

执行结果

```
===== RESTART: D:/Python/ch15/ch15_13.py
5.0
division by zero
None
unsupported operand type(s) for /: 'str' and 'str'
None
2.0
>>>
```

上述执行结果的错误信息皆是Python内部的错误信息。

15-2-5 捕捉所有异常

程序设计许多异常是我们不可预期的，很难一次设想周到，Python提供语法让我们可以一次捕捉所有异常，此时try - except语法如下：

程序实例ch15_14.py： 一次捕捉所有异常的设计。

```
try:
    指令                                     # 预先设想可能引发错误异常的指令
except:
    异常处理程序                           # 通常是 print 输出异常说明

1 # ch15_14.py
2 def division(x, y):
3     try:                                # try - except指令
4         return x / y
5     except:                             # 捕捉所有异常
6         print("异常发生")
7
8 print(division(10, 2))                   # 列出10/2
9 print(division(5, 0))                   # 列出5/0
10 print(division('a', 'b'))               # 列出'a' / 'b'
11 print(division(6, 3))                   # 列出6/3
```

执行结果

```
===== RESTART: D:\Python\ch15\ch15_14.py =====
5.0
异常发生
None
异常发生
None
2.0
>>>
```

15-3 丢出异常

前面所介绍的异常皆是Python直译器发现异常时，自行丢出异常对象，如果我们不处理程序就终止执行，如果我们使用try - except处理程序可以在异常中继续执行。这一节要探讨的是，我们设计程序时如果发生某些状况，我们自己将它定义为异常然后丢出异常信息，程序停止正常往下执行，同时让程序跳到自己设计的except去执行。它的语法如下：

```
raise Exception( 'msg' )                 # 调用 Exception, msg 是传递错误信息
...
...
try:
    指令
except Exception as err:                  # err 是任意取的变量名称，内容是 msg
    print( "message" , + str(err))        # 打印错误信息
```

程序实例ch15_15.py： 目前有些金融机构在客户建立网络账号时，会要求密码长度必须在5到8个字符间，接下来我们设计一个程序，这个程序内有passWord()函数，这个函数会检查密码长度，如果长度小于5或是长度大于8皆抛出异常。在第11行会有一系列密码供测试，然后以循环方式执行检查。

```
1 # ch15_15.py
2 def passWord(pwd):
3     """检查密码长度必须是5到8个字符"""
4     pwrlen = len(pwd)          # 密码长度
5     if pwrlen < 5:              # 密码长度不足
6         raise Exception('密码长度不足')
7     if pwrlen > 8:              # 密码长度太长
8         raise Exception('密码长度太长')
9     print('密码长度正确')
10
11 for pwd in ('aaabbbccc', 'aaa', 'aaabbb'): # 测试系列密码值
12     try:
13         passWord(pwd)
14     except Exception as err:
15         print("密码长度检查异常发生: ", str(err))
```

执行结果

```
===== RESTART: D:\Python\ch15\ch15_15.py =====
密码长度检查异常发生: 密码长度太长
密码长度检查异常发生: 密码长度不足
密码长度正确
>>>
```

上述当密码长度不足或密码长度太长，皆会抛出异常，这时passWord()函数返回的是Exception对象（第6和8行），这时原先Exception()内的字符串（‘密码长度不足’或‘密码长度太长’）会通过第14行传给err变量，然后执行第15行内容。

15-4 记录Traceback字符串

相信读者学习至今，已经经历了许多程序设计的错误，每次错误屏幕皆出现Traceback字符串，在这个字符串中指出程序错误的原因。例如，请参考程序实例ch15_2_1.py的执行结果，该程序使用Traceback列出了错误。

如果我们导入 traceback 模块，就可以使用 traceback.format_exc() 记录这个Traceback字符串。

程序实例ch15_16.py： 重新设计程序实例ch15_15.py，增加记录Traceback字符串，这个记录将被记录在errch15_16.txt内。

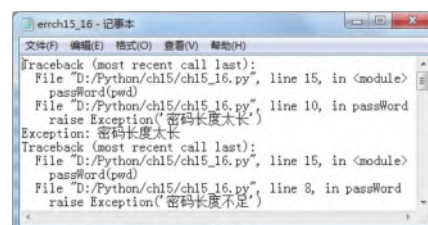
```
1 # ch15_16.py
2 import traceback                                # 导入traceback
3
4 def password(pwd):
5     """检查密码长度必须是5到8个字符"""
6     pwrlen = len(pwd)                          # 密码长度
7     if pwrlen < 5:                             # 密码长度不足
8         raise Exception('The length of pwd is too short')
9     if pwrlen > 8:                             # 密码长度太长
10        raise Exception('The length of pwd is too long')
11    print('密码长度正确')
12
13 for pwd in ('aaabbbccc', 'aaa', 'aaabbb'):    # 测试系列密码值
14     try:
15         password(pwd)
16     except Exception as err:
17         errlog = open('errch15_16.txt', 'a')    # 开启错误文件
18         errlog.write(traceback.format_exc())    # 写入错误文件
19         errlog.close()                        # 关闭错误文件
20         print("将Traceback写入错误文件errch15_16.txt完成")
21         print("密码长度检查异常发生: ", str(err))
```

执行结果

```
===== RESTART: D:/Python/ch15/ch15_16.py =====
将Traceback写入错误文件errch15_16.txt完成
密码长度检查异常发生: The length of pwd is too long
将Traceback写入错误文件errch15_16.txt完成
密码长度检查异常发生: The length of pwd is too short
密码长度正确
>>>
```

如果使用记事本打开 errch15_16.txt，可以得到下列结果。

上述程序第17行笔者使用‘a’附加文件方式打开文件，主要是程序执行期间可能有多个错误，为了记录所有错误所以使用这种方式打开文件。上述程序最关键的地方是第17至19行，在这里我们打开了记录错误的errch15_17.txt文件，然后将错误写入此文件，最后关闭此文件。这个程序记录的错误是我们抛出的异常错误，其实在15-1和15-2节中我们设计了异常处理程序，避免错误造成程序中断，实际上Python还是有记录错误，可参考下一个实例。



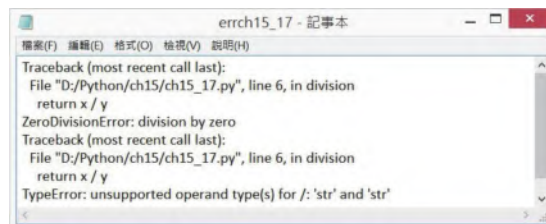
程序实例ch15_17.py: 重新设计ch15_14.py, 主要是将程序异常的信息保存在errch15_17.txt文件内, 本程序的重点是第8至10行。

```
1 # ch15_17.py
2 import traceback
3
4 def division(x, y):
5     try:                                # try - except指令
6         return x / y
7     except:                             # 捕捉所有异常
8         errlog = open('errch15_17.txt', 'a') # 开启错误文件
9         errlog.write(traceback.format_exc()) # 写入错误文件
10        errlog.close()                    # 关闭错误文件
11        print("将Traceback写入错误文件errch15_17.txt完成")
12        print("异常发生")
13
14 print(division(10, 2))                  # 列出10/2
15 print(division(5, 0))                  # 列出5/0
16 print(division('a', 'b'))              # 列出'a' / 'b'
17 print(division(6, 3))                  # 列出6/3
```

执行结果

```
===== RESTART: D:\Python\ch15\ch15_17.py =====
5.0
将Traceback写入错误文件errch15_17.txt完成
异常发生
None
将Traceback写入错误文件errch15_17.txt完成
异常发生
None
2.0
>>>
```

如果使用记事本打开errch15_17.txt, 可以得到下列结果。



```
Traceback (most recent call last):
  File "D:\Python\ch15\ch15_17.py", line 6, in division
    return x / y
ZeroDivisionError: division by zero
Traceback (most recent call last):
  File "D:\Python\ch15\ch15_17.py", line 6, in division
    return x / y
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

15-5 finally

Python的关键词finally功能是和try配合使用, 在try之后可以有except或else, 这个finally关键词必须放在except和else之后, 同时不论是否有异常发生一定会执行这个finally内的程序代码。这个功能主要是用在Python程序与数据库连接时, 输出连接相关信息。

程序实例ch15_18.py: 重新设计ch15_14.py, 增加finally关键词。

```

1 # ch15_18.py
2 def division(x, y):
3     try:
4         return x / y
5     except:
6         print("异常发生")
7     finally:
8         print("阶段任务完成")
9
10 print(division(10, 2), "\n")
11 print(division(5, 0), "\n")
12 print(division('a', 'b'), "\n")
13 print(division(6, 3), "\n")

```

try - except指令
捕捉所有异常
离开函数前先执行此程序代码
列出10/2
列出5/0
列出'a' / 'b'
列出6/3

执行结果

```

===== RESTART: D:\Python\ch15\ch15_18.py =====
阶段任务完成
5.0

异常发生
阶段任务完成
None

异常发生
阶段任务完成
None

阶段任务完成
2.0
>>>

```

上述程序执行时，如果没有发生异常，程序会先输出字符串“阶段任务完成”，然后返回主程序，输出division()的返回值。如果程序有异常会先输出字符串“异常发生”，再执行finally的程序代码输出字符串“阶段任务完成”然后返回主程序输出“None”。

15-6 程序断言assert

15-6-1 设计断言

Python的assert关键词主要功能是协助程序设计师在程序设计阶段，对整个程序的执行状态做一个全面性的安全检查，以确保程序不会发生语意上的错误。例如，我们在第12章设计银行的存款程序时，我们没有考虑到存款或提款是负值的问题，我们也没有考虑到如果提款金额大于存款金额的情况。

程序实例ch15_19.py： 重新设计ch12_4.py，这个程序主要是将第22行的存款金额改为-300和第24行提款金额大于存款金额，接着观察执行结果。


```

1 # ch15_19.py
2 class Banks():
3     # 定义银行类别
4     title = 'Taipei Bank'           # 定义属性
5     def __init__(self, uname, money): # 初始化方法
6         self.name = uname           # 设定存款者名字
7         self.balance = money        # 设定所存的钱
8
9     def save_money(self, money):      # 设计存款方法
10        self.balance += money         # 执行存款
11        print("存款 ", money, " 完成") # 打印存款完成
12
13    def withdraw_money(self, money):   # 设计提款方法
14        self.balance -= money         # 执行提款
15        print("提款 ", money, " 完成") # 打印提款完成
16
17    def get_balance(self):             # 获得存款余额
18        print(self.name.title(), " 目前余额: ", self.balance)
19
20 hungbank = Banks('hung', 100)        # 定义对象hungbank
21 hungbank.get_balance()                # 获得存款余额
22 hungbank.save_money(-300)             # 存款-300元
23 hungbank.get_balance()               # 获得存款余额
24 hungbank.withdraw_money(700)         # 提款700元
25 hungbank.get_balance()               # 获得存款余额

```

执行结果

```

===== RESTART: D:\Python\ch15\ch15_19.py
Hung 目前余额: 100
存款 -300 完成
Hung 目前余额: -200
提款 700 完成
Hung 目前余额: -900
>>>

```

上述程序语法上是没有错误，但是犯了2个程序语意上的设计错误，分别是存款金额出现了负值和提款金额大于存款金额的问题。所以我们发现存款余额出现了负值-200和-900的情况。接下来笔者将讲解如何解决上述问题。

断言(assert) 主要功能是确保程序执行的某个阶段，必须符合一定的条件，如果不符合这个条件时程序主动**抛出异常**，让程序终止同时主动打印出异常原因，方便程序设计师侦错。它的语法格式如下：

```

assert 条件
, '字符串'

```

上述意义是程序执行至此阶段时**测试条件**，如果条件响应是True，程序不理睬逗号“,”右边的字符串 正常往下执行。如果条件响应是False，程序终止同时将逗号“,”右边的字符串 输出到Traceback的字符串内。对上述程序ch15_19.py而言，很明显我们重新设计ch15_20.py时必须让assert关键词做下列2件事：

①确保存款与提款金额是正值，否则输出错误，可参考第10和15行。

②确保提款金额小于等于存款金额，否则输出错误，可参考第16行。

程序实例ch15_20.py： 重新设计ch15_19.py，在这个程序我们先测试存款金额小于0的状况，第27行。

```
1 # ch15_20.py
2 class Banks():
3     # 定义银行类别
4     title = 'Taipei Bank'          # 定义属性
5     def __init__(self, uname, money): # 初始化方法
6         self.name = uname          # 设定存款者名字
7         self.balance = money       # 设定所存的钱
8
9     def save_money(self, money):     # 设计存款方法
10        assert money > 0, '存款money必须大于0'
11        self.balance += money        # 执行存款
12        print("存款 ", money, " 完成") # 打印存款完成
13
14    def withdraw_money(self, money):  # 设计提款方法
15        assert money > 0, '提款money必须大于0'
16        assert money <= self.balance, '存款金额不足'
17        self.balance -= money        # 执行提款
18        print("提款 ", money, " 完成") # 打印提款完成
19
20    def get_balance(self):            # 获得存款余额
21        print(self.name.title(), " 目前余额: ", self.balance)
22
23 hungbank = Banks('hung', 100)      # 定义对象hungbank
24 hungbank.get_balance()              # 获得存款余额
25 hungbank.save_money(300)            # 存款300元
26 hungbank.get_balance()              # 获得存款余额
27 hungbank.save_money(-300)           # 存款-300元
28 hungbank.get_balance()              # 获得存款余额
```

执行结果

```
===== RESTART: D:\Python\ch15\ch15_20.py =====
Hung 目前余额: 100
存款 300 完成
Hung 目前余额: 400
Traceback (most recent call last):
  File "D:\Python\ch15\ch15_20.py", line 27, in <module>
    hungbank.save_money(-300) # 存款-300元
  File "D:\Python\ch15\ch15_20.py", line 10, in save_money
    assert money > 0, '存款money必须大于0'
AssertionError: 存款money必须大于0
>>>
```

上述执行结果很清楚，当程序第27行将存款金额设为负值-300时，调用save_money()方法，结果在第10行的assert断言地方出现False，所以设定的错误信息‘存款必需大余0’的字符串被打印出来，这种设计方便我们在真实的环境做最后程序语意检查。

程序实例ch15_21.py： 重新设计ch15_20.py，这个程序我们测试了当提款金额大于存款金额的状况，可参考第27行，下列只列出主程序内容。

```
23 hungbank = Banks('hung', 100)      # 定义对象hungbank
24 hungbank.get_balance()              # 获得存款余额
25 hungbank.save_money(300)            # 存款300元
26 hungbank.get_balance()              # 获得存款余额
27 hungbank.withdraw_money(700)        # 提款700元
28 hungbank.get_balance()              # 获得存款余额
```

执行结果

```
===== RESTART: D:\Python\ch15\ch15_21.py =====
Hung 目前余额: 100
存款 300 完成
Hung 目前余额: 400
Traceback (most recent call last):
  File "D:\Python\ch15\ch15_21.py", line 27, in <module>
    hungbank.withdraw_money(700) # 提款700元
  File "D:\Python\ch15\ch15_21.py", line 16, in withdraw_money
    assert money <= self.balance, '存款金额不足'
AssertionError: 存款金额不足
>>>
```

上述当提款金额大于存款金额时，这个程序将造成第16行的assert断言条件是False，所以触发了打印‘存款金额不足’的信息。由上述的执行结果，我们就可以依据需要修正程序的内容。

15-6-2 停用断言

断言assert 一般是用在程序开发阶段，如果整个程序设计好了以后，想要停用断言assert，可以在Windows的命令提示环境（可参考附录B-2-1），执行程序时使用“-O”选项停用断言。笔者在Windows 8操作系统安装Python 3.62版本，在这个版本的Python安装路径内~\Python\Python36-32内有python.exe可以执行所设计的Python程序，若以ch15_21.py为实例，如果我们要停用断言可以使用下列指令。

```
~\python.exe -O D:\Python\ch15\ch15_21.py
```

上述“~”代表安装Python的路径，若是以ch15_21.py为例，采用停用断言选项“-O”后，执行结果将不再有Traceback错误信息产生，因为断言被停用了。



```
命令提示符
Microsoft Windows [版本 6.3.9600]
(c) 2013 Microsoft Corporation.

C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\python.exe -O D:\Python\ch15\ch15_21.py
Hung 目前余额: 100
存款 300 完成
Hung 目前余额: 400
提款 700 完成
Hung 目前余额: -300

C:\Users\Jiin-Kwei>
```

15-7 程序日志模块logging

程序设计阶段难免会有错误产生，没有得到预期的结果，在产生错误期间到底发生什么事情？程序代码执行顺序是否有误或变量值如何变化？这些都是程序设计师想知道的事情。笔者过去碰上这方面的问题，常常是在程序代码几个重要节点增加`print()`函数输出关键变量，以了解程序的变化，程序修订完成后再将这几个`print()`删除，坦白说是有一点麻烦。

Python有程序日志`logging` 功能，这个功能可以协助我们执行程序的除错，有了这个功能我们可以自行设定关键变量在每一个程序阶段的变化，由这个关键变量的变化可方便我们执行程序的除错，同时未来不想要显示这些关键变量数据时，可以不用删除，只要适度加上指令就可隐藏它们，这将是本节的主题。

15-7-1 logging模块

Python内有提供`logging`模块，这个模块有提供方法可以让我们使用程序日志`logging` 功能，在使用前须先使用`import`导入此模块。

```
import logging
```

15-7-2 logging的等级

`logging`模块共分5个等级，从最低到最高等级顺序如下：

 **DEBUG**等级使用`logging.debug()`显示程序日志内容，所显示的内容是程序的小细节，最

低层级的内容，感觉程序有问题时可使用它追踪关键变量的变化过程。

- ❑ INFO等级使用`logging.info()`显示程序日志内容，所显示的内容是记录程序一般发生的事件。
- ❑ WARNING等级使用`logging.warning()`显示程序日志内容，所显示的内容虽然不会影响程序的执行，但是未来可能导致问题的发生。
- ❑ ERROR等级使用`logging.error()`显示程序日志内容，通常显示程序在某些状态将引发错误的缘由。
- ❑ CRITICAL等级使用`logging.critical()`显示程序日志内容，这是最重要的等级，通常是显示将让整个系统当掉或中断的错误。

程序设计时，可以使用下列函数设定显示信息的等级：

```
logging.basicConfig(level=logging.DEBUG) # 假设是设定DEBUG等级
```

当设定`logging`为某一等级时，未来只有此等级或更高等级的`logging`会被显示。

程序实例`ch15_22.py`： 显示所有等级的`logging`信息。

```
1 # ch15_22.py
2 import logging
3
4 logging.basicConfig(level=logging.DEBUG) # 等级是DEBUG
5 logging.debug('logging message, DEBUG')
6 logging.info('logging message, INFO')
7 logging.warning('logging message, WARNING')
8 logging.error('logging message, ERROR')
9 logging.critical('logging message, CRITICAL')
```

执行结果

```
===== RESTART: D:/Python/ch15/ch15_22.py =====
DEBUG:root:logging message, DEBUG
INFO:root:logging message, INFO
WARNING:root:logging message, WARNING
ERROR:root:logging message, ERROR
CRITICAL:root:logging message, CRITICAL
>>>
```

上述每一个输出前方有DEBUG:root:(其他依次类推)前导信息, 这是该logging输出模式默认的输出信息注明输出logging模式。

程序实例ch15_23.py: 显示WARNING等级或更高等级的输出。

```
1 # ch15_23.py
2 import logging
3
4 logging.basicConfig(level=logging.WARNING) # 等级是WARNING
5 logging.debug('logging message, DEBUG')
6 logging.info('logging message, INFO')
7 logging.warning('logging message, WARNING')
8 logging.error('logging message, ERROR')
9 logging.critical('logging message, CRITICAL')
```

执行结果

```
===== RESTART: D:/Python/ch15/ch15_23.py =====
WARNING:root:logging message, WARNING
ERROR:root:logging message, ERROR
CRITICAL:root:logging message, CRITICAL
>>>
```

当我们设定logging的输出等级是WARNING时, 较低等级的logging输出就被隐藏了。当了解了上述logging输出等级的特性后, 笔者通常在设计大型程序时, 程序设计初期阶段会将logging等级设为DEBUG, 如果确定程序大致没问题, 就将logging等级设为WARNING, 最后再设为CRITICAL。这样就可以不用再像过去一样, 在程序设计初期使用print()记录关键变量的变化, 当程序确定完成后, 还需要一个一个检查print()然后将它删除。

15-7-3 格式化logging信息输出format

从ch15_22.py和ch15_23.py可以看到输出信息前方有前导输出信息, 我们可以使用在logging.basicConfig()方法内增加format格式化输出信息为空字符串''的方式, 取消显示前导输出信息。

```
logging.basicConfig(level=logging.DEBUG, format = '')
```

程序实例ch15_24.py: 重新设计ch15_22.py, 取消显示logging的前导输出信息。


```

1 # ch15_24.py
2 import logging
3
4 logging.basicConfig(level=logging.DEBUG, format='')
5 logging.debug('logging message, DEBUG')
6 logging.info('logging message, INFO')
7 logging.warning('logging message, WARNING')
8 logging.error('logging message, ERROR')
9 logging.critical('logging message, CRITICAL')

```

执行结果

```

===== RESTART: D:/Python/ch15/ch15_24.py
logging message, DEBUG
logging message, INFO
logging message, WARNING
logging message, ERROR
logging message, CRITICAL
>>>

```

从上述执行结果很明显看到，模式前导的输出信息没有了。

15-7-4 时间信息asctime

我们可以在format内配合asctime列出系统时间，这样可以列出每一重要阶段关键变量发生的时间。

程序实例ch15_25.py： 列出每一个logging输出时的时间。

```

1 # ch15_25.py
2 import logging
3
4 logging.basicConfig(level=logging.DEBUG, format='%asctime)s')
5 logging.debug('logging message, DEBUG')
6 logging.info('logging message, INFO')
7 logging.warning('logging message, WARNING')
8 logging.error('logging message, ERROR')
9 logging.critical('logging message, CRITICAL')

```

执行结果

```

===== RESTART: D:\Python\ch15\ch15_25.py
2017-10-07 00:46:15,030
2017-10-07 00:46:15,030
2017-10-07 00:46:15,046
2017-10-07 00:46:15,046
2017-10-07 00:46:15,046
>>>

```

我们的确获得了每一个logging的输出时间，但是经过format处理后原先logging.xxx()内的输出信息却没有了，这是因为我们在format内只有留时间字符串信息。

15-7-5 format内的message

如果想要输出原先logging.xxx()的输出信息，必须在format内增加message格式。

程序实例ch15_26.py： 增加logging.xxx()的输出信息。

```
1 # ch15_26.py
2 import logging
3
4 logging.basicConfig(level=logging.DEBUG, format='%(asctime)s : %(message)s')
5 logging.debug('logging message, DEBUG')
6 logging.info('logging message, INFO')
7 logging.warning('logging message, WARNING')
8 logging.error('logging message, ERROR')
9 logging.critical('logging message, CRITICAL')
```

执行结果

```
===== RESTART: D:/Python/ch15/ch15_26.py =====
2017-10-07 00:55:47.378 : logging message, DEBUG
2017-10-07 00:55:47.378 : logging message, INFO
2017-10-07 00:55:47.394 : logging message, WARNING
2017-10-07 00:55:47.394 : logging message, ERROR
2017-10-07 00:55:47.394 : logging message, CRITICAL
>>>
```

15-7-6 列出levelname

levelname属性是记载目前logging的显示层级是哪一个等级。

程序实例ch15_27.py： 列出目前level所设定的等级。

```
1 # ch15_27.py
2 import logging
3
4 logging.basicConfig(level=logging.DEBUG,
5                     format='%(asctime)s - %(levelname)s : %(message)s')
6 logging.debug('logging message.')
7 logging.info('logging message.')
8 logging.warning('logging message')
9 logging.error('logging message')
10 logging.critical('logging message')
```

执行结果

```
===== RESTART: D:/Python/ch15/ch15_27.py =====
2017-10-07 01:07:23.543 - DEBUG : logging message.
2017-10-07 01:07:23.543 - INFO : logging message.
2017-10-07 01:07:23.558 - WARNING : logging message
2017-10-07 01:07:23.558 - ERROR : logging message
2017-10-07 01:07:23.558 - CRITICAL : logging message
>>>
```

15-7-7 使用logging列出变量变化的应用

这一节开始笔者将正式使用logging追踪变量的变化，下列是简单追踪索引值变化的程序。

程序实例ch15_28.py： 追踪索引值变化的实例。

```
1 # ch15_28.py
2 import logging
3
4 logging.basicConfig(level=logging.DEBUG,
5                     format='%(asctime)s - %(levelname)s : %(message)s')
6 logging.debug('程序开始')
7 for i in range(5):
8     logging.debug('目前索引 %s ' % i)
9 logging.debug('程序结束')
```

执行结果

```
===== RESTART: D:\Python\ch15\ch15_28.py =====
2017-12-16 12:46:29,765 - DEBUG : 程序开始
2017-12-16 12:46:29,765 - DEBUG : 目前索引 0
2017-12-16 12:46:29,765 - DEBUG : 目前索引 1
2017-12-16 12:46:29,765 - DEBUG : 目前索引 2
2017-12-16 12:46:29,765 - DEBUG : 目前索引 3
2017-12-16 12:46:29,780 - DEBUG : 目前索引 4
2017-12-16 12:46:29,780 - DEBUG : 程序结束
>>>
```

上述程序记录了整个索引值的变化过程，读者需留意第8行的输出，它的输出结果是在%(message)s定义。

15-7-8 正式追踪factorial数值的应用

在程序ch11_26.py笔者曾经使用递归函数计算阶乘factorial，接下来笔者想用一般循环方式追踪阶乘计算的过程。

程序实例ch15_29.py： 使用logging追踪factorial阶乘计算的过程。

```
1 # ch15_29.py
2 import logging
3
4 logging.basicConfig(level=logging.DEBUG,
5                     format='%(asctime)s - %(levelname)s : %(message)s')
6 logging.debug('程序开始')
7
8 def factorial(n):
9     logging.debug('factorial %s 计算开始' % n)
10    ans = 1
11    for i in range(n + 1):
12        ans *= i
13        logging.debug('i = ' + str(i) + ', ans = ' + str(ans))
14    logging.debug('factorial %s 计算结束' % n)
15    return ans
16
17 num = 5
18 print("factorial(%d) = %d" % (num, factorial(num)))
19 logging.debug('程序结束')
```

执行结果

```
===== RESTART: D:\Python\ch15\ch15_29.py =====
2017-12-16 12:49:30,468 - DEBUG : 程序开始
2017-12-16 12:49:30,468 - DEBUG : factorial 5 计算开始
2017-12-16 12:49:30,468 - DEBUG : i = 0, ans = 0
2017-12-16 12:49:30,468 - DEBUG : i = 1, ans = 0
2017-12-16 12:49:30,484 - DEBUG : i = 2, ans = 0
2017-12-16 12:49:30,484 - DEBUG : i = 3, ans = 0
2017-12-16 12:49:30,484 - DEBUG : i = 4, ans = 0
2017-12-16 12:49:30,484 - DEBUG : i = 5, ans = 0
2017-12-16 12:49:30,484 - DEBUG : factorial 5 计算结束
factorial(5) = 0
2017-12-16 12:49:30,500 - DEBUG : 程序结束
>>>
```

在上述使用logging的DEBUG过程可以发现阶乘数从0开始，造成所有阶段的执行结果皆是0，程序的错误，下列程序第11行，笔者更改此项设定为从1开始。

程序实例ch15_30.py： 修订ch15_29.py的错误，让阶乘从1开始。

```
1 # ch15_30.py
2 import logging
3
4 logging.basicConfig(level=logging.DEBUG,
5                     format='%(asctime)s - %(levelname)s : %(message)s')
6 logging.debug('程序开始')
7
8 def factorial(n):
9     logging.debug('factorial %s 计算开始' % n)
10    ans = 1
11    for i in range(1, n + 1):
12        ans *= i
13        logging.debug('i = ' + str(i) + ', ans = ' + str(ans))
14    logging.debug('factorial %s 计算结束' % n)
15    return ans
16
17 num = 5
18 print("factorial(%d) = %d" % (num, factorial(num)))
19 logging.debug('程序结束')
```

执行结果

```
===== RESTART: D:\Python\ch15\ch15_30.py =====
2017-12-16 12:52:39,994 - DEBUG : 程序开始
2017-12-16 12:52:39,994 - DEBUG : factorial 5 计算开始
2017-12-16 12:52:39,994 - DEBUG : i = 1, ans = 1
2017-12-16 12:52:40,009 - DEBUG : i = 2, ans = 2
2017-12-16 12:52:40,009 - DEBUG : i = 3, ans = 6
2017-12-16 12:52:40,009 - DEBUG : i = 4, ans = 24
2017-12-16 12:52:40,009 - DEBUG : i = 5, ans = 120
2017-12-16 12:52:40,009 - DEBUG : factorial 5 计算结束
factorial(5) = 120
2017-12-16 12:52:40,025 - DEBUG : 程序结束
>>>
```

15-7-9 将程序日志logging输出到文件

程序很长时，若将logging输出在屏幕，其实不太方便逐一核对关键变量值的变化，此时我们可以考虑将logging输出到文件，方法是在logging.basicConfig()增加filename=“文件名”，这样就可以将logging输出到指定的文件内。

程序实例ch15_31.py： 将程序实例的logging输出到out15_31.txt。

```
4 logging.basicConfig(filename='out15_31.txt', level=logging.DEBUG,  
5 format='%(asctime)s - %(levelname)s : %(message)s')
```

执行结果

```
===== RESTART: D:/Python/ch15/ch15_31.py =====  
factorial(5) = 120  
>>>
```

这时在当前工作文件夹可以看到out15_31.txt，打开后可以得到下列结果。



```
2017-12-16 13:00:23,844 - DEBUG : i = 1, ans = 1  
2017-12-16 13:00:23,844 - DEBUG : i = 2, ans = 2  
2017-12-16 13:00:23,844 - DEBUG : i = 3, ans = 6  
2017-12-16 13:00:23,844 - DEBUG : i = 4, ans = 24  
2017-12-16 13:00:23,844 - DEBUG : i = 5, ans = 120  
2017-12-16 13:00:58,771 - DEBUG : 程序开始  
2017-12-16 13:00:58,771 - DEBUG : factorial 5 计算开始  
2017-12-16 13:00:58,771 - DEBUG : i = 1, ans = 1  
2017-12-16 13:00:58,771 - DEBUG : i = 2, ans = 2  
2017-12-16 13:00:58,771 - DEBUG : i = 3, ans = 6  
2017-12-16 13:00:58,771 - DEBUG : i = 4, ans = 24  
2017-12-16 13:00:58,771 - DEBUG : i = 5, ans = 120  
2017-12-16 13:00:58,771 - DEBUG : factorial 5 计算结束  
2017-12-16 13:00:58,771 - DEBUG : 程序结束
```

15-7-10 隐藏程序日志logging的DEBUG等级使用CRITICAL

先前笔者有说明logging有许多等级，只要设定高等级，Python就会忽略低等级的输出，所以如果我们程序设计完成，也确定没有错误，其实可以将logging等级设为最高等级，所有较低等级的输出将被隐藏。

程序实例ch15_32.py： 重新设计ch15_30.py，将程序内DEBUG等级的logging隐藏。

```
4 logging.basicConfig(level=logging.CRITICAL,  
5 format='%(asctime)s - %(levelname)s : %(message)s')
```

执行结果

```
===== RESTART: D:/Python/ch15/ch15_32.py =====  
factorial(5) = 120  
>>>
```

15-7-11 停用程序日志logging

可以使用下列方法停用日志logging。

```
logging.disable(level) # level是停用logging的等级
```

上述可以停用该程序代码后指定等级以下的所有等级，如果想停用全部参数可以使用logging.CRITICAL等级，这个方法一般是放在import下方，这样就可以停用所有的logging。

程序实例ch15_33.py： 重新设计ch15_30.py，这个程序只是在原先第3行空白行加上下列程序代码。

```
3 logging.disable(logging.CRITICAL) # 停用所有logging
```

执行结果 与ch15_32.py相同。

15-8 程序除错的典故

通常我们又将程序除错称Debug，De是除去的意思，bug是指小虫，其实这是有典故的。1944年IBM和哈佛大学联合开发了Mark I计算机，此计算机重5吨，有8英尺高，51英尺长，内部线路加总长是500英里，没有中断使用了15年，下列是此计算机图片。

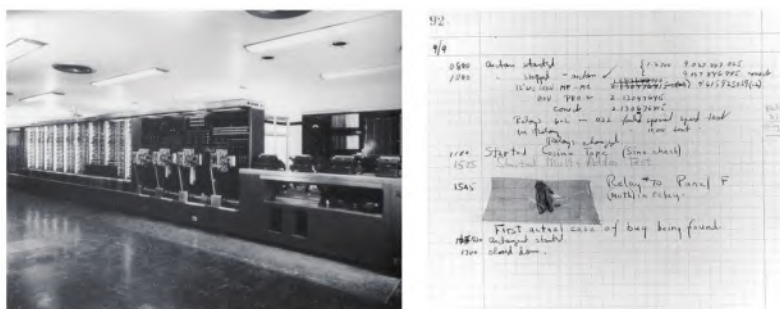
在当时有一位女性程序设计师Grace Hopper，发现了第一个计算机虫(bug)，一只死的蛾(moth)的双翅卡在继电器(relay)，促使数据读取失败，下列是当时Grace Hopper记录此事件的数据。

当时Grace Hopper写下了下列两句话。

Relay #70 Panel F (moth) in relay.

First actual case of bug being found.

大意是编号70的继电器出问题（因为蛾），这是真实计算机上所发现的第一只虫。自此，计算机界认定用debug描述“找出及删除程序错误”应归功于Grace Hopper。



本图片转载自<http://www.computersciencelab.com>

习题

1. 请重新设计ch15_11.py, 但是将2个except错误处理程序改为列出系统内置的异常处理信息。
2. 请将程序实例ch15_6.py改为由屏幕输入文字, 然后将输入的文字存入in15_6.txt, 再予以分析。
3. 请将程序实例ch15_7.py第13行的3个文件改为5个文件, 同时这5个文件的文件名是由屏幕输入。
4. 请重新设计ch15_11.py, 但是将2个except错误处理程序改为列出系统内置的异常处理信息。
5. 请重新设计ch15_11.py, 将程序改为由屏幕输入数字, 如果输入‘q’或‘Q’代表程序结束。
6. 请重新设计程序实例ch15_15.py, 将程序改为读取文件, 请用列表建立5个文件测试, 如果文件长度超过100字或小于50个字则出现

异常。最后异常发生时，请将异常结果存入errdata.txt内。

7. 请重新设计ch15_20.py，增加__init__()也具有确定开户时金额需在100元以上的断言assert。
8. 请参考程序实例ch15_30.py，将factorial(n)函数改为sumrange(n)，这个功能可以累计1+2+……+ n的总和。
9. 请将习题8的logging存入mydebug.txt内。
10. 请为ch15_30.py增加assert功能，此程序必须确保第17行的n是正值，请用列表[10, 8, -5]做测试。

第16章 正则表达式(Regular Expression)

正则表达式(Regular Expression)主要功能是执行模式的比对与搜寻，甚至Word文件也可以使用正则表达式处理搜寻 (search)与取代(replace)功能，本章首先会介绍如果没用正则表达式，如何处理搜寻文字功能，再介绍使用正则表达式处理这类问题，读者会发现整个工作变得更简洁容易。

16-1 使用Python硬功夫搜寻文字

如果现在打开手机的联络信息可以看到，台湾手机号码的格式如下：

0952-282-020 # 可以表示为xxxx-xxx-xxx，每个x代表一个0-9数字

从上述可以发现手机号码格式是由4个数字，1个连字符号，3个数字，1个连字符号，3个数字所组成。

程序实例ch16_1.py： 用传统知识设计一个程序，然后判断字符串是否含有台湾的手机号码格式。

```
1 # ch16_1.py
2 def taiwanPhoneNum(string):
3     """检查是否有含手机联络信息的台湾手机号码格式"""
4     if len(string) != 12: # 如果长度不是12
5         return False # 传回非手机号码格式
6
7     for i in range(0, 4): # 如果前4个字出现非数字字符
8         if string[i].isdecimal() == False:
9             return False # 传回非手机号码格式
10
11    if string[4] != '-': # 如果不是 '-' 字符
12        return False # 传回非手机号码格式
13
14    for i in range(5, 8): # 如果中间3个字出现非数字字符
15        if string[i].isdecimal() == False:
16            return False # 传回非手机号码格式
17
18    if string[8] != '-': # 如果不是 '-' 字符
19        return False # 传回非手机号码格式
20
21    for i in range(9, 12): # 如果最后3个字出现非数字字符
22        if string[i].isdecimal() == False:
23            return False # 传回非手机号码格式
24    return True # 通过以上测试
25
26 print("I love Ming-Chi: 是台湾手机号码", taiwanPhoneNum('I love Ming-Chi'))
27 print("0932-999-199: 是台湾手机号码", taiwanPhoneNum('0932-999-199'))
```


执行结果

```
=====RESTART: D:\Python\ch16\ch16_1.py=====
I love Ming-Chi: 是台湾手机号码 False
0932-999-199: 是台湾手机号码 True
>>>
```

上述程序第4和5行是判断字符串长度是否12，如果不是则表示这不是手机号码格式。程序第7至9行是判断字符串前4码是不是数字，如果不是则表示这不是手机号码格式。程序第11至12行是判断这个字符是不是‘-’，如果不是则表示这不是手机号码格式。程序第14至16行是判断字符串索引[5][6][7]码是不是数字，如果不是则表示这不是手机号码格式。程序第18至19行是判断这个字符是不是‘-’，如果不是则表示这不是手机号码格式。程序第21至23行是判断字符串索引[9][10][11]码是不是数字，如果不是则表示这不是手机号码格式。如果通过了以上所有测试，表示这是手机号码格式，程序第24行传回True。

在真实的环境应用中，我们可能需面临一段文字，这段文字内穿插一些数字，然后我们必须将手机号码从这段文字抽离出来。

程序实例ch16_2.py: 将电话号码从一段文字抽离出来。


```

1 # ch16_2.py
2 def taiwanPhoneNum(string):
3     """检查是否有含手机联络信息的台湾手机号码格式"""
4     if len(string) != 12: # 如果长度不是12
5         return False # 传回非手机号码格式
6
7     for i in range(0, 4): # 如果前4个字出现非数字字符
8         if string[i].isdecimal() == False:
9             return False # 传回非手机号码格式
10
11     if string[4] != '-': # 如果不是 '-' 字符
12         return False # 传回非手机号码格式
13
14     for i in range(5, 8): # 如果中间3个字出现非数字字符
15         if string[i].isdecimal() == False:
16             return False # 传回非手机号码格式
17
18     if string[8] != '-': # 如果不是 '-' 字符
19         return False # 传回非手机号码格式
20
21     for i in range(9, 12): # 如果最后3个字出现非数字字符
22         if string[i].isdecimal() == False:
23             return False # 传回非手机号码格式
24     return True # 通过以上测试
25
26 def parseString(string):
27     """解析字符串是否含有电话号码"""
28     notFoundSignal = True # 注：没有找到电话号码为True
29     for i in range(len(string)): # 用循环逐步抽取12个字符做测试
30         msg = string[i:i+12]
31         if taiwanPhoneNum(msg):
32             print("电话号码是: %s" % msg)
33             notFoundSignal = False
34     if notFoundSignal: # 如果没有找到电话号码则打印
35         print("%s 字符串不含电话号码" % string)
36
37 msg1 = 'Please call my secretary using 0930-919-919 or 0952-001-001'
38 msg2 = '请明天17:30和我一起参加明志科大教师节晚餐'
39 msg3 = '请明天17:30和我一起参加明志科大教师节晚餐, 可用0933-080-080联络我'
40 parseString(msg1)
41 parseString(msg2)
42 parseString(msg3)

```

执行结果

```

===== RESTART: D:\Python\ch16\ch16_2.py =====
电话号码是: 0930-919-919
电话号码是: 0952-001-001
请明天17:30和我一起参加明志科大教师节晚餐 字符串不含电话号码
电话号码是: 0933-080-080
>>>

```

从上述执行结果可以得知我们成功地从一个字符串分析，然后将电话号码分析出来了。分析方式的重点是程序第26行到35行的parseString函数，这个函数重点是第29至33行，这个循环会逐步抽取字符串的12个字符做比对，将比对字符串放在msg字符串变量内，下列是各循环次序的msg字符串变量内容。

```

msg = 'Please call'           # 第1次 [0:12]
msg = 'lease call m'         # 第2次 [1:13]
msg = 'ease call my'         # 第3次 [2:14]
...
msg = '0930-939-939'          # 第31次 [30:42]
...
msg = '0952-001-001'          # 第48次 [47:59]

```

程序第28行将没有找到电话号码notFoundSignal设为True，如果有找到电话号码程序33行将notFoundSignal标示为False，当parseString()函数执行完，notFoundSignal仍是True，表示没找到电话号码，所以第35行打印字符串不含电话号码。

上述使用所学的Python硬功夫虽然解决了我们的问题，但是若是将电话号码改成中国大陆手机号(XXX-XXXX-XXXX)、美国手机号(XXX-XXX-XXXX)或是一般公司行号的电话，整个号码格式不一样，要重新设计可能需要一些时间。不过不用担心，接下来笔者将讲解的Python的正则表达式可以轻松解决上述困扰。

16-2 正则表达式的基础

Python有关正则表达式的方法是在re模块内，所以使用正则表达式需要导入re模块。

```
import re # 导入re模块
```

16-2-1 建立搜寻字符串模式

在前一节我们使用isdecimal()方法判断字符是否是0—9的数字。

正则表达式是一种文本模式的表达方法，在这个方法中使用\d表示0—9的数字字符，采用这个观念我们可以将前一节的手机号码XXXX-XXX-XXX改用下列正则表达方式表示：

```
'\d\d\d\d-\d\d\d\d'
```

由逸出字符的观念可知，将上述表达式当字符串放入函数内需增加‘\’，所以整个正则表达式的使用方式如下：

```
'\\d\\d\\d\\d-\\d\\d\\d\\d'
```

在3-4-9小节笔者有介绍字符串前加r可以防止字符串内的逸出字符被转译，所以又可以将上述正则表达式简化为下列格式：

```
r'\d\d\d\d-\d\d\d-\d\d\d'
```

16-2-2 使用re.compile()建立Regex对象

Regex 是Regular expression的简称，在re模块内有compile()方法，可以将16-2-1节的欲搜寻字符串的正则表达式当作字符串参数放在此方法内，然后会传回一个Regex对象。如下所示：

```
phoneRule = re.compile(r'\d\d\d\d-\d\d\d-\d\d\d') # 建立
phoneRule对象
```

16-2-3 搜寻对象

在Regex对象内有search()方法，可以由Regex对象启用，然后将欲搜寻的字符串放在这个方法内，沿用上述观念程序片段如下：

```
phoneNum = phoneRule.search(msg) # msg是欲搜寻的字符串
```

如果找不到比对相符的字符串会传回None，如果找到比对相符的字符串会将结果传回所设定的phoneNum变量对象，这个对象在Python中称之为MatchObject对象，将在16-6节完整解说。现在笔者将介绍实用性较高的部分，处理此对象主要是将搜寻结果传回，我们可以用group()方法将结果传回，不过search()将只传回第一个比对相符的字符串。

程序实例ch16_3.py： 使用正则表达式重新设计ch16_2.py。

```
1 # ch16_3.py
2 import re
3
4 msg1 = 'Please call my secretary using 0930-919-919 or 0952-001-001'
5 msg2 = '请明天17:30和我一起参加明志科大教师节晚餐'
6 msg3 = '请明天17:30和我一起参加明志科大教师节晚餐，可用0933-080-080联络我'
7
8 def parseString(string):
9     """解析字符串是否含有电话号码"""
10    phoneRule = re.compile(r'\d\d\d\d-\d\d\d-\d\d\d')
11    phoneNum = phoneRule.search(string)
12    if phoneNum != None: # 检查phoneNum内容
13        print("电话号码是: %s" % phoneNum.group())
14    else:
15        print("%s 字符串不含电话号码" % string)
16
17 parseString(msg1)
18 parseString(msg2)
19 parseString(msg3)
```

执行结果

```
===== RESTART: D:\Python\ch16\ch16_3.py =====
电话号码是: 0930-919-919
请明天17:30和我一起参加明志科大教师节晚餐 字符串不含电话号码
电话号码是: 0933-080-080
>>>
```

在程序实例ch16_2.py我们使用了约21行做字符串解析，当我们使用Python的正则表达式时，只用第10和11行共2行就解析了字符串是否含手机号码了，整个程序变得简单许多。不过上述msg1字符串内含2组手机号码，使用search()只传回第一个发现的号码，下一节将改良此方法。

16-2-4 findall()

从方法的名字就可以知道，这个方法可以传回所有找到的手机号码。这个方法会将搜寻到的手机号码用列表方式传回，这样就不会有只显示第一个搜寻到的手机号码的缺点，如果没有比对相符的号码就传回[]空列表。要使用这个方法的关键指令如下：

```
phoneRule = re.compile(r'\d\d\d\d-\d\d\d-\d\d\d') # 建立
phoneRule对象
phoneNum = phoneRule.findall(string) # string
是欲搜寻的字符串
```

findall()函数由phoneRule对象启用，最后会将搜寻结果的列表传给phoneNum，只要打印phoneNum就可以得到执行结果。

程序实例ch16_4.py： 使用findall()搜寻字符串，第10行定义正则表达式，打印结果。

```
1 # ch16_4.py
2 import re
3
4 msg1 = 'Please call my secretary using 0930-919-919 or 0952-001-001'
5 msg2 = '请明天17:30和我一起参加明志科大教师节晚餐'
6 msg3 = '请明天17:30和我一起参加明志科大教师节晚餐，可用0933-080-080联络我'
7
8 def parseString(string):
9     """解析字符串是否含有电话号码"""
10    phoneRule = re.compile(r'\d\d\d\d-\d\d\d-\d\d\d')
11    phoneNum = phoneRule.findall(string) # 用列表传回搜寻结果
12    print("电话号码是: %s" % phoneNum) # 列表方式显示电话号码
13
14 parseString(msg1)
15 parseString(msg2)
16 parseString(msg3)
```

执行结果

```
===== RESTART: D:\Python\ch16\ch16_4.py =====
电话号码是: ['0930-919-919', '0952-001-001']
电话号码是: []
电话号码是: ['0933-080-080']
>>>
```

16-2-5 再看re模块

其实Python语言的re模块对于search()和findall()有提供更强的功能，可以省略使用re.compile()直接将比对模式放在各自的参数内，此时语法格式如下：

```
re.search(pattern, string, flags)
re.findall(pattern, string, flags)
```

上述pattern是欲搜寻的正则表达方式，string是所搜寻的字符串，flags可以省略，未来会介绍几个flags常用相关参数的应用。

程序实例ch16_5.py： 使用re.search()重新设计ch16_3.py，由于省略了re.compile()，所以读者需留意第11行内容写法。

```
1 # ch16_5.py
2 import re
3
4 msg1 = 'Please call my secretary using 0930-919-919 or 0952-001-001'
5 msg2 = '请明天17:30和我一起参加明志科大教师节晚餐'
6 msg3 = '请明天17:30和我一起参加明志科大教师节晚餐，可用0933-080-080联络我'
7
8 def parseString(string):
9     """解析字符串是否含有电话号码"""
10    pattern = r'\d\d\d\d-\d\d\d-\d\d\d\d'
11    phoneNum = re.search(pattern, string)
12    if phoneNum != None:      # 如果phoneNum不是None表示取得号码
13        print("电话号码是: %s" % phoneNum.group())
14    else:
15        print("%s 字符串不含电话号码" % string)
16
17 parseString(msg1)
18 parseString(msg2)
19 parseString(msg3)
```

执行结果 与ch16_3.py相同。

程序实例ch16_6.py： 使用re.findall()重新设计ch16_4.py，由于省略了re.compile()，所以读者需留意第11行内容写法。

```

1 # ch16_6.py
2 import re
3
4 msg1 = 'Please call my secretary using 0930-919-919 or 0952-001-001'
5 msg2 = '请明天17:30和我一起参加明志科大教师节晚餐'
6 msg3 = '请明天17:30和我一起参加明志科大教师节晚餐，可用0933-080-080联络我'
7
8 def parseString(string):
9     """解析字符串是否含有电话号码"""
10    pattern = r'\d\d\d\d-\d\d\d-\d\d\d'
11    phoneNum = re.findall(pattern, string) # 用列表传回搜寻结果
12    print("电话号码是: %s" % phoneNum)    # 列表方式显示电话号码
13
14 parseString(msg1)
15 parseString(msg2)
16 parseString(msg3)

```

执行结果 与ch16_4. py相同。

16-2-6 再看正则表达式

下列是我们目前的正则表达式所搜寻的字符串模式：

```
r'\d\d\d\d-\d\d\d-\d\d\d'
```

其中可以看到\d重复出现，对于重复出现的字符串可以用大括号内部加上重复次数方式表达，所以上述可以用下列方式表达。

```
r'\d{4}-\d{3}-\d{3}'
```

程序实例ch16_7.py： 使用本节观念重新设计ch16_6.py，下列只列出不一样程序内容。

```
10 pattern = r'\d{4}-\d{3}-\d{3}'
```

执行结果 与ch16_4. py相同。

16-3 更多搜寻比对模式

先前我们所用的实例是手机号码，想想看如果我们改用市区电话号码的比对，台北市的电话号码如下：

```
02-28350000 # 可用xx-xxxxxxx表达
```

下列将以上述电话号码模式说明。

16-3-1 使用小括号分组

依照16-2节的观念，可以用下列正则表示法表达上述市区电话号码。

```
r'\d\d-\d\d\d\d\d\d\d'
```

所谓括号分组是以连字符“-”区别，然后用小括号隔开群组，可以用下列方式重新规划上述表达式。也可简化为：

```
r'(\d\d)-(\d\d\d\d\d\d\d')
```

```
r'(\d{2})-(\d{8})'
```

当使用`re.search()`执行比对时，未来可以使用`group()`传回比对符合的不同分组，例如：`group()`或`group(0)`传回第一个比对相符的文字与`ch16_3.py`观念相同。如果`group(1)`则传回括号的第二组文字，`group(2)`则传回括号的第三组文字。

程序实例ch16_8.py： 使用小括号分组的观念，将分组内容输出。

```
1 # ch16_8.py
2 import re
3
4 msg = 'Please call my secretary using 02-26669999'
5 pattern = r'(\d{2})-(\d{8})'
6 phoneNum = re.search(pattern, msg) # 传回搜寻结果
7
8 print("完整号码是：%s" % phoneNum.group()) # 显示完整号码
9 print("完整号码是：%s" % phoneNum.group(0)) # 显示完整号码
10 print("区域号码是：%s" % phoneNum.group(1)) # 显示区域号码
11 print("电话号码是：%s" % phoneNum.group(2)) # 显示电话号码
```

执行结果

```
===== RESTART: D:\Python\ch16\ch16_8.py
完整号码是：02-26669999
完整号码是：02-26669999
区域号码是：02
电话号码是：26669999
>>>
```

如果所搜寻比对的正则表达式字符串有用小括号分组，若是使用`findall()`方法处理，会传回元组(tuple)的列表(list)，元组内的每个元素就是搜寻的分组内容。

程序实例ch16_9.py： 使用`findall()`重新设计`ch16_8.py`，这个实例会多增加一组电话号码。

```

1 # ch16_9.py
2 import re
3
4 msg = 'Please call my secretary using 02-26669999 or 02-11112222'
5 pattern = r'(\d{2})-(\d{8})'
6 phoneNum = re.findall(pattern, msg)      # 传回搜寻结果
7 print(phoneNum)

```

执行结果

```

===== RESTART: D:/Python/ch16/ch16_9.py =====
[('02', '26669999'), ('02', '11112222')]
>>>

```

16-3-2 groups()

注意这是 groups ()，有在 group 后面加上 s，当我们使用 re.search() 搜寻字符串时，可以使用这个方法取得分组的内容。这时还可以使用 2-9 节的多重指定的观念，若以 ch16_8.py 为例，在第 7 行我们可以使用下列多重指定获得区域号码和当地电话号码。

```
areaNum, localNum = phoneNum.groups( ) # 多重指定
```

程序实例 ch16_10.py：重新设计 ch16_8.py，分别列出区域号码与电话号码。

```

1 # ch16_10.py
2 import re
3
4 msg = 'Please call my secretary using 02-26669999'
5 pattern = r'(\d{2})-(\d{8})'
6 phoneNum = re.search(pattern, msg)      # 传回搜寻结果
7 areaNum, localNum = phoneNum.groups()   # 留意是groups()
8 print("区域号码是: %s" % areaNum)       # 显示区域号码
9 print("电话号码是: %s" % localNum)       # 显示电话号码

```

执行结果

```

===== RESTART: D:\Python\ch16\ch16_10.py =====
区域号码是: 02
电话号码是: 26669999
>>>

```

16-3-3 区域号码是在小括号内

在一般电话号码的使用中，常看到区域号码是用小括号包夹，如下所示：

(02)-26669999

在处理小括号时，方式是\ (和\)，可参考下列实例。

程序实例ch16_11.py： 重新设计ch16_10.py，第4行的区域号码是(02)，读者需留意第4行和第5行的设计。

```
1 # ch16_11.py
2 import re
3
4 msg = 'Please call my secretary using (02)-26669999'
5 pattern = r'\(\d{2}\)-(\d{8})'
6 phoneNum = re.search(pattern, msg) # 传回搜寻结果
7 areaNum, localNum = phoneNum.groups() # 留意是groups()
8 print("区域号码是: %s" % areaNum) # 显示区域号码
9 print("电话号码是: %s" % localNum) # 显示电话号码
```

执行结果

```
===== RESTART: D:\Python\ch16\ch16_11.py =====
区域号码是: (02)
电话号码是: 26669999
>>>
```

16-3-4 使用管道|

| (pipe) 在正规表示法称管道，使用管道我们可以同时搜寻比对多个字符串，例如，想要搜寻Mary和Tom字符串，可以使用下列表示。

```
pattern = 'Mary|Tom' # 注意单引号'或|旁不可留空白
```

程序实例ch16_12.py： 管道搜寻多个字符串的实例。

```
1 # ch16_12.py
2 import re
3
4 msg = 'John and Tom will attend my party tonight. John is my best friend.'
5 pattern = 'John|Tom' # 搜寻John和Tom
6 txt = re.findall(pattern, msg) # 传回搜寻结果
7 print(txt)
8 pattern = 'Mary|Tom' # 搜寻Mary和Tom
9 txt = re.findall(pattern, msg) # 传回搜寻结果
10 print(txt)
```

执行结果

```
===== RESTART: D:\Python\ch16\ch16_12.py =====
['John', 'Tom', 'John']
['Tom']
>>>
```

16-3-5 多个分组的管道搜寻

假设有一个字符串内容如下：

```
Johnson, Johnnason and Johnnathan will attend my party tonight.
```

由上述可知，如果想要搜寻字符串比对John后面可以是son、nason、nathan任一个字符串的组合，可以使用下列正则表达式格式：

```
pattern = John (son|nason|nathan)
```

程序实例ch16_13.py： 搜寻Johnson、Johnnason或Johnnathan任一字符串，然后列出结果，这个程序将列出第一个搜寻比对到的字符串。

```
1 # ch16_13.py
2 import re
3
4 msg = 'Johnson, Johnnason and Johnnathan will attend my party tonight.'
5 pattern = 'John(son|nason|nathan)'
6 txt = re.search(pattern,msg) # 传回搜寻结果
7 print(txt.group())          # 打印第一个搜寻结果
8 print(txt.group(1))         # 打印第一个分组
```

执行结果

```
===== RESTART: D:\Python\ch16\ch16_13.py =====
Johnson
son
>>>
```

同样的正则表达式若是使用findall()方法处理，将只传回各分组搜寻到的字符串，如果要列出完整的内容，可以用循环同时为每个分组字符串加上前导字符串John。

程序实例ch16_14.py： 使用findall()重新设计ch16_13.py。

```
1 # ch16_14.py
2 import re
3
4 msg = 'Johnson, Johnnason and Johnnathan will attend my party tonight.'
5 pattern = 'John(son|nason|nathan)'
6 txts = re.findall(pattern,msg) # 传回搜寻结果
7 print(txts)
8 for txt in txts:               # 将搜寻到内容加上John
9     print('John'+txt)
```

执行结果

```
===== RESTART: D:/Python/ch16/ch16_14.py =====
['son', 'nason', 'nathan']
Johnson
Johnnason
Johnnathan
>>>
```

16-3-6 使用？号做搜寻

在正则表达式中若某些括号内的字符串或正则表达式可有可无，执行搜寻时皆算成功，例如，na字符串可有可无，表达方式是(na)?。

程序实例ch16_15.py： 使用？搜寻的实例，这个程序会测试2次。

```
1 # ch16_15.py
2 import re
3 # 测试1
4 msg = 'Johnson will attend my party tonight.'
5 pattern = 'John((na)?son)'
6 txt = re.search(pattern,msg)      # 传回搜寻结果
7 print(txt.group())
8 # 测试2
9 msg = 'Johnnason will attend my party tonight.'
10 pattern = 'John((na)?son)'
11 txt = re.search(pattern,msg)      # 传回搜寻结果
12 print(txt.group())
```

执行结果

```
===== RESTART: D:/Python/ch16/ch16_15.py
Johnson
Johnnason
>>>
```

有时候如果居住在同一个城市，在留电话号码时，可能不会留区域号码，这时就可以使用本功能了。请参考下列实例第11行。

程序实例ch16_16.py： 这个程序在搜寻电话号码时，即使省略区域号码程序也可以搜寻到此号码，然后打印出来，正则表达式格式请留意第6行。

```
1 # ch16_16.py
2 import re
3
4 # 测试1
5 msg = 'Please call my secretary using 02-26669999'
6 pattern = r'(\d\d-)?(\d{8})'      # 增加?号
7 phoneNum = re.search(pattern, msg) # 传回搜寻结果
8 print("完整号码是: %s" % phoneNum.group()) # 显示完整号码
9
10 # 测试2
11 msg = 'Please call my secretary using 26669999'
12 pattern = r'(\d\d-)?(\d{8})'      # 增加?号
13 phoneNum = re.search(pattern, msg) # 传回搜寻结果
14 print("完整号码是: %s" % phoneNum.group()) # 显示完整号码
```

执行结果

```
===== RESTART: D:\Python\ch16\ch16_16.py :
完整号码是: 02-26669999
完整号码是: 26669999
>>>
```

16-3-7 使用*号做搜寻

在正则表达式中若某些字符串或正则表达式可从0到多次，执行搜寻时皆算成功，例如，na字符串可从0到多次，表达方式是(na)*。

程序实例ch16_17.py： 这个程序的重点是第5行的正则表达式，其中字符串na的出现次数可以是0到多次。

```
1 # ch16_17.py
2 import re
3 # 测试1
4 msg = 'Johnson will attend my party tonight.'
5 pattern = 'John((na)*son)' # 字符串na可以0到多次
6 txt = re.search(pattern,msg) # 传回搜寻结果
7 print(txt.group())
8 # 测试2
9 msg = 'Johnnason will attend my party tonight.'
10 pattern = 'John((na)*son)' # 字符串na可以0到多次
11 txt = re.search(pattern,msg) # 传回搜寻结果
12 print(txt.group())
13 # 测试3
14 msg = 'Johnnananason will attend my party tonight.'
15 pattern = 'John((na)*son)' # 字符串na可以0到多次
16 txt = re.search(pattern,msg) # 传回搜寻结果
17 print(txt.group())
```

执行结果

```
===== RESTART: D:/Python/ch16/ch16_17.py
Johnson
Johnnason
Johnnananason
>>>
```

16-3-8 使用+号做搜寻

在正则表达式中若是某些字符串或正则表达式可从1到多次，执行搜寻时皆算成功，例如，na字符串可从1到多次，表达方式是(na)+。

程序实例ch16_18.py： 这个程序的重点是第5行的正则表达式，其中字符串na的出现次数可以是1到多次。

```
1 # ch16_18.py
2 import re
3 # 测试1
4 msg = 'Johnson will attend my party tonight.'
5 pattern = 'John((na)+son)' # 字符串na可以1到多次
6 txt = re.search(pattern,msg) # 传回搜寻结果
7 print(txt) # 请注意是直接打印对象
8 # 测试2
9 msg = 'Johnnason will attend my party tonight.'
10 pattern = 'John((na)+son)' # 字符串na可以1到多次
11 txt = re.search(pattern,msg) # 传回搜寻结果
12 print(txt.group())
13 # 测试3
14 msg = 'Johnnananason will attend my party tonight.'
15 pattern = 'John((na)+son)' # 字符串na可以1到多次
16 txt = re.search(pattern,msg) # 传回搜寻结果
17 print(txt.group())
```

执行结果

```
===== RESTART: D:/Python/ch16/ch16_18.py =====
None
Johnnason
Johnnananason
>>>
```

16-3-9 搜寻时忽略大小写

搜寻时若是在 `search()` 或 `findall()` 内增加第三个参数 `re.I` 或 `re.IGNORECASE`，搜寻时就会忽略大小写，至于打印输出时将以原字符串的格式显示。

程序实例 `ch16_19.py`：以忽略大小写方式执行找寻相符字符串。

```
1 # ch16_19.py
2 import re
3
4 msg = 'john and TOM will attend my party tonight. JOHN is my best friend.'
5 pattern = 'John|Tom' # 搜寻 John 和 Tom
6 txt = re.findall(pattern, msg, re.I) # 传回搜寻忽略大小写的结果
7 print(txt)
8 pattern = 'Mary|tom' # 搜寻 Mary 和 tom
9 txt = re.findall(pattern, msg, re.I) # 传回搜寻忽略大小写的结果
10 print(txt)
```

执行结果

```
===== RESTART: D:/Python/ch16/ch16_19.py =====
['john', 'TOM', 'JOHN']
['TOM']
>>>
```

16-4 贪婪与非贪婪搜寻

16-4-1 搜寻时使用大括号设定比对次数

在 16-2-6 节我们有使用过大括号，当时讲解 `\d{4}` 代表重复 4 次，也就是大括号的数字是设定重复次数。可以将这个观念应用在搜寻一般字符串，例如，`(son){3}` 代表所搜寻的字符串是 ‘sonsonson’，如果有一字符串是 ‘sonson’，则搜寻结果是不符。大括号除了可以设定重复次数，也可以设定指定范围，例如，`(son){3,5}` 代表所搜寻的字符串如果

是‘sonsonson’ ‘sonsonsonson’ 或 ‘sonsonsonsonson’ 皆算是相符合的字符串。(son){3,5}正则表达式相当于下列表达式：

$$((\text{son}) (\text{son}) (\text{son})) \mid ((\text{son}) (\text{son}) (\text{son}) (\text{son})) \mid ((\text{son}) (\text{son}) (\text{son}) (\text{son}) (\text{son}))$$

程序实例ch16_20.py： 设定搜寻son字符串重复3-5次皆算搜寻成功。

```
1 # ch16_20.py
2 import re
3
4 def searchStr(pattern, msg):
5     txt = re.search(pattern, msg)
6     if txt == None:          # 搜寻失败
7         print("搜寻失败 ",txt)
8     else:                   # 搜寻成功
9         print("搜寻成功 ",txt.group())
10
11 msg1 = 'son'
12 msg2 = 'sonson'
13 msg3 = 'sonsonson'
14 msg4 = 'sonsonsonson'
15 msg5 = 'sonsonsonsonson'
16 pattern = '(son){3,5}'
17 searchStr(pattern,msg1)
18 searchStr(pattern,msg2)
19 searchStr(pattern,msg3)
20 searchStr(pattern,msg4)
21 searchStr(pattern,msg5)
```

执行结果

```
===== RESTART: D:/Python/ch16/ch16_20.py
搜寻失败 None
搜寻失败 None
搜寻成功 sonsonson
搜寻成功 sonsonsonson
搜寻成功 sonsonsonsonson
>>>
```

使用大括号时，也可以省略第一或第二个数字，这相当于不设定最小或最大重复次数。例如：(son){3,}代表重复3次以上皆符合，(son){,10}代表重复10次以下皆符合。有关这方面的实作，将留给读者练习，可参考习题3。

16-4-2 贪婪与非贪婪搜寻

在讲解贪婪与非贪婪搜寻前，笔者先简化程序实例ch16_20.py，使用相同的搜寻模式‘(son){3,5}’，搜寻字符串是‘sonsonsonsonson’，

看看结果。

程序实例 ch16_21.py： 使用搜寻模式 ‘(son){3,5}’， 搜寻字符串 ‘sonsonsonsonson’。

```
1 # ch16_21.py
2 import re
3
4 def searchStr(pattern, msg):
5     txt = re.search(pattern, msg)
6     if txt == None:          # 搜寻失败
7         print("搜寻失败 ",txt)
8     else:                   # 搜寻成功
9         print("搜寻成功 ",txt.group())
10
11 msg = 'sonsonsonsonson'
12 pattern = '(son){3,5}'
13 searchStr(pattern,msg)
```

执行结果

```
===== RESTART: D:\Python\ch16\ch16_21.py =====
搜寻成功  sonsonsonsonson
>>>
```

其实由上述程序所设定的搜寻模式可知3、4或5个son重复就算找到了，可是Python执行结果是列出最多重复的字符串，5次重复，这是Python的默认模式，这种模式又称贪婪(greedy)模式。

另一种是列出最少重复的字符串，以这个实例而言是重复3次，这称非贪婪模式，方法是在正则表达式的搜寻模式右边增加？符号。

程序实例ch16_22.py： 以非贪婪模式重新设计ch16_21.py，请读者留意第12行的正则表达式的搜寻模式最右边的？符号。

```
12 pattern = '(son){3,5}?'    # 非贪婪模式
```

执行结果

```
===== RESTART: D:\Python\ch16\ch16_22.py =====
搜寻成功  sonsonson
>>>
```

16-5 正则表达式的特殊字符

为了不让一开始学习正则表达式太复杂，在前面4个小节笔者只介绍了\d，同时穿插介绍一些字符串的搜寻。我们知道\d代表的是数字字符，也就是从0-9的阿拉伯数字，如果使用管道|的观念，\d相当于是下列正则表达式：

```
(0|1|2|3|4|5|6|7|8|9)
```

这一节将针对正则表达式的特殊字符做一个完整的说明。

16-5-1 特殊字符表

字符	使用说明
\d	0 ~ 9 的整数字元
\D	除了 0 ~ 9 的整数字元以外的其他字符
\s	空白、定位、Tab 键、换行、换页字符
\S	除了空白、定位、Tab 键、换行、换页字符以外的其他字符
\w	数字、字母和底线_ 字符，[A-Za-z0-9_]
\W	除了数字、字母、底线_ 字符和 [a-Za-z0-9_] 以外的其他字符

下列是一些使用上述表格观念的正则表达式的实例说明。

程序实例ch16_23.py： 将一段英文句子的单词分离，同时将英文单词前4个字母是“John”的单词筛选出来。笔者设定如下：

```
pattern = '\w+'          # 意义是把不限长度的数字、字母和底线字符当作符合
搜寻
pattern = 'John\w*'      # John开头后面接0~多个数字、字母和底线字符
```

```
1 # ch16_23.py
2 import re
3 # 测试1将字符串从句子分离
4 msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
5 pattern = '\w+'          # 不限长度的单字
6 txt = re.findall(pattern,msg)  # 传回搜寻结果
7 print(txt)
8 # 测试2将John开始的字符串分离
9 msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
10 pattern = 'John\w*'      # John开头的单字
11 txt = re.findall(pattern,msg)  # 传回搜寻结果
12 print(txt)
```

执行结果


```
===== RESTART: D:/Python/ch16/ch16_23.py =====
['John', 'Johnson', 'Johnnason', 'and', 'Johnnathan', 'will', 'attend', 'my', 'p
arty', 'tonight']
['John', 'Johnson', 'Johnnason', 'Johnnathan']
>>>
```

程序实例ch16_24.py： 正则表达式的应用，下列程序重点是第5行。

\d+：表示不限长度的数字。

\s：表示空格。

\w+：表示不限长度的数字、字母和底线字符连续字符。

```
1 # ch16_24.py
2 import re
3
4 msg = '1 cat, 2 dogs, 3 pigs, 4 swans'
5 pattern = '\d+\s\w+'
6 txt = re.findall(pattern,msg)      # 传回搜寻结果
7 print(txt)
```

执行结果

```
===== RESTART: D:/Python/ch16/ch16_24.py =====
['1 cat', '2 dogs', '3 pigs', '4 swans']
>>>
```

16-5-2 字符分类

Python可以使用中括号来设定字符，可参考下列范例。

[a-z]：代表a-z的小写字符。

[A-Z]：代表A-Z的大写字符。

[aeiouAEIOU]：代表英文发音的元音字符。

[2-5]：代表2-5的数字。

在字符分类中，中括号内可以不用放上正则表示法的反斜杠\执行.、?、*、(、)等字符的转译。例如，[2-5.]会搜寻2-5的数字和句点，这个语法不用写成[2-5\.]。

程序实例 ch16_25.py： 搜寻字符的应用，这个程序首先将搜寻[aeiouAEIOU]，然后将搜寻[2-5.]。

```

1 # ch16_25.py
2 import re
3 # 测试1搜寻[aeiouAEIOU]字符
4 msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
5 pattern = '[aeiouAEIOU]'
6 txt = re.findall(pattern,msg) # 传回搜寻结果
7 print(txt)
8 # 测试2搜寻[2-5.]字符
9 msg = '1. cat, 2. dogs, 3. pigs, 4. swans'
10 pattern = '[2-5.]'
11 txt = re.findall(pattern,msg) # 传回搜寻结果
12 print(txt)

```

执行结果

```

=====RESTART: D:/Python/ch16/ch16_25.py=====
['o', 'o', 'o', 'o', 'a', 'o', 'a', 'o', 'a', 'a', 'i', 'a', 'e', 'a', 'o', 'i']
['.', '2', '.', '3', '.', '4', '.']
>>>

```

16-5-3 字符分类的^字符

在16-5-2小节字符的处理中，如果在中括号内的左方加上^字符，意义是搜寻不在这些字符内的所有字符。

程序实例ch16_26.py： 使用字符分类的^字符重新设计ch16_25.py。

```

1 # ch16_26.py
2 import re
3 # 测试1搜寻不在[aeiouAEIOU]的字符
4 msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
5 pattern = '[^aeiouAEIOU]'
6 txt = re.findall(pattern,msg) # 传回搜寻结果
7 print(txt)
8 # 测试2搜寻不在[2-5.]的字符
9 msg = '1. cat, 2. dogs, 3. pigs, 4. swans'
10 pattern = '[^2-5.]'
11 txt = re.findall(pattern,msg) # 传回搜寻结果
12 print(txt)

```

执行结果

```

=====RESTART: D:/Python/ch16/ch16_26.py=====
['J', 'h', 'n', ' ', 'J', 'h', 'n', 's', 'n', ' ', 'J', 'h', 'n', 'n',
 's', 'n', 'n', 'd', ' ', 'J', 'h', 'n', 'n', 't', 'h', 'n', 'w', 'l',
 'l', 't', 't', 'n', 'd', ' ', 'm', 'y', 'p', 'r', 't', 'y', 't',
 'n', 'g', 'h', 't', 'n', 'd',
 'l', 'c', 'a', 't', ' ', 'd', 'o', 'g', 's', ' ', 'p', 'i',
 'g', 's', ' ', 's', 'w', 'a', 'n', 's']
>>>

```

上述第一个测试结果不会出现[aeiouAEIOU]字符，第二个测试结果不会出现[2-5.]字符。

16-5-4 正则表示法的^字符

这个`^`字符与16-5-3小节的`^`字符完全相同，但是用在不一样的地方，意义不同。在正规表示法中起始位置加上`^`字符，表示正则表示法的字符串必须出现在被搜寻字符串的起始位置，这样搜寻成功才算成功。

程序实例ch16_27.py: 正则表示法`^`字符的应用，测试1字符串John是在最前面所以可以得到搜寻结果，测试2字符串John不是在最前面，结果搜寻失败传回空字符串。

```
1 # ch16_27.py
2 import re
3 # 测试1搜寻John字符串在最前面
4 msg = 'John will attend my party tonight.'
5 pattern = '^John'
6 txt = re.findall(pattern,msg)      # 传回搜寻结果
7 print(txt)
8 # 测试2搜寻John字符串不是在最前面
9 msg = 'My best friend is John'
10 pattern = '^John'
11 txt = re.findall(pattern,msg)      # 传回搜寻结果
12 print(txt)
```

执行结果

```
===== RESTART: D:/Python/ch16/ch16_27.py
['John']
[]
>>>
```

16-5-5 正则表示法的`$`字符

正则表示法的末端放置`$`字符时，表示正则表示法的字符串必须出现在被搜寻字符串的最后位置，这样搜寻成功才算成功。

程序实例ch16_28.py: 正则表示法`$`字符的应用，测试1是搜寻字符串结尾是非英文字符、数字和底线字符，由于结尾字符是“.”，所以传回所搜寻到的字符。测试2是搜寻字符串结尾是非英文字符、数字和底线字符，由于结尾字符是“8”，所以传回搜寻结果是空字符串。测试3是搜寻字符串结尾是数字字符，由于结尾字符是“8”，所以传回搜寻结果“8”。测试4是搜寻字符串结尾是数字字符，由于结尾字符是“.”，所以传回搜寻结果空字符串。

```

1 # ch16_28.py
2 import re
3 # 测试1搜寻最后字符是非英文字母数字和底线字符
4 msg = 'John will attend my party 28 tonight.'
5 pattern = '\w$'
6 txt = re.findall(pattern,msg) # 传回搜寻结果
7 print(txt)
8 # 测试2搜寻最后字符是非英文字母数字和底线字符
9 msg = 'I am 28'
10 pattern = '\w$'
11 txt = re.findall(pattern,msg) # 传回搜寻结果
12 print(txt)
13 # 测试3搜寻最后字符是数字
14 msg = 'I am 28'
15 pattern = '\d$'
16 txt = re.findall(pattern,msg) # 传回搜寻结果
17 print(txt)
18 # 测试4搜寻最后字符是数字
19 msg = 'I am 28 year old.'
20 pattern = '\d$'
21 txt = re.findall(pattern,msg) # 传回搜寻结果
22 print(txt)

```

执行结果

```

===== RESTART: D:/Python/ch16/ch16_28.py
['.']
[]
['8']
[]
>>>

```

我们也可以将16-5-4小节的`^`字符和`$`字符混合使用，这时如果既要符合开始字符串也要符合结束字符串，所以被搜寻的句子一定要只有一个字符串。

程序实例ch16_29.py： 搜寻开始到结束皆是数字的字符串，字符串内容只要有非数字字符就算搜寻失败。测试2中由于中间有非数字字符，所以搜寻失败。读者应留意程序第5行的正则表达式的写法。

```

1 # ch16_29.py
2 import re
3 # 测试1搜寻开始或结尾皆是数字的字符串
4 msg = '09282028222'
5 pattern = '^d+$'
6 txt = re.findall(pattern,msg) # 传回搜寻结果
7 print(txt)
8 # 测试2搜寻开始或结尾皆是数字的字符串
9 msg = '0928tuyr990'
10 pattern = '^d+$'
11 txt = re.findall(pattern,msg) # 传回搜寻结果
12 print(txt)

```

执行结果

```

===== RESTART: D:/Python/ch16/ch16_29.py
['09282028222']
[]
>>>

```

16-5-6 单一字符使用通配符“.”

通配符(wildcard) “.” 表示可以搜寻除了换行字符以外的所有字符，但是只限定一个字符。

程序实例ch16_30.py: 通配符的应用，搜寻一个通配符加上at，在下列输出中，第4个由于at符合，Python自动加上空格符。第6个由于只能加上一个字符，所以搜寻结果是l at。

```
1 # ch16_30.py
2 import re
3 msg = 'cat hat sat at matter flat'
4 pattern = '.at'
5 txt = re.findall(pattern,msg)      # 传回搜寻结果
6 print(txt)
```

执行结果

```
===== RESTART: D:/Python/ch16/ch16_30.py =====
['cat', 'hat', 'sat', ' at', 'mat', 'lat']
>>>
```

如果搜寻的是真正的“.”字符，须使用反斜杠“\.”。

16-5-7 所有字符使用通配符“.*”

若是将16-3-7小节所介绍的“.”字符与“*”组合，可以搜寻所有字符，意义是搜寻0到多个通配符（换行字符除外）。

程序实例ch16_31.py: 搜寻所有字符“.*”的组合应用。

```
1 # ch16_31.py
2 import re
3
4 msg = 'Name: Jiin-Kwei Hung Address: 8F, Nan-Jing E. Rd, Taipei'
5 pattern = 'Name: (.*?) Address: (.*?)'
6 txt = re.search(pattern,msg)      # 传回搜寻结果
7 Name, Address = txt.groups()
8 print("Name: ", Name)
9 print("Address: ", Address)
```

执行结果

```
===== RESTART: D:/Python/ch16/ch16_31.py =====
Name:      Jiin-Kwei Hung
Address:    8F, Nan-Jing E. Rd, Taipei
>>>
```

16-5-8 换行字符的处理

使用16-5-7小节观念用“.”搜寻时碰上换行字符，搜寻就停止。Python的re模块提供参数re.DOTALL，功能是包括搜寻换行字符，可以将此参数放在search()、findall()或compile()。

程序实例ch16_32.py： 测试1是搜寻除换行字符以外的字符，测试2是搜寻含换行字符的所有字符。由于测试2有包含换行字符，所以输出时，换行字符主导分2行输出。

```
1 # ch16_32.py
2 import re
3 #测试1搜寻除了换行字符以外字符
4 msg = 'Name: Jiin-Kwei Hung \nAddress: 8F, Nan-Jing E. Rd, Taipei'
5 pattern = '.*'
6 txt = re.search(pattern,msg) # 传回搜寻不含换行字符结果
7 print("测试1输出: ", txt.group())
8 #测试2搜寻包括换行字符
9 msg = 'Name: Jiin-Kwei Hung \nAddress: 8F, Nan-Jing E. Rd, Taipei'
10 pattern = '.*'
11 txt = re.search(pattern,msg,re.DOTALL) # 传回搜寻含换行字符结果
12 print("测试2输出: ", txt.group())
```

执行结果

```
===== RESTART: D:\Python\ch16\ch16_32.py =====
测试1输出: Name: Jiin-Kwei Hung
测试2输出: Name: Jiin-Kwei Hung
Address: 8F, Nan-Jing E. Rd, Taipei
>>>
```

16-6 MatchObject对象

16-2节已经讲解使用re.search()搜寻字符串，搜寻成功时可以产生MatchObject对象，这里将先介绍另一个搜寻对象的方法re.match()，这个方法搜寻成功后也将产生MatchObject对象。接着本节会分成几个小节，再讲解MatchObject几个重要的方法(method)。

16-6-1 re.match()

这本书已经讲解了搜寻字符串中最重要的2个方法re.search()和re.findall()，re模块另一个方法是re.match()，这个方法其实和re.search()相同，差异是re.match()只搜寻比对字符串开始的字，如果失败就算失败。re.search()则是搜寻整个字符串。至于re.match()

搜寻成功会传回MatchObject对象，若是搜寻失败会传回None，这部分与re.search()相同。

程序实例ch16_33.py: re.match()的应用。测试1是将John放在被搜寻字符串的最前面，测试2没有将John放在被搜寻字符串的最前面。

```
1 # ch16_33.py
2 import re
3 #测试1搜寻使用re.match()
4 msg = 'John will attend my party tonight.' # John是第一个字符串
5 pattern = 'John'
6 txt = re.match(pattern,msg) # 传回搜寻结果
7 if txt != None:
8     print("测试1输出: ", txt.group())
9 else:
10    print("测试1搜寻失败")
11 #测试2搜寻使用re.match()
12 msg = 'My best friend is John.' # John不是第一个字符串
13 txt = re.match(pattern,msg,re.DOTALL) # 传回搜寻结果
14 if txt != None:
15     print("测试2输出: ", txt.group())
16 else:
17    print("测试2搜寻失败")
```

执行结果

```
===== RESTART: D:\Python\ch16\ch16_33.py =====
测试1输出: John
测试2搜寻失败
>>>
```

16-6-2 MatchObject几个重要的方法

当使用re.search()或re.match()搜寻成功时，会产生MatchObject对象。

程序实例ch16_34.py: 看看MatchObject对象是什么。

```
1 # ch16_34.py
2 import re
3 #测试1搜寻使用re.match()
4 msg = 'John will attend my party tonight.'
5 pattern = 'John'
6 txt = re.match(pattern,msg) # re.match()
7 if txt != None:
8     print("使用re.match()输出MatchObject对象: ", txt)
9 else:
10    print("测试1搜寻失败")
11 #测试1搜寻使用re.search()
12 txt = re.search(pattern,msg) # re.search()
13 if txt != None:
14     print("使用re.search()输出MatchObject对象: ", txt)
15 else:
16    print("测试1搜寻失败")
```

执行结果

```
===== RESTART: D:\Python\ch16\ch16_34.py =====
使用re.match()输出MatchObject对象: <_sre.SRE_Match object; span=(0, 4), match='John'>
使用re.search()输出MatchObject对象: <_sre.SRE_Match object; span=(0, 4), match='John'>
>>>
```

从上述可知，当使用`re.match()`和`re.search()`皆搜寻成功时，两者的`MatchObject`对象内容是相同的。`span`是注明成功搜寻字符串的起始位置和结束位置，从此处可以知道起始索引位置是0，结束索引位置是4。`match`则是注明成功搜寻的字符串内容。

Python提供下列取得`MatchObject`对象内容的重要方法。

方法	说明
<code>group()</code>	可传回搜寻到的字符串，本章已有许多实例说明。
<code>end()</code>	可传回搜寻到的字符串的结束位置。
<code>start()</code>	可传回搜寻到的字符串的起始位置。
<code>span()</code>	可传回搜寻到的字符串的(起始,结束)位置。

<code>group()</code>	可传回搜寻到的字符串，本章已有许多实例说明。
<code>end()</code>	可传回搜寻到的字符串的结束位置。
<code>start()</code>	可传回搜寻到的字符串的起始位置。
<code>span()</code>	可传回搜寻到的字符串的(起始,结束)位置。

程序实例ch16_35.py： 分别使用`re.match()`和`re.search()`搜寻字符串Joah，成功搜寻到字符串时，分别用`start()`、`end()`和`span()`方法列出字符串出现的位置。

```
1 # ch16_35.py
2 import re
3 #测试1搜寻使用re.match()
4 msg = 'John will attend my party tonight.'
5 pattern = 'John'
6 txt = re.match(pattern,msg)           # re.match()
7 if txt != None:
8     print("搜寻成功字符串的起始索引位置 :", txt.start())
9     print("搜寻成功字符串的结束索引位置 :", txt.end())
10    print("搜寻成功字符串的结束索引位置 :", txt.span())
11 #测试2搜寻使用re.search()
12 msg = 'My best friend is John.'
13 txt = re.search(pattern,msg)          # re.search()
14 if txt != None:
15     print("搜寻成功字符串的起始索引位置 :", txt.start())
16     print("搜寻成功字符串的结束索引位置 :", txt.end())
17    print("搜寻成功字符串的结束索引位置 :", txt.span())
```

执行结果


```

===== RESTART: D:\Python\ch16\ch16_35.py
搜寻成功字符串的起始索引位置 : 0
搜寻成功字符串的结束索引位置 : 4
搜寻成功字符串的结束索引位置 : (0, 4)
搜寻成功字符串的起始索引位置 : 18
搜寻成功字符串的结束索引位置 : 22
搜寻成功字符串的结束索引位置 : (18, 22)
>>>

```

16-7 抢救CIA情报员-sub()方法

Python re模块内的sub()方法可以用新的字符串取代原本字符串的内容。

16-7-1 一般的应用

sub()方法的基本使用语法如下：

```
result = re.sub(pattern, newstr, msg) # msg是整个欲处理的字符串或句子
```

pattern是欲搜寻的字符串，如果搜寻成功则用newstr取代，同时成功取代的结果回传给result变量，如果搜寻到多个相同字符串，这些字符串将全部被取代，需留意原先msg内容将不会改变。如果搜寻失败则将msg内容回传给result变量，当然msg内容也不会改变。

程序实例ch16_36.py： 这是字符串取代的应用，测试1是发现2个字符串被成功取代(Eli Nan被Kevin Thomson取代)，同时列出取代结果。测试2是取代失败，所以txt与原msg内容相同。

```

1 # ch16_36.py
2 import re
3 #测试1:取代使用re.sub()结果成功
4 msg = "Eli Nan will attend my party tonight. My best friend is Eli Nan"
5 pattern = "Eli Nan" # 欲搜寻字符串
6 newstr = "Kevin Thomson" # 新字符串
7 txt = re.sub(pattern,newstr,msg) # 如果找到则取代
8 if txt != msg: # 如果txt与msg内容不同表示取代成功
9     print("取代成功:", txt) # 列出成功取代结果
10 else:
11     print("取代失败:", txt) # 列出失败取代结果
12 #测试2:取代使用re.sub()结果失败
13 pattern = "Eli Thomson" # 欲搜寻字符串
14 txt = re.sub(pattern,newstr,msg) # 如果找到则取代
15 if txt != msg: # 如果txt与msg内容不同表示取代成功
16     print("取代成功:", txt) # 列出成功取代结果
17 else:
18     print("取代失败:", txt) # 列出失败取代结果

```

执行结果

```
===== RESTART: D:\Python\ch16\ch16_36.py =====
取代成功: Kevin Thomson will attend my party tonight. My best friend is Kevin Thomson
取代失败: Eli Nan will attend my party tonight. My best friend is Eli Nan
>>>
```

16-7-2 抢救CIA情报员

社会上有太多需要保护当事人隐私权利的场所，例如，情报机构在内部文件不可直接将情报员的名字列出来，历史上太多这类实例造成情报员的牺牲，这时可以使用*** 代替原本的姓名。使用Python的正则表示法，可以轻松协助我们执行这方面的工作。这一节将先用程序代码，然后解析此程序。

程序实例ch16_37.py: 将CIA情报员名字，用名字第一个字母和*** 取代。

```
1 # ch16_37.py
2 import re
3 # 使用隐藏文字执行取代
4 msg = 'CIA Mark told CIA Linda that secret USB had given to CIA Peter.'
5 pattern = r'CIA (\w)\w*' # 欲搜寻CIA + 空一格后的名字
6 newstr = r'\1***' # 新字符串使用隐藏文字
7 txt = re.sub(pattern,newstr,msg) # 执行取代
8 print("取代成功: ", txt) # 列出取代结果
```

执行结果

```
===== RESTART: D:\Python\ch16\ch16_37.py =====
取代成功: M*** told L*** that secret USB had given to P***.
>>>
```

上述程序第一个关键是第5行，这一行将搜寻CIA字符串外加空一格后出现不限长度的字符串（可以由英文大小写或数字或底线所组成）。观念是括号内的(\w)代表必须只有一个字符，同时小括号代表这是一个分组(group)，由于整行只有一个括号所以知道这是第一分组，同时只有一个分组，括号外的\w*表示可以有0到多个字符。所以(\w)\w*相当于是1-多个字符组成的单字，同时存在分组1。

上述程序第6行的\1 代表用分组1 找到的第一个字母当作字符串开头，后面*** 则是接在第一个字母后的字符。对CIA Mark而言所找到的第一个字母是M，所以取代的结果是M***。对CIA Linda而言所找到的第一个字母是L，所以取代的结果是L***。对CIA Peter而言所找到的第一个字母是P，所以取代的结果是P***。

16-8 处理比较复杂的正则表示法

有一个正则表示法内容如下：

```
pattern = r((\d{2})|(\d{2}\d{1}))?(\s|-)?\d{8}(\s*(ext|x|ext.)\s*\d{3,5})?
```

其实相信大部分的读者看到上述正则表示法，就想弃械投降了，坦白说的确复杂，不过不用担心，笔者将一步步解析，让事情变简单。

16-8-1 将正则表达式拆成多行字符串

在3-4-2小节笔者有介绍可以使用3个单引号（或是双引号）将过长的字符串拆成多行表达，这个观念也可以应用在正则表达式，当我们适当地拆解后，可以为每一行加上批注，整个正则表达式就变得简单了。若是将上述pattern，拆解成下列表示法，整个就变得简单了。

```
pattern = r'''(
    (\d{2})|(\d{2}\d{1}))?      # 区域号码
    (\s|-)?                    # 区域号码与电话号码的分隔符
    \d{8}                      # 电话号码
    (\s*(ext|x|ext.)\s*\d{2,4})? # 2-4位数的分机号码
)'''
```

接下来笔者分别解释相信读者就可以了解了，第一行区域号码是2位数，可以接受有括号的区域号码，也可以接受没有括号的区域号码，例如，02或(02)皆可以。第二行是设定区域号码与电话号码间的字符，可以接受空格符或-字符当作分隔符。第三行是设定8位数数字的电话号码。第四行是分机号码，分机号码可以用ext或ext.当作起始字符，空一定格数，然后接受2-4位数的分机号码。

16-8-2 re.VERBOSE

使用Python时，如果想在正则表达式中加上批注，可参考16-8-1小节，必须配合使用re.VERBOSE参数，然后将此参数放在search()、findall()或compile()。

程序实例ch16_38.py: 搜寻市区电话号码的应用, 这个程序可以搜寻下列格式的电话号码。

```
12345678          # 没有区域号码
02 12345678       # 区域号码与电话号码间没有空格
02-12345678       # 区域号码与电话号码间使用 - 分隔
(02)-12345678     # 区域号码有小括号
02-12345678 ext 123 # 有分机号
02-12345678 ext. 123 # 有分机号, ext. 右边有 .
```

```
1 # ch16_38.py
2 import re
3
4 msg = '''02-88223349, (02)-26669999, 02-29998888 ext 123,
5       12345678, 02 33887766 ext. 1222'''
6 pattern = r'''(
7     (\d{2})?(\d{2}\d{2})? # 区域号码
8     (\s|-)? # 区域号码与电话号码的分隔符
9     \d{8} # 电话号码
10    (\s*(ext|ext.)\s*\d{2,4})? # 2-4位数的分机号码
11 )'''
12 phoneNum = re.findall(pattern, msg, re.VERBOSE) # 传回搜寻结果
13 print(phoneNum)
```

执行结果

```
===== RESTART: D:\Python\ch16\ch16_38.py =====
[('02-88223349', '02', '-', '88223349'), ('(02)-26669999', '(02)', '-', '26669999'), ('02-29998888 ext 123', '02', ' ', '29998888', 'ext', '123'), ('12345678', ' ', '12345678'), ('02 33887766 ext. 1222', '02', ' ', '33887766', 'ext.', '1222')]
>>>
```

16-8-3 电子邮件地址的搜寻

在字处理过程中, 必须在文件内将电子邮件地址解析出来很常见, 下列是这方面的应用。下列是Pattern内容。

```
pattern = r'''(
    [a-zA-Z0-9_]+ # 使用者账号
    @ # @符号
    [a-zA-Z0-9-]+ # 主机域名domain
    [\.] # 符号
    [a-zA-Z]{2,4} # 可能是com或edu或其它
    ([\.]?)? # 符号, 也可能无特别是美国
    ([a-zA-Z]{2,4})? # 国别
)
```

第1行用户账号常用的有a-z字符、A-Z字符、0-9数字、底线_、点.。第2行是@符号。第3行是主机域名, 常用的有a-z字符、A-Z字符、0-9数字、分隔符-、点.。第4行是点. 符号。第5行最常见的是com或edu, 也可能是cc或其他, 这通常由2至4个字符组成, 常用的有a-z字符、A-Z字符。第6行是点. 符号, 在美国通常只要前5行就够了, 但是在其他国家

则常常需要此字段，所以此字段后面是？字符。第7行通常是国别，例如中国是cn、日本是ja，常用的有a-z字符、A-Z字符。

程序实例ch16_39.py： 电子邮件地址的搜寻。

```
1 # ch16_39.py
2 import re
3
4 msg = '''txt@deepstone.com.tw kkk@gmail.com'''
5 pattern = r'''(
6     [a-zA-Z0-9_]+           # 使用者账号
7     @                       # @符号
8     [a-zA-Z0-9-]+         # 主机域名domain
9     [\.]                   # .符号
10    [a-zA-Z]{2,4}          # 可能是com或edu或其他
11    ([\.]?)                # .符号，也可能无特别是美国
12    ([a-zA-Z]{2,4})?       # 国别
13 )'''
14 eMail = re.findall(pattern, msg, re.VERBOSE) # 传回搜寻结果
15 print(eMail)
```

执行结果

```
===== RESTART: D:/Python/ch16/ch16_39.py =====
[['txt@deepstone.com.tw', '@', '.'], ('kkk@gmail.com', '.')]
>>>
```

16-8-4 re.IGNORECASE/re.DOTALL/re.VERBOSE

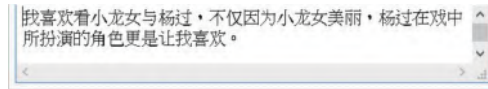
在16-3-9小节笔者介绍了re.IGNORECASE参数，在16-5-8小节笔者介绍了re.DOTALL参数，在16-8-2小节笔者介绍了re.VERBOSE参数，我们可以分别在re.search()、re.findall()、re.match()或是re.compile()方法内使用它们，可是一次只能放置一个参数，如果我们想要一次放置多个参数特性，应如何处理？方法是使用16-3-4小节的管道|观念，例如，可以使用下列方式：

```
datastr = re.search(pattern, msg,
re.IGNORECASE|re.DOTALL|re.VERBOSE)
```

其实这一章已经讲解了相当多的正则表达式的知识了，未来各位在写论文、做研究或职场上相信会有相当帮助。如果仍觉不足，可以自行到Python官网获得更多正则表达式的知识。

习题

1. 中国手机号码格式是 xxx-xxxx-xxxx，x 代表数字，请重新设计 ch16_1.py，可以判断号码是否为中国手机号码。
2. 有一文本文件 ex16_2.txt 内容如下：



请读者设计搜寻字符串小龙女，杨过，同时列出这个字符串出现的次数。这个程序应该采用交互式设计，程序执行时要求输入欲搜寻的字符串，然后列出搜寻结果，接着询问是否继续搜寻，是(y或Y)则继续，否(n或N)则结束。

其实如果将一部小说使用上述分析各个人物出现的次数，就可以知道哪些人物是主角，哪些人物是配角。

3. 请重新设计 ch16_20.py，请使用下列 pattern 做测试。
A: '(son){2,}'
B: '(son){,5}'
4. 请进入本书 ch14 目录，将扩展名是 txt 的文件打印出来，将档名是 ch14_10.py - ch14_19.py 等的 10 个文件的文件名打印出来。
5. 台湾有些地方的电话号码是区域号码 2 位数，电话号码是 7 位数，请修改 ch16_38.py，可以接受 7 位数或 8 位数的电话号码。
6. 重新设计 ch16_39.py，请在第 4 行内加上你的电子邮件地址，另外再加上其他 2 个邮件地址，请将输出结果由列表内的元组元素分离出来，处理成下列方式。

txt@deepstone.com.tw

你的邮件地址

kkk@gmail.com

.....

.....

第17章 使用Python处理Word文件

Word是二进制 (binary) 文件，同时Word还有字体格式、色彩与版面配置等，所以它的处理方式比起文本文件(txt)要复杂。不过，读者不用担心，笔者将以实例一步一步讲解，相信读完本章读者也可以很轻松学会使用Python处理Word文件。

本章内容需要使用外部模块python-docx，读者可参考附录B下载此模块，尽管模块名称是python-docx，下载时指令是：

```
pip install python-docx
```

程序导入使用import时是：

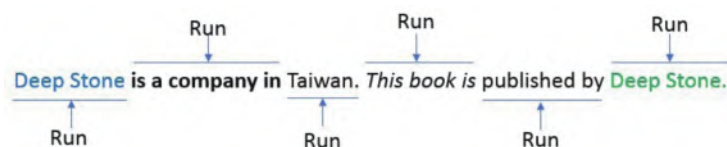
```
import docx # 这一点需留意，不是python-docx
```

17-1 从Python看Word文件结构

在python-docx模块内，将Word文件结构分成3层：

-  Document这是最高层代表整个Word文件。
-  Paragraph一个Word文件是由许多的段落所组成，在Python中整份文件的定义是Document，这些段落的定义就是Paragraph对象。我们使用Word编辑文件时，如果单击一次Enter键，会产生一个新的段落。在Python中一个段落代表一个Paragraph对象，所有段落以Paragraph对象列表(list)方式存在。
-  Run Word文件要考虑的有字号、字体样式、色彩等，我们将这些称作样式。一个Run对象所指的是Paragraph对象中

相同样式的连续文字，如果文字发生样式变化，Python将以新的Run对象代表。



上图有6个Run。

17-2 读取Word文件内容

17-2-1 建立docx对象

首先需建立Word文件(Document)的对象docx对象，可用Document()方法建立，wdoc是自行取的名称，本书实例皆以wdoc 为Document对象名称。

wdoc

```
= docx.Document('文件名') # 建立docx对象wdoc
```

17-2-2 获得Paragraph和Run数量

可以使用len()方法获得Paragraph数量。

```
len(wdoc  
.paragraphs) # wdoc
```

是前一小节所建的docx对象

下列语法可以获得第n 段Paragraph的Run数量。

```
len(wdoc.paragraphs[n  
.runs) # n
```

是第几段或称Paragraph编号

17-2-3 列出Paragraph内容

可以使用下列语法打印第n 段Paragraph内容。

```
print(wdoc.paragraphs[n  
].text)
```

17-2-4 列出Paragraph内的Run内容

可以使用下列语法打印第n 段Paragraph第m 个Run内容。

```
print(wdoc.paragraphs[n  
].runs[m  
].text)
```

17-2-5 读取和打印文件的应用

程序实例ch17_1.py: 有一个Word文件data17_1.docx内容如下:



这个程序会做下列几件事:

1. 第5行, 列出Paragraph的数量, 相当于段落的数量。
2. 第6和7行, 用循环打印各个Paragraph内容, 相当于文件内容。

3. 第9行，列出Paragraph 1的Run数量。
4. 第10和11行，用循环打印Paragraph 1 Run内容。
5. 第13行，列出Paragraph 2的Run数量。
6. 第14和15行，用循环打印Paragraph 2 Run内容。

```
1 # ch17_1.py
2 import docx
3
4 wdoc = docx.Document('data17_1.docx')
5 print("段落数，也可称Paragraph物件数量 = ", len(wdoc.paragraphs))
6 for i in range(0, len(wdoc.paragraphs)):
7     print("paragraph %d = " % i, wdoc.paragraphs[i].text)
8
9 print("Paragraph 1的Run数量 = ", len(wdoc.paragraphs[1].runs))
10 for i in range(0, len(wdoc.paragraphs[1].runs)):
11     print("Run %d = " % i, wdoc.paragraphs[1].runs[i].text)
12
13 print("Paragraph 2的Run数量 = ", len(wdoc.paragraphs[2].runs))
14 for i in range(0, len(wdoc.paragraphs[2].runs)):
15     print("Run %d = " % i, wdoc.paragraphs[2].runs[i].text)
```

执行结果

```
===== RESTART: D:\Python\ch17\ch17_1.py =====
段落数，也可称Paragraph物件数量 = 5
paragraph 0 = 使用Python操作Word
paragraph 1 = 学习Python是一个很好的经验
paragraph 2 = This book is published by Deep Stone.
paragraph 3 = 深石数字科技
paragraph 4 = DeepStone is Deep Learning.
Paragraph 1的Run数量 = 5
Run 0 = 学习
Run 1 = Python
Run 2 = 是一个
Run 3 = 很好的
Run 4 = 经验
Paragraph 2的Run数量 = 3
Run 0 = This book is
Run 1 = published by
Run 2 = Deep Stone.
>>>
```

17-2-6 读取文件与适度编排输出

在操作Word文件时，也可以适度利用一些技巧，执行文件的编排。

程序实例ch17_2.py： 这个程序主要是将所读取的Word文件，第一执行首行缩排2个字，第二段落间空一行输出。下列是data17_2.docx内容。

Word 是二进制(binary)档案, 同时 Word 还有字体格式、色彩与版面配置等, 所以它的处理方式比起文本文件(txt)要复杂。
本章内容需要使用外挂模块 python-docx, 读者可参考附录 B 下载此模块, 尽管模块名称是 python-docx, 下载时指令是:

下列是程序内容。

```
1 # ch17_2.py
2 import docx
3
4 def getFile(fn):
5     '''读取文件与适度编辑文件'''
6     wdoc = docx.Document(fn)          # 建立Word文件对象
7     txt = []
8     for paragraph in wdoc.paragraphs:
9         print(paragraph.text)         # 输出文件所读取的Paragraph内容
10        txt.append('    ' + paragraph.text) # 内缩同时将每一段Paragraph组成列表
11    return '\n\n'.join(txt)           # 将列表组成字符串同时段落隔行输出
12 print(getFile('data17_2.docx'))
```

执行结果

```
RESTART: D:\Python\ch17\ch17_2.py
Word是二进制(binary)文件, 同时Word还有字体格式、色彩与版面配置等, 所以它的
处理方式比起文本文件(txt)要复杂。
本章内容需要使用外挂模块python-docx, 读者可参考附录B下载此模块, 尽管模块名称是py
thon-docx, 下载时指令是:

    Word是二进制(binary)文件, 同时Word还有字体格式、色彩与版面配置等, 所以它
    的处理方式比起文本文件(txt)要复杂。
    本章内容需要使用外挂模块python-docx, 读者可参考附录B下载此模块, 尽管模块名称
    是python-docx, 下载时指令是:

>>>
```

上述程序第8和9行是读取段落Paragraph内容后直接输出, 可以得到前4行的输出结果。第8行至第10行是一个循环, 第10行的功能是在段落Paragraph前方加上4个英文字符宽度, 未来输出时会产生缩排2个中文字宽度的效果, 第10行另外功能是将输出的段落Paragraph组成列表(list)。

程序第11行的join()功能可以参考6-6-3小节, 这个功能是将第10行建立的列表元素组织起来变成字符串, 各字符串元素间使用 '\n\n' 分隔, 这是2个换行符号, 第一个 '\n' 可以让下一个元素换行输出, 第二个 '\n' 就可以产生空一行输出的结果。

17-3 存储文件

save()方法可以存储Document对象的文件, 如果将建立Word文件与存储Word文件整个语法串连, 整个语法如下:

```
import docx
wdoc = docx.Document( 'old_File' )      # 打开文件 old_File
...
wdoc 的编辑动作
...
wdoc.save( 'new_File' )                 # 将文件存入 new_File
```

程序实例 ch17_3.docx : 将文件 data17_1.docx 复制入 out17_3.docx。

```
1 # ch17_3.py
2 import docx
3
4 wdoc = docx.Document('data17_1.docx')
5 wdoc.save('out17_3.docx')
```

执行结果 在目前文件夹可以建立与 data17_1.docx 相同内容的 out17_3.docx。

17-4 建立文件内容

17-4-1 建立标题

可以使用下列 add_heading() 方法建立文件标题内容。

```
wdoc.add_heading('content_of_heading') # wdoc 是自建的文件对象
```

上述预设会建立 Heading 1 的标题，Word 的标题有 1-9，如果想建立不同的标题可以使用第 2 个参数 ‘level=n’。

可以使用下列语法在建立文件标题内容同时设定标题格式。

```
wdoc.add_heading('content_of_heading', level=n) # wdoc 是自建的文件对象
```

程序实例 ch17_4.docx: 建立 Word 标题的应用，这个程序会输出标题，同时将所建的 Word 文件存入 out17_4.docx。

```

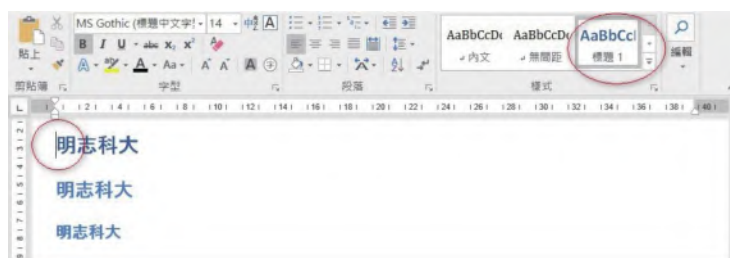
1 # ch17_4.py
2 import docx
3
4 wdoc = docx.Document()
5 wdoc.add_heading('明志科大') # 预设是标题1格式
6 wdoc.add_heading('明志科大', level=2) # 标题2格式
7 wdoc.add_heading('明志科大', level=3) # 标题3格式
8 print(wdoc.paragraphs[0].text)
9 print(wdoc.paragraphs[1].text)
10 print(wdoc.paragraphs[2].text)
11 wdoc.save('out17_4.docx')

```

执行结果 下列是Python Shell窗口执行结果与out17_4.docx。



由于Python Shell是文本模式的窗口，所以执行结果的输出看不出标题效果，但是若是用Word打开，可以看到标题效果，同时将插入点放在标题1字符串，可以在常用/样式 看到标题1 选项。



17-4-2 建立段落Paragraph内容

可以使用add_paragraph()方法建立文件的段落（也可称Paragraph）内容。

```
ptr = wdoc.add_paragraph('paragraph_content') # ptr是段落对象
```

上述ptr可以自行命名用于储存段落对象，这是可有可无的，但是如果有了这个段落对象，未来插入段落时可以将新段落插入此段落的前面，或是将Run内容插入此段落内。可以使用insert_paragraph_before()方法，将段落插在上述ptr段落对象的前方，可参考程序实例ch17_5.py第9行。

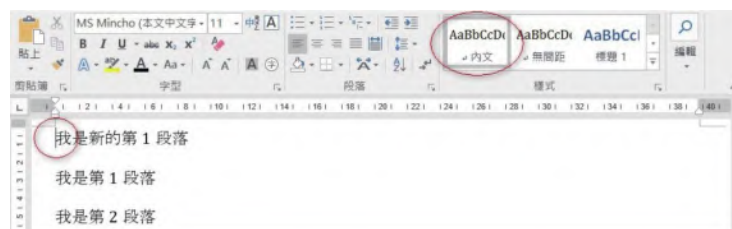
程序实例ch17_5.docx： 先插入2个段落，然后将新段落插在第一个段落的前面。

```
1 # ch17_5.py
2 import docx
3
4 wdoc = docx.Document()
5 ptr = wdoc.add_paragraph('我是第1段落')           # 回传ptr段落对象
6 print(wdoc.paragraphs[0].text)
7 wdoc.add_paragraph('我是第2段落')                 # 不加回传也可以
8 print(wdoc.paragraphs[1].text)
9 prior_ptr = ptr.insert_paragraph_before('我是新的第1段落') # 新段落插在ptr前面
10 print(wdoc.paragraphs[0].text)
11 wdoc.save('out17_5.docx')
```

执行结果 下列是Python Shell窗口执行结果与out17_5.docx。

```
===== RESTART: D:\Python\ch17\ch17_5.py =====
我是第1段落
我是第2段落
我是新的第1段落
>>>
```

将插入点放在Paragraph[0]字符串，可以在常用/样式看到正文选项，正文选项是默认插入文件内容的样式（在17-6节，笔者会介绍插入Paragraph时同时处理样式）。



17-4-3 建立Run内容

Paragraph是由Run组成，当我们建立Paragraph成功后，未来若是想要在Paragraph内插入内容，可以使用add_run()方法，此方法的语法格式如下：

```
ptr.add_run('run_content') # ptr是段落对象，插入run内容
```

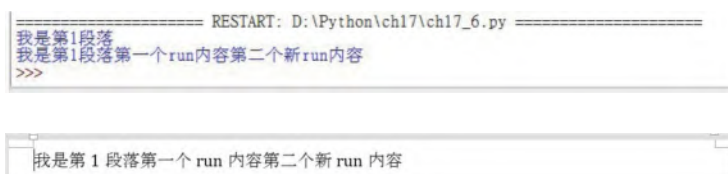
程序实例ch17_6.py： 先建立一个段落，然后将Run加在这个段落内。


```

1 # ch17_6.py
2 import docx
3
4 wdoc = docx.Document()
5 ptr = wdoc.add_paragraph('我是第1段落') # 回传ptr段落对象
6 print(wdoc.paragraphs[0].text)          # 打印段落
7 ptr.add_run('第一个run内容')           # 将run加入ptr段落对象
8 ptr.add_run('第二个新run内容')         # 将run加入ptr段落对象
9 print(wdoc.paragraphs[0].text)         # 打印段落
10 wdoc.save('out17_6.docx')

```

执行结果 下列是Python Shell窗口执行结果与out17_6.docx。



17-4-4 强制换页输出

下列add_page_break()方法可以强制Word换页。

```
wdoc.add_page_break( )
```

未来如果有插入段落时，会在新一页出现。

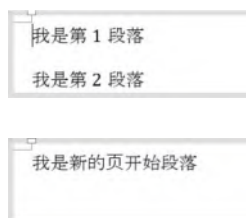
程序实例ch17_7.docx： 强制换页输出的应用。

```

1 # ch17_7.py
2 import docx
3
4 wdoc = docx.Document()
5 ptr = wdoc.add_paragraph('我是第1段落') # 回传ptr段落对象
6 wdoc.add_paragraph('我是第2段落')       # 不加回传也可以
7 wdoc.add_page_break()                   # 插入换页
8 wdoc.add_paragraph('我是新的页开始段落') # 新段落插在新的一页
9 wdoc.save('out17_7.docx')

```

执行结果 下列是out17_7.docx第一页与第二页的结果。



17-4-5 插入图片

可以使用`add_picture()`方法插入图片到Word文件内，如下所示：

```
wdoc.add_picture('image_file')
```

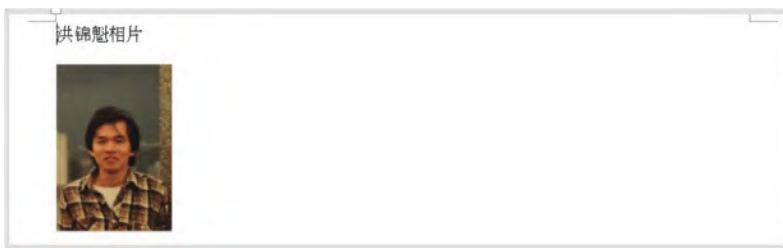
如果插入图片时想要设定图片的宽度或高度，需导入`docx.shared`模块，然后就可以在`add_picture()`方法内增加使用第2个参数`width`(宽度)或`height`(高度)，然后用`Inches()`英寸函数或`Cm()`公分函数设定图片宽度。

```
from docx.shared import Inches
wdoc.add_picture('image_file', width=Inches(宽度值))
```

程序实例ch17_8.docx： 插入图片的应用，图片宽度是1.0英寸。

```
1 # ch17_8.py
2 import docx
3 from docx.shared import Inches
4
5 wdoc = docx.Document()
6 wdoc.add_paragraph('洪锦魁相片')
7 wdoc.add_picture('洪锦魁.jpg', width=Inches(1.0))
8 wdoc.save('out17_8.docx')
```

执行结果 下列是out17_8.docx内容。



17-5 建立表格

`add_table()`方法可以建立表格。

```
table = wdoc.add_table(rows=?, cols=?) # 执行完后返回table  
表格对象
```

17-5-1 建立表格内容

建议可以一次处理一系列的表格内容，如下所示：

```
row = table.rows[0]
row.cells[0].text = '表格(0, 0)内容'
row.cells[1].text = '表格(0, 1)内容'
```

程序实例ch17_9.py： 建立一个2 rows和2 cols的表格。

```
1 # ch17_9.py
2 import docx
3
4 wdoc = docx.Document()
5 table = wdoc.add_table(rows=2, cols=2)
6 row = table.rows[0] # 建立row 0表格数据
7 row.cells[0].text = '书号'
8 row.cells[1].text = '我的著作'
9 row = table.rows[1] # 建立row 1表格数据
10 row.cells[0].text = 'XA1711'
11 row.cells[1].text = 'HTML5+CSS3王者归来'
12 wdoc.save('out17_9.docx')
```

执行结果 下列是out17_9.docx内容。



书号	我的著作
XA1711	HTML5+CSS3 王者归来

17-5-2 插入表格列

可以使用add_row()插入表格列。

程序实例ch17_10.py： 重新设计ch17_9.py，插入表格列和输入表格列数据。

```
1 # ch17_10.py
2 import docx
3
4 wdoc = docx.Document()
5 table = wdoc.add_table(rows=2, cols=2)
6 row = table.rows[0]
7 row.cells[0].text = '书号'
8 row.cells[1].text = '我的著作'
9 row = table.rows[1]
10 row.cells[0].text = 'XA1711'
11 row.cells[1].text = 'HTML5+CSS3王者归来'
12 # 增加表格列和输入数据
13 new_row = table.add_row() # 增加表格行
14 new_row.cells[0].text = 'XA1774'
15 new_row.cells[1].text = 'Python王者归来'
16 wdoc.save('out17_10.docx')
```

执行结果 下列是out17_10.docx内容。

书号	我的著作
XA1711	HTML5+CSS3 王者归来
XA1774	Python 王者归来

17-5-3 计算表格的rows和cols的长度

可以使用len()函数计算表格的rows和cols的长度。

程序实例ch17_11.py: 计算程序实例ch17_10.py所建表格的rows和cols的长度。

```
1 # ch17_11.py
2 import docx
3
4 wdoc = docx.Document()
5 table = wdoc.add_table(rows=2, cols=2)
6 row = table.rows[0]
7 row.cells[0].text = '书号'
8 row.cells[1].text = '我的著作'
9 row = table.rows[1]
10 row.cells[0].text = 'XA1711'
11 row.cells[1].text = 'HTML5+CSS3王者归来'
12 # 增加表格行和输入数据
13 new_row = table.add_row() # 增加表格行
14 new_row.cells[0].text = 'XA1774'
15 new_row.cells[1].text = 'Python王者归来'
16 print('rows = ', len(table.rows)) # 计算和打印rows
17 print('cols = ', len(table.columns)) # 计算和打印cols
```

执行结果

```
===== RESTART: D:/Python/ch17/ch17_11.py =====
rows = 3
cols = 2
>>>
```

17-5-4 打印表格内容

可以使用双层循环打印表格内容，细节可参考下列实例ch17_12.py第17至19行。

程序实例ch17_12.py: 打印ch17_10.py所建的表格内容。

```

1 # ch17_12.py
2 import docx
3
4 wdoc = docx.Document()
5 table = wdoc.add_table(rows=2, cols=2)
6 row = table.rows[0]
7 row.cells[0].text = '书号'
8 row.cells[1].text = '我的著作'
9 row = table.rows[1]
10 row.cells[0].text = 'XA1711'
11 row.cells[1].text = 'HTML5+CSS3王者归来'
12 # 增加表格行和输入数据
13 new_row = table.add_row() # 增加表格行
14 new_row.cells[0].text = 'XA1774'
15 new_row.cells[1].text = 'Python王者归来'
16 # 双层循环打印表格
17 for row in table.rows:
18     for cell in row.cells:
19         print(cell.text)

```

执行结果

```

===== RESTART: D:\Python\ch17\ch17_12.py =====
书号
我的著作
XA1711
HTML5+CSS3王者归来
XA1774
Python王者归来
>>>

```

17-5-5 表格的样式

先前所打印的表格没有框线，可以使用table.style设定框线。

程序实例ch17_13.py： 使用框线样式‘LightShading-Accent1’设定程序ch17_10.py所建立的表格。

```

1 # ch17_13.py
2 import docx
3
4 wdoc = docx.Document()
5 table = wdoc.add_table(rows=2, cols=2)
6 row = table.rows[0]
7 row.cells[0].text = '书号'
8 row.cells[1].text = '我的著作'
9 row = table.rows[1]
10 row.cells[0].text = 'XA1711'
11 row.cells[1].text = 'HTML5+CSS3王者归来'
12 # 增加表格行和输入数据
13 new_row = table.add_row() # 增加表格行
14 new_row.cells[0].text = 'XA1774'
15 new_row.cells[1].text = 'Python王者归来'
16 table.style = 'LightShading-Accent1' # 设定表格样式
17 wdoc.save('out17_13.docx')

```

执行结果 这个程序执行时有Warning信息，可以不必理会，下列是out17_13.docx内容。

书号	我的著作
XA1711	HTML5+CSS3 王者归来
XA1774	Python 王者归来

如果打开表格样式表单将鼠标光标移至样式表单，可以看到各个样式表单名称，不过目前笔者写此书时的python-docx 0.86版本尚未支持中文样式表单。上述第16行是设定表格的样式，LightShading是浅色底纹，Accent1是辅色1，若是调整为Accent2，……，Accent6将有不同的结果。

17-6 Paragraph样式

Paragraph样式 就是所谓的段落样式，下列是常见的Word样式内容。

Normal(正文)	BodyText(本文)	ListNumber(列表号码 2, 3)
Caption(书名)	Heading(标题 1 … 9)	ListBullet(项目符号 2, 3)
Title(标题)	List(清单 2, 3)	ListParagraph(清单段落)

上述list、listBullet、ListNumber如果是编号1可以省略编号，如果是编号2和3则可以标明。Heading则是由Heading1、……、Heading9所组成。我们在插入段落时，可以在add_paragraph()方法内增加第2个参数“style=样式名称”，这样就可以在插入段落同时设定段落的样式，可参考下列程序第5行。

程序实例ch17_14.py： 建立段落时，同时设定段落的样式。

```
1 # ch17_14.py
2 import docx
3
4 wdoc = docx.Document()
5 wdoc.add_paragraph('明志科大', style='ListNumber') # ListNumber
6 wdoc.add_paragraph('长庚科大', style='ListNumber') # ListNumber
7 wdoc.add_paragraph('长庚大学', style='ListNumber') # ListNumber
8 wdoc.add_paragraph('明志科大', style='ListBullet') # ListBullet
9 wdoc.add_paragraph('长庚科大', style='ListBullet') # ListBullet
10 wdoc.add_paragraph('长庚大学', style='ListBullet') # ListBullet
11 wdoc.save('out17_14.docx')
```

执行结果 执行时会有Warning消息。



17-7 Run的样式

Run的样式重点就是设定Run的文字(text)属性，下列是常见的属性。

`bold`(粗体) `italic`(斜体) `underline`(下画线) `strike`(删除线)

当我们建立一个Run对象时，会回传Run对象，此时若将此对象的样式设为True，相当于可以建立该Run对象的样式。整个细节读者可以参考第4-7行。

程序实例ch17_15.py： 建立Run内容，然后设定此内容的属性为粗体(bold)与斜体(italic)的应用。

```
1 # ch17_15.py
2 import docx
3
4 wdoc = docx.Document()
5 ptr = wdoc.add_paragraph('我是第1段落') # 回传ptr段落对象
6 run1 = ptr.add_run('我是粗体')          # 将run加入ptr段落对象
7 run1.bold = True                        # 设定粗体
8 run2 = ptr.add_run('我是斜体')          # 将run加入ptr段落对象
9 run2.italic = True                      # 设定斜体
10 wdoc.save('out17_15.docx')
```

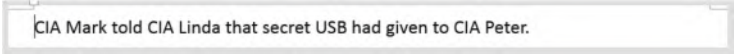
执行结果



17-8 综合应用——抢救CIA情报员

在16-7-2小节笔者有介绍这个范例，当时文字适用msg字符串方式表示，在真实的应用中所有文件皆是以Word方式表达，这一节将用读取Word文件方式重新设计ch16_37.py。

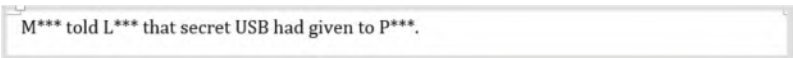
程序实例 ch17_16.py : 重新设计 ch16_37.py , 下列是 data17_16.docx 的内容。



下列是程序内容。

```
1 # ch17_16.py
2 import docx
3 import re
4
5 wdoc = docx.Document('data17_16.docx')
6 txt = docx.Document()
7 pattern = r'CIA (\w)\w*'          # 欲搜寻CIA + 空格后的名字
8 newstr = r'\1***'                # 新字符串使用隐藏文字
9 txt.add_paragraph(re.sub(pattern,newstr,wdoc.paragraphs[0].text))
10 txt.save('out17_16.docx')
```

执行结果 下列是out17_16.docx的内容。

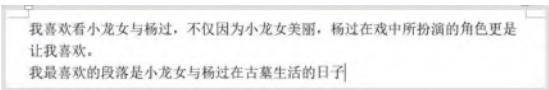


更多有关Python-docx的用法，读者可参考下列Python官方网页。

<https://python-docx.readthedocs.io/en/latest/>

习题

1. 请重新设计ch16_36.py，将第4行的msg存入ex17_1.docx，然后将修改结果存入ex17_1_1.docx。
2. 有一文本文件ex17_2.docx内容如下：



请读者设计搜寻字符串小龙女，杨过，同时列出这个字符串出现的次数。这个程序应该采用交互式设计，程序执行时要求输入欲搜

寻的字符串，然后列出搜寻结果，接着询问是否继续搜寻，是(y或Y)则继续，否(n或N)则结束。

3. 请用Python设计一份自传，内容可以自行发挥。

4. 请建立下列表格，表格样式可以自行设定。

方法	说明
group()	可传回搜寻到的字符串，本章已有许多实例说明。
end()	可传回搜寻到的字符串的结束位置。
start()	可传回搜寻到的字符串的起始位置。
span()	可传回搜寻到的字符串的(起始,结束)位置。

第18章 使用Python处理PDF文件

PDF文件和Word文件一样是二进制(binary)文件，所以处理起来步骤会多一点，不过，读者不用担心，笔者将以实例一步一步讲解，相信读完本章读者也可以很轻松学会使用Python处理PDF文件。

本章内容需要使用外部模块PyPDF2，下载此模块时指令如下：

```
pip install PyPDF2
```

程序导入使用import时是：

```
import PyPDF2
```

18-1 打开PDF文件

我们可以使用open()打开PDF文件，语法如下：

```
pdfObj = open('pdf_file', 'rb')  
# 'rb'表示以二进制打开
```

上述pdf_file是要打开的文件，开档成功后会传回所打开PDF文件的文件对象，在上述语法中笔者将所打开PDF文件的文件对象设定给pdfObj，未来就用pdfObj代表所打开的PDF文件。本书使用的PDF文件travel.pdf内容有3页，下列是第1页内容。



18-2 获得PDF文件的页数

打开PDF文件成功后，可以使用PdfFileReader()方法读取这个PDF文件，下列是语法内容：

```
pdfRd = PyPDF2.PdfFileReader(pdfObj) # 读取PDF内容
```

上述会将所读取的内容放在pdfRd对象变量内，这个对象变量内含numPages属性记录此PDF文件的页数。

程序实例ch18_1.py： 计算travel.pdf的页数，这个文件在ch18文件夹内。

```
1 # ch18_1.py
2 import PyPDF2
3
4 fn = 'travel.pdf' # 设定欲读取的PDF文件
5 pdfObj = open(fn, 'rb') # 以二进制方式打开
6 pdfRd = PyPDF2.PdfFileReader(pdfObj)
7 print("PDF页数是 =", pdfRd.numPages)
```

执行结果 读者可检查页面，这个PDF文件的确是3页。

```
===== RESTART: D:\Python\ch18\ch18_1.py :
PDF页数是 = 3
>>>
```

18-3 读取PDF页面内容

使用PdfFileReader()方法读取这个PDF文件后，可以使用getPage(n)取得第n页的PDF内容，如下所示：

```
pdfContentObj = pdfRd.getPage(n) # 读取第n页内容
```

PDF页面也是从第0页开始计算，页面内容被读入pdfContentObj对象后，可以使用extractText()取得该页的字符串内容。需留意，PyPDF2模块对于读取英文文件，比较没有障碍，对于中文内容会出现乱码。另外，PyPDF2无法读取图表或表格数据。

程序实例ch18_2.py： 读取travel.pdf的第0页内容。

```

1 # ch18_2.py
2 import PyPDF2
3
4 fn = 'travel.pdf'          # 设定欲读取的PDF文件
5 pdfObj = open(fn,'rb')    # 以二进制方式打开
6 pdfRd = PyPDF2.PdfFileReader(pdfObj) # 读取PDF文件
7 pageObj = pdfRd.getPage(0) # 将第10页内容读入pageObj
8 txt = pageObj.extractText() # 提取页面内容
9 print(txt)

```

执行结果

```

===== RESTART: D:\Python\ch18\ch18_2.py =====
Traveling in the USA
Jiin
-
Kwei
Hun
g
Kwei Travel Agency
Traffic
on
the
roa
d
Famous
scenic
are
a
>>>

```

本书有附ch18_2_1.py读取第1页内容，ch18_2_2.py读取第2页内容，由这2页的读取结果可以发现PyPDF2模块对于含中文的PDF文件尚未支持。

18-4 检查PDF是否被加密

初次执行“pdfRd = PyPDF2.PdfFileReader(pdfObj)”之后，pdfRd对象会有isEncrypted属性，如果此属性是True，表示文件有加密。如果此属性是False，表示文件没有加密。

程序实例ch18_3.py： 检查文件是否加密，在ch18文件夹内有travel.pdf和encrypttravel.pdf文件，本程序会测试这2个文件。

```

1 # ch18_3.py
2 import PyPDF2
3
4 def encryptYorN(fn):
5     '''检查文件是否加密'''
6     pdfObj = open(fn,'rb')
7     pdfRd = PyPDF2.PdfFileReader(pdfObj)
8     if pdfRd.isEncrypted: # 由这个属性判断是否加密
9         print("%s 文件有加密" % fn)
10    else:
11        print("%s 文件没有加密" % fn)
12
13 encryptYorN('travel.pdf')
14 encryptYorN('encrypttravel.pdf')

```

执行结果

```
===== RESTART: D:\Python\ch18\ch18_3.py
travel.pdf 文件没有加密
encrypttravel.pdf 文件有加密
>>>
```

18-5 解密PDF文件

对于加密的PDF文件，我们可以使用`decrypt()`执行解密，如果解密成功`decrypt()`会传回1，如果失败则传回0。

程序实例 `ch18_4.py`： 读取使用密码 ‘`jiinkwei`’ 加密的 `encrypttravel.pdf`文件。

```
1 # ch18_4.py
2 import PyPDF2
3
4 pdfObj = open('encrypttravel.pdf','rb')
5 pdfRd = PyPDF2.PdfFileReader(pdfObj)
6 if pdfRd.decrypt('jiinkwei'):          # 检查密码是否正确
7     pageObj = pdfRd.getPage(0)         # 密码正确则读取第0页
8     txt = pageObj.extractText()
9     print(txt)
10 else:
11     print('解密失败')
```

执行结果 执行结果可以参考`ch18_2.py`。

在`ch18`文件夹有`ch18_4_1.py`文件，这个文件第6行，笔者故意将密码写错，将印出‘解密失败’的信息，读者可以试着执行体会结果。读者需留意的是使用`decrypt()`解密时，是解`pdfRd`对象的密码不是整份PDF，未来如果其他程序要使用这个PDF，仍须执行解密才可阅读使用。

18-6 建立新的PDF文件

目前PyPDF2模块只能将其他的PDF页面转存成PDF文件，还无法将Word、PowerPoint等文件转成PDF文件。它的基本流程如下：

①建立一个`PdfWr`对象（名称可以自取），未来写入用。

②将已有pdfRd对象一次一页复制到pdfWr对象。

③使用write()方法将pdfWriter对象写入PDF文件。

一次一页PDF的复制可以使用addPage()，细节可参考ch18_5.pdf第7和8行。最后使用write()将pdfWr写入文件可参考第10和11行。

程序实例ch18_5.py： 将travel.pdf的第一页复制到out18_5.pdf。

```
1 # ch18_5.py
2 import PyPDF2
3
4 pdfObj = open('travel.pdf','rb')
5 pdfRd = PyPDF2.PdfFileReader(pdfObj)
6
7 pdfWr = PyPDF2.PdfFileWriter()          # 新的PDF对象
8 pdfWr.addPage(pdfRd.getPage(0))         # 将第0页放入新的PDF对象
9
10 pdfOutFile = open('out18_5.pdf', 'wb') # 开启二进制文件供写入
11 pdfWr.write(pdfOutFile)                # 执行写入
12 pdfOutFile.close()
```

执行结果 程序执行后在ch18文件夹可以看到out18_5.pdf文件。

如果要执行整个PDF文件的复制，可以将上述第8行改成for循环，就可以一次一页执行文件的复制。

程序实例ch18_6.py： 将travel.pdf复制至out18_6.pdf。

```
1 # ch18_6.py
2 import PyPDF2
3
4 pdfObj = open('travel.pdf','rb')
5 pdfRd = PyPDF2.PdfFileReader(pdfObj)
6
7 pdfWr = PyPDF2.PdfFileWriter()          # 新的PDF对象
8 for pageNum in range(pdfRd.numPages):
9     pdfWr.addPage(pdfRd.getPage(pageNum)) # 一次将一页放入新的PDF对象
10
11 pdfOutFile = open('out18_6.pdf', 'wb')  # 开启二进制文件供写入
12 pdfWr.write(pdfOutFile)                 # 执行写入
13 pdfOutFile.close()
```

执行结果 你可以在ch18文件夹看到out18_6.pdf，内容与travel.pdf相同。

18-7 PDF页面的旋转

在浏览PDF文件时，可以旋转PDF页面。rotateClockwise()可以执行页面顺时针旋转，rotateCounterClockwise()可以执行逆时针旋

转。在这2个方法内可以传入90、180、270度执行旋转工作。

程序实例ch18_7.py： 将travel.pdf的第0页旋转90度，然后存入out18_7.pdf。

```
1 # ch18_7.py
2 import PyPDF2
3
4 pdfObj = open('travel.pdf','rb')
5 pdfRd = PyPDF2.PdfFileReader(pdfObj)
6
7 pdfWr = PyPDF2.PdfFileWriter() # 新的PDF对象
8 pageR = pdfRd.getPage(0) # 原始第0页
9 pageR = pageR.rotateClockwise(90) # 第0页旋转90度
10 pdfWr.addPage(pageR) # 将旋转后的第0页放入新的PDF对象
11
12 pdfOutFile = open('out18_7.pdf', 'wb') # 开启二进制文件供写入
13 pdfWr.write(pdfOutFile) # 执行写入
14 pdfOutFile.close()
```

执行结果 这个程序会建立out18_7.pdf，下列是第0页内容。



18-8 加密PDF文件

若是想要将PDF文件加密，可以在将pdfWr对象正式使用write()方法写入前调用encrypt()执行，加密的密码当作参数放在encrypt()方法内。

程序实例ch18_8.py： 将travel.pdf文件加密储存在output.pdf内，密码是deepstone。

```

1 # ch18 8.py
2 import PyPDF2
3
4 pdfObj = open('travel.pdf', 'rb')
5 pdfRd = PyPDF2.PdfFileReader(pdfObj)
6
7 pdfWr = PyPDF2.PdfFileWriter()          # 新的PDF对象
8 for pageNum in range(pdfRd.numPages):
9     pdfWr.addPage(pdfRd.getPage(pageNum)) # 一次将一页放入新的PDF对象
10
11 pdfWr.encrypt('deepstone')             # 执行加密
12 encryptPdf = open('output.pdf', 'wb')   # 开启二进制文件供写入
13 pdfWr.write(encryptPdf)                 # 执行写入
14 encryptPdf.close()

```

执行结果 执行打开output.pdf后将看到要求输入密码的窗口。



上述程序的关键是第11行，先对pdfWr对象加密，加密完成后，第13行再将pdfWr对象写入新打开的二进制文件对象encryptPdf，最后关闭此文件。

18-9 处理PDF页面重叠

有2个PDF文件分别是sse.pdf和secret.pdf，内容如下左边两图：

sse.pdf是一般的PDF文件，secret.pdf是含水印的PDF文件，所谓的页面重叠，就是将2个PDF页面组合。如右下图所示。



要完成这个工作，步骤如下，下列是用程序实例ch18_9.py为例说明：

- ①打开一般PDF文件，然后将页面内容放入ssePage对象(4-6行)。
- ②打开一般水印文件，然后将页面内容放入secretPage对象(8-10行)。
- ③使用下列指令执行重叠。

```
ssePage.merge(secretPage) # 重叠结果放在ssePage对象(12行)
```

- ④打开新的对象pdfWr，将ssePage结果存入新对象pdfWr(14-15行)。

- ⑤打开新的文件out18_9.pdf，此文件对象名称是mergePdf(17行)。

- ⑥将pdfWr写入mergePdf(18行)。

程序实例ch18_9.py: sse.Pdf文件与secret.pdf文件合并，同时将结果存入out18_9.pdf。

```
1 # ch18_9.py
2 import PyPDF2
3
4 pdfSSE = open('sse.pdf', 'rb') # 开启一般pdf文件
5 pdfRdSSE = PyPDF2.PdfFileReader(pdfSSE)
6 ssePage = pdfRdSSE.getPage(0)
7
8 pdfSecret = open('secret.pdf', 'rb') # 开启水印pdf文件
9 pdfRdSecret = PyPDF2.PdfFileReader(pdfSecret)
10 secretPage = pdfRdSecret.getPage(0)
11
12 ssePage.mergePage(secretPage) # 执行重叠合并
13
14 pdfWr = PyPDF2.PdfFileWriter() # 新的PDF对象
15 pdfWr.addPage(ssePage) # 将重叠页放入新的PDF对象
16
17 mergePdf = open('out18_9.pdf', 'wb') # 开启二进制文件供写入
18 pdfWr.write(mergePdf) # 执行写入
19 mergePdf.close()
```

执行结果 打开out18_9.py后可以得到本节解说的文件。

18-10 破解密码的程序设计

有时候自己设计了一个PDF，但是忘记了密码怎么办？其实Python的功能可以让我们设计破解密码程序，先从简单开始吧！

破解3位数字的密码

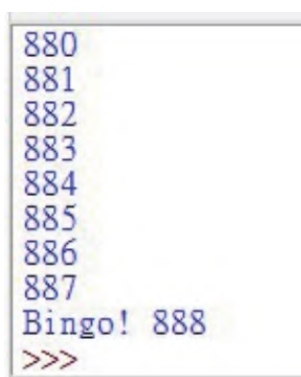
如果密码是由3个阿拉伯数字组成，表示有3个位数，每个位数是由0-1所组成，读者可以使用下列方式设计密码。

程序实例ch18_10.py： 破解3位数字的密码，程序的密码是在第3行设定，程序执行过程会将所测试失败的密码不断打印出来直到找到密码，此时会列出Bingo!字符串。为了让读者明白工作原理，所以一个密码用一行输出，在真实工作中可以不用如此，密码间空一格即可，不输出测试密码也不好，因为无法知道测试密码的进度。

```
1 # ch18_10.py
2
3 secretcode = '888'
4 codeNotFound = True
5 for i1 in range(0, 10):
6     if codeNotFound:
7         for i2 in range(0, 10):
8             if codeNotFound:
9                 for i3 in range(0, 10):
10                    code = str(i1) + str(i2) + str(i3)
11                    if code == secretcode:
12                        print('Bingo!', code)
13                        codeNotFound = False
14                        break
15                    else:
16                        print(code)
```

设定密码
尚未找到密码为True
第一位数
检查是否找到没有找到才会往下执行
第二位数
检查是否找到没有找到才会往下执行
第三位数
组成密码
比对密码
注明已经比对成功
打印无效码

执行结果



```
880
881
882
883
884
885
886
887
Bingo! 888
>>>
```

如果密码位数比较多，只要第9行下面增加循环数即可。读者可能会想密码一般是由英文字母组成，其实用英文字母也可以，只不过是增加一些转换上的问题。

程序实例ch18_11.py： 设定密码位数有3位，是由纯英文字母大写所组成。

```
1 # ch18_11.py
2
3 secretcode = 'DAY' # 设定密码
4 codeNotFound = True # 尚未找到密码为True
5 for i1 in range(1, 27): # 第一位数
6     if codeNotFound: # 检查是否找到没有找到才会往下执行
7         for i2 in range(1, 27):
8             if codeNotFound: # 检查是否找到没有找到才会往下执行
9                 for i3 in range(1, 27): # 第三位数
10                     code = chr(i1+64) + chr(i2+64) + chr(i3+64) # 组成密码
11                     if code == secretcode: # 比对密码
12                         print('Bingo!', code)
13                         codeNotFound = False # 注明已经比对成功
14                         break
15                     else:
16                         print(code, end=' ')
```

执行结果

```
COE COF COG COH COI COJ COK COL COM CON COO COP COQ COR COS COT COU COV COW COX
COY COZ CPA CPB CPC CPD CPE CPF CPG CPH CPI CPJ CPK CPL CPM CPN CPO CPP CPQ CPR
CPS CPT CPU CPV CPW CPX CPY CPZ CQA CQB CQC CQD CQE CQF CQG CQH CQI CQJ CQK CQL
COM CON COO COP COQ COR COS COT COU COV COW COX
COY COZ CRA CRB CRC CRD CRE CRF
CRG CRH CRI CRJ CRK CRL CRM CRN CRO CRP CRQ CRR CRS CRT CRU CRV CRW CRX CRY CRZ
CSA CSB CSC CSD CSE CSF CSg CSI CSJ CSL CSM CSN CSO CSP CSQ CSS CST
CSU CSV CSW CSX CSY CSZ CTA CTB CTC CTD CTE CTF CTG CTH CTI CTJ CTK CTL CTM CTN
CTO CTP CTQ CTR CTS CTT CTU CTV CTW CTX CTY CTZ CUA CUB CUC CUD CUE CUF CUG CUH
CUI CUJ CUK CUL CUM CUN CUO CUP CUQ CUR CUS CUT CUU CUV CUW CUX CUY CUZ CVA CVB
CVC CVD CVE CVF CVG CVH CVI CVJ CVK CVL CVM CVN CVO CVP CVQ CVR CVS CVT CVU CVV
CVW CVX CVY CVZ CWA CWB CWC CWD CWE CWF CWG CWH CWI CWN CXL CXM CXN CXO
CXQ CXR CXS CXT CXU CXV CXW CXX CXY CXZ CYA CYB CYC CYD
CYE CYF CYG CYH CYI CYJ CYK CYL CYM CYN CYO CYP CYQ CYR CYS CYT CYU CYV CYW CYX
CYZ CZA CZB CZC CZD CZE CZF CZG CZH CZI CZJ CZK CZL CZM CZN CZO CZP CZQ CZR
CZS CZZ CZU CZV CZW CZZ CZY CZZ DAA DAB DAC DAD DAE DAF DAG DAH DAI DAJ DAK DAL
DAM DAN DAO DAP DAQ DAR DAS DAT DAU DAV DAW DAX DAX! DAY
>>>
```

由于有26个英文字母，所以所有循环均是执行26圈，range(1, 26)，由于字母A的Unicode是65，所以i1(i2或i3也是)值加上64就会是相对应的英文字母，这个程序会执行所有可能的比对。

其实上述程序可以扩充到密码由大小写英文字母以及数字所组成，只是所花的时间会比较久，程序运行期间所花的只是CPU时间。读到这里各位应可以体会目前几乎所有银行进入网络银行需要有验证码，这是做双重保险，同时无法由机器人操作。但是最终建议是不论在哪一种场合设定密码时，尽量由英文字母大小写与数字组成，比较安全。

习题

1. PDF文件复制的应用，请输入来源文件和目标文件，这个程序会将来源文件的PDF文件复制到目标文件。来源文件请使用travel.pdf，目标文件的档名是ex18_1.pdf。

2. 请将2个PDF文件合并成一个PDF文件，其中一个主要PDF文件是你的自传，必须有3页。另一个文件是水印PDF文件，只有一页，水印放的是你的相片，所以合并后的PDF文件将是一个含有水印的自传。
3. 请搜寻travel.pdf文件，将含有Traffic的PDF页面筛选出来，然后将结果存入新的PDF文件内。
4. 将travel.pdf文件的PDF页面颠倒顺序重新排列。
5. 请利用14章介绍的os.walk()功能搜寻本章文件夹所有的PDF文件，然后将所有文件PDF文件加密，密码是‘python’，同时以‘_encry.ptf’为后置档名。
6. 请将程序ch18_10.py扩充为8位数字的暴力密码破解程序，用899998当作密码测试。
7. 请将程序ch18_11.py扩充为5位数字的暴力密码破解程序，用DFXVY当作密码测试。
8. 请将程序ch18_11.py扩充为5位数字的、每一位是由大写英文字加上0-9数字所组成的暴力密码破解程序，用DF01Y当作密码测试。
9. 请破解本文件夹的ex10_9.pdf文件，这个密码是由5个英文小写所组成。

第19章 使用Python处理Excel文件

Excel是电子表格软件，主要是做数据的统计与分析。有时候我们可能会需要从数百或更多电子表格中依条件复制一些数据到其他表格，或是从数百或更多数据表中搜寻符合特定条件的数据等，这些皆是符合使用Python处理的条件。有关Microsoft Excel的更多使用知识可参考笔者所著《看图例学Excel 2016》佳魁信息发行。

本章内容需要使用外部模块openpyxl，读者可参考附录B下载此模块，下载时指令是：

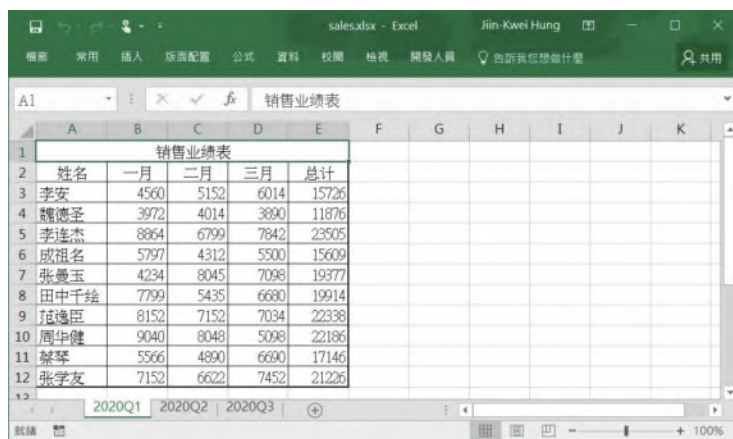
```
pip install openpyxl
```

程序导入方法如下：

```
import openpyxl
```

19-1 认识Excel窗口

下列是Microsoft Excel窗口。



The screenshot shows the Microsoft Excel application window with the title bar 'sales.xlsx - Excel' and the user name 'Jiin-Kwei Hung'. The ribbon is set to '数据' (Data). The active sheet is '销售业绩表' (Sales Performance Table). The data is as follows:

姓名	一月	二月	三月	总计
李安	4560	5152	6014	15726
魏德圣	3972	4014	3890	11876
李连杰	8864	6799	7842	23505
成祖名	5797	4312	5500	15609
张曼玉	4234	8045	7098	19377
田中千绘	7799	5435	6690	19914
范逸臣	8152	7152	7034	22338
周华健	9040	8048	5098	22186
蔡琴	5566	4890	6690	17146
张学友	7152	6622	7452	21226

Microsoft Excel文件的扩展名是xlsx，下列是一些基本名词。

工作簿(workbook)：Excel的文件又称工作簿。

工作表(worksheet)：一个工作簿由不同数量的工作表组成，若以上图为例，是由2020Q1、2020Q2、2020Q3等3个工作表所组成，其中2020Q1底色是白色，表示这是当前工作表(active sheet)。

栏(column)：工作表的栏名称 是A、B、……

行(row)：工作表的行名称 是1、2、……国人有时将row 翻译为列。

单元格(cell)：工作表内的每一个格子称单元格，用(栏名，行名)代表。

19-2 读取Excel文件

在本书ch19文件夹有sales.xlsx，本节主要以此文件为实例解说。

19-2-1 打开文件

当我们导入openpyxl模块后，可以使用openpyxl.load_workbook()方法打开Excel文件，然后可以回传Excel文件对象，本章将用wb 变量代表workbook文件对象，当然读者也可以使用其他名称。

程序实例ch19_1.py：打开sales.xlsx文件，然后列出回传Excel文件对象的文件类型。

```
1 # ch19_1.py
2 import openpyxl
3
4 fn = 'sales.xlsx'
5 wb = openpyxl.load_workbook(fn) # wb是Excel文件对象
6 print(type(wb))
```

执行结果

```
===== RESTART: D:/Python/ch19/ch19_1.py =====
<class 'openpyxl.workbook.workbook.Workbook'>
>>>
```

19-2-2 取得工作表worksheet名称

可以使用`get_sheet_names()`取得所打开工作簿文件的所有工作表，工作表将以列表数据类型回传。`get_active_sheet()`可以取得当前工作表的名称，注意，这里所指的当前工作表是打开文件后自动显示的工作表名称。本章将用`ws` 变量代表`Worksheet`文件对象，当然读者也可以使用其他名称。

程序实例ch19_2.py: 列出`sales.xlsx`文件所有的工作表和当前工作表的名称。

```
1 # ch19_2.py
2 import openpyxl
3
4 fn = 'sales.xlsx'
5 wb = openpyxl.load_workbook(fn) # wb是Excel文件对象
6 allSheets = wb.get_sheet_names() # 所有工作表对象
7 print("所有工作表 = ", allSheets)
8
9 ws = wb.get_active_sheet() # ws是当前工作表对象
10 print("当前工作表 = ", ws)
```

执行结果

```
===== RESTART: D:/Python/ch19/ch19_2.py =====
所有工作表 = ['2020Q1', '2020Q2', '2020Q3']
当前工作表 = <Worksheet "2020Q1">
>>>
```

对于当前工作表对象而言，此例是`ws`，可以使用`title`属性列出实际内容。

程序实例ch19_3.py: 重新设计`ch19_2.py`，将`title`属性应用在`ws`对象。

```
1 # ch19_3.py
2 import openpyxl
3
4 fn = 'sales.xlsx'
5 wb = openpyxl.load_workbook(fn)
6 ws = wb.get_active_sheet()
7 print("当前工作表 = ", ws.title)
```

执行结果

```
===== RESTART: D:\Python\ch19\ch19_3.py =====
当前工作表 = 2020Q1
>>>
```

19-2-3 设定当前工作的工作表

使用Python操作Excel文件时，可能需随时更改当前工作表，可以使用`get_sheet_by_name()`，然后将要设为当前工作表的名称当做这个方法的参数。

程序实例ch19_4.py： 更改当前工作表为2020Q3。

```
1 # ch19_4.py
2 import openpyxl
3
4 fn = 'sales.xlsx'
5 wb = openpyxl.load_workbook(fn)
6 ws = wb.get_sheet_by_name('2020Q3') # 设定当前工作表
7 print("当前工作表 = ", ws.title)
```

执行结果

```
===== RESTART: D:/Python/ch19/ch19_4.py =====
当前工作表 = 2020Q3
>>>
```

其实我们可以将上述`activeSheet`当作是当前工作表对象。

19-2-4 取得工作表内容

在上一小节的程序中我们有了当前工作表对象`ws`，我们可以用下列方式取得单元格内容。

`ws['栏`

`行'].value # 栏`

是A, B,、行是1, 2,

程序实例ch19_5.py： 列出一系列不同位置的单元格内容。


```

1 # ch19_5.py
2 import openpyxl
3
4 fn = 'sales.xlsx'
5 wb = openpyxl.load_workbook(fn)
6 ws = wb.get_sheet_by_name('2020Q1') # 当前工作表2020Q1
7 print("单元格A1 = ", ws['A1'].value) # A1
8 print("单元格A2 = ", ws['A2'].value) # A2
9 print("单元格B2 = ", ws['B2'].value) # B2
10 print("单元格B3 = ", ws['B3'].value) # B3
11 print("单元格C5 = ", ws['C5'].value) # C5

```

执行结果

```

===== RESTART: D:\Python\ch19\ch19_5.py =====
单元格A1 = 销售业绩表
单元格A2 = 姓名
单元格B2 = 一月
单元格B3 = 4560
单元格C5 = 6799
>>>

```

上述对于ws[‘栏 行’]而言，除了可以使用value属性取得单元格内容外，也可以使用row、column或coordinate取得单元格相对位置信息。

程序实例ch19_6.py： 列出单元格位置信息。

```

1 # ch19_6.py
2 import openpyxl
3
4 fn = 'sales.xlsx'
5 wb = openpyxl.load_workbook(fn)
6 ws = wb.get_sheet_by_name('2020Q1') # 当前工作表2020Q1
7 print("单元格A1 = ", ws['A1'].column, ws['A1'].row, ws['A1'].coordinate)
8 print("单元格A2 = ", ws['A2'].column, ws['A2'].row, ws['A2'].coordinate)
9 print("单元格B2 = ", ws['B2'].column, ws['B2'].row, ws['B2'].coordinate)
10 print("单元格B3 = ", ws['B3'].column, ws['B3'].row, ws['B3'].coordinate)
11 print("单元格C5 = ", ws['C5'].column, ws['C5'].row, ws['C5'].coordinate)

```

执行结果

```

===== RESTART: D:\Python\ch19\ch19_6.py =====
单元格A1 =  A 1 A1
单元格A2 =  A 2 A2
单元格B2 =  B 2 B2
单元格B3 =  B 3 B3
单元格C5 =  C 5 C5
>>>

```

上述每一行输出3个数据，分别是栏（column）、行（row）和坐标（coordinate）。

19-2-5 取得工作表内容的栏数和行数

对于当前工作表对象（本节实例使用ws当变量）而言，max_column和max_row可以分别传回工作表内容的栏数和行数。

程序实例ch19_7.py： 传回sales.xlsx工作簿2020Q1工作表的栏数和行数。

```
1 # ch19_7.py
2 import openpyxl
3
4 fn = 'sales.xlsx'
5 wb = openpyxl.load_workbook(fn)
6 ws = wb.get_sheet_by_name('2020Q1') # 当前工作表2020Q1
7 print("工作表栏数 = ", ws.max_column)
8 print("工作表行数 = ", ws.max_row)
```

执行结果

```
===== RESTART: D:\Python\ch19\ch19_7.py =====
工作表栏数 = 5
工作表行数 = 12
>>>
```

读者可以将上述执行结果与19-1节的sales.xlsx工作表作比较，就可以知道我们得到了正确的结果了。

19-2-6 取得单元格内容

上述我们使用“ws[‘栏列’].value”取得单元格内容，我们也可以使用cell()方法取得单元格内容，此时其语法格式如下：

`ws.cell(column=N, row=M)` # **N**

是栏编号，**M**

是行编号

程序实例ch19_8.py： 列出第5行的内容。

```
1 # ch19_8.py
2 import openpyxl
3
4 fn = 'sales.xlsx'
5 wb = openpyxl.load_workbook(fn)
6 ws = wb.get_sheet_by_name('2020Q1') # 当前工作表2020Q1
7 for i in range(1, ws.max_column+1): # column做索引增值
8     print(ws.cell(column=i, row=5).value, end=' ') # row=5, 索引不变
9 print() # 跳行打印
10 print(ws['A5'].value) # 打印A5
```

执行结果

```
===== RESTART: D:\Python\ch19\ch19_8.py =====
李连杰 8864 6799 7842 =SUM(B5:D5)
李连杰
>>>
```

从上图可以看到ws.cell(column=1, row=5).value的意义与ws. ['A5'].value意义相同。上述E5单元格内容是公式，如果想要显示值，可以在第5行打开工作簿文件时，增加data_only=True参数。

程序实例ch19_9.py： 重新设计ch19_8.py，以数值显示公式，下列只列出修改部分。

```
5 wb = openpyxl.load_workbook(fn, data_only=True)
```

执行结果

```
===== RESTART: D:\Python\ch19\ch19_9.py =====
李连杰 8864 6799 7842 23505
>>>
```

如果想要取得某一区块单元格空间可以使用双层循环的观念。

程序实例ch19_10.py： 列出某区间的单元格数据，这个程序将列出A4:E6内容。

```
1 # ch19_10.py
2 import openpyxl
3
4 fn = 'sales.xlsx'
5 wb = openpyxl.load_workbook(fn, data_only=True)
6 ws = wb.get_sheet_by_name('2020Q1') # 当前工作表2020Q1
7 for j in range(4,7): # row做索引增值
8     for i in range(1,6): # column做索引增值
9         print("%5s" % ws.cell(column=i, row=j).value, end=' ')
10    print() # 换行输出
```

执行结果

```
===== RESTART: D:\Python\ch19\ch19_10.py =====
魏德圣 3972 4014 3890 11876
李连杰 8864 6799 7842 23505
成祖名 5797 4312 5500 15609
>>>
```

19-2-7 工作表对象ws的rows和columns

当建立工作表对象ws成功后，会自动产生下列数据产生器 (generators)：

ws.rows: 工作表数据产生器

以行方式包裹，每一行用一个Tuple包裹。

ws.columns: 工作表数据产生器

以栏方式包裹，每一栏用一个Tuple包裹。

程序实例ch19_11.py: 列出ws.rows和ws.columns的数据类型。

```
1 # ch19_11.py
2 import openpyxl
3
4 fn = 'sales.xlsx'
5 wb = openpyxl.load_workbook(fn, data_only=True)
6 ws = wb.get_sheet_by_name('2020Q1') # 当前工作表2020Q1
7 print(type(ws.rows)) # 获得ws.rows数据类型
8 print(type(ws.columns)) # 获得ws.columns数据类型
```

执行结果

```
===== RESTART: D:\Python\ch19\ch19_11.py =====
<class 'generator'>
<class 'generator'>
>>>
```

由于ws.rows和ws.columns是数据产生器，若是想取得它的内容须先将它们转成列表(list)，然后就可以用索引方式取得。

程序实例ch19_12.py: 列出特定行 与 栏 的信息。需留意由于数据转成了列表，所以索引值是从0开始。本程序会列出A栏 数据和李安这行资料。

```
1 # ch19_12.py
2 import openpyxl
3
4 fn = 'sales.xlsx'
5 wb = openpyxl.load_workbook(fn, data_only=True)
6 ws = wb.get_sheet_by_name('2020Q1') # 当前工作表2020Q1
7 for cell in list(ws.columns)[0]: # A栏
8     print(cell.value)
9 for cell in list(ws.rows)[2]: # 索引是2
10     print(cell.value, end=' ')
```

执行结果

```
===== RESTART: D:\Python\ch19\ch19_12.py =====
销售业绩表
姓名
李安
魏德生
李达
成吉思汗
张曼玉
田中干绘
范逸臣
周华健
蔡琴
张学友
李安 4560 5152 6014 15726
>>>
```

对于数据产生器而言，我们可以使用逐行方式获得全部的工作表内容。

程序实例ch19_13.py： 使用逐行方式获得工作表全部的内容。

```
1 # ch19_13.py
2 import openpyxl
3
4 fn = 'sales.xlsx'
5 wb = openpyxl.load_workbook(fn, data_only=True)
6 ws = wb.get_sheet_by_name('2020Q1') # 当前工作表2020Q1
7 for row in ws.rows:
8     for cell in row:
9         print(cell.value, end=' ')
10    print()
```

执行结果

```
===== RESTART: D:\Python\ch19\ch19_13.py =====
销售业绩表 None None None None
姓名 一月 二月 三月 总计
李安 4560 5152 6014 15726
魏德生 3972 4014 3890 11876
李达 3864 6798 7842 23505
成吉思汗 5797 4312 5500 15609
张曼玉 4234 8045 7098 19377
田中干绘 7790 5435 6680 19914
范逸臣 8152 7152 7034 22338
周华健 9040 8048 5098 22186
蔡琴 5566 4890 6690 17146
张学友 7152 6622 7452 21226
>>>
```

读者可能会想是否可以使用逐栏方式获得全部的工作表内容，答案是肯定的。

程序实例ch19_14.py： 使用逐栏方式获得全部的工作表内容。

```
1 # ch19_14.py
2 import openpyxl
3
4 fn = 'sales.xlsx'
5 wb = openpyxl.load_workbook(fn, data_only=True)
6 ws = wb.get_sheet_by_name('2020Q1') # 当前工作表2020Q1
7 for col in ws.columns:
8     for cell in col:
9         print(cell.value, end=' ')
10    print()
```

执行结果

```

===== RESTART: D:\Python\ch19\ch19_14.py =====
销售业绩表 姓名 李安 魏德圣 李连杰 成祖名 张曼玉 田中千绘 范逸臣 周华健 蔡琴 张学友
None 一月 4560 3972 8864 5797 4234 7799 8152 9040 5566 7152
None 二月 5152 4014 6799 4312 8045 5435 7152 8048 4890 6622
None 三月 6014 3890 7842 5500 7098 6680 7034 5098 6690 7452
None 总计 15726 11876 23505 15609 19377 19914 22338 22186 17146 21226
>>>

```

19-2-8 用整数取代域名

在Excel中栏名称 是A、B、…、Z、AA、AB、AC、……例如，1代表A、2代表B、26代表Z、27代表AA、28代表AB。如果工作表的栏数很多，很明显我们无法清楚了解到底索引是多少，例如，BC是多少。为了解决这方面的问题，下面将介绍2个转换方法：

`get_column_letter(数值)` # 将数值

转成字母

`column_index_from_string(字母)` # 将字母

转成数值

上述方法存在于openpyxl.utils模块内，所以程序前面要加上下列指令。

```

from openpyxl.utils import get_column_letter,
column_index_from_string

```

程序实例ch19_15.py： 将字段的字母转成数值与将数值转成字母。

```

1 # ch19_15.py
2 import openpyxl
3 from openpyxl.utils import get_column_letter, column_index_from_string
4
5 fn = 'sales.xlsx'
6 wb = openpyxl.load_workbook(fn, data_only=True)
7 ws = wb.get_sheet_by_name('2020Q1') # 当前工作表2020Q1
8 print("栏数=", get_column_letter(ws.max_column))
9 print("3 = ", get_column_letter(3))
10 print("27 = ", get_column_letter(27))
11 print("100 = ", get_column_letter(100))
12 print("800 = ", get_column_letter(800))
13
14 print("A = ", column_index_from_string('A'))
15 print("E = ", column_index_from_string('E'))
16 print("AA = ", column_index_from_string('AA'))
17 print("AZ = ", column_index_from_string('AZ'))
18 print("AAA = ", column_index_from_string('AAA'))

```

执行结果

```
===== RESTART: D:\Python\ch19\ch19_15.py =====
行数= E
3 = C
27 = AA
100 = CV
800 = ADT
A = 1
E = 5
AA = 27
AZ = 52
AAA = 703
>>>
```

19-2-9 切片

这是使用切片的概念读取某区间数据，例如，读取A3:E6数据可用下列方法：

```
for row in ws[ 'A3' : 'E6' ]:          # 逐行读取
    for cell in row:                  # 读取特定行的每一单元格
        print(cell.value)
```

程序实例ch19_16.py： 采用切片观念读取单元格内容。

```
1 # ch19_16.py
2 import openpyxl
3 from openpyxl.utils import get_column_letter, column_index_from_string
4
5 fn = 'sales.xlsx'
6 wb = openpyxl.load_workbook(fn, data_only=True)
7 ws = wb.get_sheet_by_name('2020Q1') # 当前工作表2020Q1
8 for row in ws[ 'A3' : 'E6' ]:
9     for cell in row:
10         print(cell.value, end=' ')
11     print()
```

执行结果

```
===== RESTART: D:\Python\ch19\ch19_16.py =====
李安 4560 5152 6014 15726
魏德圣 3972 4014 3890 11876
李连杰 8864 6799 7842 23505
成祖名 5797 4312 5500 15609
>>>
```

19-3 写入Excel文件

openpyxl模块也有提供方法可让我们写入Excel文件。

19-3-1 建立Excel文件

`openpyxl.Workbook()`可以建立空白的工作簿，也可想成Excel文件。预设所建立的文件是可擦写，如果想要设为只写 模式，可以加上`write_only=True` 参数。

19-3-2 存储Excel文件

`save()`方法可以存储Excel文件，这个方法需由Excel文件对象启动，先前我们是使用`wb(workbook)`当作文件对象的变量，所以使用语法如下：

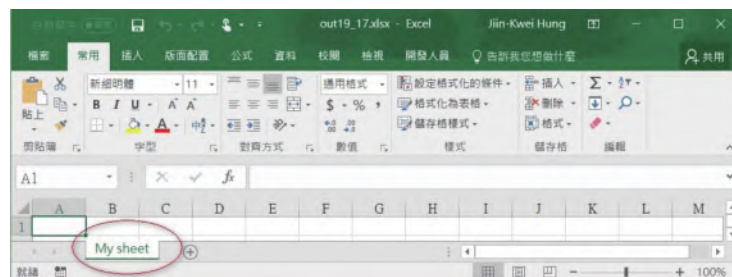
```
wb.save(文件名) # 可以存储指定文件名的文件
```

程序实例ch19_17.py： 建立一个空白的Excel文件，列出预设的工作表名称，然后将预设工作表名称改为“My sheet”，最后用out19_17.xlsx名称储存此文件。

```
1 # ch19_17.py
2 import openpyxl
3
4 wb = openpyxl.Workbook()          # 建立空白的工作簿
5 ws = wb.get_active_sheet()        # 获得当前工作表
6 print("目前工作表名称 = ", ws.title) # 打印当前工作表
7 ws.title = 'My sheet'             # 更改当前工作表名称
8 print("新工作表名称 = ", ws.title) # 打印新的当前工作表
9 wb.save('out19_17.xlsx')          # 将工作簿储存
```

执行结果 下列是执行结果与out19_17.xlsx的结果。

```
===== RESTART: D:\Python\ch19\ch19_17.py =====
目前工作表名称 = Sheet
新工作表名称 = My sheet
>>>
```



19-3-3 复制Excel文件

我们可以用打开文件，然后新名称存储文件方式达到复制Excel文件的效果。

程序实例ch19_18.py： 将sales.xlsx复制一份至out19_18.xlsx。

```
1 # ch19_18.py
2 import openpyxl
3
4 fn = 'sales.xlsx'
5 wb = openpyxl.load_workbook(fn)      # 开启sales.xlsx活页簿
6 wb.save('out19_18.xlsx')           # 将活页簿储存至out19_18.xlsx
```

执行结果 可以在当前工作文件夹看到所建的out19_18.xlsx文件，文件内容与sales.xlsx相同。

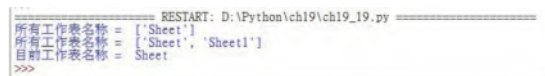
19-3-4 建立工作表

create_sheet()可以在工作簿内建立新的工作表。

程序实例ch19_19.py： 建立空白工作簿，然后打印所有工作表。接着新增工作表，再度打印所有工作表，最后将这个工作簿储存至out19_19.xlsx。

```
1 # ch19_19.py
2 import openpyxl
3
4 wb = openpyxl.Workbook()      # 建立空白的工作簿
5 print("所有工作表名称 = ", wb.get_sheet_names())  # 打印所有工作表
6 wb.create_sheet()            # 建立新工作表
7 print("所有工作表名称 = ", wb.get_sheet_names())  # 打印所有工作表
8 ws = wb.get_active_sheet()   # 取得当前工作表
9 print("当前工作表名称 = ", ws.title)              # 打印当前工作表
10 wb.save('out19_19.xlsx')     # 将工作簿储存
```

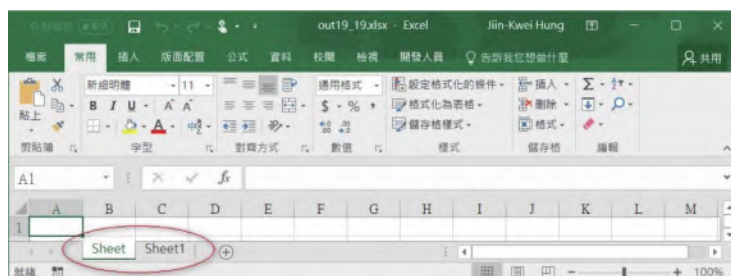
执行结果 同时在文件夹可以看到拥有2个工作表的out19_19.xlsx文件。



```
RESTART: D:\Python\ch19\ch19_19.py
所有工作表名称 = ['Sheet']
所有工作表名称 = ['Sheet', 'Sheet1']
当前工作表名称 = Sheet
```

在建立工作表时预设工作表名称是“SheetN”，N是数字编号以递增方式显示，另外新建立的工作表是放在工作表列的最右边，我们可以在create_sheet()内增加参数title和index设定新工作表的

名称 和位置。工作表的位置是从0开始，所以如果index=0，表示在最左边。



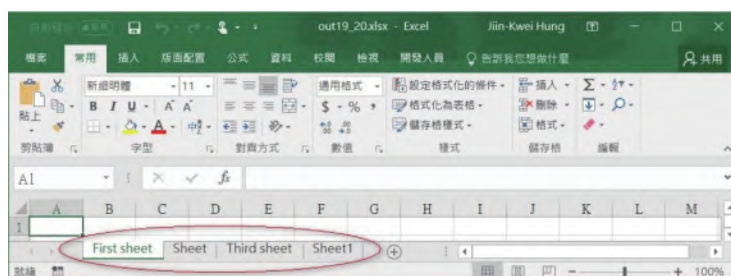
程序实例ch19_20.py: 扩充ch19_19.py，增加使用title和index关键词。

```
1 # ch19_20.py
2 import openpyxl
3
4 wb = openpyxl.Workbook()
5 print("所有工作表名称 = ", wb.get_sheet_names())
6 wb.create_sheet()
7 print("所有工作表名称 = ", wb.get_sheet_names())
8 wb.create_sheet(index=0, title='First sheet')
9 print("所有工作表名称 = ", wb.get_sheet_names())
10 wb.create_sheet(index=2, title='Third sheet')
11 print("所有工作表名称 = ", wb.get_sheet_names())
12 wb.save('out19_20.xlsx')
```

建立空白的工作簿
打印所有工作表
建立新工作表
打印所有工作表
第一个工作表
打印所有工作表
第三个工作表
打印所有工作表
将工作簿储存

执行结果

```
===== RESTART: D:\Python\ch19\ch19_20.py =====
所有工作表名称 = ['Sheet1']
所有工作表名称 = ['Sheet', 'Sheet1']
所有工作表名称 = ['First sheet', 'Sheet', 'Sheet1']
所有工作表名称 = ['First sheet', 'Sheet', 'Third sheet', 'Sheet1']
>>>
```



19-3-5 删除工作表

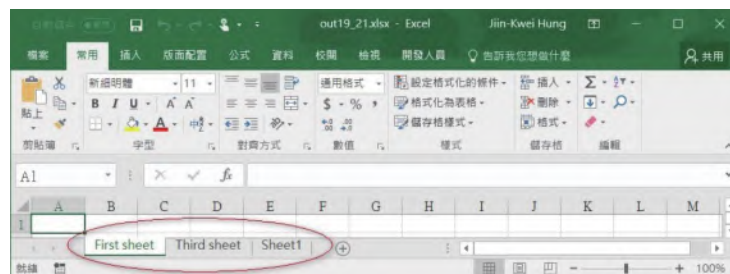
删除工作表可以使用`remove_sheet()`方法，在使用时并不是直接将工作表名称当参数，必须使用工作簿对象`wb`调用`get_sheet_by_name()`当作参数。

程序实例ch19_21.py： 重新设计ch19_20.py，主要是增加删除Sheet工作表。

```
1 # ch19_21.py
2 import openpyxl
3
4 wb = openpyxl.Workbook()          # 建立空白的工作簿
5 print("所有工作表名称 = ", wb.get_sheet_names())  # 打印所有工作表
6 wb.create_sheet()                 # 建立新工作表
7 print("所有工作表名称 = ", wb.get_sheet_names())  # 打印所有工作表
8 wb.create_sheet(index=0, title='First sheet')      # 第一个工作表
9 print("所有工作表名称 = ", wb.get_sheet_names())  # 打印所有工作表
10 wb.create_sheet(index=2, title='Third sheet')      # 第三个工作表
11 print("所有工作表名称 = ", wb.get_sheet_names())  # 打印所有工作表
12 wb.remove_sheet(wb.get_sheet_by_name('Sheet'))    # 删除Sheet工作表
13 print("所有工作表名称 = ", wb.get_sheet_names())  # 打印所有工作表
14 wb.save('out19_21.xlsx')           # 将工作簿储存
```

执行结果 由下图可以看到Sheet工作表已经被删除了。

```
===== RESTART: D:\Python\ch19\ch19_21.py =====
所有工作表名称 = ['Sheet']
所有工作表名称 = ['Sheet', 'Sheet1']
所有工作表名称 = ['First sheet', 'Sheet', 'Sheet1']
所有工作表名称 = ['First sheet', 'Sheet', 'Third sheet', 'Sheet1']
所有工作表名称 = ['First sheet', 'Third sheet', 'Sheet1']
>>>
```



19-3-6 写入单元格

我们已经学会了从特定单元格读取数据，如果想要写入资料，只要设定该单元格的值就可以了。

程序实例ch19_22.py： 将资料写入单元格。

```

1 # ch19_22.py
2 import openpyxl
3
4 wb = openpyxl.Workbook()          # 建立空白的工作簿
5 ws = wb.get_active_sheet()        # 获得当前工作表
6 ws['A1'] = 'Python'
7 ws['A2'] = 100
8 wb.save('out19_22.xlsx')          # 将工作簿储存

```

执行结果 打开out19_22.xlsx可以看到下列结果。

	A	B	C
1	Python		
2	100		

输入数据的格式与在Excel窗口是相同的，字符串靠左对齐，数值数据靠右对齐。

19-3-7 将列表数据写进单元格

我们可以使用append()方法将列表资料写入单元格，append这个名词有附加 的意义，如果当前工作表没有资料，append()可将数据从第一行(row)开始写入，如果当前工作表已经有数据，可将数据从已有数据的下一行开始写入。

程序实例ch19_23.py: 在空白工作表使用append()输入列表数据。

```

1 # ch19_23.py
2 import openpyxl
3
4 wb = openpyxl.Workbook()          # 建立空白的工作簿
5 ws = wb.get_active_sheet()        # 获得当前工作表
6 row1 = ['数学', '物理', '化学']   # 定义列表数据
7 ws.append(row1)                   # 写入列表
8 row2 = [98, 82, 89]               # 定义列表数据
9 ws.append(row2)                   # 写入列表
10 wb.save('out19_23.xlsx')          # 将工作簿储存

```

执行结果 打开out19_23.xlsx可以看到下列结果。

	A	B	C	D
1	数学	物理	化学	
2	98	82	89	

上述我们成功地一次输入一个列表数据，如果列表数据的元素也是列表，我们可以使用循环方式输入内含列表元素的列表。

程序实例ch19_24.py： 在已有数据的工作表，使用append()输入内含列表元素的列表。

```
1 # ch19_24.py
2 import openpyxl
3
4 wb = openpyxl.Workbook()          # 建立空白的工作簿
5 ws = wb.get_active_sheet()        # 获得当前工作表
6 ws['A1'] = '明志科技大学'         # 定义列表数据
7 rows = [
8     ['数学', '物理', '化学'],
9     [98, 82, 89],
10    [79, 88, 90],
11    [80, 78, 91]]
12 for row in rows:                  # 写入列表
13     ws.append(row)                # 将工作簿储存
14 wb.save('out19_24.xlsx')
```

执行结果 打开out19_24.xlsx可以看到下列结果。

	A	B	C	D
1	明志科技大学			
2	数学	物理	化学	
3	98	82	89	
4	79	88	90	
5	80	78	91	

19-4 设定单元格的字体

若是想要设定单元格的字体，建议可以导入Font()方法，如下所示：

```
from openpyxl.style import Font
```

未来就可以直接使用Font()，代替每次需写入openpyxl.styles.Font()长串的方法名称。

19-4-1 Font()

Font对象主要功能是执行字体相关的设定，可以使用Font()方法设定此对象，有了Font对象后就可以将它应用在单元格。这个方法常见的参数如下：

关键词	数据类型	说明
bold	Boolean	粗体，bold=True 可设定粗体。
italic	Boolean	斜体，italic=True 可设定斜体。
strike	Boolean	删除线，strike=True 可设定删除线。
name	字符串	字体名称，例如：Arial, Old English Text MT。
size	整数	字号
color	整数	color= 'FFFFFF'

程序实例ch19_25.py： 设计3种字体应用，分别应用在A1、A2和A3单元格。

fontTitle1： 字体名称是‘微软正黑体’、字号是24。

fontTitle2： 字体名称是‘Old English Text MT’、字号是24、粗体。

fontTitle3： 字号是24、粗体、斜体。

```

1 # ch19_25.py
2 import openpyxl
3 from openpyxl.styles import Font
4
5 wb = openpyxl.Workbook()           # 建立空白的工作簿
6 ws = wb.get_active_sheet()        # 获得当前工作表
7 fontTitle1 = Font(name='微软正黑体', size=24)
8 ws['A1'].font = fontTitle1
9 ws['A1'] = '明志科技大学'
10 fontTitle2 = Font(name='Old English Text MT', size=24, bold=True)
11 ws['A2'].font = fontTitle2
12 ws['A2'] = 'Ming-Chi Institute of Technology'
13 fontTitle3 = Font(size=24, bold=True, italic=True)
14 ws['A3'].font = fontTitle3
15 ws['A3'] = 'Ming-Chi Institute of Technology'
16 wb.save('out19_25.xlsx')          # 将工作簿存储

```

执行结果 打开out19_25.xlsx可以看到下列结果。

	A	B	C	D	E	F	G	H	I
1	明志科技大学								
2	Ming-Chi Institute of Technology								
3	<i>Ming-Chi Institute of Technology</i>								

19-4-2 字体色彩的设定

其实所有颜色可以使用3个原色red(红色)、green(绿色)和blue(蓝色)，每个颜色数值在0—255间组成。Font()方法内的参数

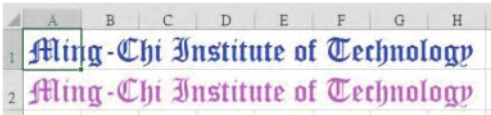
color的值“FFFFFF ”，分别代表Red、Green和Blue ，下列是常见的256种颜色组合。

本书附录D有完整的色彩说明。

程序实例ch19_26. py： 在Excel文件内建立各种字体颜色。

```
1 # ch19_26.py
2 import openpyxl
3 from openpyxl.styles import Font
4
5 wb = openpyxl.Workbook()           # 建立空白的工作簿
6 ws = wb.get_active_sheet()         # 获得当前工作表
7 fontTitle1 = Font(name='Old English Text MT', size=24, color='0000FF')
8 ws['A1'].font = fontTitle1
9 ws['A1'] = 'Ming-Chi Institute of Technology'
10 fontTitle2 = Font(name='Old English Text MT', size=24, color='FF66FF')
11 ws['A2'].font = fontTitle2
12 ws['A2'] = 'Ming-Chi Institute of Technology'
13 wb.save('out19_26.xlsx')           # 将工作簿存储
```

执行结果 打开out19_26. xlsx可以看到下列结果。



19-5 数学公式的使用

常用的数学公式如下：

公式	说明
SUM()	加总，例如：SUM(B1:B3)
AVERAGE()	平均，例如：SUM(B1:B3)
MAX()	最大值，例如：MAX(B1:B3)
MIN()	最小值，例如：MIN(B1:B3)

程序实例ch19_27. py： 计算B1:B3单元格区间的加总 、平均 、最高分 、最低分 。

```

1 # ch19_27.py
2 import openpyxl
3
4 wb = openpyxl.Workbook()          # 建立空白的工作簿
5 ws = wb.get_active_sheet()        # 获得当前工作表
6 ws['A1'] = 'Peter'                # 设定名字Peter
7 ws['B1'] = 98                      #
8 ws['A2'] = 'Janet'                 # 设定名字Janet
9 ws['B2'] = 79                      #
10 ws['A3'] = 'Nelson'                # 设定名字Nelson
11 ws['B3'] = 81                      #
12 ws['A4'] = '总分'                  #
13 ws['B4'] = '=SUM(B1:B3)'           # 计算总分
14 ws['A5'] = '平均'                  #
15 ws['B5'] = '=AVERAGE(B1:B3)'      # 计算平均
16 ws['A6'] = '最高分'               #
17 ws['B6'] = '=MAX(B1:B3)'          # 计算最高分
18 ws['A7'] = '最低分'               #
19 ws['B7'] = '=MIN(B1:B3)'          # 计算最低分
20 wb.save('out19_27.xlsx')          # 将工作簿存储

```

执行结果 打开out19_27.xlsx可以看到下列结果。

	A	B	C
1	Peter	98	
2	Janet	79	
3	Nelson	81	
4	总分	258	
5	平均	86	
6	最高分	98	
7	最低分	79	

19-6 设定单元格的高度和宽度

单元格预设的高度是12.75pt，72pt等于1英寸。可以使用column_dimensions属性设定行高。单元格默认的宽度是8.43个英文字符宽度，可以使用row_dimensions设定单元格的宽度。如果将高度或宽度设为0，则具有隐藏单元格效果。

程序实例ch19_28.py： 设定第一行(row)高度和第B栏(column)宽度。

```

1 # ch19_28.py
2 import openpyxl
3 from openpyxl.styles import Alignment
4
5 wb = openpyxl.Workbook()          # 建立空白的工作簿
6 ws = wb.get_active_sheet()        # 获得当前工作表
7 ws['A1'] = '深右数位'             #
8 ws['B2'] = 'Deep Stone'            #
9 ws.row_dimensions[1].height = 40   # 高度是40pt
10 ws.column_dimensions['B'].width = 20 # 宽度是20个英文字符宽
11 wb.save('out19_28.xlsx')          # 将工作簿存储

```

执行结果 打开out19_28.xlsx可以看到下列结果。

	A	B	C
1	深石软件		
2		Deep Stone	
3			

19-7 单元格对齐方式

可以使用Alignment()方法，所以须在程序前方导入下列模块。

```
from openpyxl.styles import Alignment
```

Alignment()方法内可以有如下2个参数：

horizontal：可以设定left（靠左）、center（居中）、right（靠右）对齐。**vertical**：可以设定top（靠上）、center（居中）、bottom（靠下）对齐。整个单元格设定的完整公式如下：

```
ws['A1'].alignment = Alignment() # 详细应用可参考下列实例
```

程序实例ch19_29.py：为了让读者更加清楚整个城市的意义，特别增加第一行高度和第2栏宽度。‘台科大’字符串是靠左靠上 对齐、‘明志科大’字符串是左右上下居中 对齐、‘北科大’字符串是靠右靠下 对齐。

```
1 # ch19_29.py
2 import openpyxl
3 from openpyxl.styles import Alignment
4
5 wb = openpyxl.Workbook() # 建立空白的工作簿
6 ws = wb.get_active_sheet() # 获得当前工作表
7 ws['A1'] = '台科大'
8 ws['B1'] = '明志科大'
9 ws['C1'] = '北科大'
10 ws.row_dimensions[1].height = 40 # 高度是40pt
11 ws.column_dimensions['B'].width = 20 # 宽度是20个英文字符宽
12 ws['A1'].alignment = Alignment(horizontal='left', vertical='top')
13 ws['B1'].alignment = Alignment(horizontal='center', vertical='center')
14 ws['C1'].alignment = Alignment(horizontal='right', vertical='bottom')
15 wb.save('out19_29.xlsx') # 将工作簿存储
```

执行结果 打开out19_29.xlsx可以看到下列结果。

	A	B	C	D
1	台科大	明志科大	北科大	
2				

19-8 合并与取消合并单元格

19-8-1 合并单元格

可以使用`merge_cells()`合并单元格，可以合并同一行(row)、同一栏(column)或一个区间的单元格，细节可以参考下列实例。

程序实例ch19_30.py: A1:E1单元格是合并的单元格，同时水平居中对齐，A2单元格未合并，读者可以比较2个字符串的执行结果。A3:A5是垂直合并的单元格，字符串“明志科大”是垂直居中对齐。C3:E4是合并一个单元格区间，字符串“机械工程系”是水平和垂直居中。

```
1 # ch19_30.py
2 import openpyxl
3 from openpyxl.styles import Alignment
4
5 wb = openpyxl.Workbook()           # 建立空白的工作簿
6 ws = wb.get_active_sheet()         # 获得当前工作表
7 ws['A1'] = 'Ming-Chi Institute of Technology'
8 ws['A2'] = 'Ming-Chi Institute of Technology'
9 ws['A3'] = '明志科大'
10 ws['C3'] = '机械工程系'
11 ws.merge_cells('A1:E1')           # 合并A1:E1单元格
12 ws.merge_cells('A3:A5')           # 合并A1:E1单元格
13 ws.merge_cells('C3:E4')           # 合并C3:E4单元格
14 ws['A1'].alignment = Alignment(horizontal='center') # A1单元格水平居中
15 ws['A3'].alignment = Alignment(vertical='center')   # A3单元格垂直居中
16 ws['C3'].alignment = Alignment(horizontal='center', vertical='center')
17 wb.save('out19_30.xlsx')           # 将工作簿存储
```

执行结果 执行结果打开out19_30.xlsx可以看到下列结果。

	A	B	C	D	E	F
1	Ming-Chi Institute of Technology					
2	Ming-Chi Institute of Technology					
3			机械工程系			
4	明志科大					
5						
6						

19-8-2 取消合并单元格

可以使用`unmerge_cells()`取消合并单元格。

程序实例ch19_31.py: 打开ch19_30.py所建立的Excel文件，然后取消合并同时将执行结果存入out19_31.py。

```

1 # ch19_31.py
2 import openpyxl
3
4 wb = openpyxl.load_workbook('out19_30.xlsx') # 开启工作簿
5 ws = wb.active # 获得当前工作表
6 ws.unmerge_cells('A1:E1') # 取消合并A1:E1单元格
7 ws.unmerge_cells('A3:A5') # 取消合并A1:E1单元格
8 ws.unmerge_cells('C3:E4') # 取消合并C3:E4单元格
9 wb.save('out19_31.xlsx') # 将工作簿存储

```

执行结果 打开out19_31.xlsx可以看到下列结果。

	A	B	C	D	E	F	G	H
1	Institute of Technology							
2	Ming-Chi Institute of Technology							
3	明志科大	机械工程系						
4								

上述程序第5行“ws=wb.active”，笔者使用另一个获得当前工作表的用法，观念与ch19_30.py的第6行用法意义是一样的。

19-9 建立图表

Python可以建立的图表有许多，所有Excel可以建立的图表皆可使用Python建立，为了建立图表可以更方便地导入图表模块，程序需导入下列图表方法。BarChart（柱形图）、BarChart3D（3D柱形图）、PieChart（饼图）、PieChart3D（3D饼图）、BubbleChart（泡泡图）、AreaChart（分区图）、AreaChart3D（3D分区图）、LineChart（线段图）、LineChart3D（3D线段图）、RadarChart（雷达图）、StockChart（股票图）。上述英文名称就是建立图表的方法，导入方法如下：

```

from openpyxl.chart import BarChart, Reference # 以导入
BarChart

```

为例

另外需导入Reference方法，这个方法主要是供我们将建立图表所需的工作表数据 或是卷标名称（有时也可称轴的卷标）数据导入所建的图表对象内。

本节将以4个最常用的图表，用程序实例做说明。

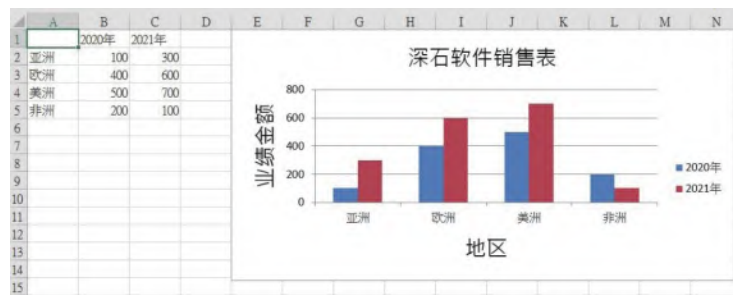
19-9-1 柱形图

这是最常见的图表应用，主要是显示多组数据于一段时间的变化，从此类型也可以了解各组资料间比较的情形，应用时通常数值数据是在纵轴(y轴)，而标记是在横轴(x轴)。

程序实例ch19_32.py： 建立深石软件2020—2021年销售报表。

```
1 # ch19_32.py
2 import openpyxl
3 from openpyxl.chart import BarChart, Reference
4
5 wb = openpyxl.Workbook() # 开启工作簿
6 ws = wb.active # 获得当前工作表
7 rows = [
8     ['', '2020年', '2021年'],
9     ['亚洲', 100, 300],
10    ['欧洲', 400, 600],
11    ['美洲', 500, 700],
12    ['非洲', 200, 100]]
13 for row in rows:
14     ws.append(row)
15
16 chart = BarChart() # 直方图
17 chart.title = '深石软件销售表' # 图表标题
18 chart.y_axis.title = '业绩金额' # y轴标题
19 chart.x_axis.title = '地区' # x轴标题
20
21 data = Reference(ws, min_col=2, max_col=3, min_row=1, max_row=5) # 图表数据
22 chart.add_data(data, titles_from_data=True) # 建立图表
23 xtitle = Reference(ws, min_col=1, min_row=2, max_row=5) # x轴标记名称
24 chart.set_categories(xtitle) # 设定x轴标记名称(亚洲欧洲美洲非洲)
25 ws.add_chart(chart, 'E1') # 放置图表位置
26 wb.save('out19_32.xlsx')
```

执行结果 打开out19_32.xlsx可以看到下列结果。



上述16行是建立图表类型，未来可以由此决定所要建立图表的类型。17—19行分别是建立图表标题、y轴和x轴的标题。21行是Reference()，主要是建立图表数据对象，在此记录建立图表的数据

范围，有ws、min_col、max_col、min_row、max_row参数，ws是代表图表数据源工作表，剩余的数据是列出数据是由哪些单元格区间组成，以此例可知是由B 1:C5所组成。22行是将21行的图表数据对象放入16行所建的图表对象内。

23行是建立x轴标记名称的数据对象，有ws、min_col、max_col、min_row、max_row参数，ws是代表图表数据源工作表，剩余的数据是列出数据是由哪些单元格区间组成，此例没有max_col参数，表示min_col与max_col值相同，以此例可知标记名称数据是由A2:A5所组成。24行set_categories()是将标记名称的数据对象填入整个图表对象Chart。25行是设定图表在工作表中的位置，一个程序可以建立许多图表，只要使用25行观念将图表放在不同位置即可。

19-9-2 3D柱形图

学会了建立柱形图后，若想要建立3D柱形图表就很简单，只要导入BarChart3D，可参考程序第3行，在16行将图表对象改成BarChart()，剩余的程序代码就完全相同了。为了方便读者阅读，笔者在程序代码内也标记了不一样的地方。

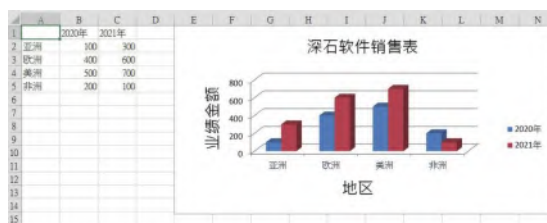
程序实例 ch19_33.py： 以 3D 柱形图 (BarChart3D) 重新设计 ch19_32.py。

```

1 # ch19_33.py
2 import openpyxl
3 from openpyxl.chart import BarChart3D, Reference
4
5 wb = openpyxl.Workbook() # 开启工作簿
6 ws = wb.active # 获得当前工作表
7 rows = [
8     ['', '2020年', '2021年'],
9     ['亚洲', 100, 300],
10    ['欧洲', 400, 600],
11    ['美洲', 500, 700],
12    ['非洲', 200, 100]]
13 for row in rows:
14     ws.append(row)
15
16 chart = BarChart3D() # 3D直方图
17 chart.title = '深石软件销售表' # 图表标题
18 chart.y_axis.title = '业绩金额' # y轴标题
19 chart.x_axis.title = '地区' # x轴标题
20
21 data = Reference(ws, min_col= 2, max_col=3, min_row=1, max_row=5) # 图表数据
22 chart.add_data(data, titles_from_data=True) # 建立图表
23 xtitle = Reference(ws, min_col=1, min_row= 2, max_row=5) # x轴标记名称
24 chart.set_categories(xtitle) # 设定x轴标记名称(亚洲欧洲美洲非洲)
25 ws.add_chart(chart, 'E1') # 放置图表位置
26 wb.save('out19_33.xlsx')

```

执行结果 打开out19_33. xlsx可以看到下列结果。



19-9-3 饼图

饼图(PieChart)只适合一个数据系列，主要是供了解单笔数据相对于整体数据的关系比。设计程序需先导入PieChart，可参考第3行。

程序实例ch19_34. py： 将深石员工旅游意向调查表处理成饼图。

```

1 # ch19_34.py
2 import openpyxl
3 from openpyxl.chart import PieChart, Reference
4
5 wb = openpyxl.Workbook() # 开启工作簿
6 ws = wb.active # 获得当前工作表
7 rows = [
8     ['地区', '人次'],
9     ['上海', 300],
10    ['东京', 600],
11    ['香港', 700],
12    ['新加坡', 400]]
13 for row in rows:
14     ws.append(row)
15
16 chart = PieChart() # 直方图
17 chart.title = '深石员工旅游意向调查表' # 图表标题
18
19 data = Reference(ws, min_col= 2, min_row=1, max_row=5) # 图表数据
20 chart.add_data(data, titles_from_data=True) # 建立图表
21 labels = Reference(ws, min_col=1, min_row= 2, max_row=5) # 标签名称
22 chart.set_categories(labels) # 设定标签名称
23 ws.add_chart(chart, 'E1') # 放置图表位置
24 wb.save('out19_34.xlsx')

```

执行结果 打开out19_34.xlsx可以看到下列结果。



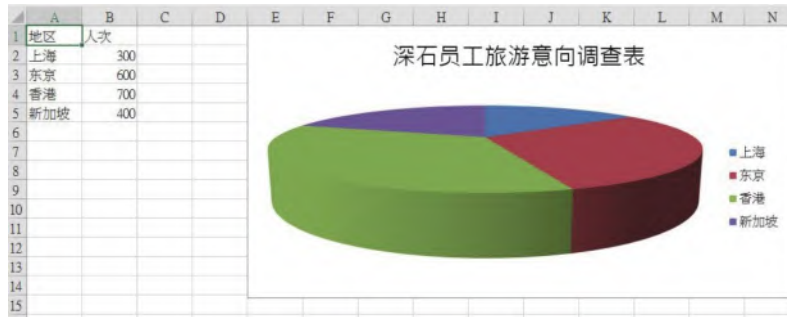
19-9-4 3D饼图

学会了建立饼图后，若想要建立3D饼图表就很简单，只要导入PieChart3D，可参考程序第3行，在16行将图表对象改成PieChart3D()，剩余的程序代码就完全相同了。为了方便读者阅读，笔者在程序代码内也标记不一样的地方。

程序实例 ch19_35.py： 以 3D 饼图 (PieChart3D) 重新设计 ch19_34.py。

```
1 # ch19_35.py
2 import openpyxl
3 from openpyxl.chart import PieChart3D, Reference
4
5 wb = openpyxl.Workbook() # 开启工作簿
6 ws = wb.active # 获得当前工作表
7 rows = [
8     ['地区', '人次'],
9     ['上海', 300],
10    ['东京', 600],
11    ['香港', 700],
12    ['新加坡', 400]]
13 for row in rows:
14     ws.append(row)
15
16 chart = PieChart3D() # 3D饼图
17 chart.title = '深石员工旅游意向调查表'
18
19 data = Reference(ws, min_col=2, min_row=1, max_row=5) # 图表数据
20 chart.add_data(data, titles_from_data=True) # 建立图表
21 labels = Reference(ws, min_col=1, min_row=2, max_row=5) # 标签名称
22 chart.set_categories(labels) # 设定标签名称
23 ws.add_chart(chart, 'E1') # 放置图表位置
24 wb.save('out19_35.xlsx')
```

执行结果 打开out19_35.xlsx可以看到下列结果。



习题

1. 请将下列列表数据写入工作表，工作表名称是mywork，其中A1单元格放置自己的**母校名称**。

Rows = [

[‘ ’, ‘国文’, ‘英文’, ‘数学’, ‘物理’, ‘化学’],

[‘张三’, 89, 90, 92, 78, 75],

[‘李四’, 90, 67, 78, 58, 94],

[‘洪五’, 77, 88, 99, 90, 69],

[‘陈六’, 98, 56, 77, 88, 91]]

2. 将习题1自己的母校用英文书写，同时改为Old English Text MT字体。
3. 请使用习题1的工作表设计各科最高分。
4. 请使用习题1的工作表加上总分功能，同时列出第一名。
5. 请读取sales.xlsx文件使用2020Q1工作表，同时计算每一个月份的销售总计，和所有人前三个月的销售总计，并将结果放在B13:E13单元格。

6. 请使用sales.xlsx文件，建立一个新的工作表2020Total，然后将2020Q1、2020Q2、2020Q3各个月份销售数据复制至2020Total工作表，然后做总计，这个2020Total工作表字段最后结果如下：

	A	B	C	D	E	F	G	H	I	J	K
1	銷售業績表										
2	姓名	一月	二月	三月	四月	五月	六月	七月	八月	九月	總計
3	李安	4560	5152	6014	4560	2152	9014	4560	5152	6014	47178
4	魏德聖	3972	4014	3890	3972	4014	3890	3972	2014	3890	33628
5	李連杰	8864	6799	7842	8864	6799	7842	8864	6799	9842	72515
6	成祖名	5797	4312	5500	5797	4312	5500	5797	4312	5500	46827
7	張曼玉	4234	8045	7098	4234	8045	7098	4234	8045	7098	58131
8	田中千繪	7799	5435	6680	7799	5435	6680	7799	5435	6680	59742
9	范逸臣	8152	7152	7034	8152	7152	7034	8152	7152	7034	67014
10	周華健	9040	8048	5098	9040	8048	5098	9840	8048	5098	67358
11	蔡琴	5566	4890	6690	5566	4890	6690	5566	4890	6690	51438
12	張學友	7152	6622	7452	2152	6622	7452	7152	6622	7452	58678
		2020Q1	2020Q2	2020Q3	2020Total						

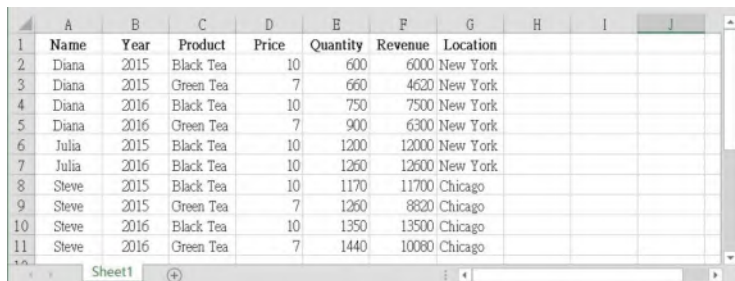
第20章 使用Python处理CSV文件

CSV是一个缩写，它的英文全名是C omma-S eparated V alues，由字面意义可以解说是逗号分隔值，当然逗号是主要数据字段间的分隔值，不过目前也有非逗号的分隔值。这是一个纯文本格式的文件，没有图片、不用考虑字体、大小、颜色等。

简单地说，CSV数据是指同一行（row）的资料彼此用逗号（或其他符号）隔开，同时每一行数据数据是一笔(record)数据，几乎所有电子表格与数据库文件均支持这个文件格式。

20-1 建立一个CSV文件

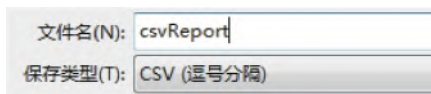
为了更详细解说，笔者先用ch20文件夹的report.xlsx文件产生一个CSV文件，未来再用这个文件做说明。目前窗口内容是report.xlsx，如下所示：



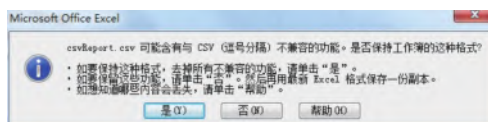
	A	B	C	D	E	F	G	H	I	J
1	Name	Year	Product	Price	Quantity	Revenue	Location			
2	Diana	2015	Black Tea	10	600	6000	New York			
3	Diana	2015	Green Tea	7	660	4620	New York			
4	Diana	2016	Black Tea	10	750	7500	New York			
5	Diana	2016	Green Tea	7	900	6300	New York			
6	Julia	2015	Black Tea	10	1200	12000	New York			
7	Julia	2016	Black Tea	10	1260	12600	New York			
8	Steve	2015	Black Tea	10	1170	11700	Chicago			
9	Steve	2015	Green Tea	7	1260	8820	Chicago			
10	Steve	2016	Black Tea	10	1350	13500	Chicago			
11	Steve	2016	Green Tea	7	1440	10080	Chicago			

请执行另存为 命令，然后选择目前D:\Python\ch20 文件夹。存档类型 选CSV(逗号分隔)(*.csv)，然后将文件名 改为csvReport

。



点击保存 按钮后，出现下列信息。



请单击是 按钮，可以得到下列结果。

	A	B	C	D	E	F	G	H	I	J
1	Name	Year	Product	Price	Quantity	Revenue	Location			
2	Diana	2015	Black Tea	10	600	6000	New York			
3	Diana	2015	Green Tea	7	660	4620	New York			
4	Diana	2016	Black Tea	10	750	7500	New York			
5	Diana	2016	Green Tea	7	900	6300	New York			
6	Julia	2015	Black Tea	10	1200	12000	New York			
7	Julia	2016	Black Tea	10	1260	12600	New York			
8	Steve	2015	Black Tea	10	1170	11700	Chicago			
9	Steve	2015	Green Tea	7	1260	8820	Chicago			
10	Steve	2016	Black Tea	10	1350	13500	Chicago			
11	Steve	2016	Green Tea	7	1440	10080	Chicago			

我们已经成功地建立一个CSV文件了，文件是csvReport.csv，可以关闭上述Excel窗口了。

20-2 用记事本打开CSV文件

CSV文件的特色是几乎可以在所有不同的电子表格内编辑，当然也可以在一般的文字编辑程序内查阅使用，如果我们现在使用记事本打开这个CSV文件，可以看到这个文件的原貌。

```
Name,Year,Product,Price,Quantity,Revenue,Location
Diana,2015,Black Tea,10,600,6000,New York
Diana,2015,Green Tea,7,660,4620,New York
Diana,2016,Black Tea,10,750,7500,New York
Diana,2016,Green Tea,7,900,6300,New York
Julia,2015,Black Tea,10,1200,12000,New York
Julia,2016,Black Tea,10,1260,12600,New York
Steve,2015,Black Tea,10,1170,11700,Chicago
Steve,2015,Green Tea,7,1260,8820,Chicago
Steve,2016,Black Tea,10,1350,13500,Chicago
Steve,2016,Green Tea,7,1440,10080,Chicago
```

20-3 CSV模块

Python有内置CSV模块，导入这个模块后，可以轻松读取CSV文件，方便未来程序的操作，所以本程序前端要加上下列指令。

```
import csv
```

20-4 读取CSV文件

20-4-1 使用open()打开CSV文件

在读取CSV文件前第一步是使用open()打开文件，语法格式如下：

```
with open(文件名
) as csvFile
# csvFile是可以自行命名的文件对象
```

相关系列指令

如果忘了with关键词的用法，可以参考14-2-2小节。当然你也可以直接使用传统方法打开文件。

```
csvFile = open(文件名
) # 打开文件建立CSV文件对象csvFile
```

20-4-2 建立Reader对象

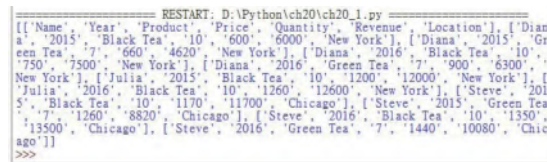
有了CSV文件对象后，下一步可以使用csv模块的reader()建立Reader对象，可以使用list()将这个Reader对象转换成列表

(list)，现在我们可以很轻松地使用这个列表资料了。

程序实例ch20_1.py： 打开csvReport.csv文件，读取csv文件可以建立Reader对象 csvReader，再将csvReader对象转成列表数据，然后打印列表数据。

```
1 # ch20_1.py
2 import csv
3
4 fn = 'csvReport.csv'
5 with open(fn) as csvFile:          # 打开csv文件
6     csvReader = csv.reader(csvFile) # 读文件建立Reader对象
7     listReport = list(csvReader)   # 将数据转成列表
8 print(listReport)                  # 打印列表方法
```

执行结果



```
RESTART: D:\Python\ch20\ch20_1.py
[['Name', 'Year', 'Product', 'Price', 'Quantity', 'Revenue', 'Location'], ['Diana', '2015', 'Black Tea', '10', '600', '6000', 'New York'], ['Diana', '2015', 'Green Tea', '7', '660', '4620', 'New York'], ['Diana', '2016', 'Black Tea', '10', '750', '7500', 'New York'], ['Diana', '2016', 'Green Tea', '7', '900', '6300', 'New York'], ['Julia', '2015', 'Black Tea', '10', '1200', '12000', 'New York'], ['Julia', '2016', 'Black Tea', '10', '1260', '12600', 'New York'], ['Steve', '2015', 'Black Tea', '10', '1170', '11700', 'Chicago'], ['Steve', '2015', 'Green Tea', '7', '1260', '8820', 'Chicago'], ['Steve', '2016', 'Black Tea', '10', '1350', '13500', 'Chicago'], ['Steve', '2016', 'Green Tea', '7', '1440', '10080', 'Chicago']]
>>>
```

上述程序需留意的是，程序第6行所建立的Reader对象 csvReader，只能在with关键区块内使用，此例是5-7行，未来我们要继续操作这个CSV文件内容，需使用第7行所建的列表listReport或是重新开档与读档。

20-4-3 用循环列出Reader对象数据

我们可以使用for循环操作Reader对象，列出各行数据，同时使用Reader对象的line_num属性列出行号。

程序实例ch20_2.py： 读取Reader对象，然后以循环方式列出对象内容。

```
1 # ch20_2.py
2 import csv
3
4 fn = 'csvReport.csv'
5 with open(fn) as csvFile:          # 打开csv文件
6     csvReader = csv.reader(csvFile) # 读文件建立Reader对象csvReader
7     for row in csvReader:           # 用循环列出csvReader对象内容
8         print("Row %s = " % csvReader.line_num, row)
```

执行结果

```
===== RESTART: D:/Python/ch20/ch20_2.py =====
Row 1 = ['Name', 'Year', 'Product', 'Price', 'Quantity', 'Revenue', 'Location']
Row 2 = ['Diana', '2015', 'Black Tea', '10', '600', '6000', 'New York']
Row 3 = ['Diana', '2015', 'Green Tea', '7', '660', '4620', 'New York']
Row 4 = ['Diana', '2016', 'Black Tea', '10', '750', '7500', 'New York']
Row 5 = ['Diana', '2016', 'Green Tea', '7', '900', '6300', 'New York']
Row 6 = ['Julia', '2015', 'Black Tea', '10', '1200', '12000', 'New York']
Row 7 = ['Julia', '2016', 'Black Tea', '10', '1260', '12600', 'New York']
Row 8 = ['Steve', '2015', 'Black Tea', '10', '1170', '11700', 'Chicago']
Row 9 = ['Steve', '2015', 'Green Tea', '7', '1260', '8820', 'Chicago']
Row 10 = ['Steve', '2016', 'Black Tea', '10', '1350', '13500', 'Chicago']
Row 11 = ['Steve', '2016', 'Green Tea', '7', '1440', '10080', 'Chicago']
>>>
```

20-4-4 用循环列出列表内容

for循环也可用于列出列表内容。

程序实例ch20_3.py: 用for循环列出列表内容。

```
1 # ch20_3.py
2 import csv
3
4 fn = 'csvReport.csv'
5 with open(fn) as csvFile:          # 打开csv文件
6     csvReader = csv.reader(csvFile) # 读文件建立Reader对象
7     listReport = list(csvReader)   # 将数据转成列表
8     for row in listReport:         # 使用循环列出列表内容
9         print(row)
```

执行结果

```
===== RESTART: D:/Python/ch20/ch20_3.py =====
['Name', 'Year', 'Product', 'Price', 'Quantity', 'Revenue', 'Location']
['Diana', '2015', 'Black Tea', '10', '600', '6000', 'New York']
['Diana', '2015', 'Green Tea', '7', '660', '4620', 'New York']
['Diana', '2016', 'Black Tea', '10', '750', '7500', 'New York']
['Diana', '2016', 'Green Tea', '7', '900', '6300', 'New York']
['Julia', '2015', 'Black Tea', '10', '1200', '12000', 'New York']
['Julia', '2016', 'Black Tea', '10', '1260', '12600', 'New York']
['Steve', '2015', 'Black Tea', '10', '1170', '11700', 'Chicago']
['Steve', '2015', 'Green Tea', '7', '1260', '8820', 'Chicago']
['Steve', '2016', 'Black Tea', '10', '1350', '13500', 'Chicago']
['Steve', '2016', 'Green Tea', '7', '1440', '10080', 'Chicago']
>>>
```

20-4-5 使用列表索引读取CSV内容

其实我们也可以使用第6章所学的列表知识，读取CSV内容。

程序实例ch20_4.py: 使用索引列出列表内容。

```

1 # ch20_4.py
2 import csv
3
4 fn = 'csvReport.csv'
5 with open(fn) as csvFile:          # 打开csv文件
6     csvReader = csv.reader(csvFile) # 读文件建立Reader对象
7     listReport = list(csvReader)    # 将数据转成列表
8
9 print(listReport[0][1], listReport[0][2])
10 print(listReport[1][2], listReport[1][5])
11 print(listReport[2][3], listReport[2][6])

```

执行结果

```

===== RESTART: D:/Python/ch20/ch20_4.py =====
Year Product
Black Tea 6000
7 New York
>>>

```

20-4-6 DictReader()

这这也是一个读取CSV文件的方法，不过传回的是排序字典(OrderedDict) 类型，所以可以用域名 当索引 方式取得数据。在美国许多文件以CSV文件储存时，常常人名的Last Name(姓)与First Name(名)是分开以不同字段储存，读取时可以使用这个方法，可参考ch20文件夹的csvPeople.csv文件。

```

first_name,last_name,city
Eli,Manning,New York
Kevin ,James,Cleveland
Mike,Jordon,Chicago

```

程序实例ch20_5.py： 使用DictReader()读取csv文件，然后列出DictReader对象内容。

```

1 # ch20_5.py
2 import csv
3
4 fn = 'csvPeople.csv'
5 with open(fn) as csvFile:          # 打开csv文件
6     csvDictReader = csv.DictReader(csvFile) # 读文件建立DictReader对象
7     for row in csvDictReader:        # 列出DictReader各行内容
8         print(row)

```

执行结果

```
===== RESTART: D:/Python/ch20/ch20_5.py =====
OrderedDict([('first_name', 'Eli'), ('last_name', 'Manning'), ('city', 'New York')])
OrderedDict([('first_name', 'Kevin '), ('last_name', 'James'), ('city', 'Cleveland')])
OrderedDict([('first_name', 'Mike'), ('last_name', 'Jordon'), ('city', 'Chicago')])
>>>
```

对于上述OrderedDict数据类型，可以使用下列方法读取。

程序实例 ch20_6.py： 将 csvPeople.csv 文件的 last_name 与 first_name解析出来。

```
1 # ch20_6.py
2 import csv
3
4 fn = 'csvPeople.csv'
5 with open(fn) as csvFile:          # 打开csv文件
6     csvDictReader = csv.DictReader(csvFile) # 读文件建立DictReader对象
7     for row in csvDictReader:        # 使用循环列出字典内容
8         print(row['first_name'], row['last_name'])
```

执行结果

```
===== RESTART: D:/Python/ch20/ch20_6.py =====
Eli Manning
Kevin James
Mike Jordon
>>>
```

20-5 写入CSV文件

20-5-1 打开欲写入的文件open()与关闭文件close()

想要将数据写入CSV文件，首先是要打开一个文件供写入，如下所示：

```
csvFile = open('文件名', 'w', newline= ' ') # w是write only模式
...
csvFile.close()                               # 执行结束关闭文件
```

当然如果使用with关键词可以省略close()，如下所示：

```
with open('文件名', 'w', newline= ' ') as csvFile:
    ...
```


20-5-2 建立writer对象

如果应用前一节的csvFile对象，接下来需建立writer对象，语法如下：

```
with open( '文件名' , 'w' , newline= ' ' ) as csvFile:
    outWriter = csv.writer(csvFile)
    ...
```

或是

```
csvFile = open( '文件名' , 'w' , newline= ' ' ) # w是write only 模式
outWriter = csv.writer(csvFile)
...
csvFile.close( ) # 执行结束关闭文件
```

上述打开文件时多加参数newline= ' '，可避免输出时每个行之间多空一行。

20-5-3 输出列表writerow()

writerow()可以输出列表数据。

程序实例ch20_7.py： 输出列表数据的应用。

```
1 # ch20_7.py
2 import csv
3
4 fn = 'out20_7.csv'
5 with open(fn, 'w', newline = '') as csvFile: # 打开csv文件
6     csvWriter = csv.writer(csvFile) # 建立Writer对象
7     csvWriter.writerow(['Name', 'Age', 'City'])
8     csvWriter.writerow(['Hung', '35', 'Taipei'])
9     csvWriter.writerow(['James', '40', 'Chicago'])
```

执行结果 下列是分别用记事本与Excel打开文件的结果。



	A	B	C	D
1	Name	Age	City	
2	Hung	35	Taipei	
3	James	40	Chicago	
4				

本书在ch20文件夹内有ch20_7_1.py文件，这个文件在第5行open()中没有加上newline= ‘\n’，造成输出时若用Excel窗口观察有跳行输出的现象，可参考out20_7_1.csv文件，至于用记事本打开文件则一切正常，下列是程序代码。

```
5 with open(fn, 'w') as csvFile: # 打开csv文件
```

out20_7_1 - 記事本			
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)			
Name, Age, City			
Hung, 35, Taipei			
James, 40, Chicago			

下列是执行结果，读者可以比较下图右边的Excel报表。

程序实例ch20_8.py： 复制CSV文件，这个程序会用读文件，然后将文件写入另一个文件的方式，达成复制的目的。

```
1 # ch20_8.py
2 import csv
3
4 infn = 'csvReport.csv' # 来源文件
5 outfn = 'out20_8.csv' # 目标文件
6 with open(infn) as csvRFile: # 打开csv文件供读取
7     csvReader = csv.reader(csvRFile) # 读文件建立Reader对象
8     listReport = list(csvReader) # 将数据转成列表
9
10 with open(outfn, 'w', newline = '') as csvOFile: # 打开csv文件供写入
11     csvWriter = csv.writer(csvOFile) # 建立Writer对象
12     for row in listReport: # 将列表写入
13         csvWriter.writerow(row)
```

执行结果 读者可以打开out20_8.csv文件，内容将和csvReport.csv文件相同。

20-5-4 delimiter关键词

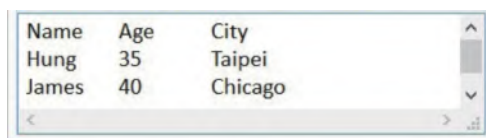
delimiter是分隔符，这个关键词是用在writer()方法内，将数据写入CSV文件时预设是同一行各栏间是逗号，可以用这个分隔符

更改各栏间的逗号。

程序实例ch20_9.py: 将分隔符改为定位点字符(\t)。

```
1 # ch20_9.py
2 import csv
3
4 fn = 'out20_9.csv'
5 with open(fn, 'w', newline = '') as csvFile:          # 打开csv文件
6     csvWriter = csv.writer(csvFile, delimiter='\t')    # 建立Writer对象
7     csvWriter.writerow(['Name', 'Age', 'City'])
8     csvWriter.writerow(['Hung', '35', 'Taipei'])
9     csvWriter.writerow(['James', '40', 'Chicago'])
```

执行结果 下列是用记事本打开out20_9.csv的结果。



Name	Age	City
Hung	35	Taipei
James	40	Chicago

当用‘\t’字符取代逗号后，Excel窗口打开这个文件时，会将每行数据挤在一起，所以最好方式是用记事本打开这类的CSV文件。

20-5-5 写入字典数据DictWriter()

DictWriter()可以写入字典数据，其语法格式如下：

dictWriter

= csv.DictWriter(csvFile, fieldnames=fields)

上述dictWriter 是字典的Writer对象，在上述指令前我们需要先设定fields列表，这个列表将包含未来字典内容的键(key)。

程序实例ch20_10.py: 使用DictWriter()将字典数据写入CSV文件。

```
1 # ch20_10.py
2 import csv
3
4 fn = 'out20_10.csv'
5 with open(fn, 'w', newline = '') as csvFile:          # 打开csv文件
6     fields = ['Name', 'Age', 'City']
7     dictWriter = csv.DictWriter(csvFile, fieldnames=fields) # 建立Writer对象
8
9     dictWriter.writeheader()                            # 写入标题
10    dictWriter.writerow({'Name': 'Hung', 'Age': '35', 'City': 'Taipei'})
11    dictWriter.writerow({'Name': 'James', 'Age': '40', 'City': 'Chicago'})
```

执行结果 下列是用Excel打开out20_10.csv的结果。

上述程序第9行的writeheader()主要是写入我们在第7行设定的fieldname。

程序实例ch20_11.py: 改写程序实例ch20_10.py, 将欲写入CSV文件的数据改成列表数据, 此列表数据的元素是字典。

	A	B	C	D
1	Name	Age	City	
2	Hung	35	Taipei	
3	James	40	Chicago	

```
1 # ch20_11.py
2 import csv
3
4 dictList = [{'Name': 'Hung', 'Age': '35', 'City': 'Taipei'},      # 定义列表,元素是字典
5             {'Name': 'James', 'Age': '40', 'City': 'Chicago'}]
6
7 fn = 'out20_11.csv'
8 with open(fn, 'w', newline = '') as csvFile:                    # 打开csv文件
9     fields = ['Name', 'Age', 'City']
10    dictWriter = csv.DictWriter(csvFile, fieldnames=fields)      # 建立Writer对象
11
12    dictWriter.writeheader()                                     # 写入标题
13    for row in dictList:                                        # 写入内容
14        dictWriter.writerow(row)
```

执行结果 打开out20_11.csv后与out20_10.csv相同。

20-6 后记

读者可能会想学习了打开个别CSV文件的用处在哪里? 现在是大数
据时代, 所有数据搜集无法完整地用某一种格式呈现, CSV是电子表格
和数据库间最常用的资料格式, 我们可以先将所搜集的各式文件转成
CSV, 然后你就可以使用Python读取所有的CSV文件, 再提取需要的数
据做大数据分析。或是利用CSV文件, 将它当作不同数据库间的桥梁或
数据库与电子表格间的桥梁。

习题

1. 在ch20文件夹内有report.xlsx文件，请读取这个文件，然后转存成ex20_1.csv文件。
2. 请读取ch20文件夹内的csvReport.csv文件，计算2015年与2016年的总业绩，输出到ex20_2.csv。
3. 请读取ch20文件夹内的csvReport.csv文件，计算2015年与2016年业绩，最好的业务员和业绩最差的业务员，输出到ex20_3.csv。
4. 请读取csvReport.csv文件，然后输出Name、Revenue、Location字段，同时名字不要重复出现。
5. 请扩充ch20_11.py，将列表数据元素个数扩充为10个，同时每一个字典元素增加手机号码字段。

第21章 网络爬虫

过去我们浏览网页是使用浏览器，例如，Microsoft公司的Internet Explorer、Google公司的Chrome、Apple公司的Safari等。现在学了Python，我们可以不再需要通过浏览器浏览网页了，除了浏览网页，本章笔者也将讲解如何从网站下载有用的信息。

一般我们将从网络搜寻资源的程序称之为**网络爬虫**，一些著名的搜索引擎公司就是不断地送出网络爬虫搜寻网络最新信息，以保持搜索引擎的热度。

21-1 上网不再需要浏览器了

这一节将介绍webbrowser模块浏览网页，在程序前方需导入此模块。

```
import webbrowser
```

Python有提供webbrowser模块，可以调用这个模块的open()方法，就可以打开指定的网页了。

程序实例 ch21_1.py： 打开上奇信息公司(<http://www.grandtech.info>)网页。

```
1 # ch21_1.py
2 import webbrowser
3 webbrowser.open('http://www.grandtech.info')
```

执行结果 请留意以下浏览程序外观，不属于目前各位已知的浏览器。



21-2 下载网页信息使用requests模块

requests是第三方模块，读者需参考附录B，使用下列指令下载此模块。

```
pip install requests
```

21-2-1 下载网页使用requests.get()方法

requests.get()方法内需放置欲下载网页信息的网址当参数，这个方法可以传回网页的HTML源文件。

程序实例ch21_2.py： 下载上奇信息网页内容做测试，这个程序会列出返回值的数据类型。

```
1 # ch21_2.py
2 import requests
3
4 url = 'http://www.grandtech.info'
5 htmlfile = requests.get(url)
6 print(type(htmlfile))
```

执行结果

```
===== RESTART: D:/Python/ch21/ch21_2.py :
<class 'requests.models.Response'>
>>>
```

由上述可以知道使用requests.get()之后传回的数据类型是Response对象。

21-2-2 认识Response对象

Response对象内有下列几个重要属性：

status_code: 如果值是requests.codes.ok, 表示获得的网页内容成功。

text: 网页内容。

程序实例ch21_3.py: 检查ch21_2.py获得的网页内容是否成功。

```
1 # ch21_3.py
2 import requests
3
4 url = 'http://www.grandtech.info'
5 htmlfile = requests.get(url)
6 if htmlfile.status_code == requests.codes.ok:
7     print("取得网页内容成功")
8 else:
9     print("取得网页内容失败")
```

执行结果

```
===== RESTART: D:/Python/ch21/ch21_3.py
取得网页内容成功
>>>
```

程序实例ch21_4.py: 扩充ch21_3.py, 取得网页内容大小。

```
1 # ch21_4.py
2 import requests
3
4 url = 'http://www.grandtech.info'
5 htmlfile = requests.get(url)
6 if htmlfile.status_code == requests.codes.ok:
7     print("取得网页内容成功")
8 else:
9     print("取得网页内容失败")
10 print("网页内容大小 = ", len(htmlfile.text))
```

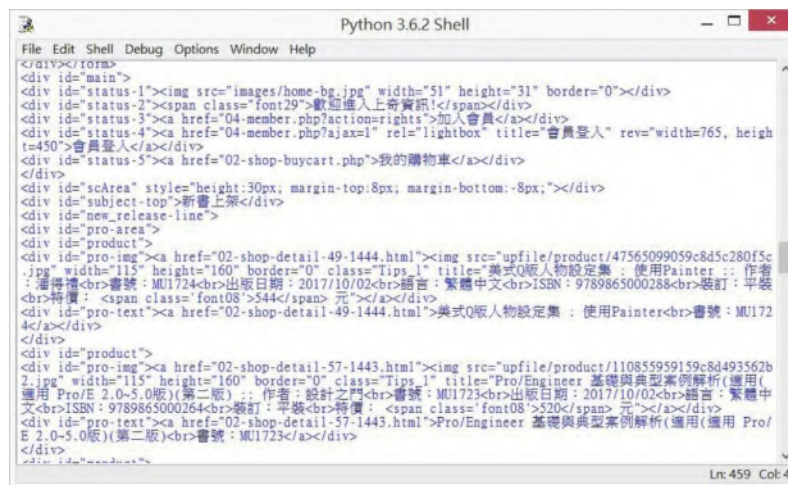
执行结果

```
===== RESTART: D:/Python/ch21/ch21_4.py
取得网页内容成功
网页内容大小 = 38818
>>>
```


程序实例ch21_5.py: 打印网页的原始码, 然后可以看到密密麻麻的网页内容(繁体中文)。

```
1 # ch21_5.py
2 import requests
3
4 url = 'http://www.grandtech.info'
5 htmlfile = requests.get(url)
6 if htmlfile.status_code == requests.codes.ok:
7     print("取得网页内容成功")
8 else:
9     print("取得网页内容失败")
10 print(htmlfile.text) # 打印网页内容
```

执行结果



```
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
</div></form>
<div id="status-1"></div>
<div id="status-2"><span class="font29">歡迎進入上奇資訊!</span></div>
<div id="status-3"><a href="04-member.php?action=rights">加入會員</a></div>
<div id="status-4"><a href="04-member.php?ajax=1" rel="lightbox" title="會員登入" rev="width=765, height=450">會員登入</a></div>
<div id="status-5"><a href="02-shop-buycart.php">我的購物車</a></div>
</div>
<div id="scArea" style="height:30px; margin-top:8px; margin-bottom:-8px;"></div>
<div id="subject-top">新書上架</div>
<div id="new_release-line">
<div id="pro-area">
<div id="product">
<div id="pro-img"><a href="02-shop-detail-49-1444.html">書號：MU1724<br>出版日期：2017/10/02<br>語言：繁體中文<br>ISBN：9789865000288<br>裝訂：平裝<br>特價：<span class="font08">$544</span> 元"></a></div>
<div id="pro-text"><a href="02-shop-detail-49-1444.html">美式Q版人物設定集：使用Painter<br>書號：MU1724</a></div>
</div>
<div id="product">
<div id="pro-img"><a href="02-shop-detail-57-1443.html">書號：MU1723<br>出版日期：2017/10/02<br>語言：繁體中文<br>ISBN：9789865000264<br>裝訂：平裝<br>特價：<span class="font08">$520</span> 元"></a></div>
<div id="pro-text"><a href="02-shop-detail-57-1443.html">Pro/Engineer 基礎與典型实例解析(適用Pro/E 2.0-5.0版)(第二版)<br>書號：MU1723</a></div>
</div>
</div>
```

21-2-3 搜索页特定内容

继续先前的内容, 网页内容下载后, 如果我们想要搜寻特定字符串, 可以使用许多方法, 下列将简单地用2个方法处理。

程序实例ch21_6.py: 搜寻字符串“洪锦魁”使用方法1, 使用方法2不仅搜寻, 如果找到同时列出执行结果。这个程序执行时, 如果网页内容下载成功, 会要求输入欲搜寻的字符串, 将此字符串放入pattern变量。使用2种方法搜寻, 方法1会列出搜寻成功或失败, 方法2会列出搜寻到此字符串的次数。


```

1 # ch21_6.py
2 import requests
3 import re
4
5 url = 'http://www.grandtech.info'
6 htmlfile = requests.get(url)
7 if htmlfile.status_code == requests.codes.ok:
8     pattern = input("请输入欲搜寻的字符串：") # pattern存放欲搜寻的字符串
9     # 使用方法1
10    if pattern in htmlfile.text: # 方法1
11        print("搜寻 %s 成功" % pattern)
12    else:
13        print("搜寻 %s 失败" % pattern)
14    # 使用方法2，如果找到放在列表name内
15    name = re.findall(pattern, htmlfile.text) # 方法2
16    if name != None:
17        print("%s 出现 %d 次" % (pattern, len(name)))
18    else:
19        print("%s 出现 0 次" % pattern)
20 else:
21    print("网页下载失败")

```

执行结果

```

===== RESTART: D:/Python/ch21/ch21_6.py =====
请输入欲搜寻的字符串：洪锦魁
搜寻 洪锦魁 成功
洪锦魁 出现 4 次
>>>
===== RESTART: D:/Python/ch21/ch21_6.py =====
请输入欲搜寻的字符串：武则天
搜寻 武则天 失败
武则天 出现 0 次
>>>

```

21-2-4 下载网页失败的异常处理

有时候我们输入网址错误或有些网页有反爬虫机制，造成下载网页失败，其实建议可以使用第15章程式除错与异常处理观念处理这类问题。Response对象有raise_for_status()，可以针对网址正确但是后续文件名错误的状况产生异常处理。下列将直接以实例解说。

程序实例ch21_7.py： 下载网页错误的异常处理，由于不存在file_not_existed造成这个程序异常发生。

```

1 # ch21_7.py
2 import requests
3
4 url = 'http://www.grandtech.info/file_not_existed' # 不存在的内容
5 htmlfile = requests.get(url)
6 try:
7     htmlfile.raise_for_status() # 异常处理
8     print("下载成功")
9 except Exception as err: # err是系统自定义的错误信息
10    print("网页下载失败： %s" % err)

```

执行结果

```
===== RESTART: D:/Python/ch21/ch21_7.py =====
网页下载失败: 404 Client Error: Not Found for url: http://www.grandtech.info/file_not_existed
>>>
```

若是忘记了try: 的用法可参考第15章，若是忘记第9行用法可以参考15-2-4小节。上述raise_for_status()可以处理网址正确但是后面附加文件错误的问题，可是无法处理网址错误的信息。

程序实例ch21_8.py: 一个错误的网址造成出现一长串的错误。

```
4 url = 'http://www.gzaxxc.com/file_not_existed' # 错误的网址
```

执行结果

```
===== RESTART: D:/Python/ch21/ch21_9.py =====
Traceback (most recent call last):
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-packages\urllib3\connection.py", line 141, in _new_conn
    (self.host, self.port), self.timeout, **extra_kw)
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-packages\urllib3\util\connection.py", line 60, in create_connection
    for res in socket.getaddrinfo(host, port, family, socket.SOCK_STREAM):
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\socket.py", line 743, in getaddrinfo
    for res in _socket.getaddrinfo(host, port, family, type, proto, flags):
socket.gaierror: [Errno 11001] getaddrinfo failed

During handling of the above exception, another exception occurred:
```

很明显执行异常处理期间又产生了异常，所以程序错误产生中断，有时候可以将requests.get()放在try: 后面。

程序实例ch21_9.py: 重新设计下载网页错误的异常处理。

```
1 # ch21_9.py
2 import requests
3
4 url = 'http://www.gzaxxc.com/file_not_existed' # 错误的网址
5 try:
6     htmlfile = requests.get(url)
7     print("下载成功")
8 except Exception as err:                                # err是系统自定义的错误信息
9     print("网页下载失败: %s" % err)
```

执行结果

```
===== RESTART: D:/Python/ch21/ch21_9.py =====
网页下载失败: HTTPConnectionPool(host='www.gzaxxc.com', port=80): Max retries exceeded with url: /file_not_existed (Caused by NewConnectionError('<urllib3.connection.HTTPConnection object at 0x0436C310>: Failed to establish a new connection: [Errno 11001] getaddrinfo failed',))
>>>
```

从上述可以看到，即使网址错误，程序还是依照我们设计的逻辑执行。

21-2-5 网页服务器阻挡造成读取错误

现在有些网页也许基于安全原因，或是不想让太多网络爬虫造访造成网络流量增加，因此会设计程序阻挡网络爬虫提取信息，碰上这类问题就会产生406的错误，如下所示：

程序实例ch21_9_1.py： 网页服务器阻挡造成编号406的错误，无法提取网页信息。

```
1 # ch21_9_1.py
2 import requests
3
4 url = 'http://aaa.24ht.com.tw/'
5 htmlfile = requests.get(url)
6 htmlfile.raise_for_status()
```

执行结果

```
===== RESTART: D:/Python/ch21/ch21_9_1.py =====
Traceback (most recent call last):
  File "D:/Python/ch21/ch21_9_1.py", line 6, in <module>
    htmlfile.raise_for_status()
  File "C:\Users\j\in-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-packages\requests\models.py", line 925, in raise_for_status
    raise HTTPError(http_error_msg, response=self)
requests.exceptions.HTTPError: 406 Client Error: Not Acceptable for url: http://aaa.24ht.com.tw/
>>>
```

上述程序第6行的raise_for_status()主要是如果Response对象htmlfile在前一行提取网页内容有错误码时，将列出错误原因，406错误就是网页服务器阻挡。用这行程序代码，可以快速中断协助我们侦错程序的错误。

21-2-6 爬虫程序伪装成浏览器

其实我们使用requests.get()方法到网络上读取网页数据，这类的程序就称网络爬虫程序，甚至你也可以将各大公司所设计的搜索引

擎称为网络爬虫程序。为了解决爬虫程序被服务器阻挡的困扰，我们可以将所设计的爬虫程序伪装成浏览器，方法是在程序前端加上headers内容。

程序实例 ch21_9_2.py： 使用伪装浏览器方式，重新设计ch21_9_1.py。

```
1 # ch21_9_2.py
2 import requests
3
4 headers = { 'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64)\
5             AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101\
6             Safari/537.36', }
7 url = 'http://aaa.24ht.com.tw/'
8 htmlfile = requests.get(url, headers=headers)
9 htmlfile.raise_for_status()
10 print("伪装浏览器提取网络数据成功")
```

执行结果



```
===== RESTART: D:/Python/ch21/ch21_9_2.py =====
伪装浏览器提取网络数据成功
>>>
```

上述的重点是第4-6行的叙述，其实这是一个标题(headers)定义，第4和5行末端的反斜杠“\”主要表达下一行与这一行是相同叙述，也就是处理同一叙述太长时分行撰写，Python会将4-6行视为同一叙述。然后第8行调用requests.get()时，第2个参数需要加上“headers=headers”，这样这个程序就可以伪装成浏览器，顺利取得网页数据了。

其实将Python程序伪装成浏览器比想象的复杂，上述headers定义碰上安全机制强大的网页也可能失效，更详细的解说超出本书范围。

21-2-7 存储下载的网页

使用requests.get()获得网页内容时，是存储在Response对象类型内，如果要将这类型的对象存入硬盘内，需使用Response对象的iter_content()方法，这个方法是采用重复迭代方式将Response对象

内容写入指定的文件内，每次写入指定扇区大小是以Bytes为单位，一般可以设定1024×5或1024×10或更多。

程序实例ch21_10.py： 下载深石数字公司网页，同时将网页内容存入out21_10.txt文件内。

```
1 # ch21_10.py
2 import requests
3
4 url = 'http://www.deepstone.com.tw'          # 网址
5 try:
6     htmlfile = requests.get(url)
7     print("下载成功")
8 except Exception as err:                      # err是系统自定义的错误信息
9     print("网页下载失败: %s" % err)
10 # 存储网页内容
11 fn = 'out21_10.txt'
12 with open(fn, 'wb') as file_Obj:             # 以二进制存储
13     for diskStorage in htmlfile.iter_content(10240): # Response对象处理
14         size = file_Obj.write(diskStorage)         # Response对象写入
15         print(size)                                # 列出每次写入大小
16     print("以 %s 存储网页HTML文件成功" % fn)
```

执行结果

```
===== RESTART: D:/Python/ch21/ch21_10.py =====
下载成功
10240
5395
以 out21_10.txt 存储网页HTML文件成功
>>>
```

由于这个网页文件内容比较小，所以笔者将每次写入文件大小设为10240bytes，程序第12行所打开的是以二进制可写入“wb”方式打开，这是怕网页内有Unicode码。程序第13~15行是一个循环，这个循环会将Response对象htmlfile以循环方式写入所打开的file_Obj，最后是存入第11行设定的out21_11.txt文件内。程序第14行每次使用write()写入Response对象时会回传所写入网页内容的大小，所以第15行会列出当次循环所写入的大小。

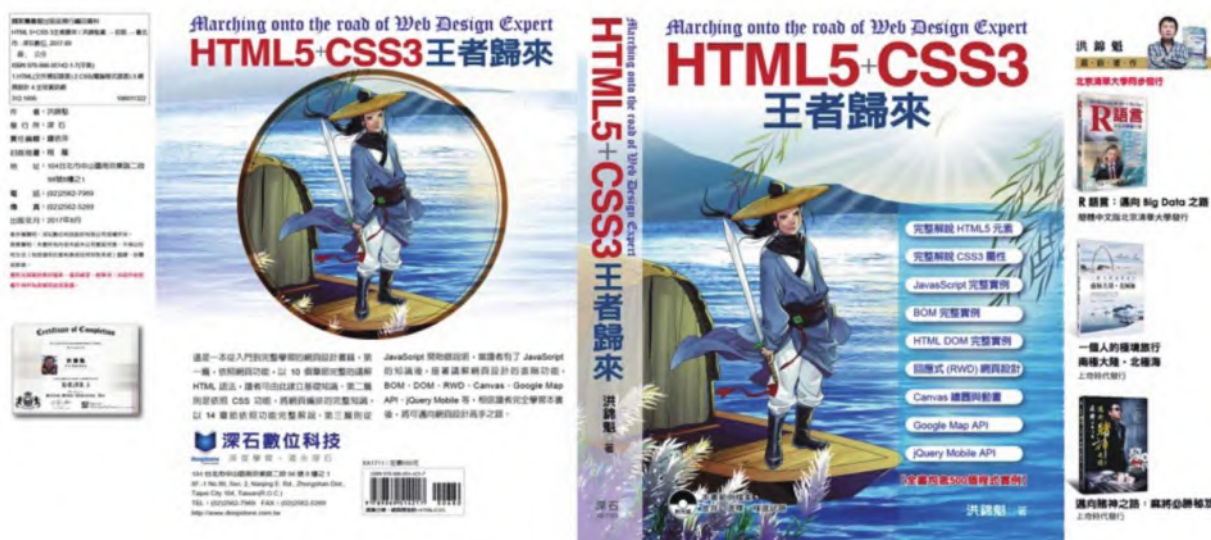
21-3 检视网页原始文件

前一节笔者教导读者利用requests.get()取得网页内容的原始HTML文件，其实也可以使用浏览器取得网页内容的原始文件。检视网

页的原始文件目的不是要模仿设计相同的网页，主要是掌握几个关键点，然后提取我们想要的数据库。

21-3-1 建议阅读书籍

也许你不必彻底了解HTML网页设计，但是若有HTML知识更佳，下列是笔者所著的HTML，以600个程序实例讲解网页设计，可供读者参考，简体中文版同步发行。



21-3-2 以Microsoft浏览器为实例

Microsoft浏览器Internet Explorer简称IE，此例是使用IE打开清华大学出版社 网页，在网页内单击鼠标右键，出现快捷菜单时，执行查看源 指令。



就可以看到此网页的原始HTML文件。

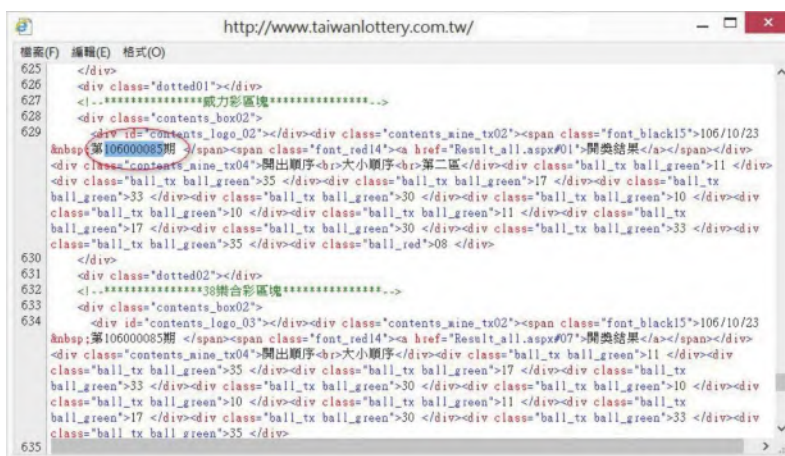


如果使用的是Chrome浏览器，将鼠标光标放在网页上单击鼠标右键，打开快捷菜单，再执行View page source 指令，也可以打开新窗口显示此网页的HTML原始文件。

21-3-3 源文件的重点

假设你想要下载某网页的图片，可以进入网页了解此网页的结构，例如，如果我们想要下载台湾彩券公司的威力彩开奖号码，我们可以先进入此公司网页。

将鼠标光标移至威力彩开奖结果，单击鼠标右键，出现快捷菜单，执行查看源指令。接着出现HTML源文件的窗口，请执行编辑 / 搜索，再输入1000085，这是笔者写本书时最新开奖期数，可以得到下列结果。



```
http://www.taiwanlottery.com.tw/
625 </div>
626 <div class="dotted01"></div>
627 <!--*****威力彩區塊*****-->
628 <div class="contents_box02">
629 <div id="contents_logo_02"></div><div class="contents_mine_tx02"><span class="font_black15">106/10/23
  第10600085期</span><span class="font_red14"><a href="Result_all.aspx#01">開獎結果</a></span></div>
  <div class="contents_mine_tx04">開出順序</div><div class="ball_tx ball_green">11 </div>
  <div class="ball_tx ball_green">35 </div><div class="ball_tx ball_green">17 </div><div class="ball_tx
  ball_green">33 </div><div class="ball_tx ball_green">30 </div><div class="ball_tx ball_green">10 </div><div
  class="ball_tx ball_green">10 </div><div class="ball_tx ball_green">11 </div><div class="ball_tx
  ball_green">17 </div><div class="ball_tx ball_green">30 </div><div class="ball_tx ball_green">33 </div><div
  class="ball_tx ball_green">35 </div><div class="ball_red">08 </div>
  </div>
  <div class="dotted02"></div>
  <!--*****38排合彩區塊*****-->
  <div class="contents_box02">
  <div id="contents_logo_03"></div><div class="contents_mine_tx02"><span class="font_black15">106/10/23
    第10600085期</span><span class="font_red14"><a href="Result_all.aspx#07">開獎結果</a></span></div>
    <div class="contents_mine_tx04">開出順序</div><div class="ball_tx ball_green">11 </div><div
    class="ball_tx ball_green">35 </div><div class="ball_tx ball_green">17 </div><div class="ball_tx
    ball_green">33 </div><div class="ball_tx ball_green">30 </div><div class="ball_tx ball_green">10 </div><div
    class="ball_tx ball_green">10 </div><div class="ball_tx ball_green">11 </div><div class="ball_tx
    ball_green">17 </div><div class="ball_tx ball_green">30 </div><div class="ball_tx ball_green">33 </div><div
    class="ball_tx ball_green">35 </div>
  </div>
635
```

由上图我们已经找到放置威力彩券号码球的地点了，接着我们必须了解此区域特性，然后再针对此，执行搜寻，最后设计可以找出彩券号码的爬虫程序，可参考ch21_20.py。

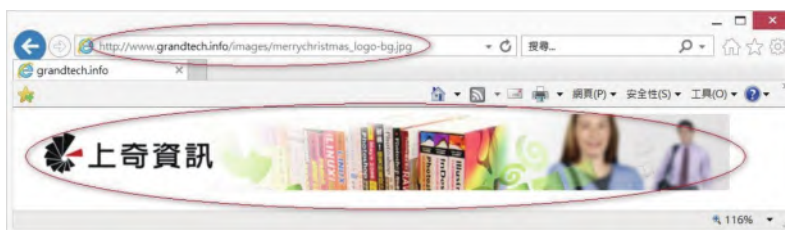
如果我们现在要下载某个网页的所有图片文件，可以进入该网页，例如，想要下载上奇信息网页 (http://www.grandtech.info) 的图片文件，可以打开该网页的HTML文件，然后请执行编辑 / 寻找，再输入 ‘<img ’，接着可以了解该网页图片文件的状况。



```
http://www.grandtech.info/
153 </head>
154 <body>
  onLoad="MM_preloadImages('images/btn/btn02-2.jpg','images/btn/btn03-2.jpg','images/btn/btn
  04-2.jpg','images/btn/btn05-2.jpg','images/btn/btn06-2.jpg','images/btn/btn01-2.jpg')">
  <div id="logo"></div>
  <div id="toolbar">
  <a href="index.php" target="_top"
  onClick="MM_nbGroup('down','group1','btn011','images/btn/btn01-2.jpg',1)"
  onMouseOver="MM_nbGroup('over','btn011','images/btn/btn01-2.jpg','images/btn/btn01-2.jpg',
  1)" onMouseOut="MM_nbGroup('out')"></a><a href="01-about.php" target="_top"
  onClick="MM_nbGroup('down','group1','btn021','images/btn/btn02-2.jpg',1)"
  onMouseOver="MM_nbGroup('over','btn021','images/btn/btn02-2.jpg','images/btn/btn02-2.jpg',
  1)" onMouseOut="MM_nbGroup('out')"></a><a href="02-shop-list.html"

```


由上图可以看到图片文件是在images文件夹内，其实我们也可以使用“网址+文件路径”，列出图文件的内容。



21-4 解析网页使用BeautifulSoup模块

从前面章节读者应该已经了解了如何下载网页HTML源文件，也应该对网页的基本架构有基本认识，本节要介绍的是使用BeautifulSoup模块解析HTML文件。目前这个模块是第4版，模块名称是beautifulsoup4，可参考附录B，以下列方式安装：

```
pip install beautifulsoup4
```

虽然安装是beautifulsoup4，但是导入模块时是用下列方式：

```
import bs4
```

21-4-1 建立BeautifulSoup对象

可以使用下列语法建立BeautifulSoup对象。

```
htmlFile

= requests.get('http://www.grandtech.info') # 下载网页内容
  objSoup = bs4.BeautifulSoup(htmlFile.text

, 'lxml

') # lxml
```

是解析HTML文件方式

上述是以下载上奇信息网页为例，当网页下载后，将网页内容的Response对象传给bs4.BeautifulSoup()方法，就可以建立BeautifulSoup对象。至于另一个参数“lxml”目的是注明解析HTML文件的方法，常用的有下列方法。

‘html.parser’：这是老旧的方法(3.2.3版本前)，兼容性比较不好。

‘lxml’：速度快，兼容性佳，这是本书采用的方法。

‘html5lib’：速度比较慢，但是解析能力强，需另外安装html5lib。

```
pip install html5lib
```

程序实例ch21_11.py：解析http://www.grandtech.info网页，主要是列出数据类型。

```
1 # ch21_11.py
2 import requests, bs4
3
4 htmlFile = requests.get('http://www.grandtech.info')
5 objSoup = bs4.BeautifulSoup(htmlFile.text, 'lxml')
6 print("打印BeautifulSoup对象数据类型", type(objSoup))
```

执行结果

```
===== RESTART: D:/Python/ch21/ch21_11.py =====
打印BeautifulSoup对象数据类型 <class 'bs4.BeautifulSoup'>
>>>
```

从上述我们获得了BeautifulSoup的数据类型了，表示我们获得初步成果了。

21-4-2 基本HTML文件解析——从简单开始

真实世界的网页是很复杂的，所以笔者想先从简单的HTML文件开始解析网页。在ch21_11.py程序第5行第一个参数htmlFile.text是网

页内容的Response对象，我们可以在ch21文件夹放置一个简单的HTML文件，然后先学习使用BeautifulSoup解析此HTML文件。

程序实例myhtml.html： 在ch21文件夹有myhtml.html文件，这个文件内容如下：

```
1 <!doctype html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>洪锦魁著作</title>
6   <style>
7     h1#author { width:400px; height:50px; text-align:center;
8       background:linear-gradient(to right,yellow,green);
9     }
10    h1#content { width:400px; height:50px;
11      background:linear-gradient(to right,yellow,red);
12    }
13    section { background:linear-gradient(to right bottom,yellow,gray); }
14  </style>
15 </head>
16 <body>
17 <h1 id="author">洪锦魁</h1>
18 
19 <section>
20   <h1 id="content">一个人的极境旅行 - 南极大陆北极海</h1>
21   <p>2015/2016年<strong>洪锦魁</strong>一个人到南极</p>
22   
23 </section>
24 <section>
25   <h1 id="content">HTML5+CSS3王者归来</h1>
26   <p>本书讲解网页设计使用HTML5+CSS3</p>
27   
28 </section>
29 </body>
30 </html>
```

执行结果



本节有几个小节将会解析此份HTML文件。

程序实例ch21_12.py: 解析本书ch21文件夹的myhtml.html文件，列出对象类型。

```
1 # ch21_12.py
2 import bs4
3
4 htmlFile = open('myhtml.html', encoding='utf-8')
5 objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6 print("打印BeautifulSoup对象数据类型 ", type(objSoup))
```

执行结果

```
===== RESTART: D:/Python/ch21/ch21_12.py =====
打印BeautifulSoup对象数据类型 <class 'bs4.BeautifulSoup'>
>>>
```

上述可以看到解析ch21文件夹的myhtml.html文件是初步成功的。

21-4-3 页标题title属性

BeautifulSoup对象的title属性可以传回页标题的<title>标签内容。

程序实例ch21_13.py: 使用title属性解析myhtml.html文件的页标题，本程序会列出对象类型与内容。

```
1 # ch21_13.py
2 import bs4
3
4 htmlFile = open('myhtml.html', encoding='utf-8')
5 objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6 print("对象类型 = ", type(objSoup.title))
7 print("打印title = ", objSoup.title)
```

执行结果

```
===== RESTART: D:/Python/ch21/ch21_13.py =====
对象类型 = <class 'bs4.element.Tag'>
打印title = <title>洪锦魁著作</title>
>>>
```

从上述执行结果可以看到所解析的objSoup.title是一个HTML卷标对象。

21-4-4 去除卷标传回文字text属性

前一节实例的确解析了myhtml.html文件，传回解析的结果是一个HTML的标签，不过我们可以使用text属性获得此卷标的内容。

程序实例ch21_14.py： 扩充ch21_13.py，列出解析的标签内容。

```
1 # ch21_14.py
2 import bs4
3
4 htmlFile = open('myhtml.html', encoding='utf-8')
5 objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6 print("打印title = ", objSoup.title)
7 print("title内容 = ", objSoup.title.text)
```

执行结果

```
===== RESTART: D:/Python/ch21/ch21_14.py
打印title = <title>洪锦魁著作</title>
title内容 = 洪锦魁著作
>>>
```

21-4-5 传回所找寻的第一个符合的标签find()

这个函数可以找寻HTML文件内第一个符合的标签内容，例如，find('h1')是要找第一个h1的标签。如果找到了就传回该卷标字符串，我们可以使用text属性获得内容，如果没找到就传回None。

程序实例ch21_15.py： 传回第一个<h1>标签。

```
1 # ch21_15.py
2 import bs4
3
4 htmlFile = open('myhtml.html', encoding='utf-8')
5 objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6 objTag = objSoup.find('h1')
7 print("数据类型= ", type(objTag))
8 print("打印Tag = ", objTag)
9 print("Tag内容 = ", objTag.text)
```

执行结果

```
===== RESTART: D:/Python/ch21/ch21_15.py
数据类型= <class 'bs4.element.Tag'>
打印Tag = <h1 id="author">洪锦魁</h1>
Tag内容 = 洪锦魁
>>>
```

21-4-6 传回所找寻的所有符合的标签find_all()

这个函数可以找寻HTML文件内所有符合的标签内容，例如，find_all('h1')是要找所有h1的标签。如果找到了就传回该标签列表，如果没找到就传回空列表。

程序实例ch21_16.py： 传回所有的<h1>标签。

```
1 # ch21_16.py
2 import bs4
3
4 htmlFile = open('myhtml.html', encoding='utf-8')
5 objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6 objTag = objSoup.find_all('h1')
7 print("数据类型 = ", type(objTag))      # 打印数据类型
8 print("打印Tag列表 = ", objTag)         # 打印列表
9 print("以下是打印列表元素：")
10 for data in objTag:                     # 打印列表元素内容
11     print(data.text)
```

执行结果

```
===== RESTART: D:/Python/ch21/ch21_16.py =====
数据类型 = <class 'bs4.element.ResultSet'>
打印Tag列表 = [<h1 id="author">洪锦魁</h1>, <h1 id="content">一个人的极境旅行 -
南极大陆北极海</h1>, <h1 id="content">HTML5+CSS3王者归来</h1>]
以下是打印列表元素：
洪锦魁
一个人的极境旅行 - 南极大陆北极海
HTML5+CSS3王者归来
>>>
```

21-4-7 认识HTML元素上下文属性与getText()

HTML元素内容的属性有下列3种。

textContent：内容，不含任何标签码。

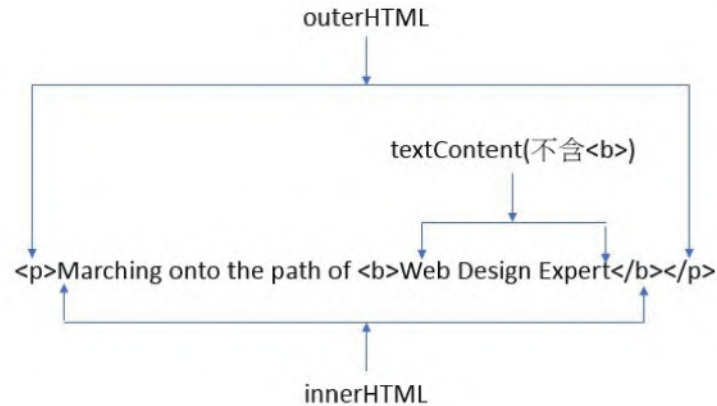
innerHTML：元素内容，含子卷标码，但是不含本身标签码。

outerHTML：元素内容，含子卷标码，也含本身标签码。

如果有一个元素内容如下：

<p>Marching onto the path of Web Design Expert</p>

则上述3个属性的观念与内容分别如下：



`textContent` : Web Design Expert

`innerHTML` : Marching onto the path of `<b>Web Design Expert</b>`

`outerHTML` : `<p>Marching onto the path of <b>Web Design Expert</b></p>`

当使用BeautifulSoup模块解析HTML文件时，如果传回的是列表，也可以配合索引应用`getText()`取得列表元素内容，所取得的内容是`textContent`。意义与21-4-4小节的`text`属性相同。

程序实例ch21_17.py： 使用`getText()`重新扩充设计ch21_16.py。

```

1  # ch21_17.py
2  import bs4
3
4  htmlFile = open('myhtml.html', encoding='utf-8')
5  objSoup = bs4.BeautifulSoup(htmlFile, 'xml')
6  objTag = objSoup.find_all('h1')
7  print("数据类型 = ", type(objTag))      # 打印数据类型
8  print("打印Tag列表 = ", objTag)         # 打印列表
9  print("\n使用Text属性打印列表元素：")
10 for data in objTag:                      # 打印列表元素内容
11     print(data.text)
12 print("\n使用getText()方法打印列表元素：")
13 for i in range(len(objTag)):
14     print(objTag[i].getText())

```

执行结果


```

===== RESTART: D:/Python/ch21/ch21_17.py =====
数据类型 = <class 'bs4.element.ResultSet'>
打印Tag列表 = [<hl id="author">洪锦魁</hl>, <hl id="content">一个人的极境旅行 - 南极大陆北极海</hl>, <hl id="content">HTML5+CSS3王者归来</hl>]

使用Text属性打印列表元素：
洪锦魁
一个人的极境旅行 - 南极大陆北极海
HTML5+CSS3王者归来

使用getText()方法打印列表元素：
洪锦魁
一个人的极境旅行 - 南极大陆北极海
HTML5+CSS3王者归来
>>>

```

21-4-8 select()

select()主要是以CSS选择器(selector)的观念寻找元素，如果找到，回传的是列表(list)，如果找不到则传回空列表。下列是使用实例：

objSoup.select('p')：找寻所有<p>卷标的元素。

objSoup.select ('img')：找寻所有卷标的元素。

objSoup.select ('. happy')：找寻所有CSS class属性为happy的元素。

objSoup.select ('# author')：找寻所有CSS id属性为author的元素。

objSoup.select ('p #author')：找寻所有<p>且id属性为author的元素。

objSoup.select ('p .happy')：找寻所有<p>且class属性为happy的元素。

objSoup.select ('div strong')：找寻所有在<section>元素内的元素。

objSoup.select ('div > strong')：找寻所有在<section>内的元素，中间没有其他元素。

objSoup.select ('input[name]'): 找寻所有<input>卷标且有name属性的元素。

程序实例ch21_18.py: 找寻id属性是author的内容。

```
1 # ch21_18.py
2 import bs4
3
4 htmlFile = open('myhtml.html', encoding='utf-8')
5 objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6 objTag = objSoup.select('#author')
7 print("数据类型 = ", type(objTag))          # 打印数据类型
8 print("列表长度 = ", len(objTag))           # 打印列表长度
9 print("元素数据类型 = ", type(objTag[0]))    # 打印元素数据类型
10 print("元素内容 = ", objTag[0].getText())   # 打印元素内容
```

执行结果

```
===== RESTART: D:/Python/ch21/ch21_18.py =====
数据类型 = <class 'list'>
列表长度 = 1
元素数据类型 = <class 'bs4.element.Tag'>
元素内容 = 洪锦魁
>>>
```

上述在使用时如果将元素内容当作参数传给str(), 将会传回含开始和结束卷标的字符串。

程序实例ch21_19.py: 将解析的列表元素传给str(), 同时打印执行结果。

```
1 # ch21_19.py
2 import bs4
3
4 htmlFile = open('myhtml.html', encoding='utf-8')
5 objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6 objTag = objSoup.select('#author')
7 print("列出列表元素的数据类型 = ", type(objTag[0]))
8 print(objTag[0])
9 print("列出str()转换过的数据类型 = ", type(str(objTag[0])))
10 print(str(objTag[0]))
```

执行结果

```
===== RESTART: D:/Python/ch21/ch21_19.py =====
列出列表元素的数据类型 = <class 'bs4.element.Tag'>
<h1 id="author">洪锦魁</h1>
列出str()转换过的数据类型 = <class 'str'>
<h1 id="author">洪锦魁</h1>
>>>
```

尽管上述第8行与第10行打印的结果相同，但是第10行是纯字符串，第8行是卷标字符串，意义不同，未来可使用的方法也不同，将在21-4-9小节解说。

列表元素有attrs属性，如果使用此属性可以得到一个字典结果。

程序实例ch21_20.py： 将attrs属性应用在列表元素，列出字典结果。

```
1 # ch21_20.py
2 import bs4
3
4 htmlFile = open('myhtml.html', encoding='utf-8')
5 objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6 objTag = objSoup.select('#author')
7 print(str(objTag[0].attrs))
```

执行结果

```
===== RESTART: D:/Python/ch21/ch21_20.py =====
{'id': 'author'}
>>>
```

在HTML文件中常常可以看到卷标内有子卷标，如果查看myhtml.html的第21行，可以看到<p>卷标内有卷标，碰上这种状况若是打印列表元素内容时，可以看到子标签存在。但是，若是使用getText()取得元素内容，可以得到没有子卷标的字符串内容。

程序实例ch21_21.py： 搜寻<p>标签，最后列出列表内容与不含子卷标的元素内容。

```
1 # ch21_21.py
2 import bs4
3
4 htmlFile = open('myhtml.html', encoding='utf-8')
5 objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6 pObjTag = objSoup.select('p')
7 print("含<p>标签的列表长度 = ", len(pObjTag))
8 for i in range(len(pObjTag)):
9     print(str(pObjTag[i]))          # 内部有子卷标<strong>字符串
10    print(pObjTag[i].getText())      # 没有子标签
11    print(pObjTag[i].text)           # 没有子标签
```

执行结果

```

===== RESTART: D:/Python/ch21/ch21_21.py =====
含<p>标签的列表长度 = 2
<p>2015/2016年<strong>洪棉魁</strong>一个人到南极</p>
2015/2016年洪棉魁一个人到南极
2015/2016年洪棉魁一个人到南极
<p>本书讲解网页设计使用HTML5+CSS3</p>
本书讲解网页设计使用HTML5+CSS3
本书讲解网页设计使用HTML5+CSS3
>>>

```

21-4-9 卷标字符串的get()

假设我们现在搜寻标签，请参考下列实例。

程序实例ch21_22.py： 搜寻标签，同时列出结果。

```

1 # ch21_22.py
2 import bs4
3
4 htmlFile = open('myhtml.html', encoding='utf-8')
5 objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6 imgTag = objSoup.select('img')
7 print("含<img>标签的列表长度 = ", len(imgTag))
8 for i in range(len(imgTag)):
9     print(imgTag[i])

```

执行结果

```

===== RESTART: D:/Python/ch21/ch21_22.py =====
含<img>标签的列表长度 = 3



>>>

```

是一个插入图片的卷标，没有结束卷标，所以没有内文，如果读者尝试使用text属性打印内容“print(imgTag[0].text)”将看不到任何结果。对网络爬虫设计很重要，因为可以由此获得网页的图文件信息。从上述执行结果可以看到，对我们而言很重要的是卷标内的属性src，这个属性设定了图片路径。这个时候我们可以使用卷标字符串的get()取得。

程序实例ch21_23.py： 扩充ch21_22.py，取得myhtml.html的所有图片文件。

```

1 # ch21_23.py
2 import bs4
3
4 htmlFile = open('myhtml.html', encoding='utf-8')
5 objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6 imgTag = objSoup.select('img')
7 print("含<img>标签的列表长度 = ", len(imgTag))
8 for i in range(len(imgTag)):
9     print("打印标签列表 = ", imgTag[i])
10    print("打印图片文件 = ", imgTag[i].get('src'))

```

执行结果

```

===== RESTART: D:/Python/ch21/ch21_23.py =====
含<img>标签的列表长度 = 3
打印标签列表 = 
打印图片文件 = hung.jpg
打印标签列表 = 
打印图片文件 = travel.jpg
打印标签列表 = 
打印图片文件 = html5.jpg
>>>

```

上述程序最重要是第10行的

21-5 网络爬虫实战

其实笔者已经用HTML文件解说网络爬虫的基本原理了，而在真实的网络世界一切比上述实例复杂困难。

程序实例ch21_24.py： 这个程序将至上奇信息网页下载所有图片，所下载的图片将放到目前文件夹的out21_24内，上奇信息网址如下：

<http://www.grandtech.info>

```

1  # ch21_24.py
2  import bs4, requests, os
3
4  url = 'http://www.grandtech.info/'          # 上奇信息网页
5  html = requests.get(url)
6  print("网页下载中 ...")
7  html.raise_for_status()                    # 验证网页是否下载成功
8  print("网页下载完成")
9
10 destDir = 'out21_25'                        # 设定未来存储图片的文件夹
11 if os.path.exists(destDir) == False:
12     os.mkdir(destDir)                      # 建立文件夹供未来存储图片
13
14 objSoup = bs4.BeautifulSoup(html.text, 'lxml') # 建立BeautifulSoup对象
15
16 imgTag = objSoup.select('img')              # 搜寻所有图片文件
17 print("搜寻到的图片数量 = ", len(imgTag))   # 列出搜寻到的图片数量
18 if len(imgTag) > 0:
19     for i in range(len(imgTag)):
20         imgUrl = imgTag[i].get('src')        # 取得图片的路径
21         print("%s 图片下载中 ..." % imgUrl)
22         finUrl = url + imgUrl                # 取得图片在Internet上的路径
23         print("%s 图片下载中 ..." % finUrl)
24         picture = requests.get(finUrl)       # 下载图片
25         picture.raise_for_status()           # 验证图片是否下载成功
26         print("%s 图片下载成功" % finUrl)
27
28 # 先开启文件, 再存储图片
29 pictFile = open(os.path.join(destDir, os.path.basename(imgUrl)), 'wb')
30 for diskStorage in picture.iter_content(10240):
31     pictFile.write(diskStorage)
32 pictFile.close()                            # 关闭文件

```

执行结果

```

===== RESTART: D:/Python/ch21/ch21_24.py =====
网页下载中 ...
网页下载完成
搜寻到的图片数量 = 74
images/merrychristmas_logo-bg.jpg 图片下载中 ...
http://www.grandtech.info/images/merrychristmas_logo-bg.jpg 图片下载中 ...
http://www.grandtech.info/images/merrychristmas_logo-bg.jpg 图片下载成功
images/btn/btn01-2.jpg 图片下载中 ...
http://www.grandtech.info/images/btn/btn01-2.jpg 图片下载中 ...
http://www.grandtech.info/images/btn/btn01-2.jpg 图片下载成功
images/btn/btn02-1.jpg 图片下载中 ...
http://www.grandtech.info/images/btn/btn02-1.jpg 图片下载中 ...
http://www.grandtech.info/images/btn/btn02-1.jpg 图片下载成功
images/btn/btn03-1.jpg 图片下载中 ...
http://www.grandtech.info/images/btn/btn03-1.jpg 图片下载中 ...
http://www.grandtech.info/images/btn/btn03-1.jpg 图片下载成功
images/btn/btn04-1.jpg 图片下载中 ...

```

上述程序大部分皆已做过解说，最重要是第29行，因为所搜寻到的图片imgURL可能是位于其他子文件夹，它的文件名前方有目录路径，os.path.basename()主要是忽略目录路径 传回程序文件名，这样就可以避免因为需要在desrDir下打开不存在的文件夹而产生错误。例如、imgUrl是\image\sample.jpg，如果没有os.path.basename()，传回结果是：

out21_24\image

\sample.jpg # 因为image文件夹不存在

打开时会有错误

有了os.path.basename()，传回结果是：

```
out21_24\sample.jpg # 可以正常打开此文件
```

在21-2-6小节笔者有介绍有些网站的服务器会挡住网络爬虫的需求，所以必须在Python程序前方加上伪装成服务器的header定义，当时有用程序实例ch21_9_2.py设计一个程序，下列是这个程序的实际应用。

程序实例ch21_25.py： 笔者将myhtml.html文件放上网络，改名index.html，这个程序会下载这个网页的图片，现在可以使用下列网址浏览此网页：

<http://aaa.24ht.com.tw/>

```
1 # ch21_25.py
2 import bs4, requests, os
3
4 headers = { 'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64)\
5             AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101\
6             Safari/537.36', }
7 url = 'http://aaa.24ht.com.tw/' # 这个服务器会挡住网页
8 html = requests.get(url, headers=headers)
9 print("网页下载中 ...")
10 html.raise_for_status() # 验证网页是否下载成功
11 print("网页下载完成")
12
13 destDir = 'out21_25' # 设定存储文件夹
14 if os.path.exists(destDir) == False:
15     os.mkdir(destDir) # 建立目录供未来存储图片
16
17 objSoup = bs4.BeautifulSoup(html.text, 'lxml') # 建立BeautifulSoup对象
18
19 imgTag = objSoup.select('img') # 搜寻所有图片文件存储
20 print("搜寻到的图片数量 = ", len(imgTag)) # 列出搜寻到的图片数量
21 if len(imgTag) > 0: # 如果找到图片则执行下载与存储
22     for i in range(len(imgTag)): # 循环下载图片与
23         imgUrl = imgTag[i].get('src') # 取得图片的路径
24         print("%s 图片下载中 ..." % imgUrl)
25         finUrl = url + imgUrl # 取得图片在Internet上的路径
26         print("%s 图片下载中 ..." % finUrl)
27         picture = requests.get(finUrl, headers=headers) # 下载图片
28         picture.raise_for_status() # 验证图片是否下载成功
29         print("%s 图片下载成功" % finUrl)
30
31 # 先开启文件，再存储图片
32 pictFile = open(os.path.join(destDir, os.path.basename(imgUrl)), 'wb')
33 for diskStorage in picture.iter_content(10240):
34     pictFile.write(diskStorage)
35 pictFile.close() # 关闭文件
```

执行结果 所下载的图片会放在out21_25文件夹。

```

===== RESTART: D:/Python/ch21/ch21_25.py =====
网页下载中 ...
网页下载完成
搜寻到的图片数量 = 3
hung.jpg 图片下载中 ...
http://aaa.24ht.com.tw/hung.jpg 图片下载中 ...
http://aaa.24ht.com.tw/hung.jpg 图片下载成功
travel.jpg 图片下载中 ...
http://aaa.24ht.com.tw/travel.jpg 图片下载中 ...
http://aaa.24ht.com.tw/travel.jpg 图片下载成功
html5.jpg 图片下载中 ...
http://aaa.24ht.com.tw/html5.jpg 图片下载中 ...
http://aaa.24ht.com.tw/html5.jpg 图片下载成功
>>>

```

程序实例ch21_26.py：找出台湾彩券公司106000085期威力彩开奖结果。这个程序在设计时，第12行我们列出先找寻Class是“contents_box02”，因为我们发现这里有记载106000085期的开奖结果。这个程序会随时间不同而有不同的日期期数开奖结果。



```

626 <div class="dotted01"></div>
627 <!-- *****威力彩區境*****-->
628 <div class="contents_box02">
629 <div id="contents_logo_02"></div><div class="contents_mine_tx02"><span
class="font_black15">106/10/23<span>第106000085期 </span><span class="font_red14">a
href="Result_all.aspx#01">開獎結果</a></span></div><div class="contents_mine_tx04">開出順序<br>大小順
序<br>第二區</div><div class="ball_tx ball_green">11 </div><div class="ball_tx ball_green">35 </div>
<div class="ball_tx ball_green">17 </div><div class="ball_tx ball_green">33 </div><div class="ball_tx
ball_green">30 </div><div class="ball_tx ball_green">10 </div><div class="ball_tx ball_green">10
</div><div class="ball_tx ball_green">11 </div><div class="ball_tx ball_green">17 </div><div><div
class="ball_tx ball_green">30 </div><div class="ball_tx ball_green">33 </div><div class="ball_tx
ball_green">35 </div><div class="ball_red">08 </div>
630 </div>
631 <div class="dotted02"></div>
632 <!-- *****38樂合彩區境*****-->
633 <div class="contents_box02">

```

结果程序第13行发现有4组Class是“contents_box02”，程序第14和15行则列出这4组列表。

```

1 # ch21_26.py
2 import bs4, requests
3
4 url = 'http://www.taiwanlottery.com.tw'
5 html = requests.get(url)
6 print("网页下载中 ...")
7 html.raise_for_status() # 验证网页是否下载成功
8 print("网页下载完成")
9
10 objSoup = bs4.BeautifulSoup(html.text, 'lxml') # 建立BeautifulSoup对象
11
12 dataTag = objSoup.select('.contents_box02') # 寻找class是contents_box02
13 print("列表长度", len(dataTag))
14 for i in range(len(dataTag)): # 列出含contents_box02的列表
15     print(dataTag[i])
16
17 # 找寻开出顺序与大小顺序的球
18 balls = dataTag[0].find_all('div', {'class': 'ball_tx ball_green'})
19 print("开出顺序：", end='')
20 for i in range(6): # 前6球是开出顺序
21     print(balls[i].text, end=' ')
22
23 print("\n大小顺序：", end='')
24 for i in range(6, len(balls)): # 第7球以后是大小顺序
25     print(balls[i].text, end=' ')
26
27 # 找出第二区的红球
28 redball = dataTag[0].find_all('div', {'class': 'ball_red'})
29 print("\n第二区：", redball[0].text)

```

执行结果

```

===== RESTART: D:\Python\ch21\ch21_27.py =====
网页下载中 ...
网页下载完成
序列长度 4
<div class="contents_box02">
<div id="contents_logo_02"></div><div class="contents_mine_tx02"><span class="font_black15">106/10/23 第106000085期 </
span><span class="font_red14"><a href="Result_all.aspx#01">開獎結果</a></span></div><div class="contents_mine_tx04">開
出順序<br/>大小順序<br/>特別號</div><div class="ball_tx ball_green">11 </div><div class="ball_tx ball_green">35 </div>
<div class="ball_tx ball_green">17 </div><div class="ball_tx ball_green">33 </div><div class="ball_tx ball_green">30 </
div><div class="ball_tx ball_green">10 </div><div class="ball_tx ball_green">10 </div><div class="ball_tx ball_green">11 </div>
<div class="ball_tx ball_green">17 </div><div class="ball_tx ball_green">30 </div><div class="ball_tx ball_g
reen">33 </div><div class="ball_tx ball_green">35 </div><div class="ball_red">08 </div>
</div>
<div class="contents_box02">
<div id="contents_logo_03"></div><div class="contents_mine_tx02"><span class="font_black15">106/10/23 第106000085期 </
span><span class="font_red14"><a href="Result_all.aspx#07">開獎結果</a></span></div><div class="contents_mine_tx04">開
出順序<br/>大小順序<br/>特別號</div><div class="ball_tx ball_green">11 </div><div class="ball_tx ball_green">35 </div><div class="
ball_tx ball_green">17 </div><div class="ball_tx ball_green">33 </div><div class="ball_tx ball_green">30 </div><div c
lass="ball_tx ball_green">10 </div><div class="ball_tx ball_green">10 </div><div class="ball_tx ball_green">11 </div>
<div class="ball_tx ball_green">17 </div><div class="ball_tx ball_green">30 </div><div class="ball_tx ball_green">33 </
div><div class="ball_tx ball_green">35 </div>
</div>
<div class="contents_box02">
<div id="contents_logo_04"></div><div class="contents_mine_tx02"><span class="font_black15">106/10/20 第106000088期 </
span><span class="font_red14"><a href="Result_all.aspx#02">開獎結果</a></span></div><div class="contents_mine_tx04">開
出順序<br/>大小順序<br/>特別號</div><div class="ball_tx ball_yellow">19 </div><div class="ball_tx ball_yellow">42 </di
v><div class="ball_tx ball_yellow">03 </div><div class="ball_tx ball_yellow">08 </div><div class="ball_tx ball_yellow"
>26 </div><div class="ball_tx ball_yellow">18 </div><div class="ball_tx ball_yellow">03 </div><div class="ball_tx ball
_yellow">08 </div><div class="ball_tx ball_yellow">18 </div><div class="ball_tx ball_yellow">19 </div><div class="ball
_tx ball_yellow">26 </div><div class="ball_tx ball_yellow">42 </div><div class="ball_red">17 </div>
</div>
<div class="contents_box02">
<div id="contents_logo_05"></div><div class="contents_mine_tx02"><span class="font_black15">106/10/20 第106000088期 </
span><span class="font_red14"><a href="Result_all.aspx#08">開獎結果</a></span></div><div class="contents_mine_tx04">開
出順序<br/>大小順序<br/>特別號</div><div class="ball_tx ball_yellow">19 </div><div class="ball_tx ball_yellow">42 </div><div class="
ball_tx ball_yellow">03 </div><div class="ball_tx ball_yellow">08 </div><div class="ball_tx ball_yellow">26 </div>
<div class="ball_tx ball_yellow">18 </div><div class="ball_tx ball_yellow">03 </div><div class="ball_tx ball_yellow">08
</div><div class="ball_tx ball_yellow">18 </div><div class="ball_tx ball_yellow">19 </div><div class="ball_tx ball_yel
low">26 </div><div class="ball_tx ball_yellow">42 </div>
</div>
</div>
開出順序： 11 35 17 33 30 10
大小順序： 10 11 17 30 33 35
第二區： 08
>>>

```

由于我们发现106000085期威力彩是在第一个列表，所以程序第18行，使用下列指令。

```
balls = dataTag[0].find_all('div', {'class': 'ball_tx ball_green'})
```

dataTag[0]代表找寻第1组列表元素，find_all()是找寻所有标签是‘div’，此标签类别class是“ball_tx ball_green”的结果。

经过这个搜寻可以得到balls列表，然后第20和21行列出开球顺序。程序第24和25行列出号码球的大小顺序。

程序第28行也可以改用find()，因为只有一个红球是特别号。这是找寻所有卷标是‘div’，此标签类别class是“ball_red”的结果。

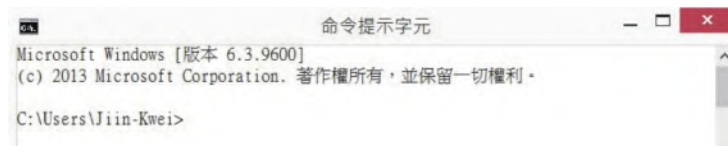
21-6 命令行窗口

其实一般的计算机用户是不会用到命令行窗口，这是最早期DOS(Disk Operating System)操作系统时的环境，现在大多数应用程序在安装时，已经将应用程序打包成一个图标，只要点选图标即可操作。但是，如果想要成为计算机高手，常会需要额外安装一些应用软件，这些应用软件需要在命令行窗口安装或设定，本节笔者将讲解Python程序在命令行窗口执行的方法，以及说明程序执行的参数。

Windows 7是在开始菜单内，输入CMD，回车。



执行后可以看到下列命令行窗口。



我们除了可以在Python的IDLE窗口执行程序，也可以在命令行环境执行Python程序，假设要执行的程序是d:\Python\ch21\ch21_27.py(这是笔者目前程序所在位置)，方法如下：

```
python安装路径\python d:\Python\ch21\ch21_27.py
```

如果程序执行时有参数，则参数在空一格后，放在右边，如下所示：

```
python安装路径\python d:\Python\ch21\ch21_27.py 参数1 ...  
参数n
```

其实在Python程序设计中d:\Python\ch21\ch21_27.py会被当作命令提示列表 的第0个元素，如果有其他参数存在，则会依次当作第1个元素，……接着笔者将测试这个观念，首先要导入sys模块，如下所示：

```
import sys
```

当我们导入上述模块后，命令提示列表 的名称是sys.argv。

程序实例ch21_27.py： 打印命令提示列表的第0个元素，这个程序需在命令行 窗口执行。

```
1 # ch21_27.py  
2 import sys  
3  
4 print(sys.argv[0])
```

执行结果



```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\python d:\Python\ch21\ch21_27.py  
d:\Python\ch21\ch21_27.py
```

习题

1. 请使用webbrowser()打开自己学校的网页。
2. 请提取自己学校的网页。
3. 请提取自己学校网页的所有图片。

第22章 Selenium网络爬虫的王者

在21-2-5小节笔者有介绍有些网页服务器会阻挡网络爬虫读取网页内容，我们可以使用headers的定义将爬虫程序伪装成浏览器，这样我们克服了读取网页内容的障碍。

Selenium功能可以控制浏览器，所以当使用Selenium当爬虫工具时，网络服务器会认为来读取数据的是浏览器，所以不会有被阻挡无法读取网页HTML原始文件的问题。当然Selenium功能不仅如此，可以使用它单击链接，填写登录信息，甚至订票系统、抢购系统等。

22-1 顺利使用Selenium工具前的安装工作

如果想要在Windows系统内顺利使用Selenium执行工作，必须安装下列3项工具以及一个设定。

- ① Selenium工具。
- ② 浏览器，使用Selenium市面上最常见是安装Firefox，也可以是Chrome或IE，本书将以Firefox为主要说明。另外，也会说明安装Chrome方式。
- ③ 驱动程序，这是指Selenium驱动浏览器的程序，其实这部分信息很重要，但是目前极少文件有说明，因此常造成读者学习上的障碍。因为依照一般说明，结果是错误信息。

22-1-1 安装Selenium

这个部分相对单纯，可以使用下列方式安装Selenium：

```
pip install selenium
```

未来程序的导入稍微不一样，如下所示：

```
from selenium

import webdriver
```

22-1-2 安装浏览器

这部分也相对单纯，可以至<https://www.mozilla.org>网页下载Firefox：

22-1-3 错误的实例

目前许多文件书写安装完上述2项后，就可以使用Selenium设计网络爬虫程序了，下列是普遍书写的第一个Selenium程序。

程序实例ch22_1.py： 建立Firefox浏览器对象，然后列出这个对象的数据类型，有关程序的细节说明，笔者将在22-2节解说。

```
1 # ch22_1.py
2 from selenium import webdriver
3
4 browser = webdriver.Firefox()
5 print(type(browser))
```

执行结果 注意，以下是Windows操作系统的执行结果。

```
===== RESTART: D:/Python/ch22/ch22_1.py =====
Traceback (most recent call last):
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-packages\selenium\webdriver\c
ommon\service.py", line 74, in start
    stdout=self.log_file, stderr=self.log_file)
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\subprocess.py", line 707, in __ini
t__
    restore_signals, start_new_session)
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\subprocess.py", line 992, in _exec
ute_child
    startupinfo)
FileNotFoundError: [WinError 2] 系统找不到指定的檔案。

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "D:/Python/ch22/ch22_1.py", line 4, in <module>
    browser = webdriver.Firefox()
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-packages\selenium\webdriver\fi
refox\webdriver.py", line 144, in __init__
    self.service.start()
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-packages\selenium\webdriver\c
ommon\service.py", line 81, in start
    os.path.basename(self.path), self.start_error_message)
selenium.common.exceptions.WebDriverException: Message: 'geckodriver' executable needs to be in PATH.

>>>
```

据说上述程序在Linux系统上可正常执行。

不过以上是笔者使用Windows操作系统的结果，错误原因是指geckodriver的驱动程序不在PATH路径内，所以产生错误。

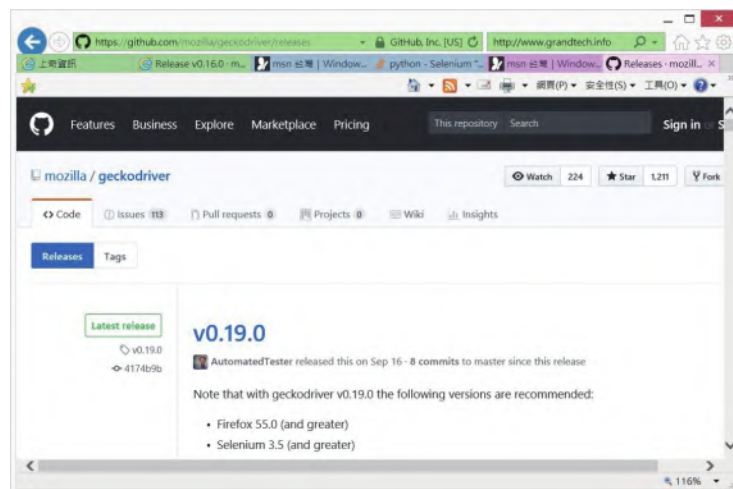
22-1-4 驱动程序的安装

驱动程序的安装分成下列步骤：

- ①安装驱动程序与解压缩。
- ②将驱动程序放在PATH路径内。
- ③将驱动程序路径放在Python程序内。

22-1-4-1以Firefox为实例

目前绝大部分的用户皆是使用Python + Selenium驱动Firefox浏览器，这时需要的驱动程序是geckodriver.exe，这个程序可以至github.com下载。



窗口往下拉可以看到不同版本与适用不同的操作系统信息。

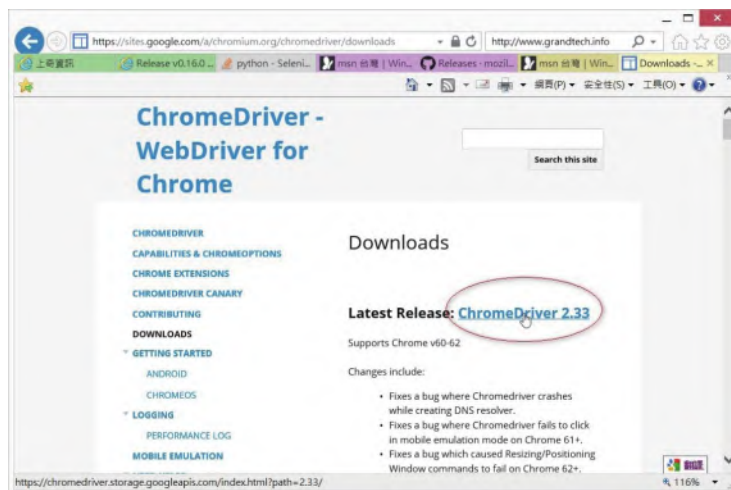
 geckodriver-v0.19.0-arm7hf.tar.gz	2.15 MB
 geckodriver-v0.19.0-linux32.tar.gz	2.19 MB
 geckodriver-v0.19.0-linux64.tar.gz	2.15 MB
 geckodriver-v0.19.0-macos.tar.gz	1.28 MB
 geckodriver-v0.19.0-win32.zip	2.62 MB
 geckodriver-v0.19.0-win64.zip	2.08 MB

v是版本信息，右边是适用操作系统的说明，由上述Windows操作系统信息可知，所下载的文件是压缩文件zip，因此压缩后需解压缩，读者可以自行依环境选择。

笔者将上述解压缩之后的geckodriver.exe放在D:/geckodriver内，未来只要将这个文件路径配合参数设定放在webdriver.Firefox()内，就可以正确执行了。

22-1-4-2以Chrome为实例

如果要使用Python + Selenium驱动Chrome浏览器，这时需要的驱动程序是chromedriver.exe，这个程序可以至下列网址下载。



这个文件下载后不用解压缩，笔者将上述解压缩之后的chromedriver.exe放在D:/geckodriver内，未来只要将这个文件路径配合参数设定放在webdriver.Chrome()内，就可以正确执行了。

22-2 获得webdriver的对象类型

使用Selenium的第一步是获得webdriver对象。

22-2-1 以Firefox浏览器为实例

程序实例ch22_2.py: 列出webdriver对象类型。

```
1 # ch22_2.py
2 from selenium import webdriver
3
4 driverPath = 'D:\geckodriver\geckodriver.exe'
5 browser = webdriver.Firefox(executable_path=driverPath)
6 print(type(browser))
```

执行结果

```
===== RESTART: D:/Python/ch22/ch22_2.py =====
<class 'selenium.webdriver.firefox.webdriver.WebDriver'>
>>>
```

这个程序在执行时，屏幕将出现D:\geckodriver\geckodriver.exe窗口，读者可以不用理会。接着会启动Firefox窗口，因为我们没有设定找寻任何网页，所以窗口是空白。不过，在Python Shell窗口可以看到程序的执行结果。

上述程序的重点是第5行，笔者将参数“executable_path=driverPath”当作参数设在webdriver.Firefox()内，dirverPath主要是设定驱动程序的位置，在第4行设定。最后程序打印出了变量browser的对象类型。

22-2-2 以Chrome浏览器为实例

程序实例ch22_3.py: 列出webdriver对象类型。

```
1 # ch22_3.py
2 from selenium import webdriver
3
4 dirverPath = 'D:\geckodriver\chromedriver.exe'
5 browser = webdriver.Chrome(dirverPath)
6 print(type(browser))
```


执行结果

```
===== RESTART: D:/Python/ch22/ch22_3.py =====<class 'selenium.webdriver.chrome.webdriver.WebDriver'>>>
```

这个程序在执行时，屏幕将出现D:\geckodriver\chromedriver.exe(这是笔者放置chromedriver.exe的文件路径)窗口，读者可以不必理会。接着会启动Chrome窗口，因为我们没有设定找寻任何网页，所以窗口是空白。不过，在Python Shell窗口可以看到程序的执行结果。

上述程序的重点是第5行，笔者将参数“driverPath”当作参数设在webdriver.Chrome()内，dirverPath主要是设定驱动程序的文件路径，在第4行设定。最后程序打印出了变量browser的对象类型。

22-3 提取网页

获得browser对象后，可以使用get()让浏览器连上网页。

程序实例ch22_4.py： 让浏览器连上网页与打印页标题。

```
1 # ch22_4.py
2 from selenium import webdriver
3
4 driverPath = 'D:\geckodriver\geckodriver.exe'
5 browser = webdriver.Firefox(executable_path=driverPath)
6 url = 'http://aaa.24ht.com.tw'
7 browser.get(url) # 网页下载至浏览器
```

执行结果



由于上述程序没有输出任何数据，所以Python Shell窗口没有任何结果，另外，由webbrowser对象启动的Firefox窗口将可以看到所加载的网页。

22-4 寻找HTML文件的元素

使用Selenium建立browser对象时，可以使用下列方法获得HTML文件的元素(WebElement)，在下列方法中find_element_*可以找到第一个符合的元素，find_elements_*则可以找到所有相符的元素同时用列表传回。

find_element_by_id(id): 传回第一个相符id的元素。

find_elements_by_id(id): 传回所有相符的id的元素，以列表方式传回。

find_element_by_class_name(name): 传回第一个相符Class的元素。

find_elements_by_class_name(name): 传回所有相符的Class的元素，以列表方式传回。

`find_element_by_name(name)`: 传回第一个相符name属性的元素。

`find_elements _by_name(name)`: 传回所有相符的name属性的元素，以列表方式传回。

`find_element_by_css_selector(selector)`: 传回第一个相符CSS selector的元素。

`find_elements _by_css_selector(selector)`: 传回所有相符的CSS selector的元素，以列表方式传回。

`find_element_by_partial_link_text(text)`: 传回第一个内含有text的<a>元素。

`find_elements _by _partial_link_text(text)`: 传回所有内含相符text的<a>元素，以列表方式传回。

`find_element_by_link_text(text)`: 传回第一个完全相同text的<a>元素。

`find_elements _by_link_text(text)`: 传回所有完全相同text的<a>元素，以列表方式传回。

`find_element_by_tag_name(name)`: 不区分大小写，传回第一个相符name的元素，例如，<p>与<P>是一样。

`find_elements _by_tag_name(name)`: 不区分大小写，传回所有相符的name的元素，以列表方式传回，例如，<p>与<P>是一样。

上述方法如果没有找到相符，会产生NoSuchElementException异常，如果我们期待没找到时程序不要列出错误而结束，可以使用try ... except执行例外处理。

找到HTML元素对象后，可以使用下列方式方法或属性获得HTML元素对象的内容。

tag_name: 元素名称。

text: 元素内容。

location: 这是字典，内含有x和y键值，表示元素在页面上的坐标。

clear(): 可以删除在文字(text)字段或文字区域(textarea)字段的文字。

get_attribute(name): 可以获得这个元素name属性的值。

is_displayed(): 如果元素可以看到传回True，否则传回False。

is_enabled(): 如果元素是可以立即使用则传回True，否则传回False。

is_selected(): 如果元素的复选框有勾选则传回True，否则传回False。

程序实例ch22_5.py: 列出找不到元素，造成程序结束的实例。

```
1 # ch22_5.py
2 from selenium import webdriver
3
4 driverPath = 'D:\geckodriver\geckodriver.exe'
5 browser = webdriver.Firefox(executable_path=driverPath)
6 url = 'http://aaa.24ht.com.tw'
7 browser.get(url)           # 网页下载至浏览器
8
9 tag = browser.find_element_by_id('main')
10 print(tag.tag_name)
```

执行结果

```

===== RESTART: D:/Python/ch22/ch22_5.py =====
Traceback (most recent call last):
  File "D:/Python/ch22/ch22_5.py", line 9, in <module>
    tag = browser.find_element_by_id('main')
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 341, in find_element_by_id
    return self.find_element(by=By.ID, value=id_)
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 843, in find_element
    'value': value)]['value']
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 308, in execute
    self.error_handler.check_response(response)
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-packages\selenium\webdriver\remote\errorhandler.py", line 194, in check_response
    raise exception_class(message, screen, stacktrace)
selenium.common.exceptions.NoSuchElementException: Message: Unable to locate element: [id="main"]
>>>

```

可以用下列方式解决。

程序实例ch22_6.py: 找不到符合条件的元素时，执行例外处理。

```

1 # ch22_6.py
2 from selenium import webdriver
3
4 driverPath = 'D:\geckodriver\geckodriver.exe'
5 browser = webdriver.Firefox(executable_path=driverPath)
6 url = 'http://aaa.24ht.com.tw'
7 browser.get(url) # 网页下载至浏览器
8
9 try:
10     tag = browser.find_element_by_id('main')
11     print(tag.tag_name)
12 except:
13     print("没有找到相符的元素")

```

执行结果

```

===== RESTART: D:\Python\ch22\ch22_6.py
没有找到相符的元素
>>>

```

程序实例ch22_7.py: 以http://aaa.24ht.com.tw网站为例，抓取不同元素的应用。

```

1 # ch22_7.py
2 from selenium import webdriver
3
4 driverPath = 'D:\geckodriver\geckodriver.exe'
5 browser = webdriver.Firefox(executable_path=driverPath)
6 url = 'http://aaa.24ht.com.tw'
7 browser.get(url) # 网页下载至浏览器
8
9 tag1 = browser.find_element_by_tag_name('title') # 传回<title>
10 print("标签名称 = %s, 内容是 = %s" % (tag1.tag_name, tag1.text))
11
12 tag2 = browser.find_element_by_id('author') # 传回<h1>
13 print("\n标签名称 = %s, 内容是 = %s" % (tag2.tag_name, tag2.text))
14
15 print()
16 tag3 = browser.find_elements_by_id('content') # 传回<h1>
17 for i in range(len(tag3)):
18     print("标签名称 = %s, 内容是 = %s" % (tag3[i].tag_name, tag3[i].text))
19
20 print()
21 tag4 = browser.find_elements_by_tag_name('p') # 传回<p>
22 for i in range(len(tag4)):
23     print("标签名称 = %s, 内容是 = %s" % (tag4[i].tag_name, tag4[i].text))
24
25 print()
26 tag5 = browser.find_elements_by_tag_name('img') # 传回<img>
27 for i in range(len(tag5)):
28     print("标签名称 = %s, 内容是 = %s" % (tag5[i].tag_name, tag5[i].get_attribute('src')))

```

执行结果

```
===== RESTART: D:\Python\ch22\ch22_7.py =====
标签名称 = title, 内容是 = 洪錦魁著作
标签名称 = h1, 内容是 = 洪錦魁
标签名称 = h1, 内容是 = 一個人的極端旅行・南極大陸北極海
标签名称 = h1, 内容是 = HTML5+CSS3王者歸來
标签名称 = p, 内容是 = 2015/2016年洪錦魁一個人到南極
标签名称 = p, 内容是 = 本書講解網頁設計使用HTML5+CSS3
标签名称 = img, 内容是 = http://aaa.24ht.com.tw/hung.jpg
标签名称 = img, 内容是 = http://aaa.24ht.com.tw/travel.jpg
标签名称 = img, 内容是 = http://aaa.24ht.com.tw/html5.jpg
>>>
```

22-5 用Python控制點選超链接

使用22-4节传回WebElement元素时，可以使用click()方法，如果执行此方法相当于我们再点击这个传回的元素。如果传回的元素是超链接的文字，这样可以产生按此超链接的结果。

程序实例ch22_8.py： 进入上奇信息(<http://www.grandtech.info>)网页，经过5秒后（第12行），设计程序自动點選认证考试 超链接。笔者设计程序暂停5秒，主要是让读者可以体会网页的变化。

```
1 # ch22_8.py
2 from selenium import webdriver
3 import time
4
5 driverPath = 'D:\geckodriver\geckodriver.exe'
6 browser = webdriver.Firefox(executable_path=driverPath)
7 url = 'http://www.grandtech.info'
8 browser.get(url) # 网页下载至浏览器
9
10 eleLink = browser.find_element_by_link_text('认证考试')
11 print(type(eleLink)) # 打印eleLink数据类型
12 time.sleep(5) # 暂停5秒
13 eleLink.click() # 执行超链接至书级的证书类别
```

执行结果

```
===== RESTART: D:/Python/ch22/ch22_8.py =====
<class 'selenium.webdriver.firefox.webelement.FirefoxWebElement'>
>>>
```

下列是经过5秒后，Python自行點選认证考试超链接的结果。



22-6 用Python填写窗体和送出

我们可以找寻<input>元素type是“text”或是找寻<textarea>元素，然后使用send_keys()方法，就可以填写窗体。填写完成后可以使用submit()方法，将窗体送出。

程序实例ch22_9.py： 用Python填写窗体，所填写的窗体是搜寻笔者的著作，本程序会经过5秒自动送出，笔者在执行结果中打印出了填写窗体以及送出的结果。

```

1  # ch22_9.py
2  from selenium import webdriver
3  import time
4
5  driverPath = 'D:\geckodriver\geckodriver.exe'
6  browser = webdriver.Firefox(executable_path=driverPath)
7  url = 'http://www.grandtech.info'
8  browser.get(url)          # 网页下载至浏览器
9
10 txtBox = browser.find_element_by_id('key')
11 txtBox.send_keys('洪锦魁') # 输入窗体数据
12 time.sleep(5)             # 暂停5秒
13 txtBox.submit()           # 送出窗体

```

执行结果 左下窗体“洪锦魁”是自动填写，过5秒后可以得到右下图结果。



了解上述自动填写窗体和送出功能，未来热门的演唱会门票、过年抢破头的高铁票就让Python处理吧！

22-7 用Python处理使用网页的特殊按键

在欣赏网页时，有时候我们可能需要滚动网页或使用一些特殊键，这些特殊键无法用Python输入，不过Python有提供下列模块，可方便我们操作。

```
selenium.webdriver.common.keys
```

使用这个模块前需在程序前方导入，语法如下：

```
from selenium.webdriver.common,keys import Keys
```

经上述声明后未来可以用Keys调用相关属性，下列是常用属性内容。

ENTER/RETURN：相当于键盘的Enter和Return按键。

PAGE_DOWN/PAGE_UP/HOME/END：相当于键盘的PAGE_DOWN、PAGE_UP、HOME、END。

UP/DOWN/LEFT/RIGHT：相当于键盘的上、下、左、右箭头键。

上述使用方式是在前方加上“Keys.”，例如，Keys.HOME。

程序实例ch22_10.py：这个程序在执行时，首先显示最上方的网页内容，过3秒后会往下滚动一页，再过3秒会滚动到最下方。经过3秒可以往上滚动，再过3秒可以将网页滚动到最上方。程序第10行先搜寻‘body’，这是网页设计网页主体的开始标签，相当于在网页的最上方。

```
1 # ch22_10.py
2 from selenium import webdriver
3 from selenium.webdriver.common.keys import Keys
4 import time
5
6 driverPath = 'D:\geckodriver\geckodriver.exe'
7 browser = webdriver.Firefox(executable_path=driverPath)
8 url = 'http://www.grandtech.info'
9 browser.get(url) # 网页下载至浏览器
10
11 ele = browser.find_element_by_tag_name('body')
12 time.sleep(3)
13 ele.send_keys(Keys.PAGE_DOWN) # 网页滚动到下一页
14 time.sleep(3)
15 ele.send_keys(Keys.END) # 网页滚动到底端
16 time.sleep(3)
17 ele.send_keys(Keys.PAGE_UP) # 网页滚动到上一页
18 time.sleep(3)
19 ele.send_keys(Keys.HOME) # 网页滚动到最上端
```

执行结果 每次间隔3秒，读者可以观察页面内容的滚动。

22-8 用Python处理浏览器运作

常见的运作有下列方法：

forward()：往前一页。

back()：往回一页。

refresh()：更新网页。

quit()：关闭网页，相当于关闭浏览器。

上述必须用Firefox浏览器对象启动，也就是我们本章的变量browser，例如，browser.refresh()可更新网页，browser.quit()可以关闭网页。

程序实例ch22_11.py： 更新网页与关闭网页的应用。

```
1 # ch22_11.py
2 from selenium import webdriver
3 from selenium.webdriver.common.keys import Keys
4 import time
5
6 driverPath = 'D:\geckodriver\geckodriver.exe'
7 browser = webdriver.Firefox(executable_path=driverPath)
8 url = 'http://www.grandtech.info'
9 browser.get(url)           # 网页下载至浏览器
10
11 time.sleep(3)
12 browser.refresh()         # 更新网页
13 time.sleep(3)
14 browser.quit()           # 关闭网页
```

执行结果 网页下载后3秒钟可以更新网页内容，再过3秒可以关闭浏览器。

习题

1. 请使用程序ch22_7.py的观念，下载该网页所有图片并存储。
2. 请使用Selenium观念，重新设计程序实例ch21_25.py。
3. 请使用Selenium观念，下载最新一期威力彩以及大乐透彩券。

第23章 数据图表的设计

本章所叙述的重点是数据图形的绘制，所使用的工具是matplotlib绘图库模块，使用前需先安装：

```
pip install matplotlib
```

matplotlib是一个庞大的绘图库模块，本章我们只导入其中的pyplot子模块就可以完成许多图表绘制，如下所示，未来就可以使用plt调用相关的方法。

```
import matplotlib.pyplot as plt
```

本章将叙述matplotlib的重点，更完整使用说明可以参考下列网站。

<http://matplotlib.org>

23-1 绘制简单的折线图

这一节将从最简单的折线图开始解说。

23-1-1 显示绘制的图形show()

这个show()方法主要是显示所绘制的图形，当我们绘制图形完成后，可以调用此方法。

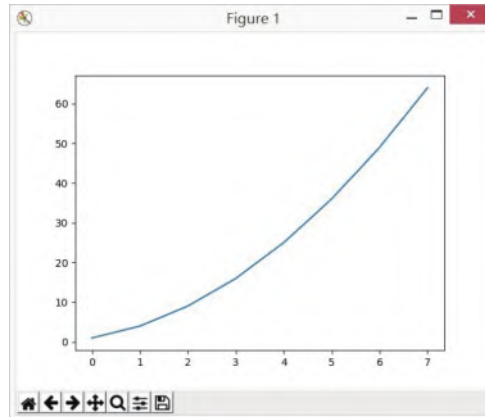
23-1-2 画线plot()

应用方式是将含数据的列表当参数传给`plot()`，列表内的数据会被视为y轴的值，x轴的值会依列表值的索引位置自动产生。

程序实例ch23_1.py： 绘制折线的应用，数据基本上是1-8的平方值序列。

```
1 # ch23_1.py
2 import matplotlib.pyplot as plt
3
4 squares = [1, 4, 9, 16, 25, 36, 49, 64]
5 plt.plot(squares)
6 plt.show()
```

执行结果

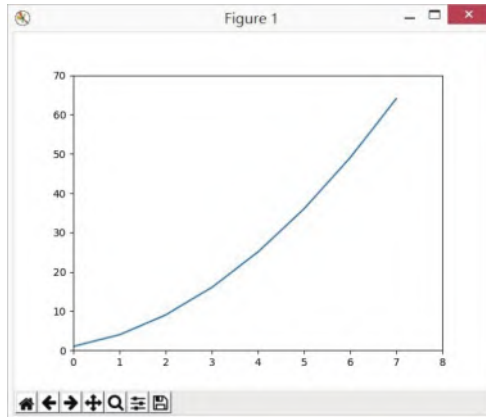


从上述执行结果可以看到左下角的轴刻度不是(0, 0)，我们可以使用`axis()`设定x, y轴的最小和最大刻度。

程序实例ch23_1_1.py： 重新设计ch23_1.py，将轴刻度x轴设为0, 8，y轴刻度设为0, 70。

```
1 # ch23_1_1.py
2 import matplotlib.pyplot as plt
3
4 squares = [1, 4, 9, 16, 25, 36, 49, 64]
5 plt.plot(squares)
6 plt.axis([0, 8, 0, 70])
7 plt.show()
```

执行结果



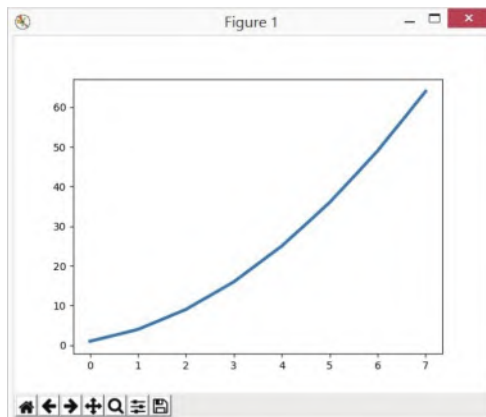
23-1-3 线条宽度linewidth

使用`plot()`时，可以多加一个`linewidth`(缩写是`lw`)参数设定线条的粗细。

程序实例`ch23_2.py`: 设定线条宽度是3。

```
1 # ch23_2.py
2 import matplotlib.pyplot as plt
3
4 squares = [1, 4, 9, 16, 25, 36, 49, 64]
5 plt.plot(squares, linewidth=3)
6 plt.show()
```

执行结果



23-1-4 标题的显示

目前matplotlib模块不支持中文显示，下列是几个图表重要的方法。

`title()`：图表标题。

`xlabel()`：x轴标题。

`ylabel()`：y轴标题。

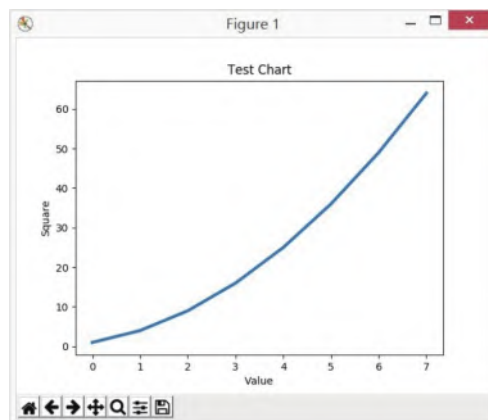
上述方法可以显示默认大小是12的字体，它的语法如下：

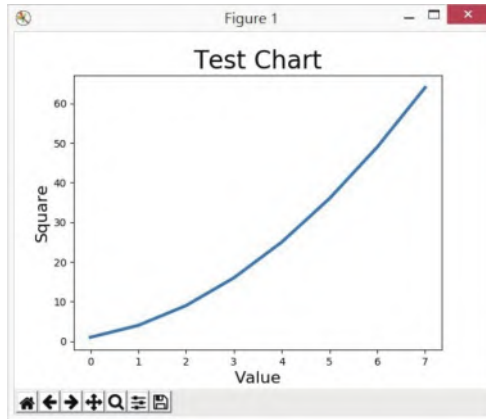
`title(标题名称, fontsize=数值大小)` # 同时可用在`xlabel()`和`ylabel()`

程序实例ch23_3.py： 使用默认字号为图表与x/y轴建立标题。

```
1 # ch23_3.py
2 import matplotlib.pyplot as plt
3
4 squares = [1, 4, 9, 16, 25, 36, 49, 64]
5 plt.plot(squares, linewidth=3)
6 plt.title("Test Chart")
7 plt.xlabel("Value")
8 plt.ylabel("Square")
9 plt.show()
```

执行结果 可参考下方左图。





程序实例ch23_4.py: 使用设定字号为图表与x/y轴建立标题。

```
1 # ch23_4.py
2 import matplotlib.pyplot as plt
3
4 squares = [1, 4, 9, 16, 25, 36, 49, 64]
5 plt.plot(squares, linewidth=3)
6 plt.title("Test Chart", fontsize=24)
7 plt.xlabel("Value", fontsize=16)
8 plt.ylabel("Square", fontsize=16)
9 plt.show()
```

执行结果 可参考上方右图。

23-1-5 坐标轴刻度的设定

在设计图表时可以使用`tick_params()`设计设定坐标轴的刻度大小、颜色 以及应用范围。

```
tick_params(axis='xx', labelsizes=xx, color='xx') # labelsizes的xx
```

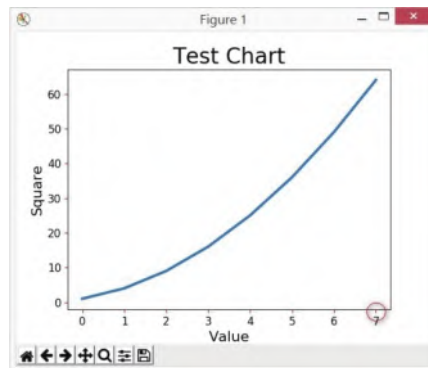
代表刻度大小

如果axis的xx是both代表应用到x和y轴，如果xx是x代表应用到x轴，如果xx是y代表应用到y轴。color则是设定刻度的线条颜色，例如，red代表红色。

程序实例ch23_5.py: 使用不同刻度与颜色的应用。

```
1 # ch23_5.py
2 import matplotlib.pyplot as plt
3
4 squares = [1, 4, 9, 16, 25, 36, 49, 64]
5 plt.plot(squares, linewidth=3)
6 plt.title("Test Chart", fontsize=24)
7 plt.xlabel("Value", fontsize=16)
8 plt.ylabel("Square", fontsize=16)
9 plt.tick_params(axis='both', labelsize=12, color='red')
10 plt.show()
```

执行结果



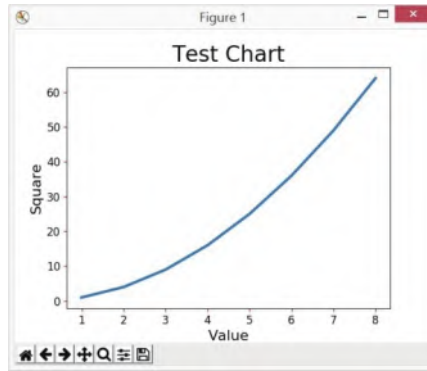
23-1-6 修订图表的起始值

从上图可以看到平方列表的值是有8个数据，依照Python语法起始数字是从0开始，所以整个数值到7结束。但是在我们日常生活呈现的报表中，通常数字是从1开始，为了要做这个修订，可以再增加一个列表，这个列表主要是设定数值索引，细节可参考下列实例的第5行。

程序实例ch23_6.py: 修订图表的起始值，读者应该注意到x轴标计从1开始。

```
1 # ch23_6.py
2 import matplotlib.pyplot as plt
3
4 squares = [1, 4, 9, 16, 25, 36, 49, 64]
5 seq = [1, 2, 3, 4, 5, 6, 7, 8]
6 plt.plot(seq, squares, linewidth=3)
7 plt.title("Test Chart", fontsize=24)
8 plt.xlabel("Value", fontsize=16)
9 plt.ylabel("Square", fontsize=16)
10 plt.tick_params(axis='both', labelsize=12, color='red')
11 plt.show()
```

执行结果



23-1-7 多组数据的应用

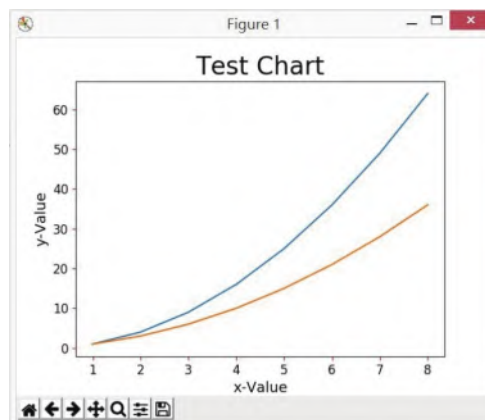
目前所有的图表皆只有一组数据，其实可以扩充多组数据，只要在`plot()`内增加数据列表参数即可。此时`plot()`的参数如下：

`plot(第一组数据, 第二组数据, ...)`

程序实例ch23_7： 设计多组数据图的应用。

```
1 # ch23_7.py
2 import matplotlib.pyplot as plt
3
4 data1 = [1, 4, 9, 16, 25, 36, 49, 64]      # data1线条
5 data2 = [1, 3, 6, 10, 15, 21, 28, 36]      # data2线条
6 seq = [1,2,3,4,5,6,7,8]
7 plt.plot(seq, data1, seq, data2)           # data1&2线条
8 plt.title("Test Chart", fontsize=24)
9 plt.xlabel("x-Value", fontsize=14)
10 plt.ylabel("y-Value", fontsize=14)
11 plt.tick_params(axis='both', labelsize=12, color='red')
12 plt.show()
```

执行结果



上述以不同颜色显示线条是系统默认，我们也可以自定义线条色彩。

23-1-8 线条色彩与样式

如果想设定线条色彩，可以在`plot()`内增加下列参数设定，下列是常见的色彩表。

色彩字符	色彩说明
‘b’	blue(蓝色)
‘c’	cyan(青色)
‘g’	green(绿色)
‘k’	black(黑色)
‘m’	magenta(品红)
‘r’	red(红色)
‘w’	white(白色)
‘y’	yellow(黄色)

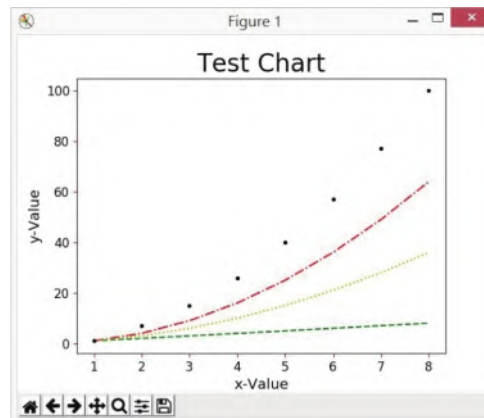
下列是常见的样式表。可以混合使用，例如，‘`r-.`’代表红色虚点线。

字符	说明
‘-’ 或 ‘solid’	这是预设实线
‘—’ 或 ‘dashed’	虚线
‘-.’ 或 ‘dashdot’	虚点线
‘:’ 或 ‘dotted’	点线
‘.’	点标记
‘,’	像素标记
‘o’	圆标记
‘v’	反三角标记
‘^’	三角标记
‘s’	方形标记
‘p’	五角标记
‘*’	星星标记
‘+’	加号标记
‘_’	减号标记

程序实例ch23_8. py: 采用不同色彩与线条样式绘制图表。

```
1 # ch23_8.py
2 import matplotlib.pyplot as plt
3
4 data1 = [1, 2, 3, 4, 5, 6, 7, 8] # data1线条
5 data2 = [1, 4, 9, 16, 25, 36, 49, 64] # data2线条
6 data3 = [1, 3, 6, 10, 15, 21, 28, 36] # data3线条
7 data4 = [1, 7, 15, 26, 40, 57, 77, 100] # data4线条
8
9 seq = [1, 2, 3, 4, 5, 6, 7, 8]
10 plt.plot(seq, data1, 'g--', seq, data2, 'r-.', seq, data3, 'y:', seq, data4, 'k.')
11 plt.title("Test Chart", fontsize=24)
12 plt.xlabel("x-Value", fontsize=14)
13 plt.ylabel("y-Value", fontsize=14)
14 plt.tick_params(axis='both', labelsize=12, color='red')
15 plt.show()
```

执行结果

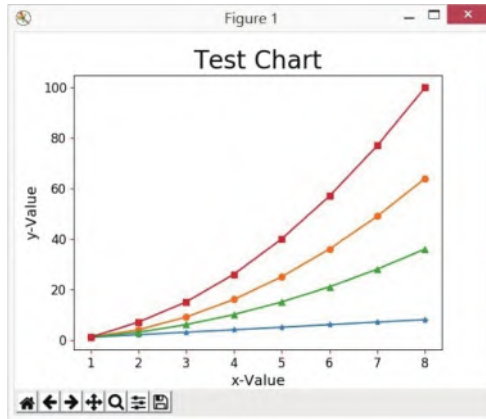


在上述第10行最右边 ‘k.’ 代表绘制黑点而不是绘制线条，由这个观念读者应该可以使用不同颜色绘制散点图(23-2节会介绍另一个方法scatter()绘制散点图)。上述格式应用是很活的，如果我们使用 ‘-*’ 可以绘制线条，同时在指定点加上星星标记。

程序实例ch23_9. py: 重新设计ch23_8. py绘制线条，同时为各个点加上标记。

```
10 plt.plot(seq, data1, '-*', seq, data2, '-o', seq, data3, '-^', seq, data4, '-s')
```

执行结果



23-1-9 刻度设计

目前所有绘制图表x轴和y轴的刻度皆是plot()方法针对所输入的参数采用默认值设定，请先参考下列实例。

程序实例ch23_10.py： 有一个假设 3大品牌车辆2018-2020的销售数据如下：

Benz 3367 4120 5539

BMW 4000 3590 4423

Lexus 5200 4930 5350

请使用上述方法将上述数据绘制成图表。

```

1 # ch23_10.py
2 import matplotlib.pyplot as plt
3
4 Benz = [3367, 4120, 5539]          # Benz线条
5 BMW = [4000, 3590, 4423]          # BMW线条
6 Lexus = [5200, 4930, 5350]        # Lexus线条
7
8 seq = [2018, 2019, 2020]          # 年度
9 plt.plot(seq, Benz, '-*', seq, BMW, '-o', seq, Lexus, '-^')
10 plt.title("Sales Report", fontsize=24)
11 plt.xlabel("Year", fontsize=14)
12 plt.ylabel("Number of Sales", fontsize=14)
13 plt.tick_params(axis='both', labelsize=12, color='red')
14 plt.show()

```

执行结果



上述程序最大的遗憾是x轴的刻度，对我们而言，其实只要有2018-2020这3个年度的刻度即可，还好可以使用pyplot模块的xticks()/yticks()分别设定x/y轴刻度，可参考下列实例。

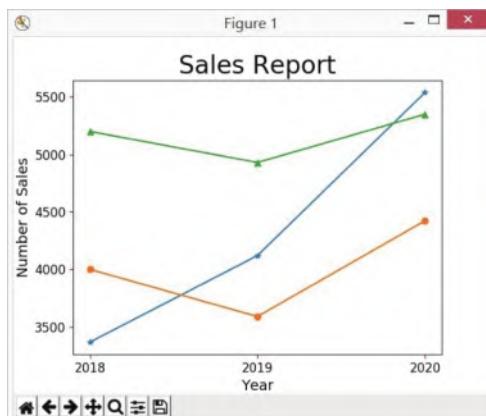
程序实例ch23_11.py： 重新设计ch23_10.py，自行设定刻度，这个程序的重点是第9行，将seq列表当参数放在plt.xticks()内。

```

1 # ch23_11.py
2 import matplotlib.pyplot as plt
3
4 Benz = [3367, 4120, 5539]          # Benz线条
5 BMW = [4000, 3590, 4423]          # BMW线条
6 Lexus = [5200, 4930, 5350]        # Lexus线条
7
8 seq = [2018, 2019, 2020]          # 年度
9 plt.xticks(seq)                    # 设定x轴刻度
10 plt.plot(seq, Benz, '-*', seq, BMW, '-o', seq, Lexus, '-^')
11 plt.title("Sales Report", fontsize=24)
12 plt.xlabel("Year", fontsize=14)
13 plt.ylabel("Number of Sales", fontsize=14)
14 plt.tick_params(axis='both', labelsize=12, color='red')
15 plt.show()

```

执行结果



23-1-10 图例legend()

程序实例ch23_11.py所建立的图表，坦白说已经很好了，缺点是缺乏各种线条代表的意义，在Excel中称图例(legend)，下列笔者将直接以实例说明。

程序实例ch23_12.py： 为ch23_11.py建立图例。

```
1 # ch23_12.py
2 import matplotlib.pyplot as plt
3
4 Benz = [3367, 4120, 5539]          # Benz线条
5 BMW = [4000, 3590, 4423]          # BMW线条
6 Lexus = [5200, 4930, 5350]        # Lexus线条
7
8 seq = [2018, 2019, 2020]          # 年度
9 plt.xticks(seq)                   # 设定x轴刻度
10 lineBenz, = plt.plot(seq, Benz, '-*', label='Benz')
11 lineBMW, = plt.plot(seq, BMW, '-o', label='BMW')
12 lineLexus, = plt.plot(seq, Lexus, '-^', label='Lexus')
13 plt.legend(handles=[lineBenz, lineBMW, lineLexus], loc='best')
14 plt.title("Sales Report", fontsize=24)
15 plt.xlabel("Year", fontsize=14)
16 plt.ylabel("Number of Sales", fontsize=14)
17 plt.tick_params(axis='both', labelsize=12, color='red')
18 plt.show()
```

执行结果



这个程序最大不同在第10-12行，以第10行解说。

```
lineBenz, = plt.plot(seq, Benz, '-*', label='Benz') # 留意  
linzBenz,
```

上述调用plt.plot()时需同时设定label，注意返回值的使用，变量右边的‘，’，最后使用第13行方式执行legend()图例的调用。其中参数loc可以设定图例的位置，可以有下列设定方式：

‘best’ : 0,
‘upper right’ : 1
‘upper left’ : 2,
‘lower left’ : 3,
‘lower right’ : 4,
‘right’ : 5, (与 ‘center right’ 相同) ‘center left’ : 6,
‘center right’ : 7,
‘lower center’ : 8,
‘upper center’ : 9,

‘center’ : 10, 如果省略loc设定, 则使用预设 ‘best’, 在应用时可以使用设定整数值, 例如, 设定loc=0与上述效果相同。若是顾虑程序可读性建议使用文字字符串方式设定, 当然也可以直接设定数字, 可以小小炫耀或迷惑不懂的人吧!

程序实例ch23_12_1.py: 省略loc设定。

```
13 plt.legend(handles=[lineBenz, lineBMW, lineLexus])
```

执行结果 与ch23_12.py相同。

程序实例ch23_12_2.py: 设定loc=0。

```
13 plt.legend(handles=[lineBenz, lineBMW, lineLexus], loc=0)
```

执行结果 与ch23_12.py相同。

程序实例ch23_12_3.py: 设定图例在右上角。

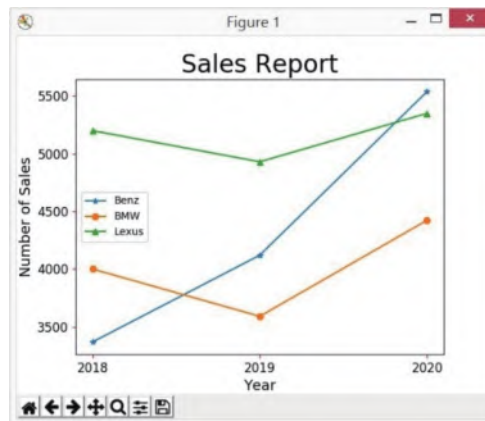
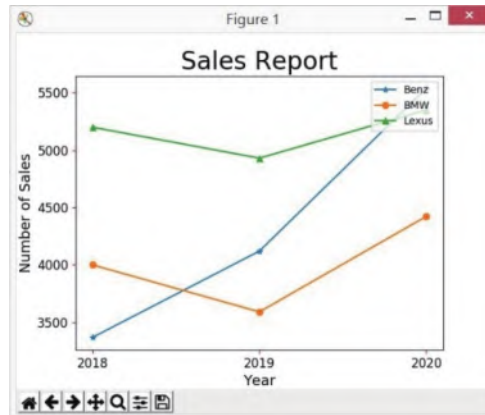
```
13 plt.legend(handles=[lineBenz, lineBMW, lineLexus], loc='upper right')
```

执行结果 下方左图。

程序实例ch23_12_4.py: 设定图例在左边中央。

```
13 plt.legend(handles=[lineBenz, lineBMW, lineLexus], loc=6)
```

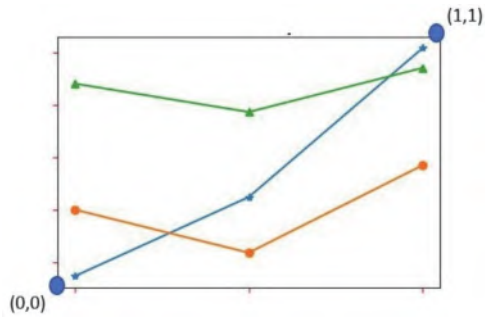
执行结果 下方右图。



经过上述解说，我们已经可以将图例放在图表内了，如果想将图例放在图表外，笔者先解释坐标，在右图表内左下角位置是(0, 0)，右上角是(1, 1)。

首先须使用**bbox_to_anchor()**当作**legend()**的一个参数，设定锚点(anchor)，也就是图例位置，例如，我们想将图例放在图表右上角外侧，需设定 **loc= 'upper left'**，然后设定 **bbox_to_anchor(1, 1)**。

程序实例ch23_12_5.py: 将图例放在图表右上角外侧。



```
13 plt.legend(handles=[lineBenz, lineBMW, lineLexus], loc='upper left',
14            bbox_to_anchor=(1,1))
```

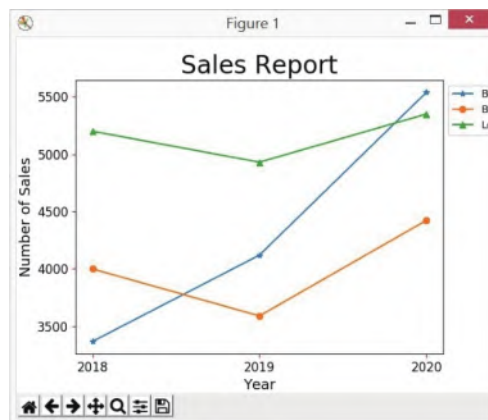
上述最大的缺点是由于图表与Figure 1的留白不足，造成无法完整显示图例。Matplotlib模块内有tight_layout()函数，可利用设定pad参数在图表与Figure 1间设定留白。

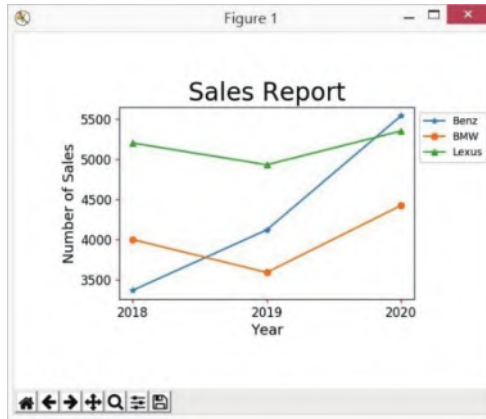
程序实例ch23_12_6.py: 设定pad=7，重新设计ch23_12_5.py。

执行结果 下方左图。

```
13 plt.legend(handles=[lineBenz, lineBMW, lineLexus], loc='upper left',
14            bbox_to_anchor=(1,1))
15 plt.tight_layout(pad=7)
```

执行结果 可参考下方右图。





很明显我们改善了图例显示不完整的问题了。如果将pad改为h_pad/w_pad可以分别设定高度/宽度的留白。

23-1-11 保存图片文件

图表设计完成，可以使用savefig()保存图片文件，这个方法需放在show()的前方，表示先存储再显示图表。

程序实例ch23_13.py: 扩充ch23_12.py，在屏幕显示图表前，先将图表存入当前文件夹的out23_13.py。执行结果读者可以在ch23文件夹看到out23_13.jpg文件。

```

1 # ch23_13.py
2 import matplotlib.pyplot as plt
3
4 Benz = [3367, 4120, 5539]          # Benz线条
5 BMW = [4000, 3590, 4423]          # BMW线条
6 Lexus = [5200, 4930, 5350]        # Lexus线条
7
8 seq = [2018, 2019, 2020]          # 年度
9 plt.xticks(seq)                   # 设定x轴刻度
10 lineBenz = plt.plot(seq, Benz, '-*', label='Benz')
11 lineBMW = plt.plot(seq, BMW, '-o-', label='BMW')
12 lineLexus = plt.plot(seq, Lexus, '-^', label='Lexus')
13 plt.legend(handles=[lineBenz, lineBMW, lineLexus])
14 plt.title("Sales Report", fontsize=24)
15 plt.xlabel("Year", fontsize=14)
16 plt.ylabel("Number of Sales", fontsize=14)
17 plt.tick_params(axis='both', labelsize=12, color='red')
18 plt.savefig('out23_13.jpg', bbox_inches='tight') # 存档
19 plt.show()

```

上述plt.savefig()第一个参数是所存的文件名，第二个参数代表将图表外多余的空间删除。

23-2 绘制散点图scatter()

23-2-1 基本散点图的绘制

绘制散点图可以使用`scatter()`，最基本语法应用如下：

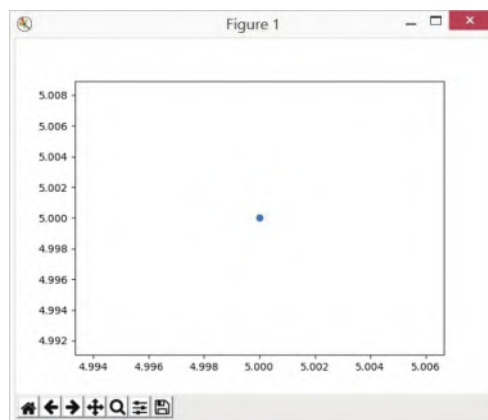
```
scatter(x, y, s, c) # 更多参数应用未来几小节会解说
```

上述相当于可以在 (x, y) 位置绘图，其中 $(0, 0)$ 位置在左下角，x轴刻度往右增加，y轴刻度往上增加。s是绘图点的大小，预设是20。c是颜色，预设是蓝色。暂时s与c皆用默认值处理，未来将一步一步解说。

程序实例ch23_14.py： 在坐标轴(5, 5)绘制一个点。

```
1 # ch23_14.py
2 import matplotlib.pyplot as plt
3
4 plt.scatter(5, 5)
5 plt.show()
```

执行结果



23-2-2 绘制系列点

如果我们想绘制系列点，可以将系列点的x轴值放在一个列表，y轴值放在另一个列表，然后将这2个列表当参数放在`scatter()`即可。

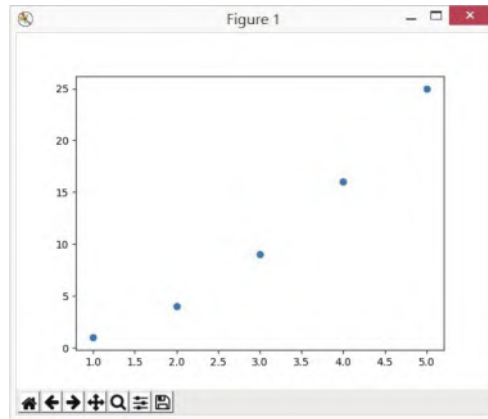
程序实例ch23_15.py： 绘制系列点的应用。

```

1 # ch23_15.py
2 import matplotlib.pyplot as plt
3
4 xpt = [1,2,3,4,5]
5 ypt = [1,4,9,16,25]
6 plt.scatter(xpt, ypt)
7 plt.show()

```

执行结果



在程序设计时，有些系列点的坐标可能是由程序产生，其实应用方式是一样的。另外，可以在`scatter()`内增加`color`（也可用`c`）参数，设定点的颜色。

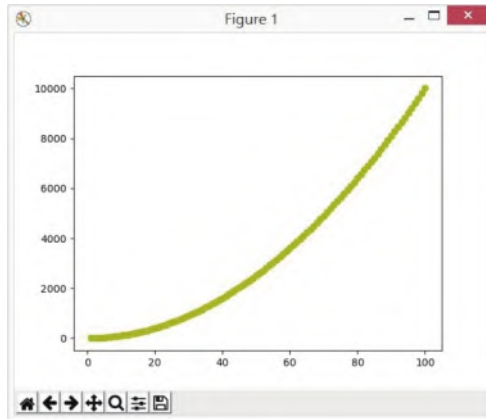
程序实例ch23_16.py： 绘制黄色的系列点，这个系列点有100个点，x轴的点由`range(1, 101)`产生，相对应y轴的值则是x的平方值。

```

1 # ch23_16.py
2 import matplotlib.pyplot as plt
3
4 xpt = list(range(1,101))      # 建立1-100序列x坐标点
5 ypt = [x**2 for x in xpt]     # 以x平方方式建立y坐标点
6 plt.scatter(xpt, ypt, color='y')
7 plt.show()

```

执行结果



上述程序第6行使用直接的指定色彩，也可以使用RGB(Red, Green, Blue)颜色模式设定色彩，RGB()内每个参数数值在0.0到1.0之间。

23-2-3 设定绘图区间

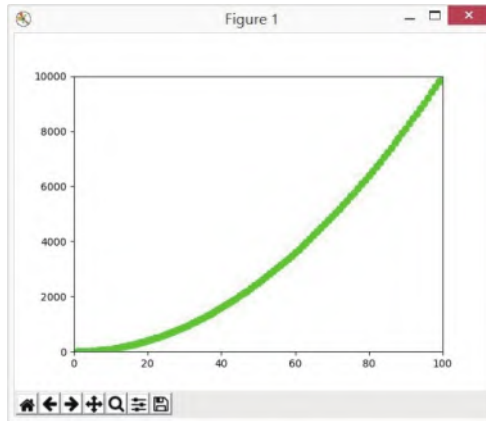
可以使用axis()设定绘图区间，语法格式如下：

axis([xmin, xmax, ymin, ymax]) # 分别代表x和y轴的最小和最大区间

程序实例ch23_17.py： 设定绘图区间为[0, 100, 0, 10000]的应用，读者可以将这个执行结果与ch23_16.py作比较。另外，第7行以不同方式建立色彩。

```
1 # ch23_17.py
2 import matplotlib.pyplot as plt
3
4 xpt = list(range(1,101))      # 建立1-100序列x坐标点
5 ypt = [x**2 for x in xpt]     # 以x平方方式建立y坐标点
6 plt.axis([0, 100, 0, 10000]) # 留意参数是列表
7 plt.scatter(xpt, ypt, c=(0, 1, 0)) # 绿色
8 plt.show()
```

执行结果



上述程序第5行是依据xpt列表产生ypt列表值的方式，由于在网络上大部分的文章使用数组方式产生图表列表，所以下一节笔者将对此做说明，期待可为读者建立基础。

23-3 Numpy模块

Numpy是Python的一个扩充模块，主要是可以支持多维度空间的数组与矩阵运算，本节笔者将使用其最简单的产生数组功能做解说，由此可以将这个功能扩充到数据图表的设计。使用前我们需导入Numpy模块，如下所示：

```
import Numpy as np
```

23-3-1 建立一个简单的数组linspace()和arange()

在Numpy模块中最基本的就是linspace()方法，使用它可以很方便产生相等等距的数组，它的语法如下：

```
linspace(start, end, num) # 这是最常用简化的语法
```

start是起始值，end是结束值，num是设定产生多少个等距的数组值，num的默认值是50。

在网络上阅读他人使用Python设计的图表时，另一个常看到产生数组的方法是`arange()`，语法如下：

```
arange(start  
  
, stop, step  
  
) # start
```

和**step**

是可以省略

`start`是起始值如果省略默认值是0，`stop`是结束值但是所产生的数组通常不包含此值，`step`是数组相邻元素的间距如果省略默认值是1。

程序实例ch23_18： 建立0, 1, ..., 9, 10的数组。

```
1 # ch23_18.py  
2 import numpy as np  
3  
4 x1 = np.linspace(0, 10, num=11) # 使用linspace()产生数组  
5 print(type(x1), x1)  
6 x2 = np.arange(0,11,1)          # 使用arange()产生数组  
7 print(type(x2), x2)  
8 x3 = np.arange(11)              # 简化语法产生数组  
9 print(type(x3), x3)
```

执行结果

```
===== RESTART: D:\Python\ch23\ch23_18.py =====  
<class 'numpy.ndarray'> [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]  
<class 'numpy.ndarray'> [ 0  1  2  3  4  5  6  7  8  9 10]  
<class 'numpy.ndarray'> [ 0  1  2  3  4  5  6  7  8  9 10]  
>>>
```

23-3-2 绘制波形

在初中数学中我们有学过`sin()`和`cos()`观念，其实有了数组数据，我们可以很方便绘制`sin`和`cos`的波形变化。

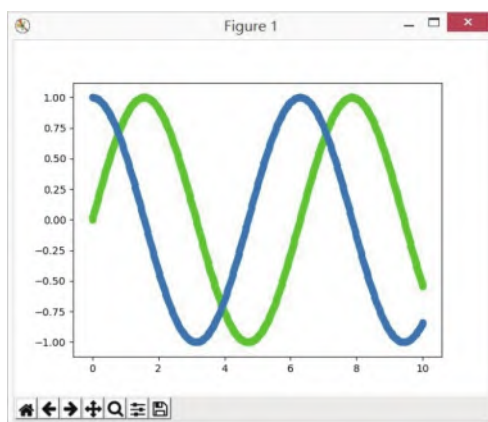
程序实例ch23_19.py： 绘制`sin()`和`cos()`的波形，在这个实例中调用`plt.scatter()`方法2次，相当于也可以绘制2次波形图表。

```

1 # ch23_19.py
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 xpt = np.linspace(0, 10, 500) # 建立含500个元素的数组
6 ypt1 = np.sin(xpt) # y数组的变化
7 ypt2 = np.cos(xpt)
8 plt.scatter(xpt, ypt1, color=(0, 1, 0)) # 绿色
9 plt.scatter(xpt, ypt2) # 预设颜色
10 plt.show()

```

执行结果



23-3-3 建立不等宽度的散点图

在scatter()方法中，(x, y)的数据可以是列表也可以是矩阵，预设所绘制点大小s的值是20，这个s可以是一个值也可以是一个数组数据，当它是一个数组数据时，利用更改数组值的大小，我们就可以建立不同大小的散点图。

在我们使用Python绘制散点图时，如果将2个点之间绘了上百或上千个点，则可以产生绘制线条的视觉，如果再加上每个点的大小不同，且依一定规律变化，则可以有特别效果。

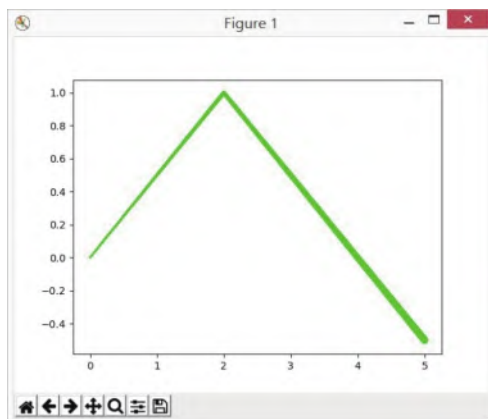
程序实例ch23_20.py： 建立一个不等宽度的图形。

```

1 # ch23_20.py
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 xpt = np.linspace(0, 5, 500) # 建立含500个元素的数组
6 ypt = 1 - 0.5*np.abs(xpt-2) # y数组的变化
7 lwidths = (1-xpt)**2 # 宽度数组
8 plt.scatter(xpt, ypt, s=lwidths, color=(0, 1, 0)) # 绿色
9 plt.show()

```

执行结果



23-3-4 色彩映射color mapping

至今我们针对一组数组（或列表）所绘制的图皆是单色，若是以ch23_20.py第8行为例，色彩设定是color=(0,1,0)，这是固定颜色的用法。在色彩的使用中是允许色彩也是数组（或列表）随着数据而变化，此时色彩的变化是根据所设定的色彩映射值（color mapping）而定，例如有一个色彩映射值是rainbow，内容如下：



在数组（或列表）中，数值低的值颜色在左边，会随着数值变高颜色往右边移动。当然在程序设计中，我们需在scatter()中增加color(也可用c)设定，这时color的值就变成一个数组（或列表）。然后我们需增加参数cmap（英文是color map），这个参数主要是指定使用哪一种色彩映射值。

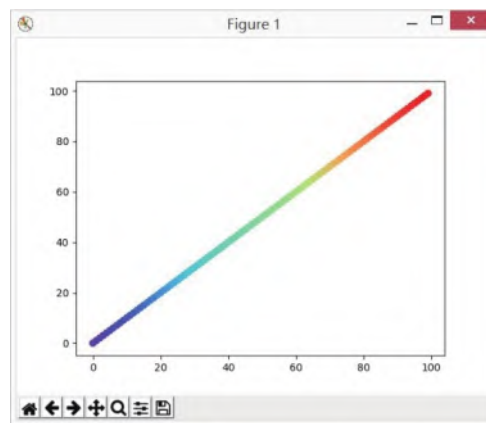
程序实例ch23_21.py：色彩映射的应用。


```

1 # ch23_21.py
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 x = np.arange(100)
6 y = x
7 t = x
8 plt.scatter(x, y, c=t, cmap='rainbow')
9 plt.show()

```

执行结果



有时候我们在程序设计时，色彩映射也可以设定是根据x轴的值变化，或是y轴的值变化，整个效果是不一样的。

程序实例ch23_22.py: 重新设计ch23_20.py，主要是设定差别是固定点的宽度为50，将色彩改为依y轴值变化，同时使用hsv色彩映射表。

```

8 plt.scatter(xpt, ypt, s=50, c=ypt, cmap='hsv') # 色彩随y轴值变化

```

程序实例ch23_23.py: 重新设计ch23_22.py，主要是将色彩改为依x轴值变化。

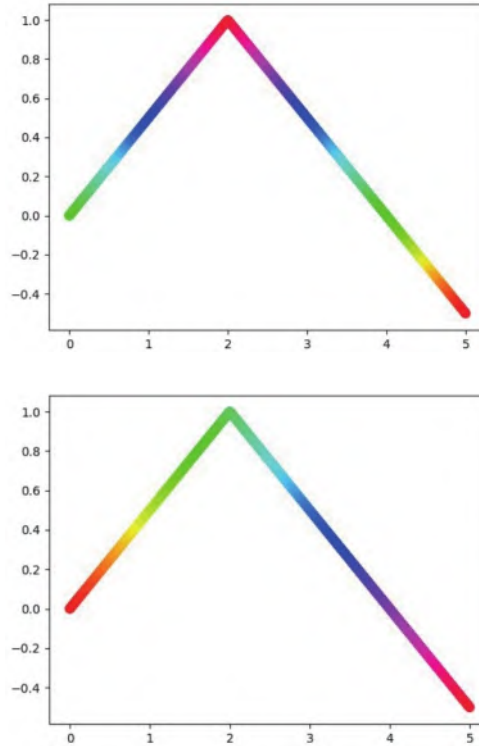
执行结果 如下方左图。

```

8 plt.scatter(xpt, ypt, s=50, c=xpt, cmap='hsv') # 色彩随x轴值变化

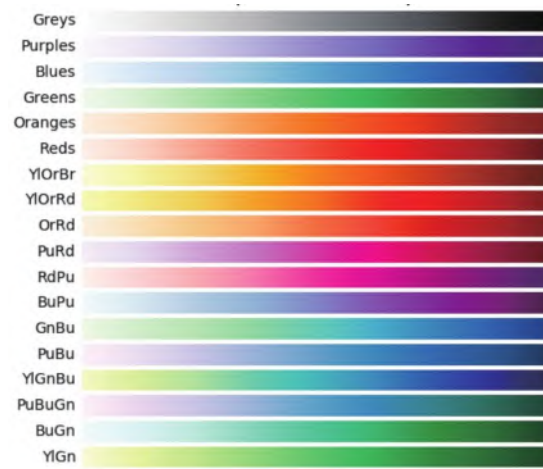
```

执行结果 如下方右图。

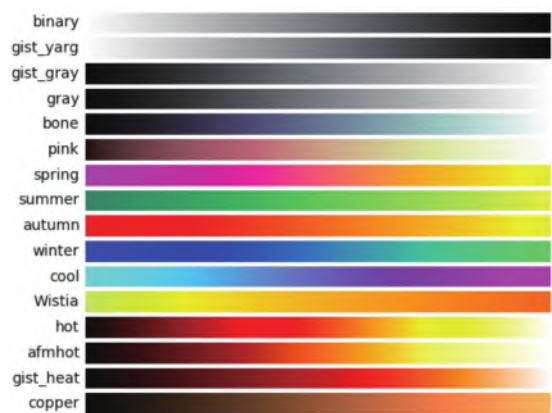


目前matplotlib协会所提供的色彩映射内容如下：

序列色彩映射表



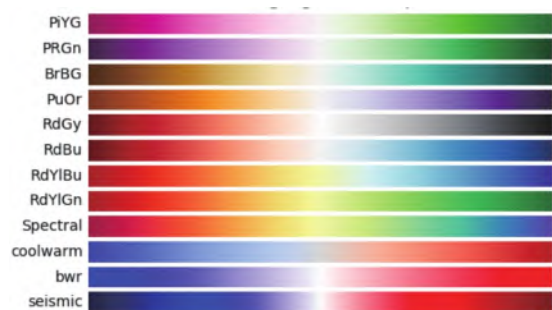
序列2色彩映射表



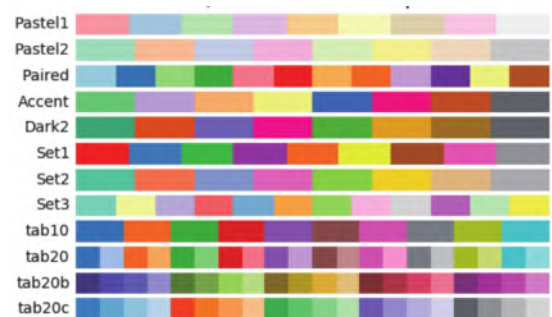
直觉一致的色彩映射表



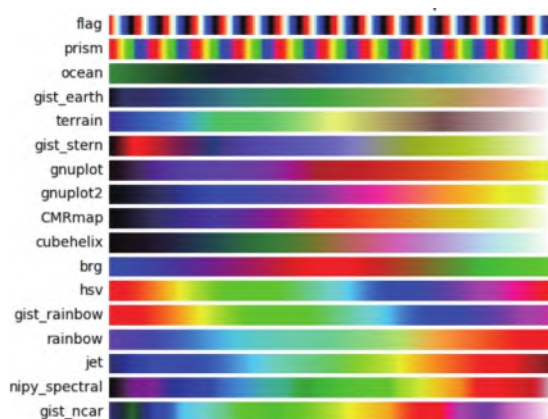
发散式的色彩映射表



定性色彩映射表



杂项色彩映射表



数据源matplotlib协会网址如下：

http://matplotlib.org/examples/color/colormaps_reference.html

如果有一天你做大数据研究，当收集了无数的数据后，可以将数据以图表显示，然后用色彩判断整个数据趋势。

23-4 随机数的应用

随机数在统计的应用中是非常重要的知识，这一节笔者试着用随机数方法，了解Python的随机数分布。这一节将介绍下列随机方法：

```
np.random.random(N) # 传回N个0.0至1.0之间的数字
```

23-4-1 一个简单的应用

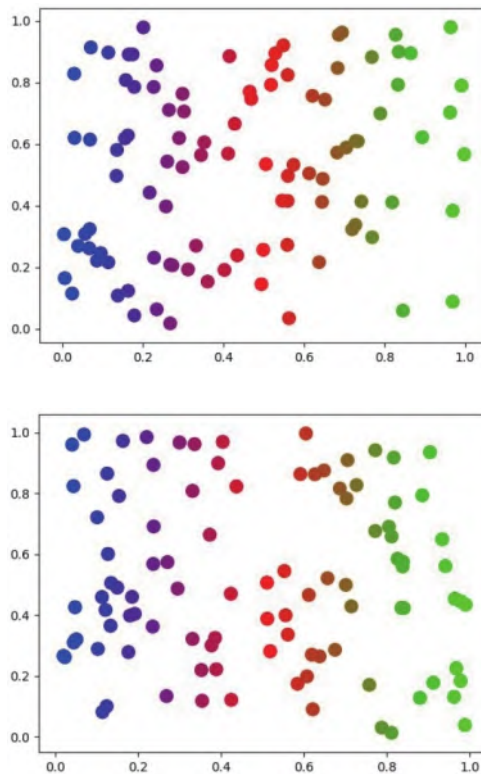
程序实例ch23_24.py：产生100个0.0至1.0之间的随机数，使用brg色彩映射表绘出这个图表。当关闭图表时，会询问是否继续，如果输入n/N则结束。其实因为数据是随机数，所以每次皆可产生不同的效果。

```

1 # ch23_24.py
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 num = 100
6 while True:
7     x = np.random.random(100)          # 可以产生num个0.0至1.0之间的数字
8     y = np.random.random(100)
9     t = x                                # 色彩随x轴变化
10    plt.scatter(x, y, s=100, c=t, cmap='brg')
11    plt.show()
12    yORn = input("是否继续 ?(y/n) ")    # 询问是否继续
13    if yORn == 'n' or yORn == 'N':      # 输入n或N则程序结束
14        break

```

执行结果



上述程序笔者使用第5行的num控制产生随机数的数量，其实读者可以自行修订，增加或减少随机数的数量，以体会本程序的运作。

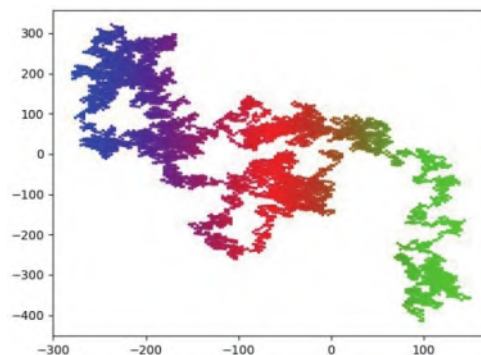
23-4-2 随机数的移动

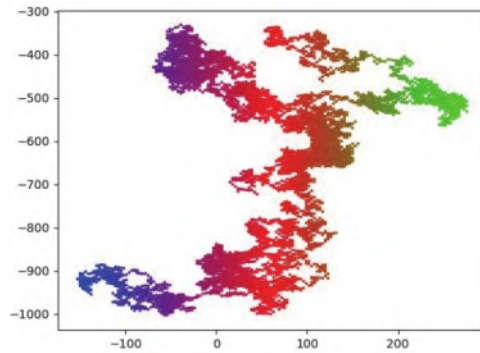
其实我们也可以针对随机数的特性，让每个点随着随机数的变化产生有序列的随机移动，经过大量值的运算后，每次均可产生不同但有趣的图形。

程序实例ch23_25.py： 随机数移动的程序设计，这个程序在设计时，最初点的起始位置是(0, 0)，程序第7行可以设定下一个点的x轴是往右移动3或是往左移动3，程序第9行可以设定下一个点的y轴是往上移动1或5或是往下移动1或5。每此执行完10000点的测试后，会询问是否继续。如果继续，先将上一回合的终点坐标当作新回合的起点坐标(27至28行)，然后清除列表索引x[0]和y[0]以外的元素(29至30行)。

```
1 # ch23_25.py
2 import matplotlib.pyplot as plt
3 import random
4
5 def loc(index):
6     ''' 处理坐标的移动 '''
7     x_mov = random.choice([-3, 3])          # 随机x轴移动值
8     xloc = x[index-1] + x_mov              # 计算x轴新位置
9     y_mov = random.choice([-5, -1, 1, 5])   # 随机y轴移动值
10    yloc = y[index-1] + y_mov               # 计算y轴新位置
11    x.append(xloc)                          # x轴新位置加入列表
12    y.append(yloc)                          # y轴新位置加入列表
13
14 num = 10000                                # 设定随机点的数量
15 x = [0]                                    # 设定第一次执行x坐标
16 y = [0]                                    # 设定第一次执行y坐标
17 while True:
18     for i in range(1, num):                # 建立点的坐标
19         loc(i)
20         t = x                              # 色彩随x轴变化
21         plt.scatter(x, y, s=2, c=t, cmap='brg')
22         plt.show()
23         yORn = input("是否继续?(y/n) ")    # 询问是否继续
24         if yORn == 'n' or yORn == 'N':      # 输入n或N则程序结束
25             break
26         else:
27             x[0] = x[num-1]                 # 上次结束x坐标成新的起点x坐标
28             y[0] = y[num-1]                 # 上次结束y坐标成新的起点y坐标
29             del x[1:]                       # 删除旧列表x坐标元素
30             del y[1:]                       # 删除旧列表y坐标元素
```

执行结果





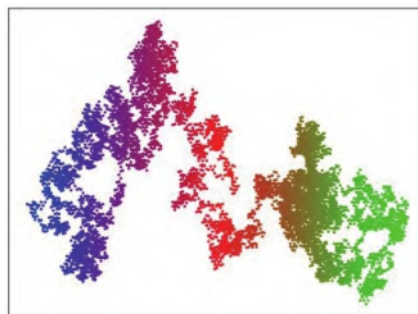
23-4-3 隐藏坐标

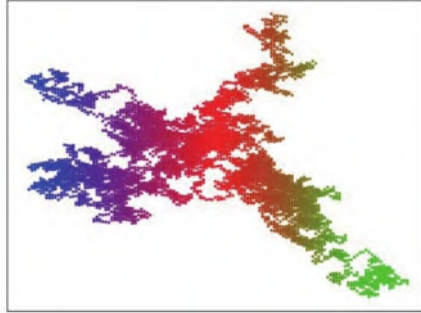
有时候我们设计随机数移动建立了美丽的图案后，觉得坐标好像很煞风景，可以使用下列程序实例 ch23_26.py 内的 `axes().get_xaxis()`、`axes().get_yaxis()`、`set_visible()` 方法隐藏坐标。

程序实例 ch23_26.py: 重新设计 ch23_25.py 隐藏坐标，这个程序只是增加下列行。

```
22 plt.axes().get_xaxis().set_visible(False) # 隐藏x轴坐标
23 plt.axes().get_yaxis().set_visible(False) # 隐藏y轴坐标
```

执行结果





23-5 绘制多个图表

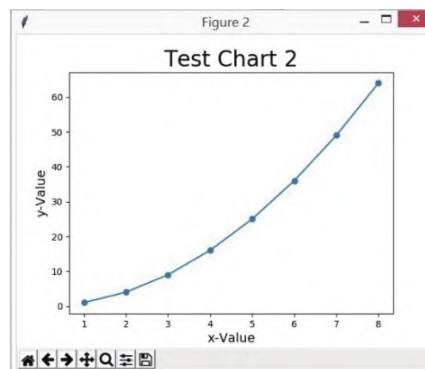
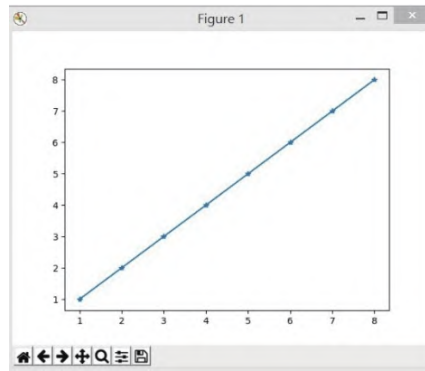
23-5-1 一个程序有多个图表

Python允许一个程序绘制多个图表，默认是一个程序绘制一个图表(Figure)，如果想要绘制多个图表，可以使用figure(N)设定图表，N是图表的序号。在建立多个图表时，只要将所要绘制的图接在欲放置的图表后面即可。

程序实例ch23_27.py： 设计2个图表，将data1线条放在图表Figure 1，将data2线条放在图表Figure 2。同时图表Figure 2将会建立图表标题与x/y轴的标签。

```
1 # ch23_27.py
2 import matplotlib.pyplot as plt
3
4 data1 = [1, 2, 3, 4, 5, 6, 7, 8]      # data1线条
5 data2 = [1, 4, 9, 16, 25, 36, 49, 64] # data2线条
6 seq = [1, 2, 3, 4, 5, 6, 7, 8]
7 plt.figure(1)                        # 建立图表1
8 plt.plot(seq, data1, '-*')          # 绘制图表1
9 plt.figure(2)                        # 建立图表2
10 plt.plot(seq, data2, '-o')          # 以下皆是绘制图表2
11 plt.title("Test Chart 2", fontsize=24)
12 plt.xlabel("x-Value", fontsize=14)
13 plt.ylabel("y-Value", fontsize=14)
14 plt.show()
```

执行结果



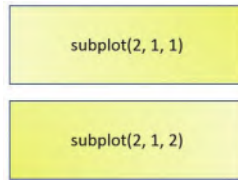
上述第8行所绘制的data1图表因为是接在plt.figure(1)后面，所以所绘制的图出现在Figure 1。上述第10-13行所绘制的data2图表因为是接在plt.figure(2)后面，所以所绘制的图出现在Figure 2。

23-5-2 含有子图的图表

要设计含有子图的图表需要使用subplot()方法，语法如下：

```
subplot(x1  
, x2, x3)
```

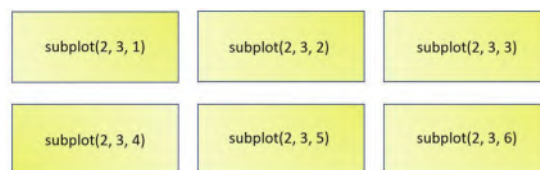
x1 代表上下（垂直）要绘几张图，x2代表左右（水平）要绘几张图。x3代表这是第几张图。如果规划是一个Figure绘制上下2张图，那么subplot()的应用如下：



如果规划是一个Figure绘制左右2张图，那么subplot()的应用如下：



如果规划是一个Figure绘制上下2张图，左右3张图，那么subplot()的应用如下：

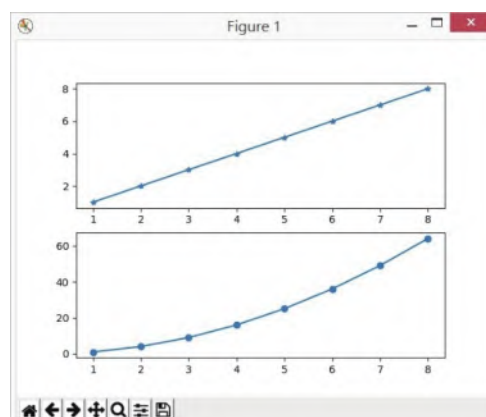


程序实例ch23_28.py： 在一个Figure内绘制上下子图的应用。

```
1 # ch23_28.py
2 import matplotlib.pyplot as plt
3
4 data1 = [1, 2, 3, 4, 5, 6, 7, 8]
5 data2 = [1, 4, 9, 16, 25, 36, 49, 64]
6 seq = [1, 2, 3, 4, 5, 6, 7, 8]
7 plt.subplot(2, 1, 1)
8 plt.plot(seq, data1, '-*')
9 plt.subplot(2, 1, 2)
10 plt.plot(seq, data2, '-o')
11 plt.show()
```

data1线条
data2线条
子图1
子图2

执行结果

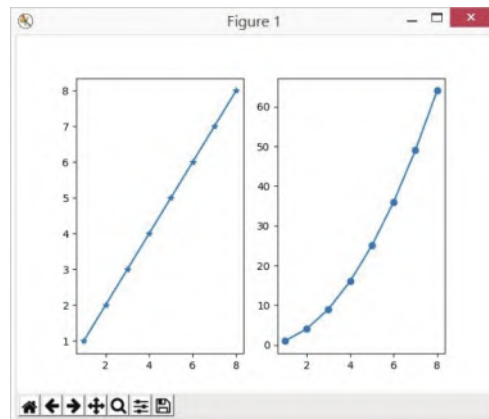


程序实例ch23_29.py： 在一个Figure内绘制上下子图的应用。

```
1 # ch23_29.py
2 import matplotlib.pyplot as plt
3
4 data1 = [1, 2, 3, 4, 5, 6, 7, 8]
5 data2 = [1, 4, 9, 16, 25, 36, 49, 64]
6 seq = [1, 2, 3, 4, 5, 6, 7, 8]
7 plt.subplot(1, 2, 1)
8 plt.plot(seq, data1, '-*')
9 plt.subplot(1, 2, 2)
10 plt.plot(seq, data2, '-o')
11 plt.show()
```

data1线条
data2线条
子图1
子图2

执行结果



23-6 直方图的制作bar()

在直方图的制作中，我们可以使用bar()方法，常用的语法如下：

```
bar(x, height, width)
```

x是一个序列，主要是直方图x轴位置。height是序列数值的大小。width是直方图的宽度，预设是0.85。

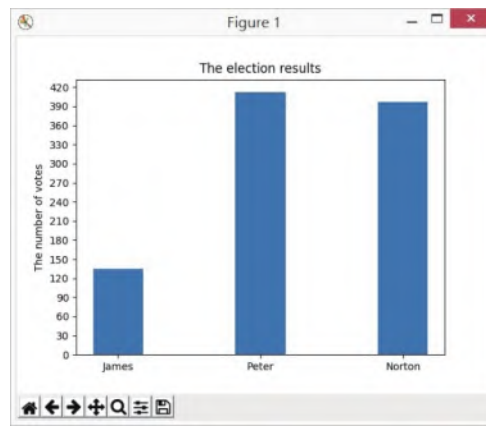
程序实例ch23_30： 有一个选举，James得票135、Peter得票412、Norton得票397，用直方图表示。

```

1 # ch23_30.py
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 votes = [135, 412, 397]      # 得票数
6 N = len(votes)              # 计算长度
7 x = np.arange(N)            # 直方图x轴坐标
8 width = 0.35                # 直方图宽度
9 plt.bar(x, votes, width)     # 绘制直方图
10
11 plt.ylabel('The number of votes')
12 plt.title('The election results')
13 plt.xticks(x, ('James', 'Peter', 'Norton'))
14 plt.yticks(np.arange(0, 450, 30))
15 plt.show()

```

执行结果



上述程序第11行是打印y轴的标签，第12行是打印直方图的标题，第13行则是打印x轴各直方图的标签，第14行是设定y轴刻度。

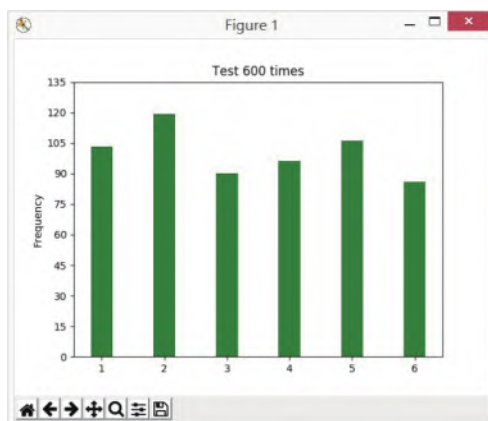
程序实例ch23_31.py： 掷骰子的机率设计，一个骰子有6面分别记载1, 2, 3, 4, 5, 6，我们这个程序会用随机数计算600次，每个数字出现的次数，同时用柱形图表示，为了让读者有不同体验，笔者将图表颜色改为绿色。

```

1 # ch23_31.py
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from random import randint
5
6 def dice_generator(times, sides):
7     ''' 处理随机数 '''
8     for i in range(times):
9         ranNum = randint(1, sides)      # 产生1-6随机数
10        dice.append(ranNum)
11
12 def dice_count(sides):
13     ''' 计算1-6个出现次数 '''
14     for i in range(1, sides+1):
15         frequency = dice.count(i)      # 计算i出现在dice列表的次数
16         frequencies.append(frequency)
17
18 times = 600      # 掷骰子次数
19 sides = 6        # 骰子有几面
20 dice = []        # 建立掷骰子的列表
21 frequencies = [] # 存储每一面骰子出现次数列表
22 dice_generator(times, sides) # 产生掷骰子的列表
23 dice_count(sides) # 将骰子列表转换成次数列表
24 x = np.arange(6) # 直方图x轴坐标
25 width = 0.35    # 直方图宽度
26 plt.bar(x, frequencies, width, color='g') # 绘制直方图
27 plt.ylabel('Frequency')
28 plt.title('Test 600 times')
29 plt.xticks(x, ('1', '2', '3', '4', '5', '6'))
30 plt.yticks(np.arange(0, 150, 15))
31 plt.show()

```

执行结果



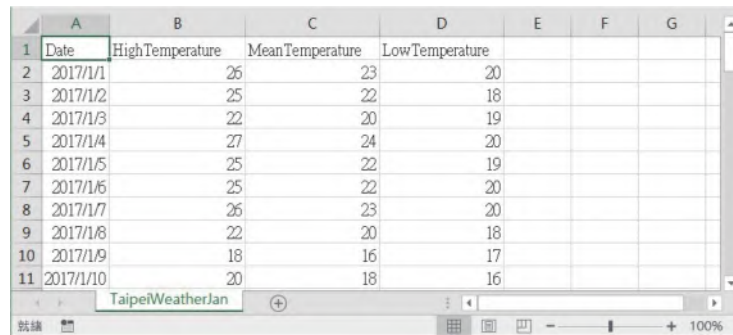
上述程序最重要的是第11-15行的dice_count()函数，这个函数主要是将含600个元素的dice列表，分别计算1, 2, 3, 4, 5, 6各出现的次数，然后将结果存储至frequencies列表。如果读者忘记count()方法的用法可以参考6-6-2小节。

23-7 使用CSV文件绘制图表

其实网络上有许多CSV文件，原始的文件有些复杂，不过我们可以使用Python读取文件，然后筛选我们要的字段，整个工作就变得比较简单了。本节主要是用实例介绍将图表设计应用在CSV文件。

23-7-1 台北2017年1月气象资料

在ch23文件夹内有TaipeiWeatherJan.csv文件，这是2017年1月份台北市的气象资料，这个文件的Excel内容如下：

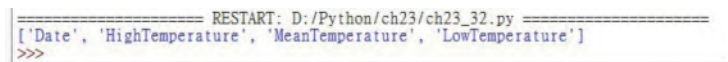


	A	B	C	D	E	F	G
1	Date	HighTemperature	MeanTemperature	LowTemperature			
2	2017/1/1	26	23	20			
3	2017/1/2	25	22	18			
4	2017/1/3	22	20	19			
5	2017/1/4	27	24	20			
6	2017/1/5	25	22	19			
7	2017/1/6	25	22	20			
8	2017/1/7	26	23	20			
9	2017/1/8	22	20	18			
10	2017/1/9	18	16	17			
11	2017/1/10	20	18	16			

程序实例ch23_32.py： 读取TaipeiWeatherJan.csv文件，然后列出标题栏。

```
1 # ch23_32.py
2 import csv
3
4 fn = 'TaipeiWeatherJan.csv'
5 with open(fn) as csvFile:
6     csvReader = csv.reader(csvFile)
7     headerRow = next(csvReader)      # 读取文件下一行
8     print(headerRow)
```

执行结果



```
===== RESTART: D:/Python/ch23/ch23_32.py =====
['Date', 'HighTemperature', 'MeanTemperature', 'LowTemperature']
>>>
```

从上图我们可以得到TaipeiWeatherJan.csv有4个字段，分别是记载日期(Date)、当天最高温(HighTemperature)、平均温度(MeanTemperature)、最低温度(LowTemperature)。

23-7-2 列出标题数据

我们可以使用6-12节所介绍的enumerate()。

程序实例ch23_33.py： 列出TaipeiWeatherJan.csv文件的标题与相对应的索引。

```
1 # ch23_33.py
2 import csv
3
4 fn = 'TaipeiWeatherJan.csv'
5 with open(fn) as csvFile:
6     csvReader = csv.reader(csvFile)
7     headerRow = next(csvReader) # 读取文件下一行
8     for i, header in enumerate(headerRow):
9         print(i, header)
```

执行结果

```
===== RESTART: D:/Python/ch23/ch23_33.py =====
0 Date
1 HighTemperature
2 MeanTemperature
3 LowTemperature
>>>
```

23-7-3 读取最高温与最低温

程序实例ch23_34.py： 读取TaipeiWeatherJan.csv文件的最高温与最低温。这个程序会将一月份的最高温放在highTemps列表，最低温放在lowTemps列表。

```
1 # ch23_34.py
2 import csv
3
4 fn = 'TaipeiWeatherJan.csv'
5 with open(fn) as csvFile:
6     csvReader = csv.reader(csvFile)
7     headerRow = next(csvReader) # 读取文件下一行
8     highTemps, lowTemps = [], [] # 设定空列表
9     for row in csvReader:
10         highTemps.append(row[1]) # 存储最高温
11         lowTemps.append(row[3]) # 存储最低温
12
13 print("最高温：", highTemps)
14 print("最低温：", lowTemps)
```

执行结果

```
===== RESTART: D:/Python/ch23/ch23_34.py =====
最高温： ['26', '25', '22', '27', '25', '25', '26', '22', '18', '20', '21', '22',
          '18', '15', '15', '16', '23', '23', '22', '18', '15', '17', '16', '17', '18',
          '19', '24', '26', '25', '27', '18']
最低温： ['20', '18', '19', '20', '19', '20', '20', '18', '17', '16', '18', '18',
          '14', '12', '13', '13', '16', '18', '18', '12', '12', '12', '13', '14', '13',
          '13', '13', '16', '17', '14', '14']
>>>
```

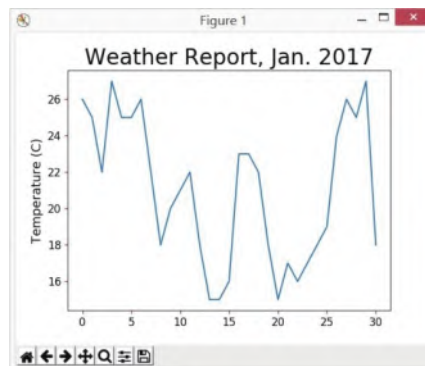
23-7-4 绘制最高温

其实这一节内容不复杂，所有绘图方法前面各小节已有说明。

程序实例ch23_35.py： 绘制2017年1月份，台北每天天气的最高温。

```
1 # ch23_35.py
2 import csv
3 import matplotlib.pyplot as plt
4
5 fn = 'TaipeiWeatherJan.csv'
6 with open(fn) as csvFile:
7     csvReader = csv.reader(csvFile)
8     headerRow = next(csvReader)      # 读取文件下一行
9     highTemps = []                  # 设定空列表
10    for row in csvReader:
11        highTemps.append(row[1])      # 存储最高温
12
13 plt.plot(highTemps)
14 plt.title("Weather Report, Jan. 2017", fontsize=24)
15 plt.xlabel("", fontsize=14)
16 plt.ylabel("Temperature (C)", fontsize=14)
17 plt.tick_params(axis='both', labelsize=12, color='red')
18 plt.show()
```

执行结果



23-7-5 设定绘图区大小

```
figure(dpi=n, figsize=(width, height))
```

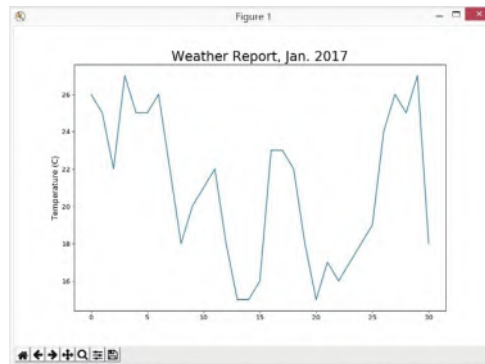
经上述设定后，绘图区的宽将是 $n \times \text{width}$ 像素，高是 $n \times \text{height}$ 像素。

程序实例ch23_36.py： 重新设计ch23_35.py，设定绘图区宽度是960，高度是640，这个程序只是增加下列行。

目前绘图区大小是使用系统默认，不过我们可以使用figure()设定绘图区大小，设定方式如下：

```
12 plt.figure(dpi=80, figsize=(12, 8))      # 设定绘图区大小
```


执行结果



23-7-6 日期格式

天气图表建立过程，我们可能想加上日期在x轴的刻度上，这时我们需要使用Python内置的datetime模块，在使用前请使用下列方式导入模块。

```
from datetime import datetime
```

然后可以使用下列方法将日期字符串解析为日期对象：

```
strptime(string, format)
```

string是要解析的日期字符串，format是该日期字符串目前格式，下表是日期格式参数的意义。

参数	说明
%Y	4 位数年份, 例如 : 2017
%y	2 位数年份, 例如 : 17
%m	月份 (1-12)
%B	月份名称, 例如 : January
%A	星期名称, 例如 : Sunday
%d	日期 (1-31)
%H	24 小时 (0-23)
%I	12 小时 (1-12)
%p	AM 或 PM
%M	分钟 (0-59)
%S	秒 (0-59)

程序实例ch23_37.py: 将字符串转成日期对象。

```

1 # ch23_37.py
2 from datetime import datetime
3
4 dateObj = datetime.strptime('2017/1/1', '%Y/%m/%d')
5 print(dateObj)

```

执行结果

```

===== RESTART: D:\Python\ch23\ch23_37.py =====
2017-01-01 00:00:00
>>>

```

有关datetime对象的更进一步使用可以参考30-1节。

23-7-7 在图表增加日期刻度

其实我们可以在plot()方法内增加日期列表参数时, 在图表增加日期刻度。

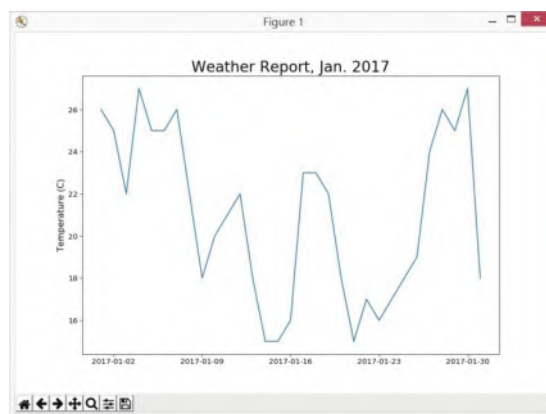
程序实例ch23_38.py: 为图表增加日期刻度。

```

1 # ch23_38.py
2 import csv
3 import matplotlib.pyplot as plt
4 from datetime import datetime
5
6 fn = 'TaipeiWeatherJan.csv'
7 with open(fn) as csvFile:
8     csvReader = csv.reader(csvFile)
9     headerRow = next(csvReader)          # 读取文件下一行
10    dates, highTemps = [], []             # 设定空列表
11    for row in csvReader:
12        highTemps.append(row[1])           # 存储最高温
13        currentDate = datetime.strptime(row[0], "%Y/%m/%d")
14        dates.append(currentDate)
15
16 plt.figure(dpi=80, figsize=(12, 8))      # 设定绘图区大小
17 plt.plot(dates, highTemps)               # 图标增加日期刻度
18 plt.title("Weather Report, Jan. 2017", fontsize=24)
19 plt.xlabel("", fontsize=14)
20 plt.ylabel("Temperature (C)", fontsize=14)
21 plt.tick_params(axis="both", labelsize=12, color='red')
22 plt.show()

```

执行结果



这个程序的第一个重点是第13行和14行，主要是将日期字符串转成对象，然后存入dates日期列表。第二个重点是第17行，在plot()方法中第一个参数是放dates日期列表。

23-7-8 日期位置的旋转

上一节的执行结果可以发现日期是水平放置，autofmt_xdate() 设定日期旋转，语法如下：

```

fig = plt.figure( xxx )          # xxx是相关设定信息
...
if g.autofmt_xdate(rotation=xx)  # rotation若省略则系统使用优化默认

```

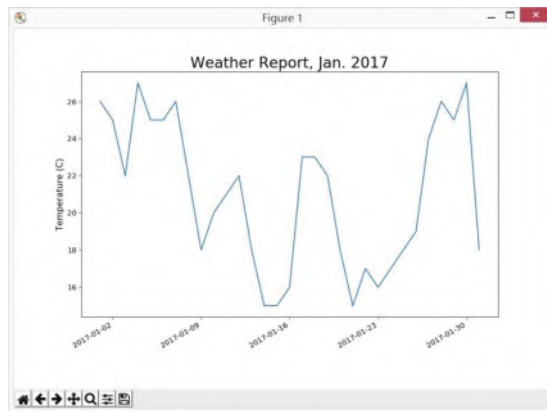
程序实例ch23_39.py： 重新设计ch23_38.py，增加将日期旋转。

```

16 fig = plt.figure(dpi=80, figsize=(12, 8)) # 设定绘图区大小
17 plt.plot(dates, highTemps)               # 图标增加日期刻度
18 fig.autofmt_xdate()                       # 日期旋转

```

执行结果



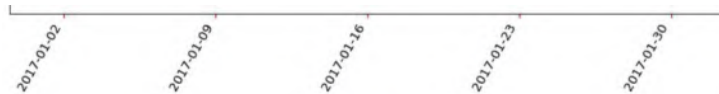
程序实例ch23_40.py： 是特别将日期字符串调整为旋转60度的结果。

```

18 fig.autofmt_xdate(rotation=60)           # 日期旋转

```

执行结果



23-7-9 绘制最高温与最低温

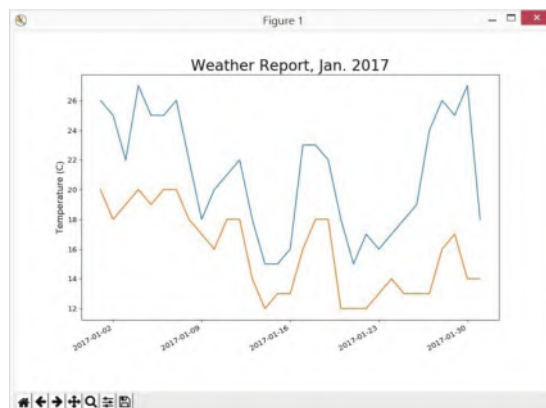
在TaipeiWeatherJan.csv文件内有最高温与最低温的字段，下列将同时绘制最高温与最低温。

程序实例ch23_41.py： 绘制最高温与最低温，这个程序第一个重点是程序第11至21行使用异常处理方式，因为读者在读取真实的网络数据时，常常会有不可预期的数据发生，例如，数据少了或是数据格式错误，往往造成程序中断，为了避免程序因数据不良，所以使用异常

处理方式。第二个重点为程序第24和25行是分别绘制最高温与最低温。

```
1 # ch23_41.py
2 import csv
3 import matplotlib.pyplot as plt
4 from datetime import datetime
5
6 fn = 'TaipeiWeatherJan.csv'
7 with open(fn) as csvFile:
8     csvReader = csv.reader(csvFile)
9     headerRow = next(csvReader) # 读取文件下一行
10    dates, highTemps, lowTemps = [], [], [] # 设定空列表
11    for row in csvReader:
12        try:
13            currentDate = datetime.strptime(row[0], "%Y/%m/%d")
14            highTemp = int(row[1]) # 设定最高温
15            lowTemp = int(row[3]) # 设定最低温
16        except Exception:
17            print('有缺值')
18        else:
19            highTemps.append(highTemp) # 存储最高温
20            lowTemps.append(lowTemp) # 存储最低温
21            dates.append(currentDate) # 存储日期
22
23 fig = plt.figure(dpi=80, figsize=(12, 8)) # 设定绘图区大小
24 plt.plot(dates, highTemps) # 绘制最高温
25 plt.plot(dates, lowTemps) # 绘制最低温
26 fig.autofmt_xdate() # 日期旋转
27 plt.title("Weather Report, Jan. 2017", fontsize=24)
28 plt.xlabel("", fontsize=14)
29 plt.ylabel("Temperature (C)", fontsize=14)
30 plt.tick_params(axis='both', labelsize=12, color='red')
31 plt.show()
```

执行结果



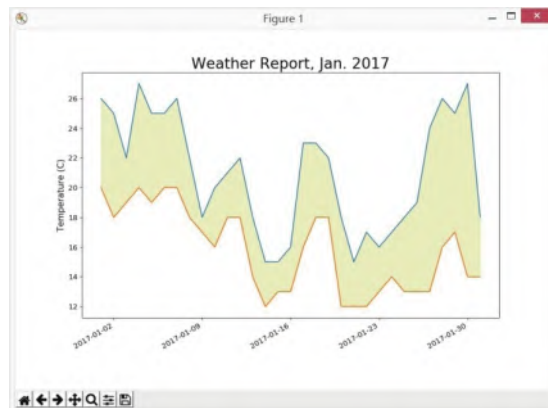
23-7-10 填满最高温与最低温之间的区域

可以使用fill_between()方法执行填满最高温与最低温之间的区域。

程序实例ch23_42.py: 使用透明度是0.2的黄色填满区间, 这个程序只是增加下列行。

```
26 plt.fill_between(dates, highTemps, lowTemps, color='y', alpha=0.2) # 填满区间
```

执行结果



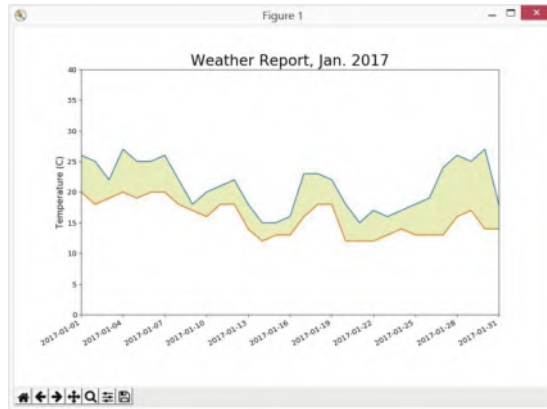
23-7-11 再谈轴刻度

在23-1-2小节笔者介绍过可以使用`axis()`设定x,y轴的最小和最大刻度，可以参考程序实例`ch23_1_1.py`，对于上述天气图表而言，x轴是以日期为刻度，如果我们想要设定x轴刻度，可以设置x轴最小刻度是2017-01-01，最大刻度是2017-01-31。

程序实例ch23_43.py：重新设计程序实例`ch23_42.py`，让x轴最小刻度是2017-01-01，最大刻度是2017-01-31。y轴（气温）则是在0℃～40℃。下列`dates[0]`就是指2017-01-01，`date[30]`就是指2017-01-31。

```
24 plt.axis([dates[0], dates[30], 0, 40]) # 设定x,y轴刻度
```

执行结果



结果发现x轴宽度变了，日期标签也变多了。

习题

1. 请参考ch23_11.py，增加2021—2022年数据如下：

Benz 6020 6620

BMW 4900 4590

Lexus 6200 6930

然后绘制图表。

2. 请分别选用 cool(sin)/hot(cos) 色彩映射表，将它应用在 ch23_19.py。
3. 请重新设计ch23_25.py，将x轴移动方式改为[-3, -2, -1, 1, 2, 3]，将y轴移动方式改为[-5, -3, -1, 1, 3, 5]，然后列出结果。
4. 扩充设计ch23_27.py，为Figure 1增加标题“Test Chart1”，x/y轴标签分别是“x-Data/y-Data”。
5. 请重新设计ch23_9.py，将4组资料绘在Figure1内以4个子图方式显示。

6. 请为ch23_9.py再增加data5数据，内容是[1, 6, 11, 16, 21, 26, 31, 36]，然后将这5组数据绘在Figure 1内分成5个子图，其中第5个子图跳过，直接绘在第6个图的位置。
7. 请读者将程序实例ch23_31.py，处理成有2个骰子，所以可以计算2—12间每个数字的出现次数，请测试1000次，以直方图表示。
8. 请读者参考程序实例ch23_31.py，在赌场最常见到的是用3个骰子，所以可以计算3—18间每个数字的出现次数，请测试1000次，以直方图表示。
9. 请读取TaipeiWeatherJan.csv，打印平均温度图。

第24章 JSON资料

JSON是一种数据格式，由美国程序设计师Douglas Crockford创建的，JSON全名是JavaScript Object Notation，由JSON英文全文字义我们可以推敲JSON的缘由，最初是为JavaScript开发的。这种数据格式由于简单好用被大量应用在Web开发与大数据数据库(NoSQL)，现在已成为一种著名数据格式，Python与许多程序语言同时采用与支持。也由此在使用Python设计程序时，可以将数据以JSON格式存储，与其他程序语言的设计师分享。

Python程序设计时需使用import json导入json模块。

24-1 认识json数据格式

json的数据格式有2种，分别是：

对象 (object)：一般用大括号{ }表示。

数组 (array)：一般用中括号[]表示。

24-1-1 对象(object)

在json中对象 就是用“键-值 (key:value)”方式配对存储，对象内容用左大括号“{”开始，右大括号“}”结束，键 (key) 和值 (value) 用“:”区隔，每一组键:值 间以逗号“,”隔开，以下是取材自json.org的官方图说明。



在json格式中键(key)是一个字符串 (string)。值可以是数值 (number)、字符串 (string)、布尔值 (bool)、数组 (array)或是null 值。

例如：下列是对象的实例。

```
{ "Name": "Hung", "Age": 25 }
```

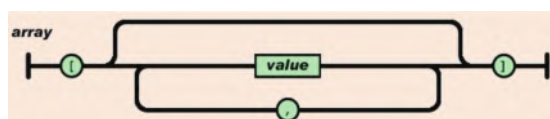
使用json时需留意，键 (key)必须是文字 ，例如下列是错误的实例。

```
{ "Name": "Hung", 25: "Key" }
```

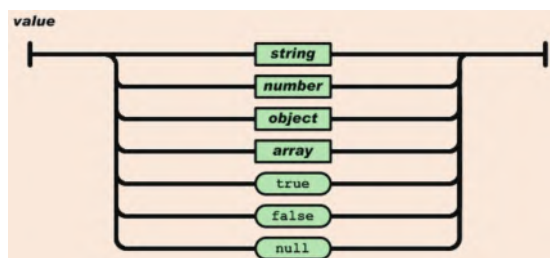
在json格式中字符串需用双引号 ，同时在json文件内不可以有批注 。

24-1-2 数组(array)

数组基本上是一系列的值 (value)所组成，用左中括号 “[” 开始，右中括号 “]” 结束。各值之间用逗号 “,” 隔开，以下是取材自json.org的官方图说明。



数组的值可以是数值 (number)、字符串 (string)、布尔值 (bool)、数组 (array)或是null 值。



24-1-3 json数据存在方式

前两节所述是json的数据格式定义，但是在Python中它存在方式是字符串(string)。

`'json数据'` # 可参考程序实例ch24_1.py的第3个输出

使用json模块执行将Python数据转成json字符串类型数据或json文件使用不同方法，下面24-2和24-3节将分别说明。

24-2 将Python应用在json字符串形式数据

本节主要说明json数据以字符串形式存在时的应用。

24-2-1 使用dumps()将Python数据转成json格式

在json模块内有dumps()，可以将Python数据转成json字符串格式，下列是转化对照表。

Python 资料	JSON 资料
dict	object
list, tuple	array
str, unicode	string
int, float, long	number
True	true
False	false
None	null

程序实例ch24_1.py： 将Python的列表与元组数据转成json的数组数据的实例。

```

1 # ch24_1.py
2 import json
3
4 listNumbers = [5, 10, 20, 1]      # 列表数据
5 tupleNumbers = (1, 5, 10, 9)      # 元组资料
6 jsonData1 = json.dumps(listNumbers) # 将列表数据转成json数据
7 jsonData2 = json.dumps(tupleNumbers) # 将列表数据转成json数据
8 print("列表转换成json的数组", jsonData1)
9 print("元组转换成json的数组", jsonData2)
10 print("json数组在Python的数据类型 ", type(jsonData1))

```

执行结果

```

===== RESTART: D:\Python\ch24\ch24_1.py =====
列表转换成json的数组 [5, 10, 20, 1]
元组转换成json的数组 [1, 5, 10, 9]
json数组在Python的数据类型 <class 'str'>
>>>

```

特别留意，上述笔者在第10行打印最终 json在Python的数据类型，结果是以字符串方式存在。若以jsonData1为例，从上述执行结果我们可以了解，在Python内它的数据如下：

```
`[5, 10, 20, 1]`
```

程序实例ch24_2.py： 将Python由字典元素所组成的列表转成json数组，转换后原先字典元素变为json的对象。

```

1 # ch24_2.py
2 import json
3
4 listObj = [{'Name': 'Peter', 'Age': 25, 'Gender': 'M'}] # 列表数据元素是字典
5 jsonData = json.dumps(listObj) # 将列表数据转成json数据
6 print("列表转换成json的数组", jsonData)
7 print("json数组在Python的数据类型 ", type(jsonData))

```

执行结果

```

===== RESTART: D:\Python\ch24\ch24_1.py =====
列表转换成json的数组 [5, 10, 20, 1]
元组转换成json的数组 [1, 5, 10, 9]
json数组在Python的数据类型 <class 'str'>
>>>

```

读者应留意json对象的字符串是用双引号 。

24-2-2 dumps()的sort keys参数

Python的字典是无序的数据，使用dumps()将Python数据转成json对象时，可以增加使用sort_keys=True，则可以将转成json格式的对象排序。

程序实例ch24_3.py： 将字典转成json格式的对象，分别是未使用排序与使用排序。最后将未使用排序与使用排序的对象作比较判断是否相同，得到的是被视为不同对象。

```
1 # ch24_3.py
2 import json
3
4 dictObj = {'b':80, 'a':25, 'c':60}          # 字典
5 jsonObj1 = json.dumps(dictObj)              # 未排序将字典转成json对象
6 jsonObj2 = json.dumps(dictObj, sort_keys=True) # 有排序将字典转成json对象
7 print("未用排序将字典转换成json的对象", jsonObj1)
8 print("使用排序将字典转换成json的对象", jsonObj2)
9 print("有排序与未排序对象是否相同", jsonObj1 == jsonObj2 )
10 print("json物件在Python的数据类型", type(jsonObj1))
```

执行结果

```
===== RESTART: D:\Python\ch24\ch24_3.py =====
未用排序将字典转换成json的对象 {"b": 80, "a": 25, "c": 60}
使用排序将字典转换成json的对象 {"a": 25, "b": 80, "c": 60}
有排序与未排序对象是否相同 False
json物件在Python的数据类型 <class 'str'>
>>>
```

从上述执行结果得出json对象在Python的存放方式也是字符串。

24-2-3 dumps()的indent参数

从ch24_3.py的执行结果可以看到数据不太容易阅读，特别是资料量如果更多的时候，在将Python的字典数据转成json格式的对象时，可以加上indent设定缩排json对象的键-值，让json对象可以更容易显示。

程序实例ch24_4.py： 将Python的字典转成json格式对象时，设定缩排4个字符宽度。

```
1 # ch24_4.py
2 import json
3
4 dictObj = {'b':80, 'a':25, 'c':60}          # 字典
5 jsonObj = json.dumps(dictObj, sort_keys=True, indent=4) # 用内缩呈现json对象
6 print(jsonObj)
```

执行结果

```
===== RESTART: D:/Python/ch24/ch24_4.py =====
{
  "a": 25,
  "b": 80,
  "c": 60
}
>>>
```

24-2-4 使用loads()将json格式数据转成Python的数据

在json模块内有loads()，可以将json格式数据转成Python数据，下列是转化对照表。

JSON 资料	Python 资料
object	dict
array	list
string	unicode
number(int)	int, long
Number(real)	float
true	True
false	False
null	None

程序实例ch24_5.py： 将json的对象数据转成Python数据的实例，需留意在建立json数据时，需加上引号，因为json数据在Python内是以字符串形式存在。

```
1 # ch24_5.py
2 import json
3
4 jsonObj = '{"b":80, "a":25, "c":60}' # json对象
5 dictObj = json.loads(jsonObj)      # 转成Python对象
6 print(dictObj)
7 print(type(dictObj))
```

执行结果

```
===== RESTART: D:/Python/ch24/ch24_5.py =====
{'b': 80, 'a': 25, 'c': 60}
<class 'dict'>
>>>
```

从上述可以看到json对象返回Python数据时的数据类型。

24-3 将Python应用在json文件

我们在程序设计时，更重要的是将Python的资料以json格式存储，未来可以供其他不同的程序语言读取。或是使用Python读取其他语言以json格式存储的数据。

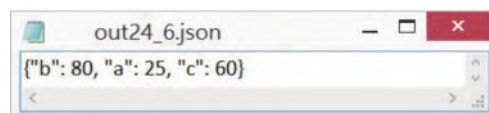
24-3-1 使用dump()将Python数据转成json文件

在json模块内有dump()，可以将Python数据转成json文件格式，这个文件格式的扩展名是json，下列将直接以程序实例解说dump()的用法。

程序实例 ch24_6.py： 将一个字典数据，使用json格式存储在out24_6.json文件内。在这个程序实例中，dump()方法的第一个参数是欲存储成json格式的数据，第二个参数是欲存储的文件对象。

```
1 # ch24_6.py
2 import json
3
4 dictObj = {'b':80, 'a':25, 'c':60}
5 fn = 'out24_6.json'
6 with open(fn, 'w') as fnObj:
7     json.dump(dictObj, fnObj)
```

执行结果 在目前工作文件夹可以新增json文件，文件名是out24_6.json。如果用记事本打开，可以得到下列结果。



24-3-2 使用load()读取json文件

在json模块内有load()，可以读取json文件，读完后这个json文件将被转换成Python的数据格式，下列将直接以程序实例解说dump()的用法。

程序实例ch24_7.py： 读取json文件out24_6.json，同时列出结果。

```
1 # ch24_7.py
2 import json
3
4 fn = 'out24_6.json'
5 with open(fn, 'r') as fnObj:
6     data = json.load(fnObj)
7
8 print(data)
9 print(type(data))
```

执行结果

```
===== RESTART: D:/Python/ch24/ch24_7.py
{'b': 80, 'a': 25, 'c': 60}
<class 'dict'>
>>>
```

24-4 简单的json文件应用

程序实例ch24_8.py： 程序执行时会要求输入账号，然后列出所输入账号同时打印欢迎使用本系统。

```
1 # ch24_8.py
2 import json
3
4 fn = 'login.json'
5 login = input("请输入账号：")
6 with open(fn, 'w') as fnObj:
7     json.dump(login, fnObj)
8     print("%s! 欢迎使用本系统!" % login)
```

执行结果

```
===== RESTART: D:\Python\ch24\ch24_8.py
请输入账号：Peter
Peter! 欢迎使用本系统!
>>>
```

上述程序同时会将所输入的账号存入login.json文件内。

程序实例ch24_9.py: 读取login.json的数据, 同时输出“欢迎回来使用本系统”。

```
1 # ch24_9.py
2 import json
3
4 fn = 'login.json'
5 with open(fn, 'r') as fnObj:
6     login = json.load(fnObj)
7     print("%s! 欢迎回来使用本系统! " % login)
```

执行结果

```
===== RESTART: D:\Python\ch24\ch24_9.py =====
Peter! 欢迎回来使用本系统!
>>>
```

程序实例ch24_10.py: 下列程序基本上是ch24_8.py和ch24_9.py的组合, 如果第一次登录会要求输入账号然后将输入账号记录在login24_10.json文件内, 如果不是第一次登录, 会直接读取已经存在login24_10.json的账号, 然后打印“欢迎回来”。这个程序用第7行是否能正常读取login24_10.json方式判断是否是第一次登录, 如果这个文件不存在表示是第一次登录, 将执行第8行except 至12行的内容。如果这个文件已经存在, 表示不是第一次登录, 将执行第13行else: 后面的内容。

```
1 # ch24_10.py
2 import json
3
4 fn = 'login24_10.json'
5 try:
6     with open(fn) as fnObj:
7         login = json.load(fnObj)
8 except Exception:
9     login = input("请输入账号 : ")
10     with open(fn, 'w') as fnObj:
11         json.dump(login, fnObj)
12         print("系统已经记录你的账号 ")
13 else:
14     print("%s 欢迎回来" % login)
```

执行结果

```
===== RESTART: D:\Python\ch24\ch24_10.py
请输入账号 : Peter
系统已经记录你的账号
>>>
===== RESTART: D:\Python\ch24\ch24_10.py
Peter 欢迎回来
>>>
```

24-5 世界人口数据的json文件

在本书ch24文件夹内有populations.json文件，这是一个非官方在2000年和2010年的人口统计数据，这一节笔者将一步一步讲解如何使用json数据文件。

24-5-1 认识人口统计的json文件

若是将这个文件用记事本打开，内容如下：



```
populations - 记事本
[[{"Country Name": "World", "Country Code": "WLD", "Year": "2000", "Numbers": "6117806174.56156"}, {"Country Name": "World", "Country Code": "WLD", "Year": "2010", "Numbers": "65258.0"}, {"Country Name": "Andorra", "Country Code": "AND", "Year": "2010", "Numbers": "85216.0"}, {"Country Name": "Australia", "Country Code": "AUS", "Year": "2000", "Numbers": "108186.0"}, {"Country Name": "Australia", "Country Code": "AUS", "Year": "2010", "Numbers": "129592417.0"}, {"Country Name": "Bangladesh", "Country Code": "BGD", "Year": "2010", "Numbers": "129592417.0"}, {"Country Name": "Bermuda", "Country Code": "BMU", "Year": "2000", "Numbers": "622000"}, {"Country Name": "Bermuda", "Country Code": "BMU", "Year": "2010", "Numbers": "174425502.0"}, {"Country Name": "Brazil", "Country Code": "BRA", "Year": "2010", "Numbers": "14139608.0"}, {"Country Name": "Cameroon", "Country Code": "CMR", "Year": "2010", "Numbers": "13.0"}, {"Country Name": "Chad", "Country Code": "TCD", "Year": "2000", "Numbers": "8223089.0"}, {"Country Name": "Comoros", "Country Code": "COM", "Year": "2010", "Numbers": "735266.0"}, {"Country Name": "Cuba", "Country Code": "CUB", "Year": "2000", "Numbers": "4418192.0"}, {"Country Name": "Cuba", "Country Code": "CUB", "Year": "2010", "Numbers": "732112.0"}, {"Country Name": "Djibouti", "Country Code": "DJI", "Year": "2000", "Numbers": "732112.0"}, {"Country Name": "Djibouti", "Country Code": "DJI", "Year": "2010", "Numbers": "6193287.0"}, {"Country Name": "El Salvador", "Country Code": "SLV", "Year": "2010", "Numbers": "49157.0"}, {"Country Name": "Fiji", "Country Code": "FJI", "Year": "2000", "Numbers": "812309.0"}, {"Country Name": "Fiji", "Country Code": "FJI", "Year": "2010", "Numbers": "1297212.0"}, {"Country Name": "Gambia, The", "Country Code": "GMB", "Year": "2010", "Numbers": "1729998"}]]
```

在网络上任何一个号称是真实统计的json数据，在用记事本打开后，初看一定是复杂的，读者碰上这个问题首先不要慌，同时分析数据的共通性，这样有助于未来程序的规划与设计。从上图基本上我们可以了解它的资料格式，这是一个列表，列表元素是字典，有些国家或地区只有2000年的资料，有些只有2010年的资料，有些则同时有这2个年度的数据，每个字典内有4个键-值，如下所示：

```
{
    "Country Name": "World",
    "Country Code": "WLD",
    "Year": "2000",
    "Numbers": "6117806174.56156"
}
```

上述字段分别是国家或地区名称(Country Name)、国家代码(Country Code)、年份(Year)和人口数(numbers)。从上述文件我们应该注意到，人口数在我们日常生活理解应该是整数，可是这个数据是用字符串表达，另外，在非官方的统计数据中，难免会有错误，例如，上述国家或地区（这是全球人口统计）的2010年人口数据资料出现了小数点，这个需要我们用程序处理。

程序实例ch24_11.py： 列出populations.json数据中的人口数据。

```
1 # ch24_11.py
2 import json
3
4 fn = 'populations.json'
5 with open(fn) as fnObj:
6     getDats = json.load(fnObj)          # 读json档案
7
8 for getData in getDats:
9     if getData['Year'] == '2000':      # 筛选2000年的数据
10         countryName = getData['Country Name']
11         countryCode = getData['Country Code']
12         population = int(float(getData['Numbers']))
13         print('代码 =', countryCode,
14               '名称 =', countryName,
15               '人口数 =', population)
```

执行结果

```
===== RESTART: D:\Python\ch24\ch24_11.py =====
代码 = WLD      名称 = World 人口数 = 6117806174
代码 = AFG      名称 = Afghanistan 人口数 = 25951672
代码 = ALB      名称 = Albania 人口数 = 3072478
代码 = DZA      名称 = Algeria 人口数 = 30534041
代码 = ASM      名称 = American Samoa 人口数 = 57995
代码 = AND      名称 = Andorra 人口数 = 65258
代码 = AGO      名称 = Angola 人口数 = 13926705
代码 = ATG      名称 = Antigua and Barbuda 人口数 = 78536
代码 = ARG      名称 = Argentina 人口数 = 36931013
代码 = ARM      名称 = Armenia 人口数 = 3076653
代码 = ABW      名称 = Aruba 人口数 = 91031
代码 = AUS      名称 = Australia 人口数 = 19153581
代码 = AUT      名称 = Austria 人口数 = 8012068
```

上述重点是第12行，当我们碰上含有小数点的字符串时，需先将这个字符串转成浮点数，然后再将浮点数转成整数。

24-5-2 认识pygal.maps.world的地区代码信息

前一节有关populations.json地区代码是3个英文字母，如果想要使用这个json数据绘制世界人口地图，需要配合pygal.maps.world模块的方法，这个模块的地区代码是2个英文字母，所以需要将populations.json地区代码转成2个英文字母。这里本节先介绍2个英文字母的信息，pygal.maps.world模块内有COUNTRIES字典，在这个字典中国码是2个英文字母，从这里我们可以列出相关地区与代码的列表。使用pygal.maps.world模块前需先安装此模块，如下所示：

```
pip install pygal.maps.world
```

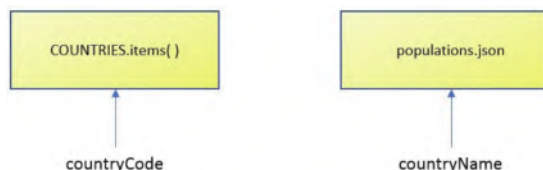
程序实例ch24_12.py： 列出pygal.maps.world模块COUNTRIES字典的2个英文字母的地区代码与完整的地区名称列表。

```
1 # ch24_12.py
2 from pygal.maps.world import COUNTRIES
3
4 for countryCode in sorted(COUNTRIES.keys()):
5     print("代码 :", countryCode, "名称 =", COUNTRIES[countryCode])
```

执行结果

```
===== RESTART: D:\Python\ch24\ch24_12.py =====
代码 : ad      名称 = Andorra
代码 : ae      名称 = United Arab Emirates
代码 : af      名称 = Afghanistan
代码 : al      名称 = Albania
代码 : am      名称 = Armenia
代码 : ao      名称 = Angola
代码 : aq      名称 = Antarctica
代码 : ar      名称 = Argentina
代码 : at      名称 = Austria
```

接着笔者将讲解，输出2个字母的地区代码时，同时输出，这个程序相当于是将2个不同来源的数据作配对。



程序实例ch24_13.py: 从populations.json取每个名称信息, 然后将每一个名称放入getCountryCode()方法中找寻相关代码, 如果找到则输出相对应的代码, 如果找不到则输出“名称不吻合”。

```
1 # ch24_13.py
2 import json
3 from pygal.maps.world import COUNTRIES
4
5 def getCountryCode(countryName):
6     '''输入名称回传代码'''
7     for dictCode, dictName in COUNTRIES.items(): # 搜寻代码字典
8         if dictName == countryName:
9             return dictCode # 如果找到则回传代码
10    return None # 找不到则回传None
11
12 fn = 'populations.json'
13 with open(fn) as fnObj:
14     getDatas = json.load(fnObj) # 读取人口数据json文件
15
16 for getData in getDatas:
17     if getData['Year'] == '2000': # 筛选2000年的数据
18         countryName = getData['Country Name']
19         countryCode = getCountryCode(countryName)
20         population = int(float(getData['Numbers'])) # 人口数
21         if countryCode != None:
22             print(countryCode, ":", population) # 名称相符
23         else:
24             print(countryName, " 名称不吻合:") # 名称不吻合
```

执行结果

```
===== RESTART: D:\Python\ch24\ch24_13.py =====
World 名称不吻合:
af : 25951672
al : 3072478
dz : 30534041
American Samoa 名称不吻合:
ad : 65258
ao : 13926705
Antigua and Barbuda 名称不吻合:
```

上述会有不吻合输出是因为这是2个不同单位的数据, 例如, Arab World在populations.json是一个记录, 在pygal.maps.world模块的COUNTRIES字典中没有这个记录。至于有关上述的更深层应用, 将在下一节解说。

习题

1. 读取populations.json文件, 将2000年的数据存入ex24_1.json文件内。

第25章 用Python传送手机短信

本章主要内容是叙述如何使用Python传送手机短信，主要是以美国Twilio公司所提供的服务为实例说明。当然本书的重点是教导读者使用免费的试用账号，它的功能是受限的，如果读者觉得好用，想要将这个功能应用在商业用途，可以使用升级，相关作业可以参考网站说明。

全球这类通信公司很多，可以用关键词free sms gateway查询，sms全名是short message service短短信服务，这是目前电信公司很普遍的一个服务。

25-1 安装twilio模块

为了要用Python设计与Twilio公司有关的网络服务，首先请安装twilio模块。

```
pip install twilio
```

25-2 到Twilio公司注册账号

为了要使用Twilio公司所提供的短信服务，需要到Twilio公司注册账号，以取得下列信息：

Account SID: Twilio API key账号。

Auth TOKEN: Twilio账号的图腾(TOKEN)。

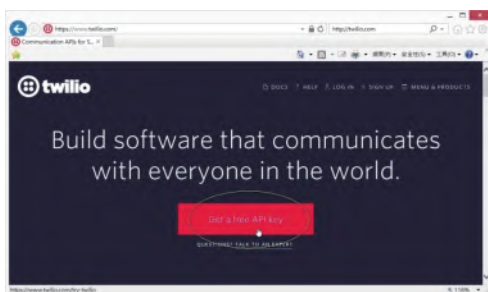
Twilio Number: Twilio电话号码。

Verified numbers: 电话号码使用地区。

上述信息我们可以称之为API key（密钥），有了上述密钥，您就可以使用Python程序发送短信了。

25-2-1 申请账号

首先请进入<http://www.twilio.com>网站。



请点选Get a free API key，然后将看到下列空白窗体。

下列是笔者的示范输入，请读者确实输入自己的数据。

填写完成后，请按Get Started按钮，然后将看到下列画面。

Sign up for free

First Name	Last Name
Company Name (optional)	
Email	
Choose a password	Password, again

WHICH PRODUCT DO YOU PLAN TO USE FIRST?

▼

WHAT ARE YOU BUILDING?

▼

CHOOSE YOUR LANGUAGE

▼

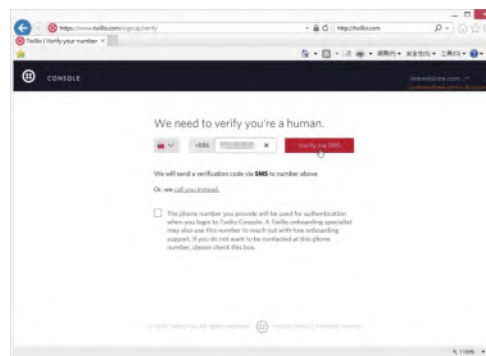
POTENTIAL MONTHLY INTERACTIONS (OVER SMS, CHAT, VOICE, & VIDEO)

▼

☐ 我不是機器人 

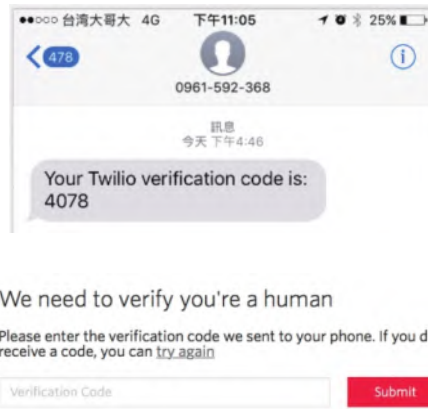
By clicking the button, you agree to our [legal policies](#).

The image shows the Twilio console registration form. It includes fields for a phone number (with a dropdown for country code), a name (Hung), an optional company name, an email address (@deepstone.com.tw), and a password (with a strength indicator showing 'Strong'). Below these are dropdown menus for 'WHICH PRODUCT DO YOU PLAN TO USE FIRST?' (I'm just exploring), 'WHAT ARE YOU BUILDING?' (I don't have a project in mind yet), 'CHOOSE YOUR LANGUAGE' (Python), and 'POTENTIAL MONTHLY INTERACTIONS (OVER SMS, CHAT, VOICE, & VIDEO)' (Not a Production App). At the bottom, there is a CAPTCHA section with a green checkmark and the text '我不是機器人' (I am not a robot), a CAPTCHA logo, and a red 'Get Started' button. A link to 'legal policies' is also present.



为了不被网络恶意软件注册攻击，所以会要求输入电话号码，Twilio网站会从你登录IP判断你的位置，然后自动勾选你的国家或地区，请在上述字段输入你的手机号码，然后按Verify via SMS按钮，Twilio公司会发送验证码到你的手机，下列是笔者手机收到验证码的图例。

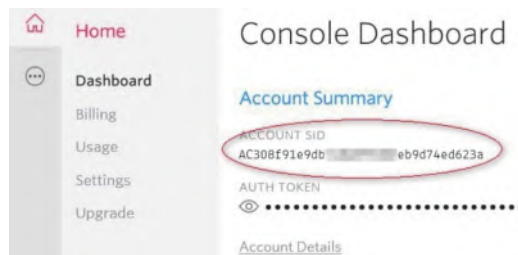
此时你的网页画面会要求将上述验证码输入下列画面。



输入完成后，请按Submit按钮，整个注册就算完成了。

25-2-2 获得Account SID

注册成功后，在画面上可以看到Dashboard，请点选可以在此看到的所申请的Account SID信息。

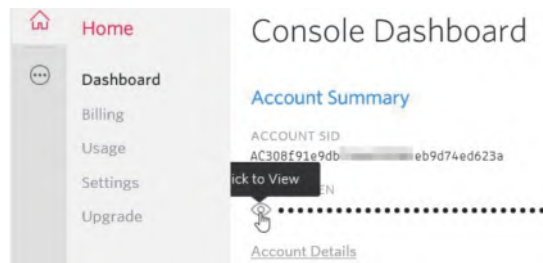


未来可以点选上述SID码，然后复制到Python程序。

25-2-3 获得Auth TOKEN

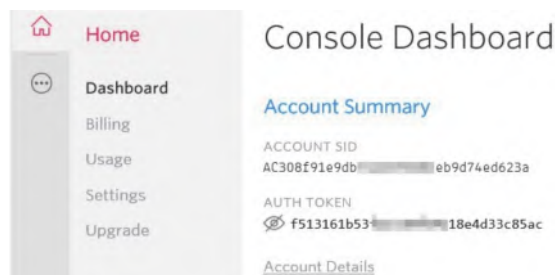
原则上图腾 (TOKEN) 信息是被保护的，可以点选 ，列出图腾 (TOKEN) 编码。

可以得到下列结果，与SID一样，使用时最好用复制方式粘贴到Python程序，比较方便，同时也不会有错误。

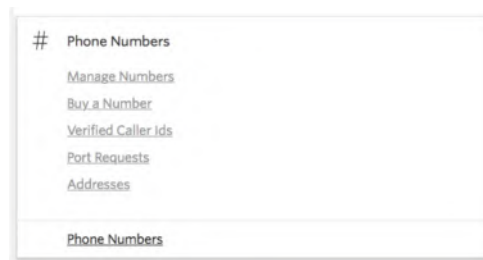


再点选一次可以再度隐藏TOKEN。

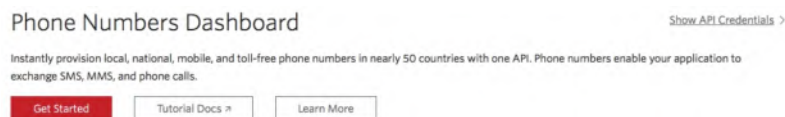
25-2-4 获得Twilio Number



在屏幕上有Phone Numbers，请点选Phone Numbers。



将看到下列画面。



请点选Get Started按钮，将看到下列画面。

Get Started with Phone Numbers

Getting started with Twilio's phone numbers is easy! Search for local, toll-free, or mobile numbers by capability, country, or prefix.

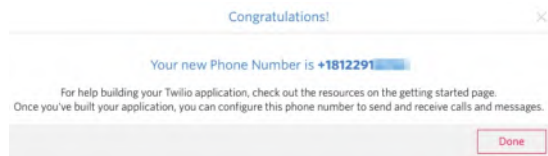
[Get your first Twilio phone number](#)

Looking for a short-code? [Apply for a short-code here](#) ➤

请点选Get yourfirst Twilio phone number按钮。



上述将列出你的Twilio号码，请按Choose this Number按钮。



上述将列出你未来程序的Twilio号码，未来你的程序需要上述格式的号码。

25-2-5 设定Twilio使用地区

由于这是试用账号，同时我们不是在美国本土，所以还需在系统设定短信适用地区。



请参考上图点选所有服务功能，然后选Programmable SMS服务，将看到下列画面。

Programmable SMS Dashboard

Programmatically send and receive SMS worldwide. Route text messages globally to and from your application over local, mobile, toll-free, and short code numbers.

[Get Started](#)[Read the Tutorial Docs »](#)[Features & Pricing](#)

You have a Trial Account

- 1 Your trial account has \$14.50 remaining
- 2 Trial accounts can only send messages to verified numbers in these countries
- 3 Messages sent in trial will begin with "Sent from a Twilio Trial Account"
- 4 While you have a trial account, you're limited to one Twilio number

请点选 `these countries` 字符串，将看到 Message Geographic Permissions 标题，请点选 Taiwan (+886) 复选框， 未来用 Python 所发的短信就可以在中国台湾地区使用了。

或是可选 China (+86)，未来用 Python 所发的短信就可以在中国大陆使用了。

25-3 使用 Python 程序设计发送短信

程序设计需要导入模块，可以使用下列指令。

```
from twilio.rest import Client
```

下列笔者将直接以程序实例做讲解方式说明发送短信的方法。

程序实例 ch25_1.py： 发送短信“Python王者归来”到手机，读者需留意下列第5、7、11行的号码皆是经过修改的，所以读者执行时将无法运作，读者必须到 Twilio 公司申请，然后输入自己的号码。第12行读者需输入自己的电话号码。

```
1 # ch25_1.py
2 from twilio.rest import Client
3
4 # 你从twilio.com申请的账号
5 accountSid='AC308f91e9dc748a59538feb9d74ed993a'
6 # 你从twilio.com获得的图腾
7 authToken='f513161b63f71720f64118e4d33ca8ac'
8
9 client = Client(accountSid, authToken)
10 message = client.messages.create (
11     from_ = "+15052070000",           # 这是twilio.com给你的号码
12     to = "+886952xxxxxx",           # 这是收短信方的号码
13     body = "Python王者归来" )       # 发送的信息
```

执行结果



由于笔者是使用测试账号，所以所有短信皆会以Sent from your Twilio trial account字符串开头，然后才是你的短信内容。

这个程序主要关键是第9行，利用所申请的SID和TOKEN当作参数，调用Client()方法传回Client对象，此程序笔者是用client当作对象。接着程序第10行client对象调用message.create()方法传送短信，这个方法需有下列3个参数。

```
from_ = "15052070000"

" # 这是Twilio公司分配给你的电话号码

to = "+886952xxxxxx"

" # 这是短信接收方的电话号码
body = "Python王者归来" # 这是短信内容
```

注意from_ 在from _右边有下画线_，这是因为from是Python的关键词，可参考程序第2行的叙述，为了有区分所以Twilio公司特别将from_右边加上底线。上述952xxxxxx 读者应改为自己的手机号码，程序第10行所设定的返回值是message，由这个返回值我们可以获得一些有用的信息。例如，下列是在Python Shell窗口列出from_、to、body属性内容的实例。

```
>>> print(message.from_)
+15052073368
>>> print(message.to)
+886952282828
>>> print(message.body)
Sent from your Twilio trial account - Python王者归来
>>>
```

短信传送后，可以使用date_created属性获得信息建立时间的信息。

```
>>> print(message.date_created)
2017-11-05 17:42:13+00:00
>>>
```

每一个信息皆有唯一的id编号，可以打印此编号。

```
>>> print(message.sid)
SM4d73fcbf0bb1489b8c05344c700c5181
>>>
```

本章只介绍了一个Twilio公司所提供的最简单的功能，Twilio是一个通信软件服务公司，它的服务有许多，例如，语音、视讯等，有兴趣的读者可以自行到该公司网页体会。

习题

1. 请修订程序实例ch25_1.py，改为将短信发给自己。
2. 请发短信给授课老师，内容是“感谢老师，我们学会了Python”。

第26章 Python与SQLite数据库

26-1 SQLite基本观念

在先前章节笔者说明了CSV、Json等数据格式，我们可以将数据以这些格式存储，不过我们使用数据时有时候只是取用一个小小的部分，如果每次皆要大费周章打开文件，处理完成再存储文件，其实不是很经济的事。

一个好的解决方式是使用轻量级的数据库程序当作存储媒体，未来我们可以使用数据库语法取得此数据库的部分有用数据，这将是一个很好的想法。本章笔者将介绍如何使用Python建立SQLite数据库，同时也将讲解使用Python插入 (insert)、提取 (select)、更新 (update)、删除 (delete) SQLite数据库的内容。

Python 3.x版安装完成后有内附SQLite数据库，这一章将以此为实例讲解，在使用此SQLite前需要导入此SQLite。

```
import sqlite3
```

26-2 安装SQLite数据库

执行Python与数据库连接方法如下：

```
conn = sqlite3.connect("数据库名称")
```

上述conn是笔者取的对象名称，读者也可以自行取不一样的名称。上述connect()方法执行时，如果connect()内的数据库名称

存在，就可以将此Python程序与此数据库名称 建立连接，然后我们可以在Python程序内做更进一步的操作。如果数据库名称不存在，就会以此为名称建立一个新的数据库，然后执行数据库连接。

数据库操作结束，我们可以在Python内使用下列方法结束Python程序与数据库的连接。

```
conn.close( )
```

程序实例ch26_1.py： 建立一个新的数据库myData.db，笔者习惯使用db当扩展名。

```
1 # ch26_1.py
2 import sqlite3
3 conn = sqlite3.connect("myData.db")
4 conn.close()
```

执行结果



26-3 SQLite数据类型

SQLite数据库内的数据可以是下列类型。

数据类型	说明
NULL	NULL 也可称空值
INTEGER	整数，例如：0, 2, ……
REAL	浮点数，例如：1.5
TEXT	字符串
BLOB	一个 blob 数据，例如：一个图片、一首歌

26-4 建立SQLite数据库表

在26-2节我们可以使用connect()方法建立数据库连接，这时会回传connect对象，笔者在该节使用conn存储此所回传的对象，这个对象可以使用下列常用的方法。

connect() 对象的方法	说明
close()	数据库连接操作结束
commit()	更新数据库内容
cursor()	建立 cursor 对象，可想成一个光标在数据库中移动，然后执行 execute() 方法
execute()	执行 SQL 数据库指令、数据库建立、新增、删除、修改、提取

下列是cursor对象的方法。

cursor 对象的方法	说明
execute()	执行 SQL 数据库指令、数据库建立、新增、删除、修改、查询

其实上述execute()所使用的是SQL数据库指令，下列将以实例解说。

程序实例ch26_2.py： 建立一个数据库data26_2.db，此数据库内有一个表，表名称是students。

```
1 # ch26_2.py
2 import sqlite3
3 conn = sqlite3.connect("data26_2.db") # 数据库连接
4 cursor = conn.cursor()
5 sql = '''Create table students(
6     id int,
7     name text,
8     gender text)'''
9 cursor.execute(sql) # 执行SQL指令
10 cursor.close() # 关闭
11 conn.close() # 关闭数据库连接
```

执行结果



上述第4行是建立cursor对象，第5~8行是一个字符串，这是SQL语法字符串，意义是建立students表，这个表有3个字段，分别是id、

name、gender，每个字段也设定它的数据类型，分别是整数、字符串、字符串。第9行是执行此SQL语法字符串，经上述设定后相当于在data26_2.db的数据库文件内有students表，这个表有3个字段。

id	name	gender
----	------	--------

需特别注意是，上述ch26_2.py执行完后，如果重复执行会产生students表已经存在的错误，如下所示：

```
===== RESTART: D:\Python\ch26\ch26_2.py =====
Traceback (most recent call last):
  File "D:\Python\ch26\ch26_2.py", line 9, in <module>
    cursor.execute(sql)          # 执行SQL指令
sqlite3.OperationalError: table students already exists
>>>
```

也就是当我们已经在数据库建立表时，无法重新建立相同的表，这样可以防止因为重新建立造成原先的数据库表遗失。

其实除了上述使用cursor()方法建立对象，然后再启动execute()方法外，我们也可以省略建立cursor()方法建立cursor对象的步骤，可以参考下列实例。

程序实例ch26_3.py： 省略cursor()方法建立cursor对象，重新设计ch26_2.py。另外这个程序所建的数据库名称是myInfo.db，未来几节我们将持续使用此数据库。

```
1 # ch26_3.py
2 import sqlite3
3 conn = sqlite3.connect("myInfo.db")      # 数据库连接
4 sql = '''Create table students(
5     id int,
6     name text,
7     gender text)'''
8 conn.execute(sql)                        # 执行SQL指令
9 conn.close()                             # 关闭数据库连接
```

执行结果 此程序会建立myInfo.db文件。

上述虽然省略了建立cursor对象，其实系统内部有建立一个隐含的cursor对象，协助程序可以继续执行。

26-5 增加SQLite数据库表纪录

在SQL语法中可以使用INSERT指令增加表数据，这个表数据 我们称记录 (record)，它的相关语法用法可以参考下列实例。

程序实例ch26_4.py： 读者可以由屏幕输入students表的内容，笔者将键盘输入内容建立成一个循环，每笔记录 输入完成后，按N键可以让输入结束。

```
1 # ch26_4.py
2 import sqlite3
3 conn = sqlite3.connect("myInfo.db") # 数据库连接
4 cursor = conn.cursor()
5 print("请输入myInfo数据库students窗体数据")
6 while True:
7     new_id = int(input("请输入id : ")) # 转成整数
8     new_name = input("请输入name : ")
9     new_gender = input("请输入gender : ")
10    x = (new_id, new_name, new_gender)
11    sql = '''INSERT into students values(?,?,?)'''
12    cursor.execute(sql,x)
13    conn.commit() # 更新数据库
14    again = input("继续(y/n)? ")
15    if again[0].lower() == "n":
16        break
17 cursor.close() # 关闭
18 conn.close() # 关闭数据库连接
```

执行结果

```
===== RESTART: D:\Python\ch26\ch26_4.py =====
请输入myInfo数据库students窗体数据
请输入id : 1
请输入name : John
请输入gender : M
继续(y/n)? y
请输入id : 2
请输入name : Linda
请输入gender : F
继续(y/n)? y
请输入id : 3
请输入name : Kathy
请输入gender : F
继续(y/n)? n
>>>
```

上述程序第7~9行是读取表记录，插入表最重要的语法格式如下：

```
10 x = (new_id, new_name, new_gender)
11 sql = '''INSERT into students values(?,?,?)'''
12 cursor.execute(sql,x)
13 conn.commit() # 更新数据库
```

上述可以将每笔记录处理成元组 (tuple)，然后将SQL语法 处理成字符串，最后将元组与字符串当作execute()方法的参数。

其实在真实世界建立表时，最重要的关键词段id并不一定是数字，甚至更多时候是使用字符串，这时id输入时可以使用001, 002……方式，本实例笔者使用整数int，主要目的是丰富此表的数据类型。

26-6 查询SQLite数据库表

查询表的关键词是SELECT，下列是列出所有表的SQL语法。

```
SELECT * from students
```

程序实例ch26_5.py： 列出所有students表属性。

```
1 # ch26_5.py
2 import sqlite3
3 conn = sqlite3.connect("myInfo.db")    # 数据库连接
4 cursor = conn.cursor()
5 results = cursor.execute("SELECT * from students")
6 for record in results:
7     print("id = ", record[0])
8     print("name = ", record[1])
9     print("gender = ", record[2])
10
11 cursor.close()                        # 关闭
12 conn.close()                          # 关闭数据库连接
```

执行结果

```
===== RESTART: D:/Python/ch26/ch26_5.py =====
id = 1
name = John
gender = M
id = 2
name = Linda
gender = F
id = 3
name = Kathy
gender = F
>>>
```

在sqlite3模块内有fetchall()方法，这个方法可以将所获得的学生数据存储到元组内，可以参考下列实例。

程序实例ch26_6.py： 以元组方式列出所有查询到的学生数据。

```

1 # ch26_6.py
2 import sqlite3
3 conn = sqlite3.connect("myInfo.db") # 数据库连接
4 cursor = conn.cursor()
5 results = cursor.execute("SELECT * from students")
6 allstudents = results.fetchall() # 结果转成元组
7 for student in allstudents:
8     print(student)
9
10 cursor.close() # 关闭
11 conn.close() # 关闭数据库连接

```

执行结果

```

===== RESTART: D:/Python/ch26/ch26_6.py =====
(1, 'John', 'M')
(2, 'Linda', 'F')
(3, 'Kathy', 'F')
>>>

```

如果查询数据时，只想列出部分字段数据，在使用SELECT时可以直接列出域名取代“*”符号。

程序实例ch26_7.py： 重新设计ch26_6.py，只列出name字段数据。

```

1 # ch26_7.py
2 import sqlite3
3 conn = sqlite3.connect("myInfo.db") # 数据库连接
4 cursor = conn.cursor()
5 results = cursor.execute("SELECT name from students")
6 allstudents = results.fetchall() # 结果转成元组
7 for student in allstudents:
8     print(student)
9
10 cursor.close() # 关闭
11 conn.close() # 关闭数据库连接

```

执行结果

```

===== RESTART: D:/Python/ch26/ch26_7.py =====
('John',)
('Linda',)
('Kathy',)
>>>

```

上述如果要列出2个字段或更多字段数据，可以将第5行的name旁边增加字段即可。如果想要查询符合条件的表属性，则SQL语法如下，为了简单化笔者将此语法字符串分行解说：

SELECT

字段, ...

from

表

where

条件'''

程序实例ch26_8.py: 查询所有女生的记录(record)，此程序只列出name和gender字段。

```
1 # ch26_8.py
2 import sqlite3
3 conn = sqlite3.connect("myInfo.db") # 数据库连接
4 cursor = conn.cursor()
5 sql = '''SELECT name, gender
6         from students
7         where gender = "F"'''
8 results = cursor.execute(sql)
9 allstudents = results.fetchall() # 结果转成元组
10 for student in allstudents:
11     print(student)
12
13 cursor.close() # 关闭
14 conn.close() # 关闭数据库连接
```

执行结果

```
===== RESTART: D:/Python/ch26/ch26_8.py =====
('Linda', 'F')
('Kathy', 'F')
>>>
```

26-7 更新SQLite数据库表记录

更新SQLite表记录的关键词是UPDATE，SQL语法如下，为了简单化笔者将此语法字符串分行解说：

``UPDATE

表

set

字段=新内容

where

标明那一笔记录'''

上述完成后记得需要使用commit()更新数据库。

程序实例ch26_9.py: 将id为1的记录name名字改为Tomy。


```

1 # ch26_9.py
2 import sqlite3
3 conn = sqlite3.connect("myInfo.db") # 数据库连接
4 cursor = conn.cursor()
5 sql = '''UPDATE students
6         set name = "Tomy"
7         where id = 1'''
8 results = cursor.execute(sql)
9 conn.commit() # 更新数据库
10
11 results = cursor.execute("SELECT name from students")
12 allstudents = results.fetchall() # 结果转成元组
13 for student in allstudents:
14     print(student)
15
16 cursor.close() # 关闭
17 conn.close() # 关闭数据库连接

```

执行结果

```

===== RESTART: D:/Python/ch26/ch26_9.py =====
('Tomy',)
('Linda',)
('Kathy',)
>>>

```

26-8 删除SQLite数据库表记录

删除SQLite表记录的关键词是DELETE，SQL语法如下，为了简单化笔者将此语法字符串分行解说：

DELETE

from

表

where

标明是哪一笔记录'''

上述完成后记得需要使用commit()更新数据库。

程序实例ch29_10.py： 删除id=2的记录。

```

1 # ch26_10.py
2 import sqlite3
3 conn = sqlite3.connect("myInfo.db") # 数据库连接
4 cursor = conn.cursor()
5 sql = '''DELETE
6         from students
7         where id = 2'''
8 results = cursor.execute(sql)
9 conn.commit() # 更新数据库
10
11 results = cursor.execute("SELECT name from students")
12 allstudents = results.fetchall() # 结果转成元组
13 for student in allstudents:
14     print(student)
15
16 cursor.close() # 关闭
17 conn.close() # 关闭数据库连接

```

执行结果

```

===== RESTART: D:/Python/ch26/ch26_10.py =====
('Tomy',)
('Kathy',)
>>>

```

习题

1. 请扩充设计myInfo数据库的表students，增加年龄(age)、电话(tel)字段，同时建立10笔记录，然后列出结果。
2. 请使用习题1的数据库表，列出所有女生，需列出name和电话字段。
3. 请设计一个学生数据库表管理程序，此管理程序需有下列功能。
 - A. 增加记录
 - B. 修改记录
 - C. 删除记录
 - D. 列出所有记录
 - E. 程序结束

操作细节则可以自行规划与发挥。

第27章 用Python处理图像文件

当前，高画质的手机已经普及，也许你可以使用许多图像软件处理手机所拍摄的相片，本章笔者将教导您以Python处理这些相片。本章将使用Pillow模块，所以请先导入此模块。

```
pip install pillow
```

注意在程序设计中需导入的是PIL模块，主要原因是要向旧版Python Image Library兼容，如下所示：

```
from PIL import ImageColor
```

27-1 认识Pillow模块的RGBA

在Pillow模块中RGBA分别代表红色(R ed)、绿色(G reen)、蓝色(B lue)和透明度(A lpha)，这4个与颜色有关的数值组成元组(tuple)，每个数值都是在0~255之间。如果Alpha的值是255，代表完全不透明，值越小透明度越高。其他有关颜色的细节可参考附录D。其实它的色彩使用方式也与HTML相同，笔者在所著的《HTML5+CSS3王者归来》一书的附录E有完整的色彩名称与颜色值相对应色彩表。

27-1-1 getrgb()

这个函数可以将颜色符号或字符串转为元组，在这里可以使用英文名称（例如，“red”）、色彩数值（例如，#00ff00）、rgb函数（例如，rgb(0, 255, 0)或rgb函数以百分比代表颜色（例如，rgb(0%, 100%, 0%)）。这个函数在使用时，如果字符串无法被解析判别，将造成ValueError异常。这个函数的使用格式如下：

程序实例ch27_1.py: 使用getrgb()方法返回色彩的元组。

```
(r, g, b) = getrgb(color) # 返回色彩元组
```

```
1 # ch27_1.py
2 from PIL import ImageColor
3
4 print(ImageColor.getrgb("#0000ff"))
5 print(ImageColor.getrgb("rgb(0, 0, 255)"))
6 print(ImageColor.getrgb("rgb(0%, 0%, 100%)"))
7 print(ImageColor.getrgb("Blue"))
8 print(ImageColor.getrgb("blue"))
```

27-1-2 getcolor()

执行结果

```
===== RESTART: D:/Python/ch27/ch27_1.py
(0, 0, 255)
(0, 0, 255)
(0, 0, 255)
(0, 0, 255)
(0, 0, 255)
>>>
```

功能基本上与getrgb()相同，它的使用格式如下：

```
(r, g, b) = getcolor(color, "mode") # 返回色彩元组
```

mode若是填写“RGBA”则可返回RGBA元组，如果填写“RGB”则返回RGB元组，如果mode不是色彩，此函数将返回一个整数值。

程序实例ch27_2.py: 测试使用getcolor()函数，了解返回值。

```
1 # ch27_2.py
2 from PIL import ImageColor
3
4 print(ImageColor.getcolor("#0000ff", "RGB"))
5 print(ImageColor.getcolor("rgb(0, 0, 255)", "RGB"))
6 print(ImageColor.getcolor("Blue", "RGB"))
7 print(ImageColor.getcolor("#0000ff", "RGBA"))
8 print(ImageColor.getcolor("rgb(0, 0, 255)", "RGBA"))
9 print(ImageColor.getcolor("Blue", "RGBA"))
```

执行结果

```
===== RESTART: D:/Python/ch27/ch27_2.py
(0, 0, 255)
(0, 0, 255)
(0, 0, 255)
(0, 0, 255, 255)
(0, 0, 255, 255)
(0, 0, 255, 255)
>>>
```

27-2 Pillow模块的盒子元组(Box tuple)

下图是Pillow模块的图像坐标的观念。

最左上角的像素是(x, y)是(0, 0)，x轴像素值往右递增，y轴像素值往下递增。盒子元组的参数是(left, top, right, bottom)，意义如下：

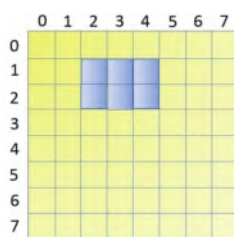
left: 盒子左上角的x轴坐标。

top: 盒子左上角的y轴坐标。

right: 盒子右下角的x轴坐标。

bottom: 盒子右下角的y轴坐标。

若是上图蓝底是一张图片，则可以用(2, 1, 4, 2)表示它的盒子元组(box tuple)，可想成它的图像坐标。



27-3 图像的基本操作

本节使用的图像文件是rushmore.jpg，在ch27文件夹可以找到，此图片内容如下。



27-3-1 打开图像对象

可以使用`open()`方法打开一个图像对象，参数是放置欲打开的图像文件。

27-3-2 图像大小属性

可以使用`size` 属性获得图像大小，这个属性可传回图像宽(`width`)和高(`height`)。

程序实例`ch27_3.py`： 在`ch27`文件夹有`rushmore.jpg`文件，这个程序会列出此图像文件的宽和高。

```
1 # ch27_3.py
2 from PIL import Image
3
4 rushMore = Image.open("rushmore.jpg") # 建立Pillow对象
5 print("列出对象类型：", type(rushMore))
6 width, height = rushMore.size # 获得图像宽度和高度
7 print("宽度 = ", width)
8 print("高度 = ", height)
```

执行结果

```
===== RESTART: D:\Python\ch27\ch27_3.py =====
列出对象类型： <class 'PIL.JpegImagePlugin.JpegImageFile'>
宽度 = 270
高度 = 161
>>>
```

27-3-3 取得图像对象文件名

可以使用filename 属性获得图像的源文件名称。

程序实例ch27_4.py: 获得图像对象的文件名。

```
1 # ch27_4.py
2 from PIL import Image
3
4 rushMore = Image.open("rushmore.jpg") # 建立Pillow对象
5 print("列出物件文件名 :", rushMore.filename)
```

执行结果

```
===== RESTART: D:\Python\ch27\ch27_4.py =====
列出物件文件名 :  rushmore.jpg
>>>
```

27-3-4 取得图像对象的文件格式

可以使用format 属性获得图像文件格式（可想成图像文件案的扩展名），此外，可以使用format_description 属性获得更详细的文件格式描述。

程序实例ch27_5.py: 获得图像对象的扩展名 与描述。

```
1 # ch27_5.py
2 from PIL import Image
3
4 rushMore = Image.open("rushmore.jpg") # 建立Pillow对象
5 print("列出对象扩展名 :", rushMore.format)
6 print("列出对象描述 : ", rushMore.format_description)
```

执行结果

```
===== RESTART: D:\Python\ch27\ch27_5.py =====
列出对象扩展名 :  JPEG
列出对象描述 :  JPEG (ISO 10918)
>>>
```

27-3-5 存储文件

可以使用save()方法存储文件，甚至我们也可以将jpg文件转存成png文件，同样是图片文件但是以不同格式存储。

程序实例ch27_6.py: 将rushmore.jpg转存成out27_6.png。

```

1 # ch27_6.py
2 from PIL import Image
3
4 rushMore = Image.open("rushmore.jpg")      # 建立Pillow对象
5 rushMore.save("out27_6.png")

```

执行结果 在ch27文件夹将可以看到所建的out27_6.png。

27-3-6 建立新的图像对象

可以使用new()方法建立新的图像对象，它的语法格式如下：

```

new(mode
, size, color=0)

```

mode 可以有多种设定，一般建议用“RGBA ”(建立png文件)或“RGB ”(建立jpg文件)即可。size参数是一个元组(tuple)，可以设定新图像的宽度和高度。color预设是黑色，不过我们可以参考附录D建立不同的颜色。

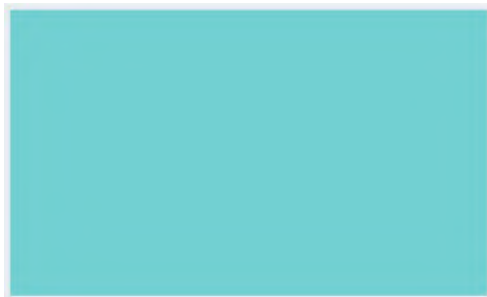
程序实例 ch27_7.py： 建立一个水蓝色(aqua)的图像文件out27_7.jpg。

```

1 # ch27_7.py
2 from PIL import Image
3
4 pictObj = Image.new("RGB", (300, 180), "aqua") # 建立aqua颜色图像
5 pictObj.save("out27_7.jpg")

```

执行结果 在ch27文件夹可以看到下列out27_7.jpg文件。



程序实例 ch27_8.py： 建立一个透明的黑色的图像文件 out27_8.png。

```
1 # ch27_8.py
2 from PIL import Image
3
4 pictObj = Image.new("RGBA", (300, 180)) # 建立完全透明图像
5 pictObj.save("out27_8.png")
```

执行结果 文件打开后因为透明，看不出任何效果。

27-4 图像的编辑

27-4-1 更改图像大小

Pillow模块提供resize()方法可以调整图像大小，它的使用语法如下：

```
resize((width, heigh), Image.BILINEAR) # 双线取样法，也可以省略
```

第一个参数是新图像的宽与高，以元组表示。第二个参数主要是设定更改图像所使用的方法，常见的除上述方法外，也可以设定Image.NEAREST 最低质量，Image.ANTIALIAS 最高质量，Image.BISCUBIC三次方取样法，一般可以省略。

程序实例ch27_9.py： 分别将图片宽度与高度增加为原先的2倍。

```
1 # ch27_9.py
2 from PIL import Image
3
4 pict = Image.open("rushmore.jpg") # 建立Pillow对象
5 width, height = pict.size
6 newPict1 = pict.resize((width*2, height)) # 宽度是2倍
7 newPict1.save("out27_9_1.jpg")
8 newPict2 = pict.resize((width, height*2)) # 高度是2倍
9 newPict2.save("out27_9_2.jpg")
```

执行结果 下列分别是out27_9_1.jpg(左)与out27_9_2.jpg(右)的执行结果。



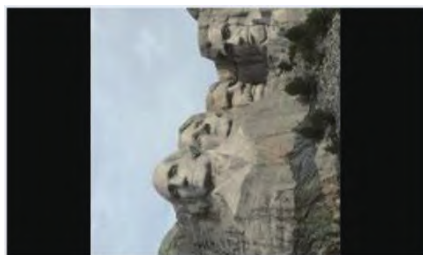
27-4-2 图像的旋转

Pillow模块提供`rotate()`方法可以逆时针旋转图像，如果旋转是90度或270度，图像的宽度与高度会有变化，图像本身比率不变，多的部分以黑色图像替代，如果是其他角度则图像维持不变。

程序实例ch27_10.py： 将图像分别旋转90度、180度和270度。

```
1 # ch27_10.py
2 from PIL import Image
3
4 pict = Image.open("rushmore.jpg")      # 建立Pillow对象
5 pict.rotate(90).save("out27_10_1.jpg")  # 旋转90度
6 pict.rotate(180).save("out27_10_2.jpg") # 旋转180度
7 pict.rotate(270).save("out27_10_3.jpg") # 旋转270度
```

执行结果 下列分别是旋转90、180、270度的结果。





在使用rotate()方法时也可以增加第2个参数expand=True, 如果有这个参数会放大图像, 让整个图像显示, 多余部分用黑色填满。

程序实例ch27_11.py: 没有使用expand=True参数与使用此参数的比较。

```
1 # ch27_11.py
2 from PIL import Image
3
4 pict = Image.open("rushmore.jpg") # 建立Pillow对象
5 pict.rotate(45).save("out27_11_1.jpg") # 旋转45度
6 pict.rotate(45, expand=True).save("out27_11_2.jpg") # 旋转45度图像扩充
```

执行结果 下列分别是out27_11_1.jpg与out27_11_2.jpg图像内容。



27-4-3 图像的翻转

可以使用transpose()让图像翻转，这个方法使用语法如下：

程序实例ch27_12.py： 图像左右翻转与上下翻转的实例。

```
transpose(Image.FLIP_LEFT_RIGHT)    # 图像左右翻转
transpose(Image.FLIP_TOP_BOTTOM)     # 图像上下翻转

1 # ch27_12.py
2 from PIL import Image
3
4 pict = Image.open("rushmore.jpg")    # 建立Pillow对象
5 pict.transpose(Image.FLIP_LEFT_RIGHT).save("out27_12_1.jpg") # 左右
6 pict.transpose(Image.FLIP_TOP_BOTTOM).save("out27_12_2.jpg") # 上下
```

执行结果 下列分别是左右翻转与上下翻转的结果。



27-4-4 图像像素的编辑

Pillow 模块的getpixel()方法可以取得图像某一位置像素(pixel)的色彩。

```
getpixel((x,y)) # 参数是元组(x,y)，这是像素位置
```

程序实例ch27_13.py： 先建立一个图像，大小是(300,100)，色彩是Yellow，然后列出图像中心点的色彩。最后将图像存储至out27_13.png。

```

1 # ch27_13.py
2 from PIL import Image
3
4 newImage = Image.new('RGBA', (300, 100), "Yellow")
5 print(newImage.getpixel((150, 50))) # 打印中心点的色彩
6 newImage.save("out27_13.png")

```

执行结果 下列是执行结果与out27_13.png内容。

```

===== RESTART: D:\Python\ch27\ch27_13.py =====
(255, 255, 0, 255)
>>>

```



Pillow模块的putpixel()方法可以在影响的某一个位置填入色彩，常用的使用语法如下：

```

putpixel((x,y)
, (r, g, b, a)) # 2个参数分别是位置

```

与色彩元组

上述色彩元组的值是在0~255间，若是省略a代表是不透明。另外我们也可以用27-1-2小节的getcolor()当做第2个参数，用这种方法可以直接用附录D的色彩名称填入指定像素位置，例如，下列是填入蓝色(blue)的方法：

```

putpixel((x,y), ImageColor
.getcolor("Blue", "RGBA")) # 需先导入
ImageColor

```

程序实例ch27_14.py： 建立一个300×300的图像，底色是黄色(Yellow)，然后(50, 50, 250, 150)是填入青色 (Cyan)，此时将上述执行结果存入out27_14_1.png。然后将蓝色 (Blue) 填入 (50, 151, 250, 250)，最后将结果存入out27_14_2.png。

```

1 # ch27_14.py
2 from PIL import Image
3 from PIL import ImageColor
4
5 newImage = Image.new('RGBA', (300, 300), "Yellow")
6 for x in range(50, 251):
7     for y in range(50, 151):
8         newImage.putpixel((x, y), (0, 255, 255, 255))
9 newImage.save("out27_14_1.png")
10 for x in range(50, 251):
11     for y in range(151, 251):
12         newImage.putpixel((x, y), ImageColor.getcolor("Blue", "RGBA"))
13 newImage.save("out27_14_2.png")

```

执行结果 下列分别是第一阶段与第二阶段的执行结果。



27-5 裁切、复制与图像合成

27-5-1 裁切图像

Pillow模块有提供crop()方法可以裁切图像，其中参数是一个元组，元组内容是（左, 上, 右, 下）的区间坐标。

程序实例ch27_15.py: 裁切(80, 30, 150, 100)区间。

```

1 # ch27_15.py
2 from PIL import Image
3
4 pict = Image.open("rushmore.jpg")      # 建立Pillow对象
5 cropPict = pict.crop((80, 30, 150, 100)) # 裁切区间
6 cropPict.save("out27_15.jpg")

```

执行结果 右图是out27_15. jpg的裁切结果。



27-5-2 复制图像

假设我们想要执行图像合成处理，为了不破坏原图像内容，建议可以先保存图像，再执行合成动作。Pillow模块有提供copy()方法可以复制图像。

程序实例ch27_16. py： 复制图像，再将所复制的图像存储。

```

1 # ch27_16.py
2 from PIL import Image
3
4 pict = Image.open("rushmore.jpg")      # 建立Pillow对象
5 copyPict = pict.copy()                  # 复制
6 copyPict.save("out27_16.jpg")

```

执行结果 下列是out27_16. jpg的执行结果。



27-5-3 图像合成

Pillow模块有提供paste()方法可以图像合成，它的语法如下：

底图图像.paste(插入图像, (x, y)) # (x, y) 元组是插入位置

程序实例ch27_17.py： 使用rushmore.jpg图像，为这个图像复制一份copyPict，裁切一份cropPict，将cropPict合成至copyPict内2次，将结果存入out27_17.jpg。

```
1 # ch27_17.py
2 from PIL import Image
3
4 pict = Image.open("rushmore.jpg")          # 建立Pillow对象
5 copyPict = pict.copy()                      # 复制
6 cropPict = copyPict.crop((80, 30, 150, 100)) # 裁切区间
7 copyPict.paste(cropPict, (20, 20))          # 第一次合成
8 copyPict.paste(cropPict, (20, 100))         # 第二次合成
9 copyPict.save("out27_17.jpg")               # 存储
```

执行结果



27-5-4 将裁切图片填满图像区间

在Windows操作系统使用中常看到图片填满某一区间，其实我们可以用双层循环完成这个工作。

程序实例ch27_18.py： 将一个裁切的图片填满某一个图像区间，最后存储此图像，在这个图像设计中，笔者也设定了留白区间，这区间是图像建立时的颜色。

```
1 # ch27_18.py
2 from PIL import Image
3
4 pict = Image.open("rushmore.jpg")          # 建立Pillow对象
5 copyPict = pict.copy()                      # 复制
6 cropPict = copyPict.crop((80, 30, 150, 100)) # 裁切区间
7 cropWidth, cropHeight = cropPict.size      # 获得裁切区间的宽与高
8
9 width, height = 600, 320                   # 新图像宽与高
10 newImage = Image.new('RGB', (width, height), "Yellow") # 建立新图像
11 for x in range(20, width-20, cropWidth):  # 双层循环合成
12     for y in range(20, height-20, cropHeight):
13         newImage.paste(cropPict, (x, y))    # 合成
14
15 newImage.save("out27_18.jpg")               # 保存
```

执行结果



27-6 在图像内绘制图案

Pillow模块内有一个ImageDraw模块，可以利用此模块绘制点 (Points)、线 (Lines)、矩形 (Rectangles)、椭圆 (Ellipses)、多边形 (Polygons)。

在图像内建立图案对象方式如下：

```
from PIL import Image, ImageDraw
newImage = Image.new('RGBA', (300, 300), "Yellow") # 建立300×300黄色底的图像
drawObj = ImageDraw.Draw(newImage)
```

27-6-1 绘制点

ImageDraw模块的point()方法可以绘制点，语法如下：

```
point([(x1,y1), ... (xn,yn)], fill) #fill是设定颜色
```

第一个参数是由元组(tuple)组成的列表，(x, y)是欲绘制的点坐标。fill可以是RGBA()或是直接指定颜色。

27-6-2 绘制线条

ImageDraw模块的line()方法可以绘制线条，语法如下：


```
line([(x1,y1), ... (xn,yn)], width  
,fill) # width
```

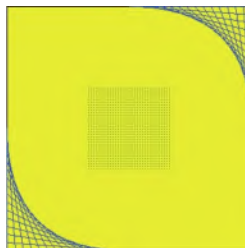
是宽度，预设是1

第一个参数是由元组(tuple)组成的列表，(x,y)是欲绘制线条的点坐标，如果多于2个点，则这些点会串接起来。fill可以是RGBA()或是直接指定颜色。

程序实例ch27_19.py: 绘制点和线条的应用。

```
1 # ch27_19.py  
2 from PIL import Image, ImageDraw  
3  
4 newImage = Image.new('RGBA', (300, 300), "Yellow") # 建立300*300黄色底的图像  
5 drawObj = ImageDraw.Draw(newImage)  
6  
7 # 绘制点  
8 for x in range(100, 200, 3):  
9     for y in range(100, 200, 3):  
10         drawObj.point([(x,y)], fill='Green')  
11  
12 # 绘制线条，绘外框线  
13 drawObj.line([(0,0), (299,0), (299,299), (0,299), (0,0)], fill="Black")  
14 # 绘制右上角美工线  
15 for x in range(150, 300, 10):  
16     drawObj.line([(x,0), (300,x-150)], fill="Blue")  
17 # 绘制左下角美工线  
18 for y in range(150, 300, 10):  
19     drawObj.line([(0,y), (y-150,300)], fill="Blue")  
20 newImage.save("out27_19.png")
```

执行结果



27-6-3 绘制圆或椭圆

ImageDraw模块的ellipse()方法可以绘制圆或椭圆，语法如下：

```
ellipse((left,top,right,bottom),fill, outline  
) # outline
```


是外框颜色

第一个参数是由元组(tuple)组成的, (left, top, right, bottom)是包住圆或椭圆的矩形左上角与右下角的坐标。fill可以是RGBA()或是直接指定颜色, outline是可选择是否加上。

27-6-4 绘制矩形

ImageDraw模块的rectangle()方法可以绘制矩形, 语法如下:

```
rectangle((left,top,right,bottom),fill, outline
) # outline
```

是外框颜色

第一个参数是由元组(tuple)组成的, (left, top, right, bottom)是矩形左上角与右下角的坐标。fill可以是RGBA()或是直接指定颜色, outline是可选择是否加上。

27-6-5 绘制多边形

ImageDraw模块的polygon()方法可以绘制多边形, 语法如下:

```
polygon([(x1,y1), ... (xn,yn)],fill, outline) # outline是外  
框颜色
```

第一个参数是由元组(tuple)组成的列表, (x, y)是欲绘制多边形的点坐标, 在此需填上多边形各端点坐标。fill可以是RGBA()或是直接指定颜色, outline是可选择是否加上。

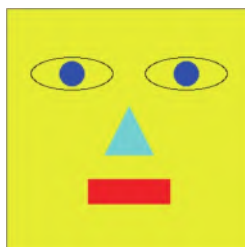
程序实例ch27_20.py: 设计一个图案。

```

1 # ch27_20.py
2 from PIL import Image, ImageDraw
3
4 newImage = Image.new('RGBA', (300, 300), 'Yellow') # 建立300*300黄色底的图像
5 drawObj = ImageDraw.Draw(newImage)
6
7 drawObj.rectangle((0,0,299,299), outline='Black') # 图像外框线
8 drawObj.ellipse((30,60,130,100),outline='Black') # 左眼外框
9 drawObj.ellipse((65,65,95,95),fill='Blue') # 左眼
10 drawObj.ellipse((170,60,270,100),outline='Black') # 右眼外框
11 drawObj.ellipse((205,65,235,95),fill='Blue') # 右眼
12 drawObj.polygon([(150,120),(180,180),(120,180),(150,120)],fill='Aqua') # 鼻子
13 drawObj.rectangle((100,210,200,240), fill='Red') # 嘴
14 newImage.save("out27_20.png")

```

执行结果



27-7 在图像内填写文字

ImageDraw模块也可以用于在图像内填写英文或中文，所使用的函数是text()，语法如下：

```

text((x,y), text
,fill, font) # text

```

是要写入的文字

如果要使用默认方式填写文字，可以省略font参数，可以参考ch27_21.py第8行。如果想要使用其他字体填写文字，需调用ImageFont.truetype()方法选用字体，同时设定字号。在使用ImageFont.truetype()方法前需在程序前方导入ImageFont模块，可参考ch27_21.py第2行，这个方法的语法如下：

```

text(字体路径, 字号)

```

在Windows系统字体是放在C:\Windows\Fonts文件夹内，在此你可以选择想要的字体。

点选字体，单击鼠标右键，执行内容，再选安全性 卷标可以看到此字体的文件名。下列是点选Old English Text的示范输出。



读者可以用复制方式获得字体的路径，有了字体路径后，就可以轻松地在图像内输出各种字体了。

程序实例ch27_21.py：在图像内填写文字，第8～9行是使用默认字体，执行英文字符串“Ming-Chi Institute of Technology”的输出。第10～11行是设定字体为Old English Text，字号是36，输出相同的字符串。第13～15行是设定字体为华康新综艺体，字号是48，输出中文字符串“明志科技大学”。

注 如果你的计算机没有华康新综艺体，执行这个程序会有错误，所以笔者有附一个ch27_21_1.py是使用Microsoft的新细明体字体，你可以自行体会中文的输出。

```

1 # ch27_21.py
2 from PIL import Image, ImageDraw, ImageFont
3
4 newImage = Image.new('RGBA', (600, 300), 'Yellow') # 建立300*300黄色底的图像
5 drawObj = ImageDraw.Draw(newImage)
6
7 strText = 'Ming-Chi Institute of Technology' # 设定欲打印英文字符串
8 drawObj.text((50,50), strText, fill='Blue') # 使用默认字体与字号
9 # 使用古老英文字体, 字号是36
10 fontInfo = ImageFont.truetype('C:\Windows\Fonts\OLDENGL.TTF', 36)
11 drawObj.text((50,100), strText, fill='Blue', font=fontInfo)
12 # 处理中文字体
13 strCtext = '明志科技大学' # 设定欲打印中文字符串
14 fontInfo = ImageFont.truetype('C:\Windows\Fonts\DFZongYiStd-W9.otf', 48)
15 drawObj.text((50,180), strCtext, fill='Blue', font=fontInfo)
16 newImage.save("out27_21.png")

```

执行结果



27-8 建立QR code

QR code是目前最流行的二维扫描码，1994年日本Denso-Wave公司发明的，英文字QR所代表的意义是Quick Response(快速反应)。它的最大特色是可以存储比普通条形码更多的资料，同时也不需对准扫描仪。使用前需安装模块：

```
pip install qrcode
```

常用的几个方法如下：

img

```
= qrcode.make("网址数据") # 产生网址数据的QR code对象img
img
```

```
.save("filename") #filename是存储QR code的文件名
```

程序实例 ch27_22.py： 建立 <http://www.deepstone.com.tw> 的 QR code，这个程序会先列出img对象的数据类型，同时将此对象存入

out27_22.jpg文件内。

```
1 # ch27_22.py
2 import qrcode
3
4 codeText = 'http://www.deepstone.com.tw'
5 img = qrcode.make(codeText) # 建立QR code 对象
6 print("文件格式", type(img))
7 img.save("out27_22.jpg")
```

执行结果 下列分别是执行结果与out27_22.jpg的QR code结果。

```
===== RESTART: D:\Python\ch27\ch27_22.py =====
文件格式 <class 'qrcode.image.pil.PilImage'>
>>>
```



习题

1. 请用自己的大头照，通过设置照片高度和宽度来调整其规格，然后贴在图像文件中，设置方式如下：
 - a): 高度不变，宽度是1.2倍。
 - b): 高度不变，宽度是1.5倍。
 - c): 高度不变，宽度是50%。
 - d): 高度不变，宽度是80%。
 - e): 宽度不变，高度是1.2倍。

f): 宽度不变, 高度是1.5倍。

g): 宽度不变, 高度是80%。

h): 宽度不变, 高度是50%。

最后请在图像最上方加上自己的中文名字。

2. 请参考护照照片规格, 将自己的大头贴参考27_18.py方式布局在图像文件内, 用高级相片纸在7-11或其他便利商店打印, 这样就可以省下护照照片的钱了, 请交出所布局的图像文件。
3. 设计自己签名的图像文件, 用循环搜寻自己计算机的硬盘, 打开所有图像文件, 将自己签名的图像文件填在每一个图像文件右下角。
4. 请参考ch27_19.py, 扩充此程序功能, 将美工线条的观念应用在左上角与右下角。
5. 请参考ch27_21.py, 选用10种字体输出就读学校的中英文名称, 各5种。
6. 请建立自己母校的QR code。

第28章 用Python控制鼠标、屏幕与键盘

本章主要说明使用Python控制鼠标、屏幕与键盘的应用。为了执行本章的程序，请安装pyautogui模块。

```
pip install pyautogui
```

28-1 鼠标的控制

28-1-1 提醒事项

由于这一章将讲解鼠标的控制，用户可能会因为程序设计错误造成对鼠标失去控制，造成程序无法控制，甚至无法使用鼠标结束程序，最后可能需使用下列方式结束计算机。

方法1:

Windows: 同时按Ctrl + Alt + Del。

Mac OS: 同时按Command + Shift + Option + Q。

方法2:

或是在设计程序时，每次启用pyautogui的方法设定暂停3秒再执行。

```
>>> pyautogui.PAUSE = 3
>>>
```

这时快速处理移动鼠标关闭程序。

方法3:

也可以使用下列语法先设定Python的安全防护功能失效。

```
>>> import pyautogui
>>> pyautogui.PAUSE = 3
>>> pyautogui.FAILSAFE = True
>>>
```

首先在暂停3秒钟期间，你可以快速将鼠标光标移至屏幕左上角，这时会产生`pyautogui.FailSafeException`异常，可以设计让程序终止。

28-1-2 屏幕坐标

我们操作鼠标时可以看到鼠标光标在屏幕上移动，对鼠标而言，屏幕坐标的基准点(0,0)位置在左上角，往右移动x轴坐标会增加，往左移动x轴坐标会减少。往下移动y轴坐标会增加，往上移动y轴坐标会减少。

坐标的单位是Pixel(像素)，每一台计算机的像素可能不同，可以用`size()`方法获得计算机屏幕的像素，这个方法传回2个值，分别是屏幕宽度和高度。

程序实例`ch28_1.py`：列出目前使用计算机的像素。

```
1 # ch28_1.py
2 import pyautogui
3
4 width, height = pyautogui.size() # 设定屏幕宽度和高度
5 print(width, height)             # 打印屏幕宽度和高度
```

执行结果

```
===== RESTART: D:\Python\ch28\ch28_1.py
1920 1080
>>>
```

由上图笔者可以得到目前所用计算机屏幕像素规格如下：



28-1-3 获得鼠标光标位置

在`pyautogui`模块内有`position()`方法可以获得鼠标光标位置，这个方法会传回2个值，分别是鼠标光标的x轴和y轴坐标。

程序实例ch28_2.py: 获得鼠标光标位置。

```
1 # ch28_2.py
2 import pyautogui
3
4 xloc, yloc = pyautogui.position() # 获得鼠标光标位置
5 print(xloc, yloc)               # 打印鼠标光标位置
```

执行结果

```
===== RESTART: D:\Python\ch28\ch28_2.py =====
559 663
>>>
```

程序实例ch28_3.py: 这个程序会持续打印鼠标光标位置，直到鼠标光标x轴位置到达1000(含)以上才停止。

```
1 # ch28_3.py
2 import pyautogui
3
4 xloc = 0
5 while xloc < 1000:
6     xloc, yloc = pyautogui.position() # 获得鼠标光标位置
7     print(xloc, yloc)                # 打印鼠标光标位置
```

执行结果 下列是部分画面。

```
===== RESTART: D:\Python\ch28\ch28_3.py =====
643 521
642 520
642 520
642 520
642 520
```

28-1-4 绝对位置移动鼠标

在pyautogui模块内有moveTo()方法可以将鼠标移至光标设定位置，它的使用格式如下。

moveTo(x坐标, y坐标, duration=xx) # xx是移动至此坐标的时间

```
1 # ch28_4.py
2 import pyautogui
3
4 x, y = 300, 300
5 for i in range(5):
6     pyautogui.moveTo(x, y, duration=0.5) # 左上角
7     pyautogui.moveTo(x+1200, y, duration=0.5) # 右上角
8     pyautogui.moveTo(x+1200, y+400, duration=0.5) # 右下角
9     pyautogui.moveTo(x, y+400, duration=0.5) # 左下角
```

程序实例ch28_4.py： 控制光标在一个矩形区间移动，下列程序duration是设定光标移动至此坐标的时间，我们可以自行设定此时间。可以得到鼠标光标在左上角(300, 300)、右上角(1500, 300)、右下角(1500, 700)和左下角(300, 700)间移动5次。

执行结果

28-1-5 相对位置移动鼠标

在pyautogui模块内有moveRel()方法可以将鼠标移至相较于前一次光标的相对位置，一般是适用在移动距离较短的情况，它的使用格式如下。

moveRel(x位移, y位移, duration=xx) # xx是移动至此坐标相对位置的时间

程序实例ch28_5.py： 控制光标在一个正方形区间移动，程序执行会以光标位置为左上角，然后在正方形区间移动。程序执行期间，你将发现我们无法自主控制鼠标光标。

```
1 # ch28_5.py
2 import pyautogui
3
4 for i in range(5):
5     pyautogui.moveRel(300, 0, duration=0.5) # 往右上角移动
6     pyautogui.moveRel(0, 300, duration=0.5) # 往右下角移动
7     pyautogui.moveRel(-300, 0, duration=0.5) # 往左下角移动
8     pyautogui.moveRel(0, -300, duration=0.5) # 往左上角移动
```

执行结果 本程序执行结果与ch28_4.py相同。

28-1-6 键盘Ctrl-C键

如果我们现在执行ch28_3.py，可以发现除了鼠标光标在x轴超出1000像素坐标可以终止程序外，如果按下Ctrl-C键，也可以产生KeyboardInterrupt异常，造成程序终止。



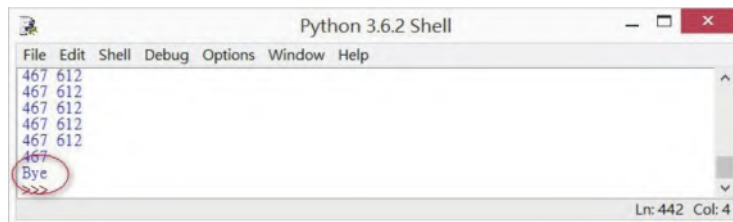
了解了上述特性，我们可以改良ch28_3.py。

程序实例ch28_6.py： 重新设计ch28_3.py，增加若是读者按键盘的Ctrl-C键，也可以让程序终止执行，当然要设计这类程序需借用异常处理。这个程序如果是异常结束将跳一行输出Bye字符串。

```

1 # ch28_6.py
2 import pyautogui
3
4 xloc = 0
5 print('按Ctrl-C 可以中断本程序')
6 try:
7     while xloc < 1000:
8         xloc, yloc = pyautogui.position() # 获得鼠标光标位置
9         print(xloc, yloc) # 打印鼠标光标位置
10 except KeyboardInterrupt:
11     print('\nBye')
```

执行结果



其实在教导读者时总是想一步一步引导读者，现在我们已经可以控制让键盘产生异常，让程序终止，所以设计程序已经不需要侦测限制鼠标光标所在位置，让程序终止。

程序实例ch28_7.py： 重新设计ch28_6.py，让鼠标可以在所有屏幕区间移动，程序只有按Ctrl-C键才会终止。

```

1 # ch28_7.py
2 import pyautogui
3
4 print('按Ctrl-C 可以中断本程序')
5 try:
6     while True:
7         xloc, yloc = pyautogui.position() # 获得鼠标光标位置
8         print(xloc, yloc)                # 打印鼠标光标位置
9     except KeyboardInterrupt:
10        print('\nBye')

```

执行结果 程序将不断显示鼠标光标位置，直至按Ctrl-C键。

28-1-7 让鼠标位置的输出在固定位置

在讲解本节功能前，笔者想先以实例介绍一个字符串的方法`rjust()`，这个方法可以让字符串在设定的区间靠右输出。

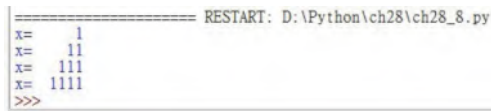
程序实例ch28_8.py： 设定4格空间，让数字靠右对齐输出，下列`print()`函数内有`str()`主要是将数字转成字符串。

```

1 # ch28_8.py
2
3 x1 = 1
4 x2 = 11
5 x3 = 111
6 x4 = 1111
7 print("x= ", str(x1).rjust(4))
8 print("x= ", str(x2).rjust(4))
9 print("x= ", str(x3).rjust(4))
10 print("x= ", str(x4).rjust(4))

```

执行结果



```

===== RESTART: D:\Python\ch28\ch28_8.py
x=    1
x=   11
x=  111
x= 1111
>>>

```

相信读者应该了解了上述`rjust()`的用法了，如果我们要将上述输出固定在同一行，也就是后面输出要遮盖住前面的输出，可以在`print()`函数内设定输出后，不执行跳行，而是使用`end=“\r”`参数，`\r`是逸出字符，可参考3-4-3节，主要是让鼠标光标到最左位置，然后再增加`flush=True`。

程序实例ch28_9.py: 每次输出后可以暂停1秒, 下一个输出将遮盖住前一个的输出。不过这个程序在Python Shell窗口将无效, 必须在DOS模式执行。

```
1 # ch28_9.py
2 import time, sys
3
4 x1 = 1
5 x2 = 11
6 x3 = 111
7 x4 = 1111
8 print("x= ", str(x1).rjust(4), end="\r", flush=True)
9 time.sleep(1)
10 print("x= ", str(x2).rjust(4), end="\r", flush=True)
11 time.sleep(1)
12 print("x= ", str(x3).rjust(4), end="\r", flush=True)
13 time.sleep(1)
14 print("x= ", str(x4).rjust(4), end="\r", flush=True)
```

执行结果 下方分别是DOS模式输出和Python Shell窗口输出的结果。

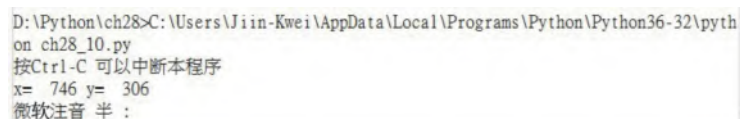


有了上述观念我们很容易设计下列观念的程序。

程序实例ch28_10.py: 鼠标光标在屏幕移动, 同时在固定位置输出鼠标光标的坐标。

```
1 # ch28_10.py
2 import pyautogui
3 import time
4
5 print('按Ctrl-C 可以中断本程序')
6 try:
7     while True:
8         xloc, yloc = pyautogui.position() # 获得鼠标光标位置
9         xylocStr = "x= " + str(xloc).rjust(4) + " y= " + str(yloc).rjust(4)
10        print(xylocStr, end="\r", flush=True) # 设定同一行最左边输出
11        time.sleep(1)
12 except KeyboardInterrupt:
13     print('\nBye')
```

执行结果 下方分别是DOS模式输出和Python Shell窗口输出的结果。



```

===== RESTART: D:\Python\ch28\ch28_10.py =====
按Ctrl-C 可以中断本程序
x= 1902 y= 624 x= 1902 y= 624 x= 719 y= 194 x= 981 y= 151 x= 818 y= 203
x= 931 y= 220 x= 927 y= 220 x= 1118 y= 235
Bye
>>>

```

28-1-8 单击鼠标click()

click()方法主要是可以设定在目前鼠标光标位置单击，所谓的单击通常是指单击鼠标左键。基本语法如下：

```

click(x, y, button='xx
') # xx

```

是left, middle或right, 预设是left

若是省略x, y, 则使用目前鼠标位置单击，若不指定按哪一个键，则默认是单击鼠标右键。

程序实例ch28_11.py： 让鼠标光标在(500, 450)位置产生单击的效果。

```

1 # ch28_11.py
2 import pyautogui
3
4 pyautogui.moveTo(500, 450)
5 pyautogui.click()

```

执行结果 由于我们没有设定任何动作，所以将只看到鼠标光标移至(500, 450)。

其实也可以在click()内增加位置参数，这时方法内容是click(x, y)，这样就可以用一个click()方法代替需使用2个方法的ch28_11.py。

程序实例 ch28_12.py： 在click()内增加位置参数重新设计ch28_11.py。

```

1 # ch28_12.py
2 import pyautogui
3
4 pyautogui.click(500, 450)

```

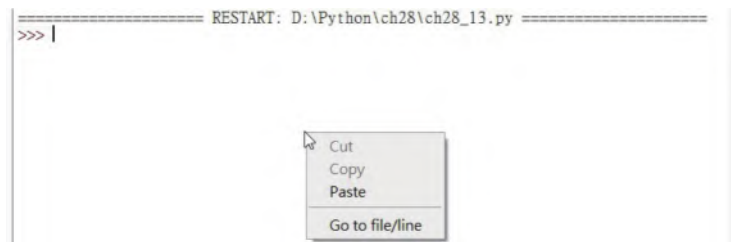

执行结果 由于我们没有设定任何动作，所以将只看到鼠标光标移至(500, 450)。

`click()` 函数默认是单击鼠标左键，也可以更改所按的键。

程序实例ch28_13.py: 重新设计ch28_12.py，改为单击鼠标右键，在许多窗口单击鼠标右键相当于有打开快捷菜单的效果。

```
1 # ch28_13.py
2 import pyautogui
3
4 pyautogui.click(500, 450, button='right')
```

执行结果 由于笔者鼠标是在Python Shell窗口，所以可以打开快捷菜单。



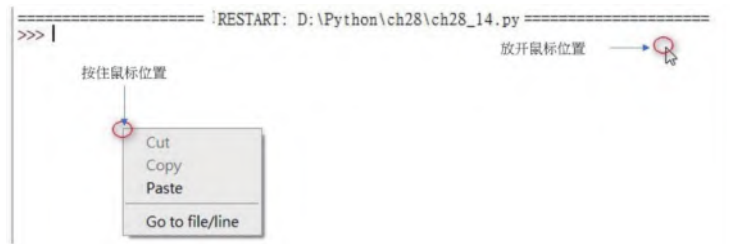
28-1-9 按住与放开鼠标`mouseDown()`和`mouseUp()`

`click()` 是指单击鼠标键然后放开，其实单击鼠标键时也可以用`mouseDown()`代替，放开鼠标键时可以用`mouseUp()`代替。这2个方法所使用的参数意义与`click()`相同。

程序实例ch28_14.py: 控制在目前鼠标光标位置按着鼠标右键，1秒后，放开所按的鼠标右键，同时鼠标光标移至(800, 300)位置。

```
1 # ch28_14.py
2 import pyautogui
3 import time
4
5 pyautogui.mouseDown(button='right') # 在鼠标光标位置按住鼠标右键
6 time.sleep(1)
7 pyautogui.mouseUp(800, 300, button='right') # 放开后鼠标光标在(800, 300)
```

执行结果



28-1-10 拖曳鼠标

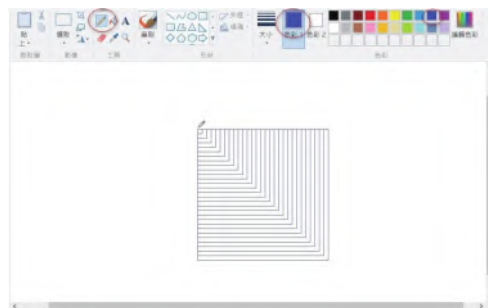
拖曳是指按着鼠标左键不放，然后移动鼠标，这个移动会在窗口画面留下轨迹，拖曳到目的位置后再放开鼠标按键。所使用的方法是`dragTo()`/`dragRel()`，这2个参数的使用与`moveTo()`/`moveRel()`相同。

有了以上观念，我们可以打开Windows系统的画图，然后绘制图形。

程序实例ch28_15.py: 这个程序在执行最初有10秒钟可以选择让绘图软件变成当前工作窗口，选择画笔和颜色，完成后请让鼠标光标停留在绘图起始点。

```
1 # ch28_15.py
2 import pyautogui
3 import time
4
5 time.sleep(10) # 这10秒需要绘图窗口取得焦点,选择画笔和选择颜色
6 pyautogui.click() # 单击设定绘图起始点
7 displacement = 10
8 while displacement < 300:
9     pyautogui.dragRel(displacement, 0, duration=0.2)
10    pyautogui.dragRel(0, displacement, duration=0.2)
11    pyautogui.dragRel(-displacement, 0, duration=0.2)
12    pyautogui.dragRel(0, -displacement, duration=0.2)
13    displacement += 10
```

执行结果



28-1-11 窗口滚动scroll()

可以使用scroll()执行窗口的滚动，我们在Windows系统内可能打开很多窗口，这个方法会针对当前鼠标光标所在窗口执行滚动，它的语法如下：

```
scroll(clicks, x
="xpos", y
="ypos") # x,y
```

是鼠标光标移动位置，可以省略

上述clicks是窗口滚动的单位数，单位大小会因不同平台而不同，正值 是往上滚动，负值 是往下滚动。如果有x,y，则先将鼠标光标移至指定位置，然后才开始滚动。

程序实例ch28_16.py： 窗口滚动的应用，如果程序执行期间鼠标光标切换新的工作窗口，将造成新的窗口滚动。

```
1 # ch28_16.py
2 import pyautogui
3 import time
4
5 for i in range(1,10):
6     pyautogui.scroll(30) # 往上滚动
7     time.sleep(1)
8     pyautogui.scroll(-30) # 往下滚动
9     time.sleep(1)
```

执行结果 读者可以试着切换工作窗口以体会窗口的滚动。

28-2 屏幕的处理

在pyautogui模块内有屏幕截图功能，截取屏幕图形后将产生Pillow的Image对象，可参考Pillow模块的功能，本节将分析这个实用的功能。

28-2-1 截取屏幕画面

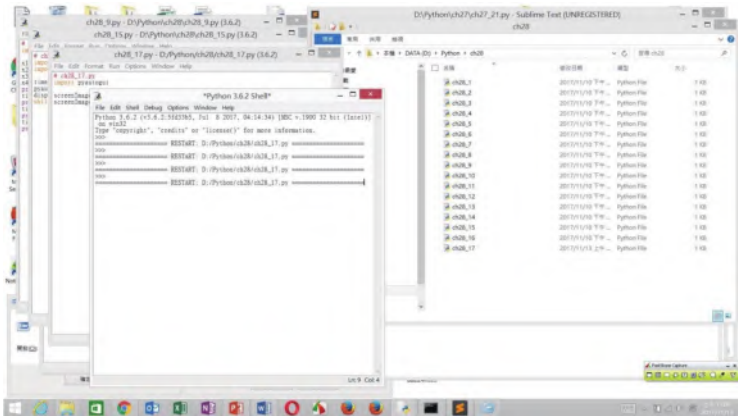
在pyautogui模块内有screenshot()方法，可用这个方法执行屏幕截图，屏幕截取后可以将它视为一个图像对象，所以可以使用save()方法

存储此对象，也可以直接在screenshot()的参数中设定欲存的文件名。

程序实例 ch28_17.py： 截取屏幕，同时存入 out28_17_1.jpg 和 out28_17_2.jpg。

```
1 # ch28_17.py
2 import pyautogui
3
4 screenImage = pyautogui.screenshot("out28_17_1.jpg") # 方法1
5 screenImage.save("out28_17_2.jpg") # 方法2
```

执行结果 下列是笔者截取屏幕画面的执行结果。



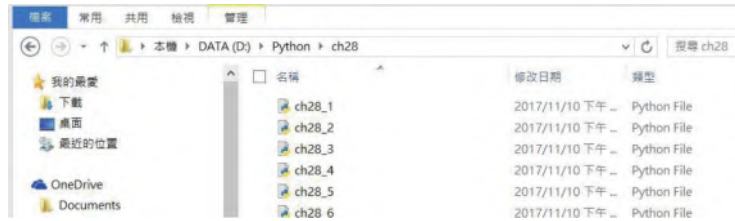
28-2-2 裁切屏幕图形

我们可以参考前一章的crop()方法裁切屏幕图形，此方法的参数是一个定义裁切画面区间的元组。

程序实例ch28_18.py： 裁切屏幕图形，下列是笔者屏幕的执行结果。

```
1 # ch28_18.py
2 import pyautogui
3
4 screenImage = pyautogui.screenshot()
5 cropPict = screenImage.crop((960,210,1900,480))
6 cropPict.save("out28_18.jpg")
```

执行结果



28-2-3 获得图像某位置的像素色彩

可以使用`getpixel((x,y))`获得`x,y`坐标的像素色彩，由于屏幕截图完全没有透明，所以所获得的是RGB的色彩元组。

程序实例ch28_19.py： 列出固定位置的RGB色彩元组。

```
1 # ch28_19.py
2 import pyautogui
3
4 screenImage = pyautogui.screenshot()
5 x, y = 200, 200
6 print(screenImage.getpixel((x,y)))
```

执行结果 下列是笔者屏幕的执行结果，读者屏幕可能有不一样的结果。

```
===== RESTART: D:\Python\ch28\ch28_19.py =====
(255, 255, 255)
>>>
```

28-2-4 色彩的比对

有时候我们可能需要确定某一个像素坐标的色彩是否是某种颜色，这时可以使用色彩比对功能`pixelMatchesColor()`，它的语法格式如下：

```
boolean = pyautogui.pixelMatchesColor(x,y, (Rxx,Gxx,Bxx))
```

上述`x,y`参数是坐标，会将此坐标的色彩取回，然后和第2个参数的色彩比对，如果相同则返回`True`，否则返回`False`。

程序实例ch28_20.py： 像素色彩比对的应用，读者计算机可能会有不一样的结果。

```
1 # ch28_20.py
2 import pyautogui
3
4 x, y = 200, 200
5 trueFalse = pyautogui.pixelMatchesColor(x,y,(255,255,255))
6 print(trueFalse)
7 trueFalse = pyautogui.pixelMatchesColor(x,y,(0,0,255))
8 print(trueFalse)
```

执行结果

```
===== RESTART: D:/Python/ch28/ch28_20.py =====
False
False
>>>
```

28-3 使用Python控制键盘

我们也可以利用pyautogui模块对键盘做一些控制。

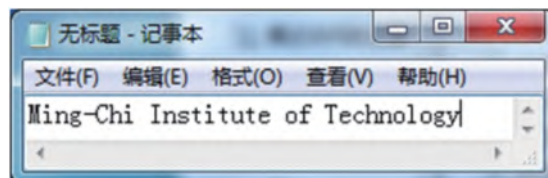
28-3-1 基本传送文字

pyautogui模块内有typewrite()方法，可以对目前焦点窗口传送文字，不过经笔者测试这个功能目前无法传送中文字。

程序实例ch28_21.py：请在10秒之内打开一个新的记事本编辑窗口，将输入环境设为英文输入，同时设为当前焦点窗口，这个程序会在此窗口输入“Ming-Chi Institute of Technology”。

```
1 # ch28_21.py
2 import pyautogui
3 import time
4
5 print("请在10秒内开启记事本并设为焦点窗口")
6 time.sleep(10)
7 pyautogui.typewrite('Ming-Chi Institute of Technology')
```

执行结果



typewrite()函数的参数是要传输的字符串，我们也可以增加第2个参数，所增加的参数是数字，代表相隔多少秒输出一个字符。

程序实例ch28_22.py： 重新设计ch28_11.py，输出相同字符串，但是每隔0.1秒输出一个字母。

```
1 # ch28_22.py
2 import pyautogui
3 import time
4
5 print("请在10秒内开启记事本并设为焦点窗口")
6 time.sleep(10)
7 pyautogui.typewrite('Ming-Chi Institute of Technology', 0.1)
```

执行结果 每隔0.1秒输出一个字母，最后结果与ch28_21.py相同。

28-3-2 键盘按键名称

我们也可以使用输入单一字符方式执行键盘数据的输入，此时typewrite()的第一个参数是字符列表。

程序实例ch28_23.py： 每隔1秒输入一个英文字符。

```
1 # ch28_23.py
2 import pyautogui
3 import time
4
5 print("请在10秒内开启记事本并设为焦点窗口")
6 time.sleep(10)
7 pyautogui.typewrite(['M', 'i', 'n', 'g'], 1)
```

执行结果

A screenshot of a text input field, likely from a Windows Notepad application, showing the word "Ming" typed into it. The text is in a standard black font on a white background.

有些键盘是具有特殊功能的，例如，backspace应如何使用Python表达，光标左移应如何使用Python表达？在pyautogui.KEYBOARD_KEYS列表有完整说明，下列是使用相同名称英文的列表。

Python 输入	意义
'a' , 'A' , '\$'	相同字义
'enter' 或 '\n'	键盘 Enter
'backspace'	键盘 Backspace
'delete'	键盘 Del
'esc'	键盘 Esc
'f1' , 'f2' ,	键盘 F1, F2,
'tab' 或 '\t'	键盘 Tab
'printscreen'	键盘 PrtSc
'insert'	键盘 Ins

下列是比较特殊的Python输入与意义表。

Python 输入	意义
'altleft' , 'altright'	键盘左右 Alt
'ctrlleft' , 'ctrlright'	键盘左右 Ctrl
'shiftright' , 'shiftright'	键盘左右 Shift
'home' , 'end'	键盘 Home, End
'pageup' , 'pagedown'	键盘 PgUp, PgDn
'up' , 'down' , 'left' , 'right'	键盘上, 下, 左, 右
'winleft' , 'winright'	键盘左 Win 键, 右 Win 键
'command'	Mac OS 系统 command 键
'option'	Mac OS 系统 option 键

程序实例ch28_24.py： 使用特殊按键输出字符串“Ming”，由于每隔1秒才输出一个字符，所以读者可以注意它的执行变化。

```

1 # ch28_24.py
2 import pyautogui
3 import time
4
5 print("请在10秒内开启记事本并设为焦点窗口")
6 time.sleep(10)
7 pyautogui.typewrite(['M', 'i', 'm', 'g', 'left', 'left', 'del', 'n'], 1)

```

执行结果



程序实例ch28_25.py： 使用Python控制键盘输入，同时让光标在不同行间执行工作。第一行输出笔者故意输入错误，第二行则去修改第一行的错误。

```
1 # ch28_25.py
2 import pyautogui
3 import time
4
5 print("请在10秒内开启记事本并设为焦点窗口")
6 time.sleep(10)
7 pyautogui.typewrite(['M', 'i', 'n', 'k', 'enter'], 1)
8 pyautogui.typewrite(['M', 'i', 'n', 'g', 'up', 'backspace', 'g'], 1)
```

执行结果



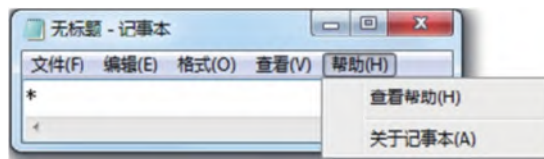
28-3-3 按下与放开按键

在pyautogui模块中keyDown()是按下键盘按键同时不放开按键，keyUp()是放开所按的键盘按键。keyPress()则是按下并放开。

程序实例ch28_26.py： 这个程序会输出“*”字符和打开记事本的帮助菜单。

```
1 # ch28_26.py
2 import pyautogui
3 import time
4
5 print("请在10秒内开启记事本并设为焦点窗口")
6 time.sleep(10)
7 # 以下输出*
8 pyautogui.keyDown('shift')
9 pyautogui.press('8')
10 pyautogui.keyUp('shift')
11 # 以下开启帮助菜单
12 pyautogui.keyDown('alt')
13 pyautogui.press('H')
14 pyautogui.keyUp('alt')
```

执行结果



28-3-4 快速组合键

在pyautogui模块中hotkey()可以用于按键的组合，下列将直接以实例说明。

程序实例ch28_27.py： 使用hotkey()重新设计ch28_26.py。

```
1 # ch28_27.py
2 import pyautogui
3 import time
4
5 print("请在10秒内开启记事本并设为焦点窗口")
6 time.sleep(10)
7 pyautogui.hotkey('shift', '8') # 输出*
8 pyautogui.hotkey('alt', 'H') # 开启帮助菜单
```

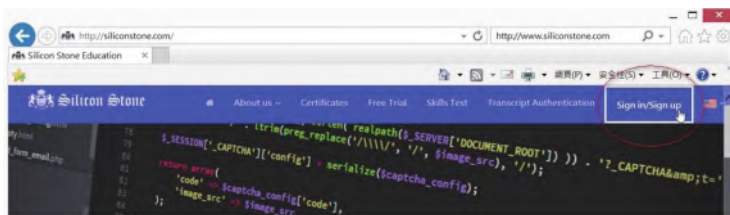
执行结果 与ch28_26.py相同。

28-4 网络窗体的填写

当我们使用Python可以控制鼠标、键盘和屏幕后，其实就可以利用键盘执行网络窗体的输入了，这一节笔者将以实例说明网络窗体的输入。请进入下列网站：

<http://www.siliconstone.com>

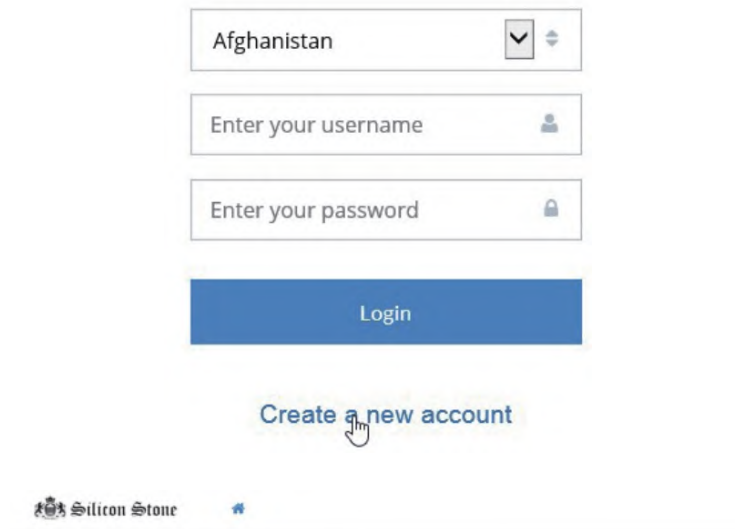
这是国外著名国际认证公司的网址，笔者将以此为实例说明如何利用Python填写窗体，然后点选Sign in/Sign up。



将看到下列画面：

请单击Create a new account。

Login



A screenshot of a web application's login page. At the top, there is a dropdown menu showing 'Afghanistan' with a downward arrow icon. Below this are two input fields: 'Enter your username' with a person icon and 'Enter your password' with a lock icon. A blue 'Login' button is positioned below the password field. Under the button is a link that says 'Create a new account' with a hand cursor icon pointing to it. At the bottom, there is a logo for 'Silicon Stone' and a small blue icon.

Language

Select your preferred language.

- | | | | |
|-----------|--------------------|----------|------------|
| » English | » Bahasa Indonesia | » German | » Romanian |
| » 中国-简体中文 | » Bahasa Malaysia | » Greek | » Russian |

请点选中国-简体中文，然后可以看到下列窗体。



A screenshot of a web application's registration page. At the top, there is a logo for 'Silicon Stone' and a small blue icon. Below this is a header that says 'Register (* Mandatory fields)'. The main section is titled '基本信息' (Basic Information). It contains four input fields: '国籍' (Nationality) with a dropdown menu showing 'Afghanistan', '名字' (First name) with the placeholder 'Enter your First name', '中间名字' (Middle name) with the placeholder 'Enter your Middle name', and '姓' (Last name) with the placeholder 'Enter your Last name'.

程序实例ch28_28.py： 这个窗体有许多字段，笔者将以实例说明输入字段实例。注意：由于无法输入中文，所以程序以输入英文取代。

```

1 # ch28_28.py
2 import pyautogui
3 import time
4
5 print("请在10秒内打开记事本并设为焦点窗口")
6 time.sleep(10)
7
8 pyautogui.typewrite('Taiwan\t')      # Taiwan
9 pyautogui.typewrite('Hung\t')        # 姓
10 pyautogui.typewrite('Jiin-Kwei\t')   # 名
11 pyautogui.typewrite('Jiin-Kwei\t')   # 名
12 pyautogui.typewrite('Hung\t')        # 姓
13 pyautogui.typewrite('1975\t')        # 出生年
14 pyautogui.typewrite('01\t')          # 月
15 pyautogui.typewrite('01\t')          # 日
16 pyautogui.typewrite('\t')            # 选男生
17 pyautogui.typewrite('Ming-Chi Inst. of Tech\t') # 学校
18 pyautogui.typewrite('Deartment of ME') # 科系

```

执行结果 首先程序执行时操作系统需在英文输入模式下，然后有10秒钟可以点选国籍字段，请将鼠标光标放在此字段。

基本信息

国籍 *

名字 *

未来程序将可以自动填写下列窗体。

基本信息

国籍 *

名字 *

中间名字

姓 *

您的全名会被打印在证书上: Jiin-Kwei Hung

出生日期 *

性别 *
☒ 男 ☐ 女

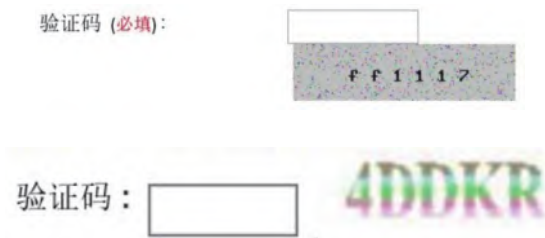
学校 *

部 *

上述程序设计是将给10秒执行启动网页窗体为焦点窗口，将鼠标光标移至第一个字段，再执行输入。另一种方式是将窗口放到最大，先侦测第一个字段的屏幕坐标，再将鼠标光标移至此字段，然后执行输入。上述是

只填写一个表单，我们可以将所要填写的数据以字典列表存储，然后用循环填写。

由于Python可以控制窗体的输入，这也代表黑客可以利用循环不断更改输入账号（记住：笔者在ch18_11.py曾经设计暴力解密码程序），同时更改密码方式攻击网站，特别是金融机构网站，当然一般要求输入账号的网站也很容易受到攻击。所以目前一般网站为了防止黑客利用此特性会要求使用者输入确认码，这样黑客就无法使用程序（我们又称**机器人**）连续进入系统测试账号。下列是2个不同单位所谓的账号确认码画面。



科技的隐忧，现在文字识别功能已经变得很容易，所以上述验证码，其实也可以利用程序处理，所以有些更加严谨的机构在设计确认码时，会将数字设计得很奇怪，虽好处，但是常常真正用户看不清楚数字造成困扰，所以是两难。所以现在金融机构处理个人网络银行登录时，除了用户身份证号还会增加用户名和密码，主要是防黑客攻击。

习题

1. 可参考ch28_10.py，再参考28-2节，增加列出每一个鼠标光标位置的RGB色彩。
2. 可参考ch28_24.py，请用输入‘backspace’方式，重新设计此程序。
3. 请扩增程序实例ch28_28.py，可以填写所有字段，除了验证码。

第29章 文字识别系统

Tesseract OCR是一个文字识别 (OCR , Optical Character Recognition)的系统,可以在多个平台上运作,目前这是一个开放资源的免费软件。1985-1994年间由惠普(HP)实验室开发,1996年开发为适用Windows系统。有接近十年期间,这个软件没有太大进展,在2005年惠普公司将这个软件释为免费使用(open source),2006年起这个软件改由Google赞助与维护。

本章笔者将简单介绍使用Python处理文字识别,在上一章笔者有说明目前有许多网站在进入前需要输入验证码,这一章将用实例说明,如何处理这些验证码。同时也将说明使用这个系统识别繁体 和 简体 中文图文件。

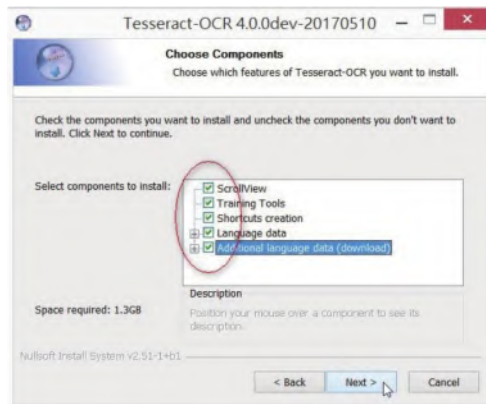
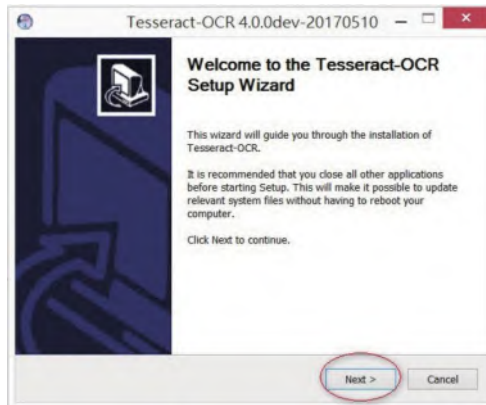
29-1 安装Tesseract OCR

使用这套软件需要下载,请至下列网站。

<http://digi.bib.uni-mannheim.de/tesseract/tesseract-ocr-setup-4.00.00dev.exe>

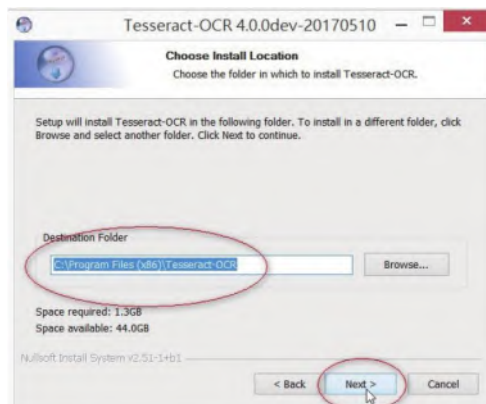
①首先将看到下列左图画面。

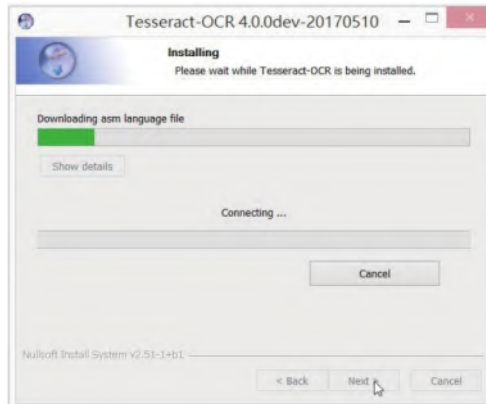
②请按Next 按钮,于第4个画面你将看到下列右图。



③请选择全部，然后按Next 按钮，如下列左图。

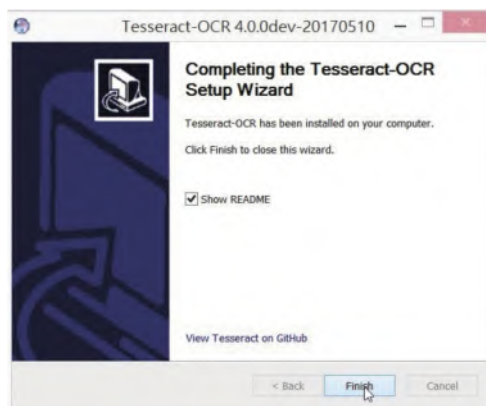
④上述请使用默认目录 安装，请按Next按钮，接着画面可以使用预设，下列右图是安装过程画面。





⑤下列左图是安装结束画面。

⑥安装完成后，下一步是将Tesseract-OCR所在的目录设定在Windows操作系统的path 路径内，这样就不会有找不到文件的问题。首先打开控制面板的系统 设置，如下列右图。



⑦选择高级系统设置 ，在高级 选项卡单击环境变量 按钮，在系统变量 栏点path 选项，会出现编辑系统变量 对话框，请在变量值 字段输入所安装Tesseract安装目录，如果是依照默认模式输入，路径如下：

C:\Program Files (x86)\Tesseract-OCR

上述路径建议用复制方式处理，需留意不同路径的设定彼此以“;”隔开。

⑧完成后，请单击**确定** 按钮。如果想要确定是否安装成功，可以在**命令行** 窗口输入“tesseract -v”，如果列出版本信息，就表示设定成功了。



29-2 安装pytesseract模块

pytesseract是一个Python与Tesseract-OCR之间的接口程序，这个程序的官网就自称是Tesseract-OCR的wrapper，它会自行调用Tesseract-OCR的内部程序执行识别功能，我们调用pytesseract的方法，就可以完成识别工作，可以使用下列方式安装这个模块。

```
pip install pytesseract
```

29-3 文字识别程序设计

安装完Tesseract-OCR后，预设情况下是可以执行英文和阿拉伯数字的识别，下列是笔者采用数字与英文的图片文件执行识别，并将结果印出(ch29_1.py)与印出和存储(ch29_2.py)，在使用pytesseract前，需要导入pytesseract模块。

```
import pytesseract
```

由于这个pytesseract会自行处理和tesseract-OCR的接口，所以程序可以不用导入tesseract模块。这个模块主要是使用image_to_string()方法，执行图像识别，然后将结果传回，如果识别英文或数字 可以不必额外参数，如果识别其他语言，则需加上lang= 'chi_tra' (这是识别繁体中文)参数，chi_tra是繁体中文的参数名称，细节可参考29-4节。

程序实例ch29_1.py： 这个程序会识别图片的文字，同时输出执行结果，下列是内含文字要识别的图片内容。



下列是程序内容。

```
1 # ch29_1.py
2 from PIL import Image
3 import pytesseract
4
5 text = pytesseract.image_to_string(Image.open('d:\\Python\\ch29\\data29_1.jpg'))
6 print(text)
```

执行结果 这个程序无法在Python idle环境执行，下列在命令提示符 模式执行。

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch29\ch29_1.py
4DDKR
C:\Users\Jiin-Kwei>
```


程序实例ch29_2.py: 这个程序会执行识别图片的文字, 除了输出文字, 也会将文字存入out29_2.txt文件内, 下列是内含文字要识别的图片内容。

```
\Users\Jiin-Kwei>echo %path%
```

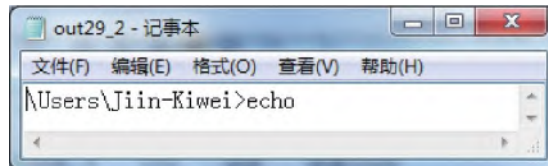
下列是程序内容。

```
1 # ch29_2.py
2 from PIL import Image
3 import pytesseract
4
5 text = pytesseract.image_to_string(Image.open('d:\\Python\\ch29\\data29_2.jpg'))
6 print(text)
7 with open('d:\\Python\\ch29\\out29_2.txt', 'w') as fn:
8     fn.write(text)
```

执行结果 下列是程序执行结果。

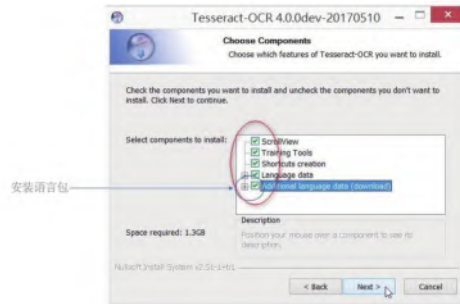
```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch29\ch29_2.py
\Users\Jiin-Kwei>echo
C:\Users\Jiin-Kwei>
微软注音 半 :
```

下图是所建立的out29_2.txt文件内容。



29-4 识别繁体中文

Tesseract-OCR也可以识别繁体中文, 这需要指示程序引用中文数据文件, 这个繁体中文数据文件名称是chi-tra.traineddata, 在29-1节的安装画面中, 笔者指出了需要设定安装语言文件。



如果读者依照上面指示安装，可以在\tesdata文件夹下看到chi_tri.trianeddata文件，下面将以实例ch29_3.py说明识别下列繁体中文的图片文件。

1：从无到有一步一步教导读者R语言的使用

2：学习本书不需要有统计基础，但在无形中

本书已灌输了统计知识给你

程序实例ch29_3.py： 执行繁体中文图片文字的识别，这个程序最重要的是笔者在image_to_string()方法内增加了第2个参数lang=‘chi_tra’ 参数，这个参数会引导程序使用繁体中文数据文件 做识别。

```
1 # ch29_3.py
2 from PIL import Image
3 import pytesseract
4
5 text = pytesseract.image_to_string(Image.open('d:\\Python\\ch29\\data29_3.jpg'),
6                                     lang='chi_tra')
7 print(text)
8 with open('d:\\Python\\ch29\\out29_3.txt', 'w') as fn:
9     fn.write(text)
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch29\ch29_3.py
1：从无到有一步一步教导读者R语言的使用
2：学习本书不需要有统计基础，但在无形中本
书已灌输了统计知识给你
C:\Users\Jiin-Kwei>
```

在上述识别处理中，只有一个字错，这是一个非常好的识别结果。不过笔者在使用时发现，如果图片文件的字比较小，会有较多识

别错误情况。

29-5 识别简体中文

识别简体中文和繁体中文步骤相同，只是导入的是`chi_sim`。`trianeddata`简体中文数据文件，下面将以实例`ch29_4.py`说明识别下列简体中文的图片文件。

1: 从无到有一步一步教导读者 R 语言的使用

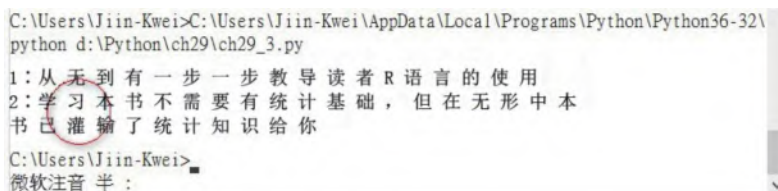
2: 学习本书不需要有统计基础，但在无形中本

书已灌输了统计知识给你

程序实例`ch29_4.py`： 执行简体中文图片文字的识别，这个程序最重要的是笔者在`image_to_string()`方法内增加了第2个参数“`lang='chi_sim'`”参数，这个参数会引导程序使用简体中文数据文件做识别。这个程序另外需留意的是，第8行在打开文件时需要增加`encoding='utf-8'`，才可以将简体中文写入文件。

```
1 # ch29_4.py
2 from PIL import Image
3 import pytesseract
4
5 text = pytesseract.image_to_string(Image.open('d:\\Python\\ch29\\data29_4.jpg'),
6                                     lang='chi_sim')
7 print(text)
8 with open('d:\\Python\\ch29\\out29_4.txt', 'w', encoding='utf-8') as fn:
9     fn.write(text)
```

执行结果



```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\python d:\Python\ch29\ch29_3.py
1: 从无到有一步一步教导读者 R 语言的使用
2: 学习本书不需要有统计基础，但在无形中本
书已灌输了统计知识给你
C:\Users\Jiin-Kwei>
微软注音 半:
```

在使用时，笔者也发现如果发生无法识别情况，程序将响应空白。

习题

1. 请用手机拍下今天报纸的头条新闻，执行识别并输出。
2. 请截取母校网页的首页，执行识别并输出。

第30章 多任务与多线程

如果我们打开计算机可以看到在Windows操作系统下可以同时执行多个应用程序，例如，当你使用浏览器下载数据期间，可以使用Word编辑文件，同时间可能Outlook告诉你收到了一封电子邮件，其实这种作业类型就称多任务作业。

相同的观念可以应用在程序设计，我们可以使用Python设计一个程序执行多个子程序，这个观念称一个程序内有好几个进程（process），然后我们也可以使用Python设计一个进程（process）内含有多个线程（threading）。过去我们使用Python所设计的程序只专注执行一件事情，我们也可以称之为单进程 内有单线程，这一章我们将讲解一个程序可以执行多个工作进程（process）的概念，同时也介绍一个进程内含有多个线程。

另外，这一章也将讲解另一个时间模块datetime和如何从Python启动其他应用程序。

30-1 时间模块datetime

在13-6节笔者讲解了时间模块time，这一节将讲解另一个时间模块datetime，在使用前需导入此模块。

```
import datetime
```

30-1-1 datetime模块的数据类型datetime

datetime模块内有一个数据类型datetime，可以用它代表一个特定时间，有一个now()方法可以列出现在时间。

程序实例ch30_1.py： 列出现在时间。

```

1 # ch30_1.py
2 import datetime
3
4 timeNow = datetime.datetime.now()
5 print(type(timeNow))
6 print("列出现在时间 :", timeNow)

```

执行结果

```

===== RESTART: D:\Python\ch30\ch30_1.py
<class 'datetime.datetime'>
列出现在时间 : 2017-12-19 22:58:53.924934
>>>

```

我们也可以使用属性year、month、day、hour、minute、second、microsecond(百万分之一秒)，获得上述时间的个别内容。

程序实例ch30_2.py: 列出时间的个别内容。

```

1 # ch30_2.py
2 import datetime
3
4 timeNow = datetime.datetime.now()
5 print(type(timeNow))
6 print("列出现在时间 :", timeNow)
7 print("年 :", timeNow.year)
8 print("月 :", timeNow.month)
9 print("日 :", timeNow.day)
10 print("时 :", timeNow.hour)
11 print("分 :", timeNow.minute)
12 print("秒 :", timeNow.second)

```

执行结果

```

===== RESTART: D:\Python\ch30\ch30_2.py
<class 'datetime.datetime'>
列出现在时间 : 2017-12-19 23:01:03.459012
年 : 2017
月 : 12
日 : 19
时 : 23
分 : 1
秒 : 3
>>>

```

另一个属性百万分之一秒microsecond，程序一般比较少用。

30-1-2 设定特定时间

当你了解了获得现在时间的方式后，其实可以用下列方法设定一个特定时间。

```

xtime = datetime.datetime(年, 月, 日, 时, 分, 秒)

```

上述xtime就是一个特定时间。

程序实例ch30_3.py: 设定程序循环执行到2017年11月16日14点54分0秒将停止打印“program is sleeping.”，同时打印“Wake up”。

```
1 # ch30_3.py
2 import datetime
3
4 timeStop = datetime.datetime(2017, 11, 16, 14, 54, 0)
5 while datetime.datetime.now() < timeStop:
6     print("program is sleeping.", end="")
7     print("Wake up")
```

执行结果

```
program is sleeping.program is sleeping.program is sleeping.program is sleeping.
program is sleeping.program is sleeping.program is sleeping.program is sleeping.
program is sleeping.program is sleeping.program is sleeping.program is sleeping.
program is sleeping.program is sleeping.program is sleeping.program is sleeping.
program is sleeping.program is sleeping.program is sleeping.program is sleeping.
program is sleeping.program is sleeping.program is sleeping.program is sleeping.
program is sleeping.program is sleeping.program is sleeping.program is sleeping.
program is sleeping.Wake up
>>> |
```

30-1-3 一段时间timedelta

这是datetime的数据类型，代表的是一段的时间，可以用下列方式指定一段时间。

```
deltaTime=datetime.timedelta(weeks=xx,days=xx,hours=xx,minutes=xx,seconds=xx)
```

上述xx代表设定的单位数。

一段时间的对象只有3个属性，days代表天数、seconds代表秒数、microseconds代表百万分之一秒。

程序实例ch30_4.py: 打印一段时间的天数、秒数和百万分之几秒。

```
1 # ch30_4.py
2 import datetime
3
4 deltaTime = datetime.timedelta(days=3, hours=5, minutes=8, seconds=10)
5 print(deltaTime.days, deltaTime.seconds, deltaTime.microseconds)
```

执行结果

```
===== RESTART: D:/Python/ch30/ch30_4.py =====
3 18490 0
>>>
```

上述5小时8分10秒被总计为18940秒。有一个方法total_second()可以将一段时间转成秒数。

程序实例ch30_5.py: 重新设计ch30_4.py, 将一段时间转成秒数。

```
1 # ch30_5.py
2 import datetime
3
4 deltaTime = datetime.timedelta(days=3, hours=5, minutes=8, seconds=10)
5 print(deltaTime.total_seconds())
```

执行结果

```
===== RESTART: D:/Python/ch30/ch30_5.py =====
277690.0
>>>
```

30-1-4 日期与一段时间相加的应用

Python允许时间相加, 例如, 想要知道过了n天之后的日期, 可以使用这个应用。

程序实例ch30_6.py: 列出过了100天后的日期。

```
1 # ch30_6.py
2 import datetime
3
4 deltaTime = datetime.timedelta(days=100)
5 timeNow = datetime.datetime.now()
6 print("现在是 : ", timeNow)
7 print("100天后是 : ", timeNow + deltaTime)
```

执行结果

```
===== RESTART: D:\Python\ch30\ch30_6.py =====
现在是 : 2017-12-19 23:06:07.989064
100天后是 : 2018-03-29 23:06:07.989064
>>>
```

当然利用上述方法也可以推算100天前是几月几号。

30-1-5 将datetime对象转成字符串

strftime()方法可以将datetime对象转成字符串, 这个指令的参数定义如下:

strftime() 参数	意义
%Y	含世纪的年份，例如：‘2020’
%y	不含世纪的年份，例如：‘20’ 代表 2020
%B	用完整英文代表月份，例如：‘January’ 代表 1 月
%b	用缩写英文代表月份，例如：‘Jan’ 代表 1 月
%m	用数字代表月份，‘01’ - ‘12’
%j	该年的第几天，‘001’ - ‘366’
%d	该月的第几天，‘01’ - ‘31’
%A	用完整英文代表星期几，例如：‘Sunday’ 代表星期日
%a	用缩写英文代表星期几，例如：‘Sun’ 代表星期日
%w	用数字代表星期几，‘0’ 星期日 - ‘6’ 星期六
%H	24 小时制，‘00’ - ‘23’
%I	12 小时制，‘01’ - ‘12’
%M	分，‘00’ - ‘59’
%S	秒，‘00’ - ‘59’
%p	‘AM’ 或 ‘PM’

程序实例ch30_7.py： 将现在日期转成字符串格式，同时用不同格式显示。

```

1 # ch30_7.py
2 import datetime
3
4 timeNow = datetime.datetime.now()
5 print(timeNow.strftime("%Y/%m/%d %H:%M:%S"))
6 print(timeNow.strftime("%y-%b-%d %H-%M-%S"))

```

执行结果

```

===== RESTART: D:/Python/ch30/ch30_7.py =====
2017/11/16 16:22:18
17-Nov-16 16-22-18
>>>

```

有关字符串转成日期观念可以参考23-7-6小节。

30-2 多线程

在商业化的应用设计时，通常会为一个程序设计多个线程，大都不会让一个线程占据系统所有资源，例如，Word设计时，有一个线程是处理编辑窗口随时监听是否有屏幕输入可实时编排版面，同一时间也有Word的线

程在做编辑字数统计随时更新Word的窗口状态栏。这一节将讲解这方面的设计观念。

30-2-1 一个睡眠程序设计

在讲解多线程前，我们可以先看下列程序实例。

程序实例ch30_8.py： 假设现在是2020年1月1日，你太在乎女朋友，想要程序在女朋友生日1月20日当天提醒自己送礼物，可能你的程序可以这样设计。

```
1 # ch30_8.py
2 import datetime
3
4 timeStop = datetime.datetime(2020, 1, 1, 8, 0, 0)
5 while datetime.datetime.now() < timeStop:
6     pass
7 print("女朋友生日")
```

执行结果 这个程序要到2020年1月1日早上8点才会苏醒，可以用Ctrl-C键中断执行。

30-2-2 建立一个简单的多线程

为了解决程序被霸占资源无法执行的后果，我们可以使用多线程的观念，例如，给上述调用循环一个线程，然后我们程序可以作为主线程继续执行应有的工作。建立线程需要导入threading模块，如下所示：

```
import threading
```

我们可以使用下列方式导入线程：

```
def threadWork():          # 用函数定义线程的工作内容
    xxx                     # 这个线程的工作内容
threadObj = threading.Thread(target=threadWork)  # 建立线程对象
threadObj.start()           # 启动线程
```

从上述我们可以发现要建立与执行一个线程，需要threading模块的Thread()方法定义一个Thread对象，同时又需设定此Thread对象所

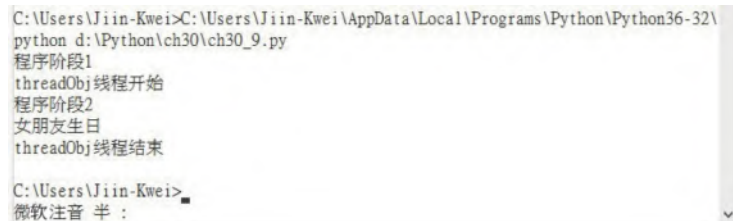
要执行的工作，用函数设定工作内容。此处threadObj是一个对象名称，读者可以自己取任意名称，同时这个方法内需用关键词target设定所要调用的函数，此处threadWork 是函数名称，读者可以自己取这个名称。所以这一行定义了线程 的对象名称 和所要执行的工作 。

要启动线程则需使用start()方法。

程序实例ch30_9.py： 设计一个线程单独执行工作，程序本身也执行工作。

```
1 # ch30_9.py
2 import threading, time
3
4 def wakeUp():
5     print("threadObj线程开始")
6     time.sleep(10)          # threadObj线程休息10秒
7     print("女朋友生日")
8     print("threadObj线程结束")
9
10 print("程序阶段1")
11 threadObj = threading.Thread(target=wakeUp)
12 threadObj.start()          # threadObj线程开始工作
13 time.sleep(1)              # 主线程休息1秒
14 print("程序阶段2")
```

执行结果



```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\python d:\Python\ch30\ch30_9.py
程序阶段1
threadObj线程开始
程序阶段2
女朋友生日
threadObj线程结束
C:\Users\Jiin-Kwei>
```

其实在测试多线程工作时，通常会在命令提示符模式下执行，这也是未来本书使用方式。

30-2-3 参数的传送

从ch30_9.py可以看到在Thread()调用函数时，只是填上函数的名称，如果函数需要有传递参数时应如何设计传递参数的方法呢？此时可以增加Thread()的参数，如下所示：

```
threadObj = threading.Thread(target=函数名称, args=['xx', ..., 'yy'])
```

程序实例ch30_10.py: 线程调用函数传递参数的应用。

```
1 # ch30_10.py
2 import threading, time
3
4 def wakeUp(name, blessingWord):
5     print("threadObj线程开始")
6     time.sleep(10) # threadObj线程休息10秒
7     print(name, " ", blessingWord)
8     print("threadObj线程结束")
9
10 print("程序阶段1")
11 threadObj = threading.Thread(target=wakeUp, args=('NaNa', '生日快乐'))
12 threadObj.start() # threadObj线程开始工作
13 time.sleep(1) # 主线程休息1秒
14 print("程序阶段2")
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\python d:\Python\ch30\ch30_10.py
程序阶段1
threadObj线程开始
程序阶段2
NaNa 生日快乐
threadObj线程结束

C:\Users\Jiin-Kwei>
```

设计多线程 程序最重要的观念是，各线程 间不要使用相同的变量，每个线程 最好使用本身的局部变量，这可以避免变量值互相干扰。

30-2-4 线程的命名与取得

每一个线程在产生的时候，如果我们没有给它命名，为了方便日后的管理，Python会自动给这个线程预设名称Thread-n，n是序列号，由1开始编号。可以使用currentThread().getName()获得线程的名称。

程序实例ch30_11.py: 建立线程同时列出线程的名称。

```
1 # ch30_11.py
2 import threading
3 import time
4
5 def worker():
6     print(threading.currentThread().getName(), 'Starting')
7     time.sleep(2)
8     print(threading.currentThread().getName(), 'Exiting')
9
10 def manager():
11     print(threading.currentThread().getName(), 'Starting')
12     time.sleep(3)
13     print(threading.currentThread().getName(), 'Exiting')
14
15 m = threading.Thread(target=manager)
16 w = threading.Thread(target=worker)
17 m.start()
18 w.start()
```

执行结果

```

C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_11.py
Thread-1 Starting
Thread-2 Starting
Thread-2 Exiting
Thread-1 Exiting
C:\Users\Jiin-Kwei>
微软注音 半：

```

当然我们也可以在使用Thread()建立线程时，在参数字段用name=“名称”，直接输入线程的名称，这相当于为线程命名。

程序实例ch30_12.py： 扩充设计ch30_11.py自行为线程命名，读者可以留意第16行为线程的命名方式。

```

1 # ch30_12.py
2 import threading
3 import time
4
5 def worker():
6     print(threading.currentThread().getName(), 'Starting')
7     time.sleep(2)
8     print(threading.currentThread().getName(), 'Exiting')
9 def manager():
10    print(threading.currentThread().getName(), 'Starting')
11    time.sleep(3)
12    print(threading.currentThread().getName(), 'Exiting')
13
14 m = threading.Thread(target=manager)
15 w = threading.Thread(target=worker)
16 w2 = threading.Thread(name='Manager',target=worker)
17 m.start()
18 w.start()
19 w2.start()

```

执行结果

```

C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_12.py
Thread-1 Starting
Thread-2 Starting
Manager Starting
Thread-2 Exiting
Manager Exiting
Thread-1 Exiting
C:\Users\Jiin-Kwei>
微软注音 半：

```

另外也可以使用currentThread().setName()为线程命名。

30-2-5 Daemon线程

在预设情况下，所有的线程皆不是Daemon线程（可以翻译为守护线程）。在默认情况下，如果一个程序建立了主线程与其他子线程，在所有线程工作结束后，程序才会结束。因为如果主线程若是先结束，将退回所有所占据的资源给操作系统，如果子线程仍在执行将会因没有资源造成程序崩溃。

但是当我们设计一个线程是Daemon线程时，主线程若是想要结束执行会检查Daemon线程的属性。

(1) 如果此时Daemon线程的属性是True，即使Daemon线程仍在执行，其他非Daemon线程执行结束，程序将不等待Daemon线程，也会自行结束同时终止此Daemon线程工作。

(2) 如果此时Daemon线程的属性是False，主线程会等待Daemon线程结束，再结束工作。

程序实例30_13.py： 这个程序在执行时，将不等待daemon线程结束，而自行结束工作，由于程序已经结束，所以我们看不到第8行 daemon Exiting的输出。

```
1 # ch30_13.py
2 import threading
3 import time
4
5 def daemonFun():
6     print(threading.currentThread().getName(), 'Starting')
7     time.sleep(5)
8     print(threading.currentThread().getName(), 'Exiting')
9
10 def non_daemon():
11     print(threading.currentThread().getName(), 'Starting')
12     print(threading.currentThread().getName(), 'Exiting')
13
14 d = threading.Thread(name='daemon', target=daemonFun)
15 d.setDaemon(True)
16 nd = threading.Thread(name='non-daemon', target=non_daemon)
17
18 d.start()
19 nd.start()
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\python d:\Python\ch30\ch30_13.py
daemon Starting
non-daemon Starting
non-daemon Exiting

C:\Users\Jiin-Kwei>
微软注音 半：
```

程序实例ch30_14.py： 重新设计ch30_13.py，但是将Daemon线程的属性设为False，在观察执行结果时可以发现主线程等待Daemon线程结束后才结束工作。

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\python d:\Python\ch30\ch30_14.py
daemon Starting
non-daemon Starting
non-daemon Exiting
daemon Exiting
C:\Users\Jiin-Kwei>
```

30-2-6 堵塞主线程join()

主线程在工作时，如果想要安插一个子线程进来，可以使用join()，这时安插进来的子线程可以先工作，直到所邀请的子线程结束，主线程才开始工作。

程序实例ch30_15.py： 这个程序执行时会因为worker线程的加入（第13行），主线程会等待此worker线程工作结束，再开始往下工作。

```
1 # ch30_15.py
2 import threading
3 import time
4
5 def worker():
6     print(threading.currentThread().getName(), 'Starting')
7     time.sleep(3)
8     print(threading.currentThread().getName(), 'Exiting')
9
10 w = threading.Thread(name='worker', target=worker)
11 w.start()
12 print('start join')
13 w.join()          # worker线程工作完成才往下执行
14 print('end join')
```

执行结果

```

C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_15.py
worker Starting
start join
worker Exiting
end join

C:\Users\Jiin-Kwei>
微软注音 半：

```

为了怕等待所安插进来的子线程工作太久，可以在`join()`内增加秒数的实数参数，代表所等待的时间，当时间到时主线程恢复工作，这时所安插进来的子线程仍是继续工作。

程序实例ch30_16.py：重新设计ch30_15.py，设计等待时间是1.5秒，当等待时间超过1.5秒后，主线程将恢复工作，所以在执行结果可以看到会先打印end join字符串，然后worker Exiting才被打印。

```

13 w.join(1.5)           # 等待worker线程1.5秒工作完成才往下执行

```

执行结果

```

C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_16.py
worker Starting
start join
end join
worker Exiting

C:\Users\Jiin-Kwei>
微软注音 半：

```

30-2-7 检查子线程是否仍在工作isAlive()

通常在使用`join(time unit)`方法，同时设定等待一段时间后程序设计师会在`join()`后面加上`isAlive()`方法，检查子线程是否工作结束了，如果是则传回False或因为时间到交出执行的权利给主线程，表示仍在工作此时会传会True。

程序实例ch30_17.py：扩充设计ch30_16.py，列出主线程取得控制权时，子线程是否仍在工作，读者应注意第14和17行。


```

1 # ch30_17.py
2 import threading
3 import time
4
5 def worker():
6     print(threading.currentThread().getName(), 'Starting')
7     time.sleep(3)
8     print(threading.currentThread().getName(), 'Exiting')
9
10 w = threading.Thread(name='worker',target=worker)
11 w.start()
12 print('start join')
13 w.join(1.5)      # 等待worker线程1.5秒工作完成才往下执行
14 print("是否working线程仍在工作？", w.isAlive())
15 time.sleep(2)    # 主线程休息2秒
16 print("是否working线程仍在工作？", w.isAlive())
17 print('end join')

```

执行结果

```

C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_17.py
worker Starting
start join
是否working线程仍在工作？ True
worker Exiting
是否working线程仍在工作？ False
end join
C:\Users\Jiin-Kwei>
微软注音 半：

```

30-2-8 了解正在工作的线程

下列是与正在工作线程相关的方法。

方法	说明
<code>threading.active_count()</code>	在工作中的线程的数量
<code>threading.enumerate()</code>	可迭代列出所有工作中的线程
<code>threading.current_thread()</code>	实时在执行的线程

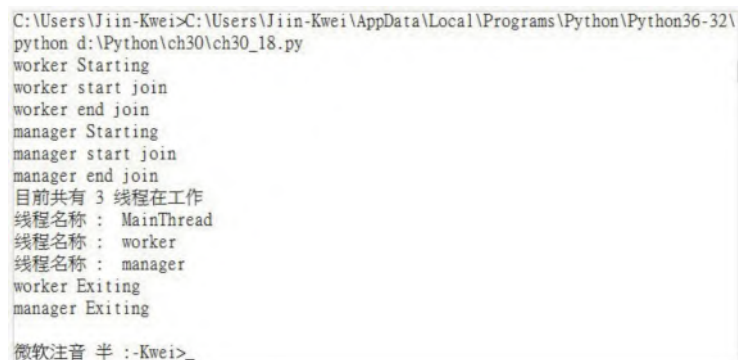
程序实例ch30_18.py： 列出在工作中的线程数量和这些线程名称。

```

1 # ch30_18.py
2 import threading
3 import time
4
5 def worker():
6     print(threading.currentThread().getName(), 'Starting')
7     time.sleep(5)
8     print(threading.currentThread().getName(), 'Exiting')
9 def manager():
10    print(threading.currentThread().getName(), 'Starting')
11    time.sleep(5)
12    print(threading.currentThread().getName(), 'Exiting')
13
14 w = threading.Thread(name='worker',target=worker)
15 w.start()
16 print('worker start join')
17 w.join(1.5) # 等待worker线程1.5秒工作完成才往下执行
18 print('worker end join')
19 m = threading.Thread(name='manager',target=worker)
20 m.start()
21 print('manager start join')
22 m.join(1.5) # 等待manager线程1.5秒工作完成才往下执行
23 print('manager end join')
24 print("目前共有 %d 线程在工作" % threading.active_count())
25 for thread in threading.enumerate():
26     print("线程名称 : ", thread.name)

```

执行结果



```

C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\python d:\Python\ch30\ch30_18.py
worker Starting
worker start join
worker end join
manager Starting
manager start join
manager end join
目前共有 3 线程在工作
线程名称 : MainThread
线程名称 : worker
线程名称 : manager
worker Exiting
manager Exiting
微软注音 半 :-Kwei>

```

30-2-9 自行定义线程和run()方法

其实threading.Thread是threading模块内的一个类别，我们可以自行设计一个类别，让这个类别继承threading.Thread类别，接着需在def _init_()内调用threading.Thread._init_()方法，然后在所设计的类别内可以设计run()方法，这个观念就称自行定义线程。假设所设计的类别是MyThread ，未来只要声明所设计类别的对象，如下所示：

```
obj = MyThread
```

() # 建立自行定义线程对象

然后执行run()方法，就可以启动自行定义的线程。

```
obj.run( )    # 启动自行定义的线程
```

过去几节我们使用threading.Thread()声明一个线程对象时，再执行start()可以建立一个线程，其实start()就是辗转调用此threading.Thread类别的run()方法开始执行工作。不过这种方式线程只能调用一次start()方法，重复调用时会有错误。我们使用自定义的线程时，可以调用run()方法多次，不会引发错误。

程序实例ch30_19.py： 测试自行定义的线程a，启动run()，2次，结果可以正常执行。测试自行定义的线程b，启动start()，1次，可以正常。

```
1  # ch30_19.py
2  import threading
3
4  class MyThread(threading.Thread):      # 这是threading.Thread的子类别
5      def __init__(self):
6          threading.Thread.__init__(self) # 建立线程
7      def run(self):                      # 定义线程的工作
8          print(threading.Thread.getName(self))
9          print("Happy Python")
10
11 a = MyThread()                         # 建立线程对象a
12 a.run()                               # 启动线程a
13 a.run()                               # 启动线程a
14 b = MyThread()                         # 建立线程对象b
15 b.start()                             # 启动线程b
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_19.py
Thread-1
Happy Python
Thread-1
Happy Python
Thread-2
Happy Python
C:\Users\Jiin-Kwei>
```

程序实例ch30_20.py： 如果我们在第16行再增加一个b.start()，就会产生错误。

```
15 b.start()          # 启动线程b
16 b.start()          # 启动线程b
```

执行结果

```

C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_20.py
Thread-1
Happy Python
Thread-1
Happy Python
Thread-2
Happy Python
Traceback (most recent call last):
  File "d:\Python\ch30\ch30_20.py", line 16, in <module>
    b.start()
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\threadi
ng.py", line 842, in start
    raise RuntimeError("threads can only be started once")
RuntimeError: threads can only be started once

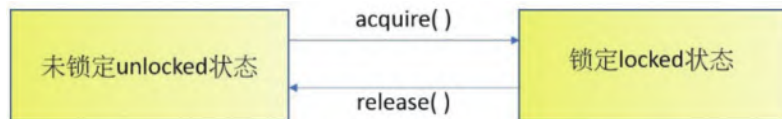
C:\Users\Jiin-Kwei>
微软注音 半：

```

错误原因是start()只能被启动一次。

30-2-10 资源锁定与解锁Threading.Lock

在多线程的程序设计中，可能会有多个线程皆要存取相同的资源，为了确保线程在处理资源期间，可以完成处理不被干扰，此时可以使用Python的锁定功能Threading.Lock，这个锁定功能有锁定 与未锁定 2种状态。在未锁定状态可以使用acquire()方法进入锁定状态，此时所锁定的资源别的线程无法存取，当处理资源完成，可以使用release()方法，将锁定状态改为未锁定状态。



程序实例ch30_21.py： 这个程序会对全局变量进行存取，为了保护顺序处理原则，存取前先锁定全局变量，处理完成后再解锁。

```

1 # ch30_21.py
2 import threading
3
4 class MyThread(threading.Thread):          # 这是threading.Thread的子类别
5     def __init__(self):
6         threading.Thread.__init__(self) # 建立线程
7     def run(self):
8         global data                      # 定义全局数据
9         datalock.acquire()              # 锁定
10        data += 5
11        print("data = ", data)
12        datalock.release()               # 解锁
13
14 data = 10                                # 全局最初值
15 datalock = threading.Lock()              # 建立对象
16 ts = []                                  # 建立线程列表
17 for t in range(10):
18     t = MyThread()
19     ts.append(t)                          # 加入线程列表
20
21 for t in ts:                              # 启动所有线程
22     t.start()
23
24 for t in ts:                              # 等待所有线程退出
25     t.join()

```

执行结果 这个程序在Python Shell窗口更可以看出差异。

```
===== RESTART: D:\Python\ch30\ch30_21.py =====
data = 15
data = 20
data = 25
data = 30
data = 35
data = 40
data = 45
data = 50
data = 55
data = 60
>>>
```

从上图可以看到数据符合预期，依序列出。下列实例是，我们不对数据进行锁定，各线程无法预期谁会先取得资源然后进行数据处理，这种现象称为竞速（race condition）。

程序实例ch30_22.py: 重新设计ch30_21.py, 但是取消第9和12行。

```
9  #      datalock.acquire()          # 锁定
10      data += 5
11      print("data = ", data)
12  #      datalock.release()         # 解锁
```

执行结果 每次执行都获得不一样的结果。

[illegible]

```
===== RESTART: D:/Python/ch30/ch30_22.py =====
data = data = data = data = data = data = data = data =
15204525503055604035
```

30-2-11 产生锁死

在使用Threading.Lock时，如果目前是锁定状态(locked)，再执行一次acquire()这样会产生锁死(dead lock)，造成程序错误。

程序实例ch30_23.py： 程序产生锁死(dead lock)测试，笔者使用15-7节的logging模块做追踪。

```
1 # ch30_23.py
2 import threading, logging
3 logging.basicConfig(level=logging.DEBUG)
4 datalock = threading.Lock() # Lock对象
5 datalock.acquire() # 进入锁定
6 logging.debug('Enter locked mode')
7 datalock.acquire() # 进入锁死程序
8 logging.debug('Trying to locked again')
9 datalock.release()
10 datalock.release()
```

执行结果 由于锁死产生，所以无法显示Trying to lock again字符串。

```
===== RESTART: D:/Python/ch30/ch30_23.py =====
DEBUG:root:Enter locked mode
```

30-2-12 资源锁定与解锁Threading.RLock

这是另一种资源锁定与解锁，在相同线程下这种锁允许在锁定状态时，再度执行一次acquire()，差异是acquire()和release()需要成对出现，如果使用n次acquire()，就必须使用n次release()解锁。

程序实例ch30_24.py： 使用Threading.Rlock重新设计ch30_23.py，程序不会产生锁死(dead lock)测试。

```
1 # ch30_24.py
2 import threading, logging
3 logging.basicConfig(level=logging.DEBUG)
4 datalock = threading.RLock() # RLock对象
5 datalock.acquire() # 进入锁定
6 logging.debug('Enter locked mode')
7 datalock.acquire() # 不会进入锁死
8 logging.debug('Trying to locked again')
9 datalock.release()
10 datalock.release()
```


执行结果

```
===== RESTART: D:/Python/ch30/ch30_24.py =====  
DEBUG:root:Enter locked mode  
DEBUG:root:Trying to locked again  
>>>
```

30-2-13 高级锁定threading.Condition

这是Python的另一种锁定，就像它的名称一样是可以有条件的(condition)。首先程序使用acquire()进入锁定状态，如果需要符合一定的条件才处理数据，此时可以调用wait()，让自己进入睡眠状态。程序设计时需用notify()通知其他线程，然后放弃锁定release()。

此时其他在等待的线程因为收到通知notify()，这时被激活了，就可以开始运作。

程序实例ch30_25.py： 生产者和消费者的设计，这个程序用producer()方法叙述生产者运作方式，基本上是需要生产5个数据（在data列表）才让自己进入睡眠状态，然后通知其他线程（第14行），再解锁（第15行）。consumer()方法则是当data列表没有数据时，才让自己进入睡眠状态，然后通知其他线程（第27行），再解锁（第28行）。这个程序首先建立threading.Condition()（第30行），然后设定资源列表data是空的（第31行），程序接着是建立线程与启动线程。由于producer()和consumer()方法皆是一个无限循环（第5~15行，第18~28行）所以程序将持续进行。

```

1  # ch30_25.py
2  import threading, time, random
3
4  def producer():
5      while True:
6          condition.acquire()
7          if len(data) >= 5:
8              print("生产线是 waiting ...")
9              condition.wait()
10         else:
11             data.append(random.randint(1, 100))
12             print("生产线库存", data)
13             time.sleep(1)
14             condition.notify()
15             condition.release()
16
17 def consumer():
18     while True:
19         condition.acquire()
20         if not data:
21             print("消费者是 waiting ...")
22             condition.wait()
23         else:
24             print("消费者取走商品 :", data.pop(0))
25             print("目前库存", data)
26             time.sleep(1)
27             condition.notify()
28             condition.release()
29
30 condition = threading.Condition()
31 data = []
32
33 p = threading.Thread(name='producer', target=producer)
34 c = threading.Thread(name='consumer', target=consumer)
35
36 p.start()
37 c.start()
38 p.join()
39 c.join()

```

执行结果 下列是部分执行过程。

```

===== RESTART: D:\Python\ch30\ch30_25.py =====
生产线库存 [97]
消费者取走商品 : 97
目前库存 []
生产线库存 [62]
消费者取走商品 : 62
目前库存 []
消费者是 waiting ...
生产线库存 [54]
生产线库存 [54, 62]
消费者取走商品 : 54
目前库存 [62]
生产线库存 [62, 66]
生产线库存 [62, 66, 20]
生产线库存 [62, 66, 20, 6]
生产线库存 [62, 66, 20, 6, 97]
消费者取走商品 : 62
目前库存 [66, 20, 6, 97]
消费者取走商品 : 66
目前库存 [20, 6, 97]

```

在程序设计中也可以在wait()设定等待秒数的参数，另外，以上述实例而言若是另外增加一个消费者时，则可以在通知时可以使用notifyAll()。

30-2-14 queue

在Python内有一个queue模块，这是一种先进先出的数据结构，可以使用put()方法插入元素，使用get()方法取得元素，元素取得后此元素将在queue内被移除，它的基本观念图如下：



由于在设计queue的逻辑上，要在queue中使用put()插入元素时，系统处理锁定逻辑，另外，如果queue空间已满，put()会在内部调用wait()进行等待。使用get()取得元素和移除元素时，系统内部也会进行锁定。如果queue空间是空的，get()会在内部调用wait()进行等待。

基于以上特性，一般也可以使用queue处理生产者(producer)和消费者(consumer)的问题。另外，使用queue时，需要导入queue。

程序实例ch30_26.py： 使用queue观念，应用到生产者和消费者的问题。这个程序在执行时，首先定义queue最大空间是10(第4~5行)，第7~13行是producer线程设计，只要queue空间尚未满，就会生产数据然后存入queue(第10~11行)。第15~20行是consumer线程设计，只要queue空间不是空的，就会读取和移除数据（第18行）。

```
1 # ch30_26.py
2 import threading, time, random, queue
3
4 bufSize = 10
5 q = queue.Queue(bufSize)                                # 建立queue,最多10
6
7 def producer():                                         # 生产者状况
8     while True:
9         if not q.full():                                # 如果queue有空间
10             item = random.randint(1,100)               # 生产产品
11             q.put(item)                                  # 将产品放入库存
12             print('生产者Putting存入 %2s : queue数量 %s' % (str(item), str(q.qsize())))
13             time.sleep(2)                                # 休息2秒
14
15 def consumer():                                         # 消费者状况
16     while True:
17         if not q.empty():                                # 如果queue不是空的
18             item = q.get()                                # 消费产品
19             print('消费者Getting取得 %2s : queue数量 %s' % (str(item), str(q.qsize())))
20             time.sleep(2)                                # 休息2秒
21
22 p = threading.Thread(name='producer',target=producer)   # 建立producer线程
23 c = threading.Thread(name='consumer',target=consumer)   # 建立consumer线程
24 p.start()
25 time.sleep(2)
26 c.start()
27 time.sleep(2)
```

执行结果 这是一个无限循环的设计。

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\python d:\Python\ch30\ch30_26.py
生产者Putting存入 98 : queue数量 1
消费者Getting取得 98 : queue数量 0
生产者Putting存入 78 : queue数量 1
消费者Getting取得 78 : queue数量 0
生产者Putting存入 65 : queue数量 1
消费者Getting取得 65 : queue数量 0
生产者Putting存入 36 : queue数量 1
消费者Getting取得 36 : queue数量 0
生产者Putting存入 4 : queue数量 1
消费者Getting取得 4 : queue数量 0
生产者Putting存入 8 : queue数量 1
消费者Getting取得 8 : queue数量 0
生产者Putting存入 30 : queue数量 1
消费者Getting取得 30 : queue数量 0
生产者Putting存入 49 : queue数量 1
消费者Getting取得 49 : queue数量 0
生产者Putting存入 83 : queue数量 1
```

30-2-15 Semaphore

Semaphore可以翻译为信号量，这个信号量代表最多允许线程访问的数量，可以使用Semaphore(n)设定，n是信号数量。这是一个更高级的锁机制，Semaphore管理一个计数器，每次使用acquire()计数器将减1，表示可允许线程访问的数量少了一个。使用release()计数器将加1，表示可允许线程访问的数量增加了一个。只有占用信号量的线程数量超过信号量时，才会阻塞，也就是说计数器为0时，若还有线程要访问，则发生阻塞。

发生阻塞后就需要等待其他线程使用release()，这时计数器会加1，然后被阻塞的线程就可以访问了。

在应用Semaphore过程，有时候可能会因为bug造成调用多次release()，因此有所谓的BoundedSemaphore，可以保证计数器次数不超过特定值，这时使用BoundedSemaphore(n)设定，n是信号数量。

程序实例ch30_27.py：这个程序在建立semaphore时就设定了最大计数值是3，程序第8~13行记录了计数值响应线程取得资源的情形。

```

1 # ch30_27.py
2 import time
3 import threading
4
5 semaphore = threading.BoundedSemaphore(3)          # 限制计数器最大值
6
7 def func():
8     if semaphore.acquire():                        # 如果取得锁
9         print (threading.currentThread().getName() + ' 取得锁')
10        print("Working ...")
11        time.sleep(2)
12        semaphore.release()
13        print (threading.currentThread().getName() + ' 释出锁')
14
15 for i in range(5):
16     t = threading.Thread(target=func)
17     t.start()

```

执行结果

```

C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_27.py
Thread-1 取得锁
Thread-2 取得锁
Working ...
Thread-3 取得锁
Working ...
Working ...
Thread-3 释出锁
Thread-4 取得锁
Thread-2 释出锁
Thread-1 释出锁
Thread-5 取得锁
Working ...
Working ...
Thread-5 释出锁
Thread-4 释出锁
C:\Users\Jiin-Kwei>
微软注音 半：

```

30-2-16 Barrier

Barrier可以翻译为**栅栏**，可以想成赛马的栅栏，当线程抵达时需等待其他线程，当所有线程抵达时，才放开栅栏，这些线程才可以往下执行。

程序实例ch30_28.py： 这个程序第11行会使用Barrier()将等待线程数量设为4，这时会建立Barrier对象b，然后可以使用b.wait()执行等待。

```

1 # ch30_28.py
2 import random, time
3 import threading
4
5 def player():
6     name = threading.current_thread().getName()
7     time.sleep(random.randint(2,5))
8     print('%s 抵达栅栏时间 : %s' % (name, time.ctime()))
9     b.wait()
10
11 b = threading.Barrier(4)                # 等待的线程数量
12 print('比赛开始 ...')
13 for i in range(4):
14     t = threading.Thread(target=player)
15     t.start()
16 for i in range(4):                      # 等待线程结束
17     t.join()
18 print('比赛结束!')

```

执行结果

```

C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_28.py
比赛开始 ...
Thread-2 抵达栅栏时间 : Wed Dec 20 02:25:23 2017
Thread-4 抵达栅栏时间 : Wed Dec 20 02:25:24 2017
Thread-1 抵达栅栏时间 : Wed Dec 20 02:25:24 2017
Thread-3 抵达栅栏时间 : Wed Dec 20 02:25:25 2017
比赛结束!

C:\Users\Jiin-Kwei>
微软注音 半 :

```

30-2-17 Event

这是一种线程的通信技术，通常会有2个线程，一个线程主要是设定Event的flag，可以使用set()设定flag。另一个线程则是等待Event的flag，可以使wait()等待，当接收到flag信号工作完成后，可以使用clear()清除flag信号。操作上用Event()建立Event对象。

程序实例ch30_29.py： 分别建立w线程(waiter)和s线程(setter)，w线程会等待s线程将flag打开（第14行），打开后第7行w线程的等待就结束，第8行列出等待完成时间，第9行将flag重置，所以下一个循环新的w线程又会进入等待状态。

```

1 # ch30_29.py
2 import random, time
3 import threading
4
5 def waiter(event, loop):
6     for i in range(loop):
7         print('%s. 等待flag被设定' % (i+1))
8         event.wait() # 等待flag
9         print('等待完成时间：', time.ctime())
10        event.clear() # 重置flag.
11        print()
12
13 def setter(event, loop):
14     for i in range(loop):
15         time.sleep(random.randint(2, 5)) # 休息一段时间再工作
16         event.set() # 设定flag
17
18 event = threading.Event() # 建立Event对象
19 loop = random.randint(3, 6) # 循环次数
20
21 w = threading.Thread(target=waiter, args=(event, loop))
22 w.start()
23 s = threading.Thread(target=setter, args=(event, loop))
24 s.start()
25 w.join()
26 s.join()
27 print('工作完成!')

```

执行结果

```

C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_29.py
1. 等待flag被设定
等待完成时间： Wed Dec 20 02:30:32 2017

2. 等待flag被设定
等待完成时间： Wed Dec 20 02:30:36 2017

3. 等待flag被设定
等待完成时间： Wed Dec 20 02:30:39 2017

4. 等待flag被设定
等待完成时间： Wed Dec 20 02:30:42 2017

5. 等待flag被设定
等待完成时间： Wed Dec 20 02:30:45 2017

6. 等待flag被设定
等待完成时间： Wed Dec 20 02:30:49 2017

工作完成!

C:\Users\Jiin-Kwei>
微软注册 半：

```

30-3 启动其他应用程序subprocess模块

subprocess是Python的内置模块，主要是可以在程序内建立子进程，使用前需导入此模块。

```
import subprocess
```

30-3-1 Popen()

Popen()方法可以打开计算机内其他应用程序，有的是Windows系统内置的应用程序或是自己开发的应用程序。当我们所设计的Python程序使用Popen()打开其他应用程序时，我们也可以将所设计的Python程序称是多进程 的应用程序。

当我们安装Windows操作系统后，在C:\Windows\System32文件夹内可以看到许多Windows应用程序，这一节将使用下列3个应用程序为实例说明。

计算器：calc.exe

记事本：notepad.exe

写字板：write.exe

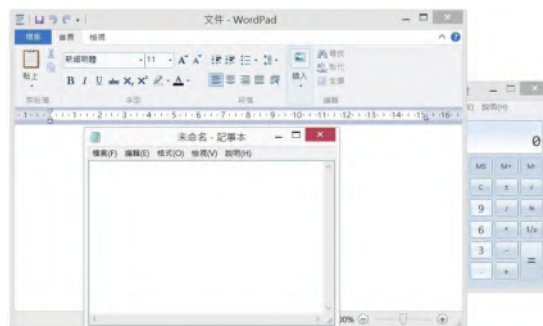
由于C:\Windows\System32在Windows安装时已经主动被设在path路径内，所以我们应用时，直接使用文件名即可。如果打开的是其他应用程序，其路径未被设在path，则需要填上完整的路径名称。

程序实例ch30_30.py： 打开计算器、记事本、写字板 (WordPad) 应用程序，这个程序同时会列出应用程序的数据类型，当打印程序时，可以看到这个程序在内存的位置。

```
1 # ch30_30.py
2 import subprocess
3
4 calcPro = subprocess.Popen('calc.exe') # 返回值是子进程
5 notePro = subprocess.Popen('notepad.exe') # 返回值是子进程
6 writePro = subprocess.Popen('write.exe') # 返回值是子进程
7 print("数据类型 = ", type(calcPro))
8 print("打印calcPro = ", calcPro)
9 print("打印notePro = ", notePro)
10 print("打印writePro = ", writePro)
```

执行结果 下列分别是Python Shell窗口与所打开应用程序的结果。

```
===== RESTART: D:\Python\ch30\ch30_30.py =====
数据类型 = <class 'subprocess.Popen'>
打印calcPro = <subprocess.Popen object at 0x03038B70>
打印notePro = <subprocess.Popen object at 0x0064AC70>
打印writePro = <subprocess.Popen object at 0x02BCEBF0>
>>>
```



其实上述3个应用程序皆是独立的子进程，而主进程则是先执行结束了。

30-3-2 poll()

这个方法会传回子进程是否已经完成工作结束了。如果仍在继续工作会传回None，如果已经执行结束 且正常结束 会传回0，如果已经执行结束 但不正常结束 会传回1。

下列是在执行完ch30_1.py后，立即执行poll()的结果，因为calcPro(计算器)仍在屏幕执行，所以传回None。

```
>>> print(calcPro.poll())
None
>>>
```

如果我们现在关闭计算器应用程序，再执行poll()，可以得到下列结果。

```
>>> print(calcPro.poll())
0
>>>
```

30-3-3 wait()

这个方法会让这个子进程暂停执行，直到启动它的进程结束才开始工作。下列是验证记事本 notePro仍在工作的实例。

```
>>> print(notePro.poll())
None
>>>
```

下列是执行wait()时，整个暂停，你只看见游标在闪烁。

```
>>> print(notePro.wait())
|
```

假设我们现在关闭窗口的记事本应用程序，将看到下列结果。

```
>>> print(notePro.wait())
0
>>>
```

如果子进程 正常结束执行，在我们执行wait()后也将传回0，如上所示。这一节的观念在设计大型多进程时是很有帮助的，因为可以了解各进程的工作状态，也可以控制是否让某个进程暂停工作。

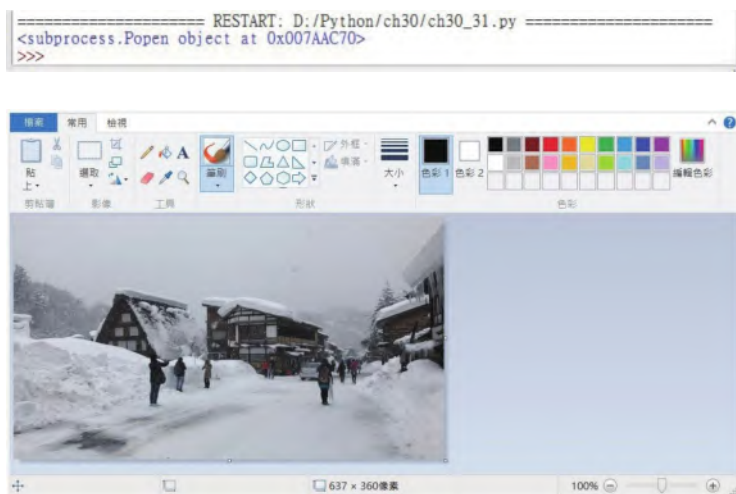
30-3-4 Popen()方法参数的传递

使用Popen()方法时，也可以传递参数，此时会将所传递的参数用列表(list)处理，列表的第一个元素是想要打开的应用程序，第二个元素是这个应用程序相关的文件，下列将以实例解说。

程序实例ch30_31.py： 打开画图mspaint.exe应用程序时，同时打开位于ch30文件夹内的winter.jpg。

```
1 # ch30_31.py
2 import subprocess
3
4 paintPro = subprocess.Popen(['mspaint.exe', 'winter.jpg'])
5 print(paintPro)
```

执行结果 下列分别是Python Shell窗口与所打开应用程序的执行结果。

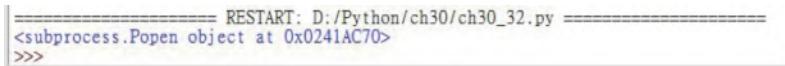


当然在使用时Python程序也可以打开其他Python程序执行工作，这时彼此的变量独立运作行，不会互相干扰也无法共享。

程序实例ch30_32.py： 在程序内启动ch30_30.py，程序执行后计算器、记事本、写字板（WordPad）应用程序将被启动。这个程序在执行时，读者需将第4行改为自己计算机的python.exe的路径。

```
1 # ch30_32.py
2 import subprocess
3
4 path = r'C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\python.exe'
5 pyPro = subprocess.Popen([path, 'ch30_30.py'])
6 print(pyPro)
```

执行结果



```
===== RESTART: D:/Python/ch30/ch30_32.py =====
<subprocess.Popen object at 0x0241AC70>
>>>
```

所打开应用程序的结果可参考ch30_30.py的执行结果。

30-3-5 使用默认应用程序打开文件

当我们在使用Windows操作系统时，若是连按某个文件图标两下，系统会自动打开相关联的应用程序，然后将此文件图示打开。这是因为操作系统已经将常见类型的文件与相关应用程序做关联，在Windows操作系统这个程序是start，在Mac OS操作系统是open。在Windows操作系统下，我们也可以利用这个特性打开文件。

程序实例ch30_33.py： 在Windows操作系统下，使用start 程序打开trip.txt、book.jpg和pegium.m4v文件。

```
1 # ch30_33.py
2 import subprocess
3
4 txtPro = subprocess.Popen(['start', 'trip.txt'], shell=True)
5 pictPro = subprocess.Popen(['start', 'book.jpg'], shell=True)
6 m4vPro = subprocess.Popen(['start', 'pegium.m4v'], shell=True)
7 print("txt文件程序 = ", txtPro)
8 print("pict文件程序 = ", pictPro)
9 print("m4v文件程序 = ", m4vPro)
```

执行结果 下列分别是Python Shell窗口与各应用程序的执行结果。

```

===== RESTART: D:\Python\ch30\ch30_33.py =====
txt文件程序 = <subprocess.Popen object at 0x00FAAC70>
pict文件程序 = <subprocess.Popen object at 0x03778F90>
m4v文件程序 = <subprocess.Popen object at 0x037821D0>
>>>

```



pegium.m4v是笔者一个人到南极所拍摄的影片
可参考
一个人的极境旅行 南极大陆-北极海



记住这个程序执行时，需要在Popen()内增加shell=True参数。

30-3-6 subprocess.run()

从Python 3.5版起，新增可以使用run()调用子进程。

程序实例ch30_34.py： 使用run()调用子进程。

```

1 # ch30_34.py
2 import subprocess
3
4 calcPro = subprocess.run('calc.exe')
5 print("数据类型 = ", type(calcPro))
6 print("打印calcPro = ", calcPro)

```

执行结果 可以启动计算器，关闭计算器时可以看到下列结果。

```

===== RESTART: D:\Python\ch30\ch30_34.py =====
数据类型 = <class 'subprocess.CompletedProcess'>
打印calcPro = CompletedProcess(args='calc.exe', returncode=0)
>>>

```

请读者留意返回值是CompletedProcess数据类型，如果启动的是命令字符模式的指令，需增加参数shell=True，未来这个命令模式指令的返回值会存入CompletedProcess数据类型结构内，如果想要未来获得执行结果，可以增加stdout=subprocess.PIPE参数。

程序实例ch30_35.py： 列出目前系统时间。

```
1 # ch30_35.py
2 import subprocess
3
4 ret = subprocess.run('echo %time%', shell=True, stdout=subprocess.PIPE)
5 print("数据类型 = ", type(ret))
6 print("打印ret = ", ret)
7 print("打印ret.stdout", ret.stdout)
```

执行结果

```
===== RESTART: D:\Python\ch30\ch30_35.py =====
数据类型 = <class 'subprocess.CompletedProcess'>
打印ret = CompletedProcess(args='echo %time%', returncode=0, stdout=b' 2:51:39
.33\r\n')
打印ret.stdout b' 2:51:39.33\r\n'
>>>
```

习题

1. 假设今天是2020年1月1日，请列出100天前是几月几号。
2. 请在主线程内建立2个线程，其中一个线程将在10分钟后打开，通知发简讯给客户，另一个线程在20分钟后打开，通知订机票到北京。
3. 请在程序内建立5个子进程，其中第一个子进程将在10分钟后打开，每隔10分钟一个子进程工作，这5个子进程皆是播放歌曲，相当于每隔10分钟播放一首歌。

第31章 海龟绘图

海龟绘图是一个很早期的绘图函数库，出现在1966年的Logo计算机语言，在笔者学生时期就曾经使用Logo语言控制海龟绘图。很高兴现在已经成为Python的模块，我们可以使用它绘制计算机图形。与先前介绍的绘图模块比较，最大的差异在我们可以看到海龟绘图的过程，增加动画效果。

31-1 基本观念与安装模块

海龟有3个关键属性，**方向**、**位置**和**笔**，笔也有属性，**色彩**、**宽度**和**开/关**状态。海龟绘图是Python内置的模块，使用前需导入此模块。

```
import turtle
```

31-2 绘图初体验

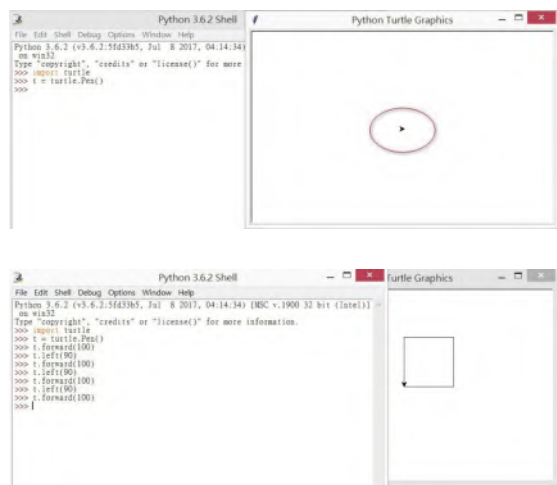
可以使用Pen()设定海龟绘图对象，例如：

```
t = turtle.Pen( )
```

上述代码执行后，就可以建立画布，同时屏幕中间就可以看到**箭头** (arrow)，这就是所谓的**海龟**，31-6节笔者将列出所有海龟外型。例如，右图是使用Python Shell执行时的画面。

在海龟绘图中，画布中央是(0,0)，往右x轴递增往左x轴递减，往上y轴递增往下y轴递减，海龟的起点在(0,0)位置，移动的单位是像素(pixel)。如果现在输入右图指令，可以看到海龟在Python Turtle Graphics画布上绘图。

上述我们画了一个正方形，其实每输入一条指令，都可以看到海龟转向或前进绘图。



31-3 绘图基本练习

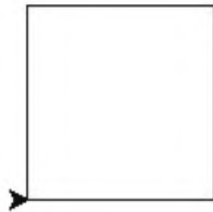
下列是海龟绘图基本方法的说明表。

方法	说明
left(angle) lt()	逆时针旋转角度
right(angle) rt()	顺时针旋转角度
forward(number) fd()	往前移动，number 是移动量
backward(number) bk() back()	往后移动，number 是移动量
setpos(x,y) goto() setposition()	更改海龟坐标至 (x,y)
hideturtle() ht()	隐藏海龟
showturtle() st()	显示海龟
isvisible()	海龟可见返回 True，否则返回 False
speed(n)	海龟速度，n=1(慢) ~ 10(快), 0(最快)

程序实例ch31_1.py： 使用海龟绘制正方形。

```
1 # ch31_1.py
2 import turtle
3
4 t = turtle.Pen()
5 t.forward(100)
6 t.left(90)
7 t.forward(100)
8 t.left(90)
9 t.forward(100)
10 t.left(90)
11 t.forward(100)
12 t.left(90)
```

执行结果

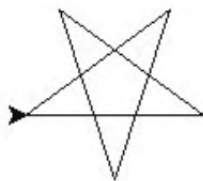


其实适度使用循环，可以创造一些有趣的图。

程序实例ch31_2.py： 绘制五角星。

```
1 # ch31_2.py
2 import turtle
3
4 t = turtle.Pen()
5 for x in range(1, 6):
6     t.forward(100)
7     t.left(144)
```

执行结果



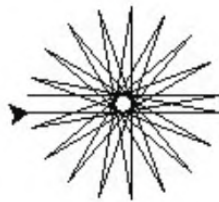
程序实例ch31_3.py： 绘制有趣图形，这次将旋转角度改成顺时针。

```

1 # ch31_3.py
2 import turtle
3
4 t = turtle.Pen()
5 for x in range(1, 20):
6     t.forward(100)
7     t.right(170)

```

执行结果



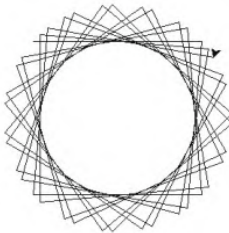
程序实例ch31_4.py: 绘制有趣图形。

```

1 # ch31_4.py
2 import turtle
3
4 t = turtle.Pen()
5 for x in range(1, 40):
6     t.forward(200)
7     t.right(95)

```

执行结果



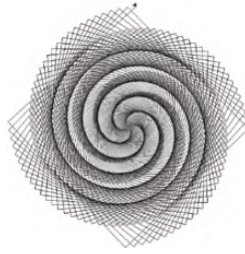
程序实例ch31_5.py: 绘制有趣图形。

```

1 # ch31_5.py
2 import turtle
3
4 t = turtle.Pen()
5 for x in range(1, 500):
6     t.forward(x)
7     t.right(91)

```

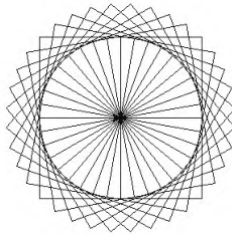
执行结果



程序实例ch31_6.py: 绘制有趣图形。

```
1 # ch31_6.py
2 import turtle
3
4 t = turtle.Pen()
5
6 for x in range(1, 37):
7     t.forward(100)
8     t.left(90)
9     t.forward(100)
10    t.left(90)
11    t.forward(100)
12    t.left(90)
13    t.forward(100)
14    t.left(100)
```

执行结果



31-4 控制画笔色彩与线条粗细

可以参考下面列表。

方法	说明
pencolor(color string)	选择彩色绘笔，颜色字符串可参考附录 D
color(r, g, b)	由 r, g, b 控制颜色，取值范围为 0 ~ 1
color((r,g,b))	这是元组 r,g,b，取值范围为 0 ~ 255
color(color string)	例如：red、green，颜色字符串可参考附录 D
pensize(size) width(size)	size 选择画笔粗细大小
penup() pu() up()	画笔是关闭
pendown() pd() down()	画笔是打开
isdown()	画笔是否打开，是则传回 True，否传回 False

由上图可知，色彩处理时我们可以使用选择彩色画笔pencolor()，也可以直接用color()方法更改目前画笔的颜色，color()方法的颜色可以是r, g, b组合，也可以是色彩字符串。在选择画笔粗细时可以使用pensize()，也可以使用width()。

程序实例ch31_7.py： 这个程序是重新设计ch31_6.py，首先将画笔粗细改为5，其次在使用循环绘图时，r=0.5，g=1，b则是由1逐渐变小。

```

1  # ch31_7.py
2  import turtle
3
4  t = turtle.Pen()
5  t.pensize(5)                # 画笔宽度
6  colorValue = 1.0
7  colorStep = colorValue / 36
8  for x in range(1, 37):
9      colorValue -= colorStep
10     t.color(0.5, 1, colorValue)  # 色彩调整
11     t.forward(100)
12     t.left(90)
13     t.forward(100)
14     t.left(90)
15     t.forward(100)
16     t.left(90)
17     t.forward(100)
18     t.left(100)

```

执行结果



程序实例ch31_8.py: 使用不同颜色与不同粗细画笔的应用。

```

1 # ch31_8.py
2 import turtle
3
4 t = turtle.Pen()
5 colorsList = ['red', 'orange', 'yellow', 'green', 'blue', 'cyan', 'purple', 'violet']
6 tWidth = 1 # 最初画笔宽度
7 for x in range(1, 41):
8     t.color(colorsList[x % 8]) # 选择画笔颜色
9     t.forward(2 + x * 5) # 每次移动距离
10    t.right(45) # 每次旋转角度
11    tWidth += x * 0.05 # 每次画笔宽度递增
12    t.width(tWidth)

```

执行结果



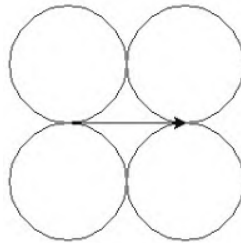
31-5 绘制圆或弧形

要绘制圆可以使用`circle(r)`，`r`是圆半径。绘制圆时当前海龟面对方向的左侧半径位置将是圆的中心。例如，海龟在(0, 0)位置，海龟方向是向东，则绘制半径为50的圆时，圆中心是在(0, 50)的位置。如果半径是正值 绘制圆时是海龟目前位置开始以逆时针方式绘制，如果半径是负值，则绘制圆时是从海龟目前位置开始以顺时针方式绘制。

程序实例ch31_9. py： 绘制4个圆其中半径是50或-50各2个，海龟位置是(0, 0)与(100, 0)。

```
1 # ch31_9.py
2 import turtle
3
4 t = turtle.Pen()
5 t.circle(50)          # 绘制第1个左上方圆
6 t.circle(-50)         # 绘制第2个左下方圆
7 t.forward(100)
8 t.circle(50)          # 绘制第3个右上方圆
9 t.circle(-50)         # 绘制第4个右下方圆
```

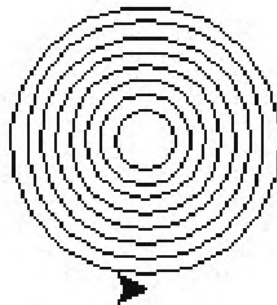
执行结果



程序实例ch31_10. py： 绘制同心圆。

```
1 # ch31_10.py
2 import turtle
3
4 t = turtle.Pen()
5 step = 5          # 每次增加距离
6 for r in range(10, 50, step):
7     t.circle(r)    # 绘制圆
8     t.penup()      # 将笔提起
9     t.right(90)     # 方向往下
10    t.forward(step) # 移动海龟位置起绘点
11    t.right(270)    # 方向往右
12    t.pendown()     # 将笔放下准备绘制
```

执行结果



上述penup()将笔提起不绘制任何线条也可想成关闭笔，pendown()将笔放下准备画图也可想成打开笔。在circle()方法内若有第2个参数，如果这个参数是360，则是一个圆；如果是180，则是半个圆弧。

程序实例ch31_11.py： 绘制弧度的应用。

```
1 # ch31_11.py
2 import turtle
3
4 t = turtle.Pen()
5 step = 5 # 每次增加距离
6 for r in range(10, 90, step):
7     t.circle(r, 90 + r*2) # 绘制圆
8     t.penup() # 将笔提起
9     t.home() # 海龟回到原点(0,0)
10    t.pendown() # 将笔放下准备绘制
```

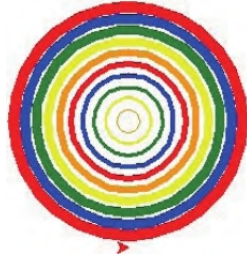
执行结果



程序实例ch31_12.py： 基本上是重新设计ch31_10.py，但是使用不同颜色与线条宽度，每次循环线条宽度加1的应用。

```
1 # ch31_12.py
2 import turtle
3
4 t = turtle.Pen()
5 colors = ['red', 'orange', 'yellow', 'green', 'blue']
6 step = 10 # 每次增加距离
7 twidth = 0 # 最初宽度0
8 for r in range(1, 11):
9     t.color(colors[r % 5]) # 选画笔颜色
10    twidth += 1 # 每次循环宽度加1
11    t.width(twidth) # 设定宽度
12    t.circle(r*step) # 绘制圆
13    t.penup() # 将笔关闭
14    t.right(90) # 方向往下
15    t.forward(step) # 移动海龟位置起绘点
16    t.right(270) # 方向往右
17    t.pendown() # 将笔开启准备绘制
```

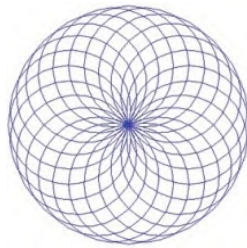
执行结果



程序实例ch31_13.py: 绘制圆线条的应用。

```
1 # ch31_13.py
2 import turtle
3
4 t = turtle.Pen()
5 t.color('blue')
6 for angle in range(0, 360, 15):
7     t.setheading(angle)      # 调整海龟方向
8     t.circle(100)
```

执行结果



上述用到了一个尚未讲解的方法setheading(), 也可以缩写seth(), 这是调整海龟方向, 海龟初始是向右, 相当于0度。

31-6 认识与操作海龟图像

在turtle模块内shape('turtle')方法可以让海龟呈现, stamp()可以使用海龟在画布盖章。

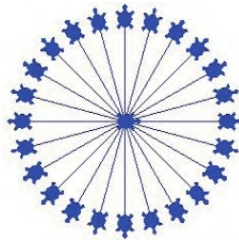
程序实例ch31_14.py: 让海龟呈现同时在画布盖章。

```

1 # ch31_14.py
2 import turtle
3
4 t = turtle.Pen()
5 t.color('blue')
6 t.shape('turtle')
7 for angle in range(0, 361, 15):
8     t.forward(100)
9     t.stamp()
10    t.home()
11    t.seth(angle)          # 调整海龟方向

```

执行结果



`clearstamps(n)` 如果 `n=None` 可以清除画布上所有的海龟，如果 `n` 是正值可以清除前 `n` 个海龟，如果 `n` 是负值可以清除后 `n` 个海龟。如果海龟在画布盖章时有设定返回值，例如，`stampID = t.stamp()`，未来也可以使用 `clearstamp(stampID)` 将这个特定的海龟盖章删除。

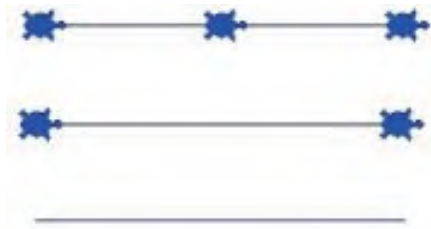
程序实例 ch31_15.py： 这个程序首先将绘制3个海龟（第7～11行），然后将自己隐藏（第12行），过5秒先删除第2只海龟（第14行），再过5秒将删除其他2只海龟（第16行）。

```

1 # ch31_15.py
2 import turtle, time
3
4 t = turtle.Pen()
5 t.color('blue')
6 t.shape('turtle')
7 firstStamp = t.stamp()      # 盖章第1只海龟
8 t.forward(100)
9 secondStamp = t.stamp()     # 盖章第2只海龟
10 t.forward(100)
11 thirdStamp = t.stamp()      # 盖章第3只海龟
12 t.hideturtle()              # 隐藏目前海龟
13 time.sleep(5)
14 t.clearstamp(secondStamp)    # 删除第2只海龟
15 time.sleep(5)
16 t.clearstamps(None)         # 删除所有海龟

```

执行结果 下列分别是显示3只海龟，删除第2只后剩2只以及全部删除的结果。



31-6-1 隐藏与显示海龟

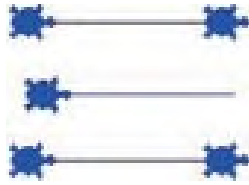
上述第12行`hideturtle()`是隐藏海龟，未来若是想显示海龟可以使用`showturtle()`方法。`isvisible()`可以检查目前程序是否有显示海龟，如果有显示可以返回`True`，如果没有显示则返回`False`。

程序实例`ch31_16.py`：这个程序会先盖章第1只海龟（第7行），第8行是打印是否显示海龟光标，结果是`True`。然后盖章第2只海龟（第10行），隐藏海龟光标，所以第13行打印是否显示海龟光标，结果是`False`。第14行是删除最后一只海龟，相当于是删除第2只海龟。第16行是显示海龟光标，所以第17行打印是否显示海龟光标，结果是`True`。

```
1 # ch31_16.py
2 import turtle, time
3
4 t = turtle.Pen()
5 t.color('blue')
6 t.shape('turtle')
7 t.stamp()          # 盖章第1只海龟
8 print("目前有显示海龟 :", t.isvisible())
9 t.forward(100)
10 secondStamp = t.stamp() # 盖章第2只海龟
11 time.sleep(3)
12 t.hideturtle()      # 隐藏目前海龟
13 print("目前有显示海龟 :", t.isvisible())
14 t.clearstamps(-1)   # 删除后面1个海龟
15 time.sleep(3)
16 t.showturtle()     # 显示海龟
17 print("目前有显示海龟 :", t.isvisible())
```

执行结果

```
===== RESTART: D:\Python\ch31\ch31_16.py
目前有显示海龟 : True
目前有显示海龟 : False
目前有显示海龟 : True
>>>
```



31-6-2 认识所有的海龟光标

`screen.getshapes()` 方法可以列出所有的海龟光标。

程序实例ch31_17.py: 列出所有海龟光标字符串, 与相对应的光标外型。

```
1 # ch31_17.py
2 import turtle, time
3
4 t = turtle.Pen()
5 t.color('blue')
6 print(t.screen.getshapes())          # 打印海龟光标字符串
7
8 for cursor in t.screen.getshapes():
9     t.shape(cursor)                 # 更改海龟光标
10    t.stamp()                       # 海龟光标盖章
11    t.forward(30)
```

执行结果

```
===== RESTART: D:/Python/ch31/ch31_17.py =====
['arrow', 'blank', 'circle', 'classic', 'square', 'triangle', 'turtle']
>>>
```



我们也可以使用下列方式将任意图片当作海龟光标, 不过图片不会在我们转动海龟时随着转动。

```
screen.register_shape("图片名称")
```

或者我们也可以使用下列方式自建一个外型当海龟光标。

```
screen.('myshape', ((3,-3), (0,3), (-3,-3)))
```

31-7 填满颜色

可以参考下表。

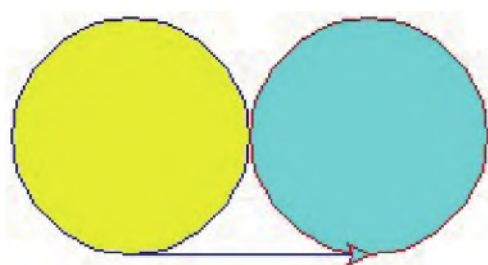
方法	说明
<code>begin_fill()</code>	想要开始填充前调用
<code>end_fill()</code>	对应 <code>begin_fill()</code> ，结束填充
<code>filling()</code>	如果填充则返回 <code>True</code> ，没有填充则返回 <code>False</code>
<code>fillcolor()</code>	填入当前色彩
<code>fillcolor(color string)</code>	例如： <code>red</code> 、 <code>green</code> ，颜色字符串可参考附录 D
<code>fillcolor((r,g,b))</code>	这是元组 <code>r,g,b</code> ，取值范围为 <code>0 ~ 255</code>
<code>fillcolor(r,g,b)</code>	由 <code>r, g, b</code> 控制颜色，取值范围为 <code>0 ~ 1</code>

在程序设计时，也可以使用 `color()` 包含 2 个参数，分别代表图形轮廓与内部颜色。

程序实例 `ch31_18.py`：设计填充 2 个圆，第一个填充轮廓颜色是蓝色填充颜色是黄色，分成 5 和 6 行撰写。第二个填充轮廓颜色是红色填充颜色是海蓝色，在第 11 行撰写。

```
1 # ch31_18.py
2 import turtle
3
4 t = turtle.Pen()
5 t.color('blue')      # 设定轮廓颜色
6 t.fillcolor('yellow') # 设定填充颜色
7 t.begin_fill()       # 开始填充
8 t.circle(50)         # 绘制左方圆
9 t.end_fill()         # 结束填充
10 t.forward(100)
11 t.color('red', 'aqua') # 设定轮廓颜色是red，填充颜色是aqua
12 t.begin_fill()       # 开始填充
13 t.circle(50)         # 绘制第2个右方圆
14 t.end_fill()         # 结束填充
```

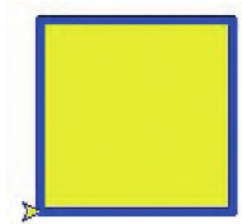
执行结果



程序实例ch31_19.py： 填充矩形的应用，轮廓宽度是5颜色是蓝色，所填充的是黄色。

```
1 # ch31_19.py
2 import turtle
3
4 t = turtle.Pen()
5 t.color('blue')      # 设定轮廓颜色
6 t.width(5)           # 轮廓宽度
7 t.fillcolor('yellow') # 设定填充颜色
8 t.begin_fill()        # 开始填充
9 t.forward(100)
10 t.left(90)
11 t.forward(100)
12 t.left(90)
13 t.forward(100)
14 t.left(90)
15 t.forward(100)
16 t.left(90)
17 t.end_fill()         # 结束填充
```

执行结果



31-8 颜色动画的设计

其实我们可以每隔一段时间更改填充区间颜色，达到颜色区间动画设计。

程序实例ch31_20.py： 每隔3秒更改填充的颜色。

```
1 # ch31_20.py
2 import turtle, time
3 colorsList = ['green', 'yellow', 'red']
4
5 t = turtle.Pen()
6 for i in range(0,3):
7     t.fillcolor(colorsList[i%3]) # 更改色彩
8     t.begin_fill()               # 开始填充
9     t.circle(50)                 # 绘制左方圆
10    t.end_fill()                 # 结束填充
11    time.sleep(3)                # 每隔3秒执行一次循环
```

执行结果 下列分别是每隔3秒的执行结果。



如果我们使用白色绘制圆轮廓可以达到隐藏轮廓颜色，若是再隐藏海龟光标，整个效果将更好。

程序实例ch31_21.py： 隐藏轮廓线和海龟（第7行），重新设计ch31_20.py，程序第9行使用白色线条绘制轮廓线相当于隐藏了轮廓，同时用指定颜色填满圆。

```
1 # ch31_21.py
2 import turtle, time
3 colorslist = ['green', 'yellow', 'red']
4
5 t = turtle.Pen()
6 t.speed(10)
7 t.ht()
8 for i in range(0,3):
9     t.color('white', colorslist[i%3])
10    t.begin_fill()
11    t.circle(50)
12    t.end_fill()
13    time.sleep(3)
```

执行结果



31-9 绘图窗口的相关知识

下列是相关方法使用表：

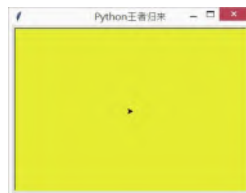
方法	说明
screen.title()	可设定窗口标题
screen.bgcolor()	窗口背景颜色
screen.bgpic(fn)	gif 文件当背景
screen.window_width()	窗口宽度
screen.window_height()	窗口高度
screen.setup(width,height)	重设窗口宽度与高度
screen.setworldcoorindates(x1,y1,x2,y2)	(x1,y1),(x2,y2) 分别是画布左上与右下的坐标

31-9-1 更改海龟窗口标题与背景颜色

程序实例ch31_22.py： 默认窗口标题是Python Turtle Graphics，这个程序会更改窗口标题为Python王者归来，同时设定背景颜色是黄色。

```
1 # ch31_22.py
2 import turtle
3
4 t = turtle.Pen()
5 t.screen.title('Python王者归来')
6 t.screen.bgcolor('yellow')
```

执行结果



31-9-2 取得/更改窗口宽度与高度

程序实例ch31_23.py： 分别传回海龟窗口的宽度与高度，3秒后这个程序会更改窗口宽度与高度分别为600与480。

```
1 # ch31_23.py
2 import turtle,time
3
4 t = turtle.Pen()
5 width = t.screen.window_width()
6 height = t.screen.window_height()
7 print("窗口width = ", width)
8 print("窗口height = ", height)
9 time.sleep(3)
10 t.screen.setup(600, 480) # 更改窗口宽和高
11 width = t.screen.window_width()
12 height = t.screen.window_height()
13 print("新窗口width = ", width)
14 print("新窗口height = ", height)
```

执行结果 读者可以在屏幕看到窗口大小更改结果。

```
===== RESTART: D:\Python\ch31\ch31_23.py
窗 width = 960
窗 height = 810
新窗 width = 600
新窗 height = 480
>>>
```

31-9-3 重设世界坐标

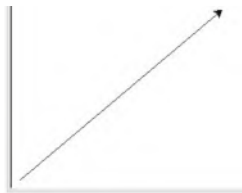
当使用`screen.setworldcoordinates()`重设海龟窗口为世界坐标时，海龟光标位置仍是在(0,0)。

程序实例ch31_24.py： 重设海龟窗口为世界坐标时，让窗口左下角坐标是(0,0)，先打印目前海龟光标坐标，然后画一条左下到右上的线，最后再打印一次当前海龟光标坐标。

```
1 # ch31_24.py
2 import turtle,time
3
4 t = turtle.Pen()
5 t.screen.setworldcoordinates(0,0,800,800)
6 print("打印海龟位置 = ", t.pos())
7 t.left(45)
8 t.forward(300)
9 print("打印新海龟位置 = ", t.pos())
```

执行结果

```
===== RESTART: D:\Python\ch31\ch31_24.py
打印海龟位置 = (0.00,0.00)
打印新海龟位置 = (212.13,212.13)
>>>
```



31-10 文字的输出生

可以使用`write()`输出文字。

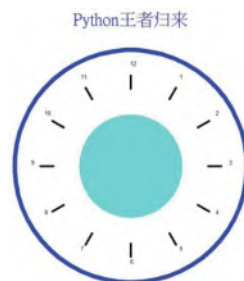
```
write(arg, move=False, align="left", font=( ))
```

arg是要写入海龟窗口的文字对象，move默认是False，如果是True画笔将移到本文右下角，align是“left”“center”或“right”。如果想自定义字体，可以在font=()内设定(fontname, fontsize, fonttype)。

程序实例ch31_25.py： 绘制时钟，同时在时钟上方输出文字。

```
1 # ch31_25.py
2 import turtle
3
4 t = turtle.Pen()
5 t.shape('turtle')
6 # 绘制时钟中间颜色
7 t.color('white', 'aqua')
8 t.setpos(0, -120)
9 t.begin_fill()
10 t.circle(120) # 绘制时钟内圆盘
11 t.end_fill()
12 t.penup() # 画笔关闭
13 t.home()
14 t.pendown() # 画笔打开
15 t.color('black')
16 t.pensize(5)
17 # 绘制时钟刻度
18 for i in range(1, 13):
19     t.penup() # 画笔关闭
20     t.seth(-30*i+90) # 设定刻度的角度
21     t.forward(180)
22     t.pendown() # 画笔打开
23     t.forward(30) # 画时间轴
24     t.penup()
25     t.forward(20)
26     t.write(str(i), align="left") # 写上刻度
27     t.home()
28 # 绘制时钟外框
29 t.home()
30 t.setpos(0, -270)
31 t.pendown()
32 t.pensize(10)
33 t.pencolor('blue')
34 t.circle(270)
35 # 写上名字
36 t.penup()
37 t.setpos(0, 320)
38 t.pendown()
39 t.write('Python王者归来', align="center", font=('新细明体', 24))
40 t.ht() # 隐藏光标
```

执行结果



31-11 鼠标与键盘信号

Python的turtle模块也提供简单的方法可以允许我们在Python Turtle Graphics窗口接收鼠标按键信号，然后可以设计成是针对这些信号做出反应。

31-11-1 onclick()

这个方法主要是在Python Turtle Graphics窗口有鼠标按键发生时，会执行参数的内容，而所放的参数是我们设计的函数：

```
onclick(fun, btn=1, add=None)
```

fun是发生onclick事件时所要执行的函数名称，它会传递按键发生的x, y位置给fun函数，btn默认是鼠标左键，可参考下列实例说明。

程序实例ch31_26.py： 当在Python Turtle Graphics窗口有按键发生时，在Python的Python Shell窗口将列出鼠标光标被按的x, y位置。下列是笔者在Python Turtle Graphics窗口单击鼠标键时的位置。

```
1 # ch31_26.py
2 import turtle
3
4 def printStr(x, y):
5     print(x, y)
6
7 t = turtle.Pen()
8 t.screen.onclick(printStr)
9 t.screen.mainloop()
```

执行结果

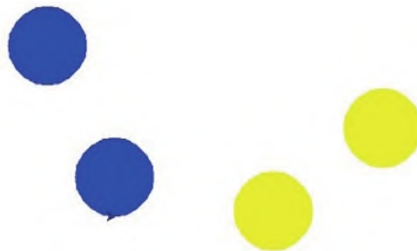
```
===== RESTART: D:/Python/ch31/ch31_26.py
160.0 50.0
122.0 -20.0
114.0 -29.0
208.0 -48.0
>>>
```


上述`screen.mainloop()`方法必须在程序最后一行，让程序不结束，直到Python Turtle Graphics窗口关闭，才执行结束。

程序实例ch31_27.py: 当在x轴大于0位置单击，绘半径是50的黄色圆，如果在x轴小于0位置单击，绘制半径为50的蓝色圆。

```
1 # ch31_27.py
2 import turtle
3
4 def drawSignal(x, y):
5     if x > 0:
6         t.fillcolor('yellow')
7     else:
8         t.fillcolor('blue')
9     t.penup()
10    t.setpos(x,y-50)      # 设定绘图起点
11    t.begin_fill()
12    t.circle(50)
13    t.end_fill()
14
15 t = turtle.Pen()
16 t.screen.onclick(drawSignal)
17 t.screen.mainloop()
```

执行结果



31-11-2 onkey()和listen()

`onkey()`主要是关注键盘的信号，语法如下：

```
onkey(fun, key) # fun是所要执行的函数，key是键盘按键
```

`onkey()`无法单独运作，需要`listen()`倾听将信号传给`onkey()`。

程序实例ch31_28.py: 单击up键海龟往上移50，单击down键海龟往下移50。


```

1  # ch31_28.py
2  import turtle
3
4  def keyUp():
5      t.seth(90)
6      t.forward(50)
7  def keyDn():
8      t.seth(270)
9      t.forward(50)
10
11 t = turtle.Pen()
12 t.screen.onkey(keyUp, 'Up')
13 t.screen.onkey(keyDn, 'Down')
14 t.screen.listen()
15 t.screen.mainloop()

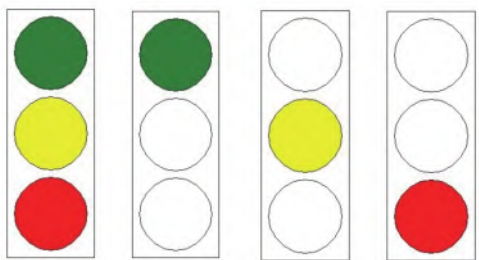
```

执行结果

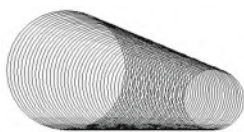


习题

1. 请设计一个 `line(x1, y1, x2, y2)` 函数，可以从 $(x1, y1)$ 绘线至 $(x2, y2)$ 。
2. 请设计一个绘制正方形的函数 `mysquare(n)`， n 是边框长度，这个函数可以在目前海龟光标位置绘制正方形。
3. 请设计红绿灯程序，下列最左边是3个灯皆亮的情况，但是期待结果是绿灯亮的时间是10秒，黄灯亮的时间是3秒，红灯亮的时间是10秒，如此循环10次。



4. 请绘制下列图形，最大半径100，画50次，半径每次递减1，起绘点每次往右移5。



5. 请扩充ch31_27.py：先绘出x, y为0的十字坐标轴，同时，如果x, y皆为正值，则绘半径为50的黄色圆；如果x正值y负值，则绘制半径为50的红色圆；如果x负值y正值，则绘制半径为50的蓝色圆；如果x, y皆为负值，则绘制半径为50的绿色圆。

第32章 动画与游戏

我们可以使用前一章的海龟绘制图案，同时可以清楚地看到海龟绘图的过程，不过它的最大缺点是速度不够快。这一章我们将介绍Python内置的模块tkinter制作动画，而动画也是设计游戏的基础，其实tkinter模块提供的功能有很多，本章将介绍动画与游戏设计所需的方法。

32-1 建立tkinter对象

虽然本章重点是动画，还是先介绍用tkinter模块建立一个按钮，让读者了解这个模块多元化的功能。当然使用tkinter模块需先导入此模块。

```
from tkinter import *
```

接着需建立tkinter对象，可以使用下列指令。

```
tk = Tk( )    # tk是自行定义的对象名称
```

32-2 建立按钮

可以使用Button()建立按钮：

```
btn  
  
= Button(tk, text="按钮名称", command=fun  
)    # fun
```

定义按钮的工作

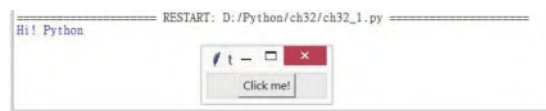
```
btn
```

`.pack()` # 可以将按钮包装好，这是必要的

程序实例ch32_1.py: 单击Click me!按钮，列出Hi! Python。

```
1 # ch32_1.py
2 from tkinter import *
3
4 def buttonClick():
5     print("Hi! Python")
6
7 tk = Tk()
8 btn = Button(tk, text="Click me!", command=buttonClick)
9 btn.pack()
10 mainloop()
```

执行结果



上述`mainloop()`可以让程序持续进行，直到关闭按钮窗口。

32-3 绘图功能

32-3-1 建立画布

可以使用`Canvas()`方法建立画布对象。

```
canvas = Canvas(tk, width=xx
, height=yy
)
# xx
, yy
```

是画布宽

与高

`canvas.pack()` # 可以将画布包装好，这是必要的

画布建立完成后，左上角是坐标(0, 0)，向右x轴递增，向下y轴递增。

32-3-2 绘线条create_line()

它的使用方式如下：

```
create_line(x1, y1  
            , x2, y2  
            , options) # x1,y1
```

是线条起点

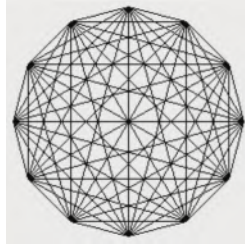
```
, x2,y2
```

是线条终点

程序实例ch32_2.py： 在半径为100的圆外围建立12个点，然后将这些点彼此连接。

```
1 # ch32_2.py  
2 from tkinter import *  
3 import math  
4  
5 tk = Tk()  
6 canvas = Canvas(tk, width=640, height=480)  
7 canvas.pack()  
8 x_center, y_center, r = 320, 240, 100  
9 x, y = [], []  
10 for i in range(12): # 建立圆外围12个点  
11     x.append(x_center + r * math.cos(30*i*math.pi/180))  
12     y.append(y_center + r * math.sin(30*i*math.pi/180))  
13 for i in range(12): # 执行12个点彼此连接  
14     for j in range(12):  
15         canvas.create_line(x[i],y[i],x[j],y[j])
```

执行结果



上述程序使用了数学函数 $\sin()$ 和 $\cos()$ 以及 π ，这些是在`math`模块。使用`create_line()`时，在`options`参数字段可以用`fill`设定线条颜色，用`width`设定线条宽度。

程序实例`ch32_3.py`： 不同线条颜色与宽度。

```
1 # ch32_3.py
2 from tkinter import *
3 import math
4
5 tk = Tk()
6 canvas = Canvas(tk, width=640, height=480)
7 canvas.pack()
8 canvas.create_line(100,100,500,100)
9 canvas.create_line(100,125,500,125,width=5)
10 canvas.create_line(100,150,500,150,width=10,fill='blue')
```

执行结果



32-3-3 绘矩形`create_rectangle()`

它的使用方式如下：

```
create_rectangle(x1, y1
, x2, y2
, options) #x1,y1,x2,y2
```

是矩形左上角

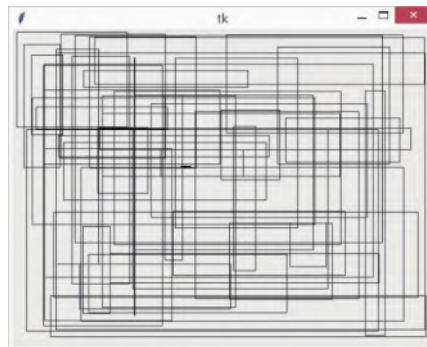
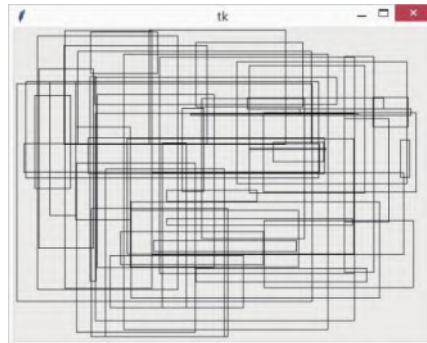
和右下角

坐标

程序实例ch32_4.py: 在画布内随机产生不同位置与大小的矩形。

```
1 # ch32_4.py
2 from tkinter import *
3 from random import *
4
5 tk = Tk()
6 canvas = Canvas(tk, width=640, height=480)
7 canvas.pack()
8 for i in range(50):          # 随机绘50个不同位置与大小的矩形
9     x1, y1 = randint(1, 640), randint(1, 480)
10    x2, y2 = randint(1, 640), randint(1, 480)
11    if x1 > x2: x1,x2 = x2,x1    # 确保左上角x坐标小于右下角x坐标
12    if y1 > y2: y1,y2 = y2,y1  # 确保左上角y坐标小于右下角y坐标
13    canvas.create_rectangle(x1, y1, x2, y2)
```

执行结果



这个程序每次执行时皆会产生不同的结果，有一点艺术画的效果。使用 `create_rectangle()` 时，在 `options` 参数字段可以用 `fill='color'` 设定矩形填充颜色，用 `outline='color'` 设定矩形轮廓颜色。

程序实例ch32_5.py: 绘制3个矩形，第一个使用红色填充轮廓色是预设，第二个使用黄色填充轮廓是蓝色，第三个使用绿色填充轮廓是

灰色。

```
1 # ch32_5.py
2 from tkinter import *
3 from random import *
4
5 tk = Tk()
6 canvas = Canvas(tk, width=640, height=480)
7 canvas.pack()
8 canvas.create_rectangle(10, 10, 120, 60, fill='red')
9 canvas.create_rectangle(130, 10, 200, 80, fill='yellow', outline='blue')
10 canvas.create_rectangle(210, 10, 300, 60, fill='green', outline='grey')
```

执行结果



由执行结果可以发现由于画布底色是浅灰色，所以第三个矩形用灰色轮廓，几乎看不到轮廓线，另外也可以用width设定矩形轮廓的宽度。

32-3-4 绘圆弧create arc()

它的使用方式如下：

```
create_arc(x1, y1
, x2, y2
, extent=angle, style=ARC, options)
```

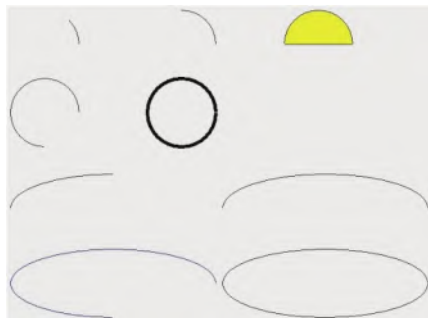
x1,y1,x2,y2 分别是包围圆形的矩形的左上角 和右下角 坐标。如果要绘圆形extent值是359，如果写360会视为0度。如果extent是介于1~359，则绘制这个角度的圆弧。上述style=ARC表示绘制圆弧，如果是要使用options参数填满圆弧则需舍去此参数。此外，options参数可以使用width设定轮廓线条宽度（可参考第12行），outline设定轮廓线条颜色（可参考第16行），fill设定填充颜色（可参考第10行）。目前预设绘圆弧的起点是右边，也可以用start=0代表，也可以

由设定start的值更改圆弧的起点，方向是逆时针，可参考ch32_6.py第14行。

程序实例ch32_6.py： 绘制各种不同的圆和椭圆，以及圆弧和椭圆弧。

```
1 # ch32_6.py
2 from tkinter import *
3
4 tk = Tk()
5 canvas = Canvas(tk, width=640, height=480)
6 canvas.pack()
7 # 以下以圆形为基础
8 canvas.create_arc(10, 10, 110, 110, extent=45, style=ARC)
9 canvas.create_arc(210, 10, 310, 110, extent=90, style=ARC)
10 canvas.create_arc(410, 10, 510, 110, extent=180, fill='yellow')
11 canvas.create_arc(10, 110, 110, 210, extent=270, style=ARC)
12 canvas.create_arc(210, 110, 310, 210, extent=359, style=ARC, width=5)
13 # 以下以椭圆形为基础
14 canvas.create_arc(10, 250, 310, 350, extent=90, style=ARC, start=90)
15 canvas.create_arc(320, 250, 620, 350, extent=180, style=ARC)
16 canvas.create_arc(10, 360, 310, 460, extent=270, style=ARC, outline='blue')
17 canvas.create_arc(320, 360, 620, 460, extent=359, style=ARC)
```

执行结果



32-3-5 绘制圆或椭圆create oval()

它的使用方式如下：

```
create_oval(x1, y1
, x2, y2
, options)
```

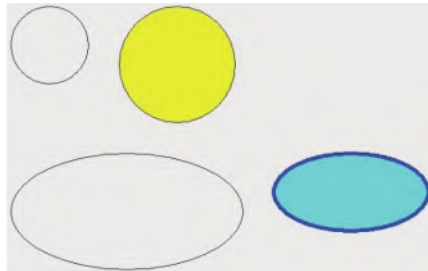
x1, y1, x2, y2 分别是包围圆形的矩形的左上角 和右下角 坐标，在options参数可以使用width设定轮廓线条宽度，outline设定轮廓线

条颜色，fill设定填充颜色。

程序实例ch32_7.py: 圆和椭圆的绘制。

```
1 # ch32_7.py
2 from tkinter import *
3
4 tk = Tk()
5 canvas = Canvas(tk, width=640, height=480)
6 canvas.pack()
7 # 以下是圆形
8 canvas.create_oval(10, 10, 110, 110)
9 canvas.create_oval(150, 10, 300, 160, fill='yellow')
10 # 以下是椭圆形
11 canvas.create_oval(10, 200, 310, 350)
12 canvas.create_oval(350, 200, 550, 300, fill='aqua', outline='blue', width=5)
```

执行结果



32-3-6 绘制多边形create_polygon()

它的使用方式如下：

```
create_polygon(x1, y1, x2, y2, x3, y3, ... xn, yn, options)
```

x1,y1, ... xn,yn 是多边形各角的x,y坐标，在options参数可以使用width设定轮廓线条宽度，outline设定轮廓线条颜色，fill设定填充颜色。

程序实例ch32_8.py: 绘制多边形的应用。

```
1 # ch32_8.py
2 from tkinter import *
3
4 tk = Tk()
5 canvas = Canvas(tk, width=640, height=480)
6 canvas.pack()
7 canvas.create_polygon(10,10, 100,10, 50,80, fill='', outline='black')
8 canvas.create_polygon(120,10, 180,30, 250,100, 200,90, 130,80)
9 canvas.create_polygon(200,10, 350,30, 420,70, 360,90, fill='aqua')
10 canvas.create_polygon(400,10,600,10,450,80,width=5,outline='blue',fill='yellow')
```

执行结果



32-3-7 输出文字create text()

它的使用方式如下：

`create_text(x,y,text=字符串, options)` # x,y是文字字符串输出的中心

在options参数可以使用fill设定字体颜色，使用font设定字体和字号。

程序实例ch32_9.py: 输出文字的应用。

```
1 # ch32_9.py
2 from tkinter import *
3
4 tk = Tk()
5 canvas = Canvas(tk, width=640, height=480)
6 canvas.pack()
7 canvas.create_text(200, 50, text='Ming-Chi Institute of Technology')
8 canvas.create_text(200, 80, text='Ming-Chi Institute of Technology', fill='blue')
9 canvas.create_text(300, 120, text='Ming-Chi Institute of Technology', fill='blue',
10                        font=('Old English Text MT',20))
11 canvas.create_text(300, 160, text='Ming-Chi Institute of Technology', fill='blue',
12                        font=('华康新综艺体 Std W7',20))
13 canvas.create_text(300, 200, text='明志科技大学', fill='blue',
14                        font=('华康新综艺体 Std W7',20))
```

执行结果



32-3-8 图像的输出create image()

它的使用方式如下：

myPict

```

= PhotoImage(file='path')          # path是完整的文件路径
    create_image(x,y,image=myPict

, options)                          # x,y是图像输出的位置

```

图像输出必须是gif文件，同时使用PhotoImage()建立对象，再将此对象放入create_image()方法内，在options参数可以使用anchor=NW参数，表示左上角当作图像起始点，如果没有此参数，表示起始点是图片中心。

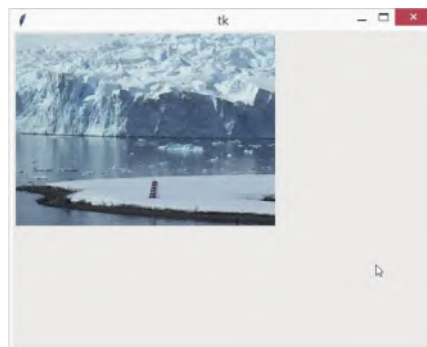
程序实例ch32_10.py： 图像输出的应用，此例由于笔者工作环境是当前文件夹，所以第7行可以省略路径。

```

1 # ch32_10.py
2 from tkinter import *
3
4 tk = Tk()
5 canvas = Canvas(tk, width=640, height=480)
6 canvas.pack()
7 myPict = PhotoImage(file='antarctica.gif')
8 canvas.create_image(0,0,image=myPict, anchor=NW)

```

执行结果



32-3-9 tk窗口标题的设定title()

窗口默认标题是tk，可以用title()设定窗口标题，可参考下列实例。

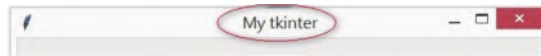
程序实例ch32_11.py： 将窗口标题改为My tkiner。

```

1 # ch32_11.py
2 from tkinter import *
3
4 tk = Tk()
5 tk.title('My tkinter')
6 canvas = Canvas(tk, width=640, height=480)
7 canvas.pack()

```

执行结果



32-3-10 更改画布背景颜色

在使用Canvas()方法建立画布时，可以加上bg参数设置画布背景颜色。

程序实例ch32_12.py： 将画布背景改成黄色。

```

1 # ch32_12.py
2 from tkinter import *
3
4 tk = Tk()
5 canvas = Canvas(tk, width=640, height=240, bg='yellow')
6 canvas.pack()

```

执行结果



32-4 滚动条控制画布背景颜色

tkinter模块有滚动条方法Scale()，利用这个方法我们可以获得滚动条 的值，它的使用格式如下：

```
obj = Scale(tk, from_=start, to=end, command=fun) # fun定义按钮的工作
```

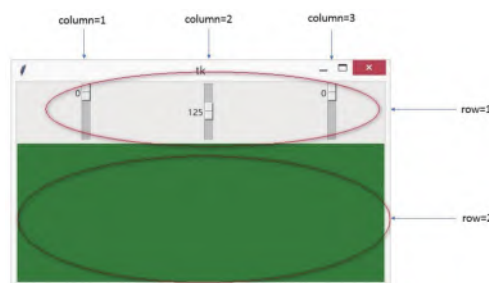
start是滚动条的起始值，end是滚动条的终点值，obj是滚动条(slider)对象，如果滚动滚动条，会执行fun()函数。经过上述设定后，可以使用下列方法设定滚动条的初值，与读取滚动条的值。

```
obj.set(initial_value)  # 设定初值
obj.get( )              # 读取滚动条值
```

程序实例ch32_13.py： 使用滚动条控制画布背景颜色，其中为了让读者了解设定滚动条初值的方法，第17行特别设定gSlider的滚动条初值为125。这个程序在执行时，若是有滚动滚动条将调用bgUpdate(source)函数，source在此是语法需要，实质没有作用。第10行config()方法是需要使用16进位方式设定背景色，格式是“#007d00”。第18~20行的grid()方法是定义滚动条和画布的位置，第20行的columnspan=3是设定将3个字段组成一个字段。

```
1 # ch32_13.py
2 from tkinter import *
3 def bgUpdate(source):
4     ''' 更改画布背景颜色 '''
5     red = rSlider.get()          # 读取red值
6     green = gSlider.get()        # 读取green值
7     blue = bSlider.get()         # 读取blue值
8     print("R=%d, G=%d, B=%d" % (red, green, blue)) # 打印色彩数值
9     myColor = "#%02x%02x%02x" % (red, green, blue) # 将颜色转成16进位字符串
10    canvas.config(bg=myColor)    # 设定画布背景颜色
11
12 tk = Tk()
13 canvas = Canvas(tk, width=640, height=240) # 初始化背景
14 rSlider = Scale(tk, from_=0, to=255, command=bgUpdate)
15 gSlider = Scale(tk, from_=0, to=255, command=bgUpdate)
16 bSlider = Scale(tk, from_=0, to=255, command=bgUpdate)
17 gSlider.set(125)                # 设定green是125
18 rSlider.grid(row=1, column=1)  # 第一行第一栏
19 gSlider.grid(row=1, column=2)  # 第一行第二栏
20 bSlider.grid(row=1, column=3)  # 第一行第三栏
21 canvas.grid(row=2, column=1, columnspan=3) # 第二行全部
22 mainloop()
```

执行结果



32-5 动画设计

32-5-1 基本动画

动画设计所使用的方法是`move()`，使用格式如下：

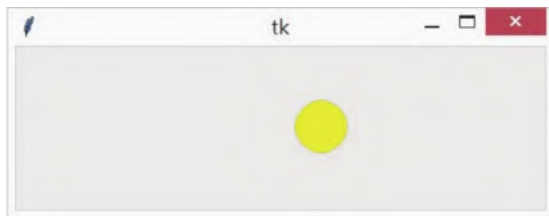
```
canvas.move(ID, xMove
, yMove
) # ID是物件编号
    canvas.update() # 强制重绘画布
```

`xMove` , `yMove` 是x, y轴移动距离，单位是像素。

程序实例ch32_14.py： 移动球的设计，每次移动5像素。

```
1 # ch32_14.py
2 from tkinter import *
3 import time
4
5 tk = Tk()
6 canvas= Canvas(tk, width=500, height=150)
7 canvas.pack()
8 canvas.create_oval(10,50,60,100,fill='yellow', outline='lightgray')
9 for x in range(0, 80):
10     canvas.move(1, 5, 0) # ID=1 x轴移动5像素，y轴不变
11     tk.update() # 强制tkinter重绘
12     time.sleep(0.05)
```

执行结果



上述执行时笔者使用循环，第12行相当于定义每隔0.05秒移动一次。其实我们只要设定`move()`方法的参数就可以往任意方向移动。

程序实例ch32_15： 扩大画布高度为300，每次移动x轴移动5，y轴移动2。


```
10 canvas.move(1, 5, 2) # ID=1 x轴移动5像素, y轴移动2像素
```

执行结果 读者可以自行体会球往右下方移动。

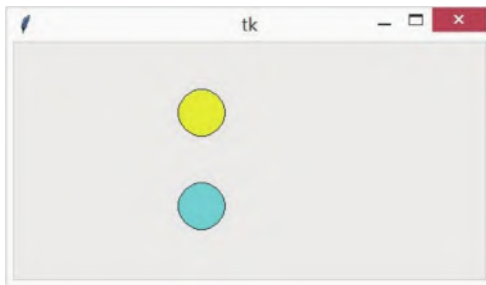
32-5-2 多个球移动的设计

在建立球对象时，可以设定id值，未来可以利用这个id值放入move()方法内，告知是移动这个球。

程序实例ch32_16.py： 一次移动2个球，第8行设定黄色球是id1，第9行设定水蓝色球是id2。

```
1 # ch32_16.py
2 from tkinter import *
3 import time
4
5 tk = Tk()
6 canvas= Canvas(tk, width=500, height=250)
7 canvas.pack()
8 id1 = canvas.create_oval(10,50,60,100,fill='yellow')
9 id2 = canvas.create_oval(10,150,60,200,fill='aqua')
10 for x in range(0, 80):
11     canvas.move(id1, 5, 0) # id1 x轴移动5像素, y轴移动0像素
12     canvas.move(id2, 5, 0) # id2 x轴移动5像素, y轴移动0像素
13     tk.update()           # 强制tkinter重绘
14     time.sleep(0.05)
```

执行结果



32-5-3 将随机数应用在多个球体的移动

在拉斯维加斯或是澳门赌场，常可以看到机器赛马的赌具，其实我们若是将球改成赛马，意义是相同的。



观念1：赌场可以作弊方式

假设笔者想让黄色球跑的速度快一些，他赢的机率是70%，可以利用randint()产生1~100的随机数，让随机数1~70间移动黄球，71~100间移动水蓝色球，这样笔者就动手脚了。

观念2：赌场作弊现形

当我们玩赛马赌具时必须下注，如果赌场要作弊最佳方式是，让下注最少的马匹有较高机率的移动机会，这样钱潮就滚滚而来了，很久以来笔者已经不碰这类的游戏了。

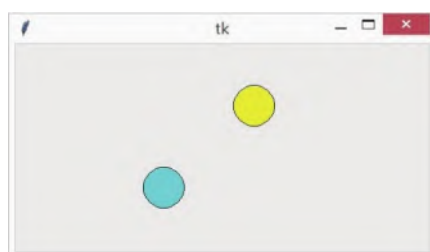
观念3：不作弊

我们可以设计随机数1~50间移动黄球，51~100间移动水蓝色球。

程序实例ch32_17.py： 让循环跑100次看哪一个球跑得快，让黄色球有70%赢的机会。

```
11 for x in range(0, 100):
12     if randint(1,100) > 70:
13         canvas.move(id2, 5, 0) # id2 x轴移动5像素, y轴移动0像素
14     else:
15         canvas.move(id1, 5, 0) # id1 x轴移动5像素, y轴移动0像素
16     tk.update() # 强制tkinter重绘
17     time.sleep(0.05)
```

执行结果



32-5-4 信息绑定

主要观念是可以利用系统接收到键盘的信息，做出反应。例如，当发生按下右移键时，可以控制球往右边移动，我们可以这样设计函数。

```
def ballMove(event):
    canvas.move(1, 5, 0)          # 假设移动 5 像素
```

在程序设计函数中对于按下右移键移动球可以这样设计。

```
def ballMove(event):
    if event.keysym == 'Right':
        canvas.move(1, 5, 0)
```

对于主程序而言需使用`canvas.bind_all()`函数，执行信息绑定工作，它的写法如下：

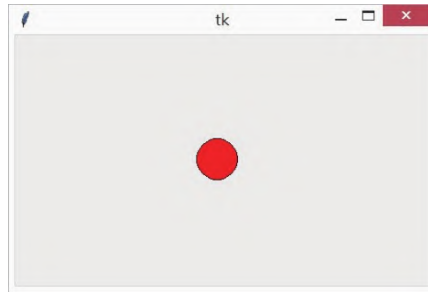
```
canvas.bind_all('<Key-Press-Right
>', ballMove)
```

上述函数主要是告知程序所接收到的键盘信息是什么，然后调用`ballMove()`函数执行键盘信息的工作。

程序实例ch32_18.py： 程序开始执行时，在画布中央有一个红球，可以按键盘的向右、向左、向上、向下键，往右、往左、往上、往下移动球，每次移动5个像素。

```
1 # ch32_18.py
2 from tkinter import *
3 import time
4 def ballMove(event):
5     if event.keysym == 'Left': # 左移
6         canvas.move(1, -5, 0)
7     if event.keysym == 'Right': # 右移
8         canvas.move(1, 5, 0)
9     if event.keysym == 'Up': # 上移
10        canvas.move(1, 0, -5)
11    if event.keysym == 'Down': # 下移
12        canvas.move(1, 0, 5)
13 tk = Tk()
14 canvas= Canvas(tk, width=500, height=300)
15 canvas.pack()
16 canvas.create_oval(225,125,275,175,fill='red')
17 canvas.bind_all('<KeyPress-Left>', ballMove)
18 canvas.bind_all('<KeyPress-Right>', ballMove)
19 canvas.bind_all('<KeyPress-Up>', ballMove)
20 canvas.bind_all('<KeyPress-Down>', ballMove)
21 mainloop()
```

执行结果



32-6 弹球游戏设计

这一节笔者将一步一步引导读者设计一个弹球游戏。

32-6-1 设计球往下移动

程序实例ch32_19.py: 定义画布窗口名称为Bouncing Ball, 同时定义画布宽度(14行)与高度(15行)分别为640, 480。这个球将往下移动然后消失, 移到超出画布范围就消失了。

```
1 # ch32_19.py
2 from tkinter import *
3 from random import *
4 import time
5
6 class Ball:
7     def __init__(self, canvas, color, winW, winH):
8         self.canvas = canvas
9         self.id = canvas.create_oval(0, 0, 20, 20, fill=color) # 建立球对象
10        self.canvas.move(self.id, winW/2, winH/2) # 设定球最初位置
11
12        def ballMove(self):
13            self.canvas.move(self.id, 0, step) # step是正值表示往下移动
14
15 winW = 640 # 定义画布宽度
16 winH = 480 # 定义画布高度
17 step = 3 # 定义速度可想成位移步伐
18 speed = 0.03 # 设定移动速度
19
20 tk = Tk()
21 tk.title("Bouncing Ball") # 游戏窗口标题
22 tk.wm_attributes('-topmost', 1) # 确保游戏窗口在屏幕最上层
23 canvas = Canvas(tk, width=winW, height=winH)
24 canvas.pack()
25 tk.update()
26
27 ball = Ball(canvas, 'yellow', winW, winH) # 定义球对象
28
29 while True:
30     ball.ballMove()
31     tk.update()
32     time.sleep(speed) # 可以控制移动速度
```

执行结果



这个程序由于是一个无限循环（第28～31行），所以我们强制关闭画布窗口时，将在Python Shell窗口看到错误信息，这无所谓，本章最后实例笔者会改良程序此情况。整个程序可以用球每次移动的步伐（16行）和循环第31行`time.sleep(speed)`指令的`speed`值，控制球的移动速度。

上述程序笔者建立了`Ball`类别，这个类别在初始化`__init__()`方法中，我们在第9行建立了球对象，第10行先设定球大约是在中间位置。另外我们建立了`ballMove()`方法，这个方法会依`step`变量移动，在此例每次往下移动。

32-6-2 设计让球上下反弹

如果想让所设计的球上下反弹，首先需了解Tkinter模块如何定义对象的位置，其实以这个实例而言，可以使用`coords()`方法获得对象位置，它的返回值是对象的左上角和右下角坐标。

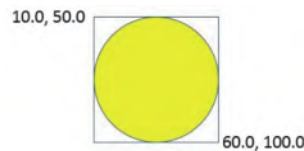
程序实例ch32_20.py： 主要是建立一个球，然后用`coords()`方法列出球位置的信息。

```
1 # ch32_20.py
2 from tkinter import *
3
4 tk = Tk()
5 canvas = Canvas(tk, width=500, height=150)
6 canvas.pack()
7 id = canvas.create_oval(10,50,60,100,fill='yellow', outline='lightgray')
8 ballPos = canvas.coords(id)
9 print(ballPos)
```

执行结果

```
===== RESTART: D:/Python/ch32/ch32_20.py =====
[10.0, 50.0, 60.0, 100.0]
>>>
```

若以上述执行结果为例，可以用下列图示做解说。



相当于可以用coords()方法获得下列结果。

ballPos[0]: 球的左边x轴坐标，未来可用于判别是否撞到画布左方。

ballPos[1]: 球的上边y轴坐标，未来可用于判别是否撞到画布上方。

ballPos[2]: 球的右边x轴坐标，未来可用于判别是否撞到画布右方。

ballPos[3]: 球的左边y轴坐标，未来可用于判别是否撞到画布下方。

程序实例ch32_21.py: 改良ch32_19.py，设计让球可以上下方移动，其实这个程序只是更改Ball类别内容。读者可以观察屏幕，球上下移动的结果。

```
6 class Ball:
7     def __init__(self, canvas, color, winW, winH):
8         self.canvas = canvas
9         self.id = canvas.create_oval(0, 0, 20, 20, fill=color) # 建立球对象
10        self.canvas.move(self.id, winW/2, winH/2) # 设定球最初位置
11        self.x = 0 # 水平不移动
12        self.y = step # 垂直移动单位
13    def ballMove(self):
14        self.canvas.move(self.id, self.x, self.y) # step是正值表示往下移动
15        ballPos = self.canvas.coords(self.id)
16        if ballPos[1] <= 0: # 侦测球是否超过画布上方
17            self.y = step
18        if ballPos[3] >= winH: # 侦测球是否超过画布下方
19            self.y = -step
```

程序第11行定义球x轴不移动，第12行定义y轴移动单位是step。第15行获得球的位置信息，第16~17行侦测如果球撞到画布上方未来

球移动是往下移动step单位，第18~19行侦测如果球撞到画布下方未来球移动是往上移动step单位（因为是负值）。

执行结果

32-6-3 设计让球在画布四面反弹

在弹球游戏中，我们必须让球在四面皆可反弹，这时需考虑到球在x轴移动，这时原先Ball类别的__init__()函数需修改下列2行。

```
11 self.x = 0 # 水平不移动
12 self.y = step # 垂直移动单位
```

下列是更改结果。

```
11 startPos = [-4, -3, -2, -1, 1, 2, 3, 4] # 球最初x轴位移的随机数
12 shuffle(startPos) # 打乱排列
13 self.x = startPos[0] # 球最初水平移动单位
14 self.y = step # 垂直移动单位
```

上述修改的观念是球局开始时，每个循环x轴的移动单位是随机数产生。至于在ballMove()方法中，我们需考虑到水平轴的移动可能碰撞画布左边与右边的状况，观念是如果球撞到画布左边，设定球未来x轴移动是正值，也就是往右移动。

```
18 if ballPos[0] <= 0: # 侦测球是否超过画布左方
19     self.x = step
```

如果球撞到画布右边，设定球未来x轴移动是负值，也就是往左移动。

```
22 if ballPos[2] >= winW: # 侦测球是否超过画布右方
23     self.x = -step
```

程序实例ch32_22.py： 改良ch32_21.py程序，现在球可以在四周移动。

```

6 class Ball:
7     def __init__(self, canvas, color, winW, winH):
8         self.canvas = canvas
9         self.id = canvas.create_oval(0, 0, 20, 20, fill=color) # 建立球对象
10        self.canvas.move(self.id, winW/2, winH/2) # 设定球最初位置
11        startPos = [-4, -3, -2, -1, 1, 2, 3, 4] # 球最初x轴位移的随机数
12        shuffle(startPos) # 打乱排列
13        self.x = startPos[0] # 球最初水平移动单位
14        self.y = step # 垂直移动单位
15    def ballMove(self):
16        self.canvas.move(self.id, self.x, self.y) # step是正值表示往下移动
17        ballPos = self.canvas.coords(self.id)
18        if ballPos[0] <= 0: # 侦测球是否超过画布左方
19            self.x = step
20        if ballPos[1] <= 0: # 侦测球是否超过画布上方
21            self.y = step
22        if ballPos[2] >= winW: # 侦测球是否超过画布右方
23            self.x = -step
24        if ballPos[3] >= winH: # 侦测球是否超过画布下方
25            self.y = -step

```

执行结果 读者可以观察屏幕，球在画布四周移动的结果。

32-6-4 建立球拍

首先我们先建立一个静止的球拍，此时可以建立Racket类别，在这个类别中我们设定了它的初始大小与位置。

程序实例ch32_23.py： 扩充ch32_22.py，主要是增加球拍设计，在这里我们先增加球拍类别。在这个类别中，我们在第29行设计了球拍的大小和颜色，第30行设定了最初球拍的位置。

```

26 class Racket:
27     def __init__(self, canvas, color):
28         self.canvas = canvas
29         self.id = canvas.create_rectangle(0,0,100,15, fill=color) # 球拍对象
30         self.canvas.move(self.id, 270, 400) # 球拍位置

```

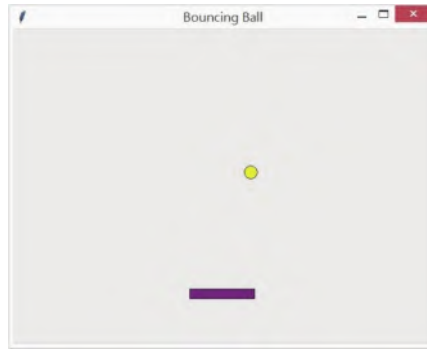
另外，在主程序增加了建立一个球拍对象。

```

44 racket = Racket(canvas, 'purple') # 定义紫色球拍

```

执行结果



32-6-5 设计球拍移动

由于是假设使用键盘的右移和左移键移动球拍，所以可以在Ractet的__init__()函数内增加，使用bind_all()方法绑定键盘按键发生时的移动方式。

```
32 self.canvas.bind_all('<KeyPress-Right>', self.moveRight) # 绑定按往右键
33 self.canvas.bind_all('<KeyPress-Left>', self.moveLeft) # 绑定按往左键
```

所以在Ractet类别内增加下列moveRight()和moveLeft()的设计。

```
41 def moveLeft(self, event): # 球拍每次向左移动的单位数
42     self.x = -3
43 def moveRight(self, event): # 球拍每次向右移动的单位数
44     self.x = 3
```

上述设计相当于每次的位移量是3，如果游戏有设等级，可以让新手位移量增加，随等级增加让位移量减少。此外这个程序增加了球拍移动主体设计如下：

```
34 def racketMove(self): # 设计球拍移动
35     self.canvas.move(self.id, self.x, 0)
36     pos = self.canvas.coords(self.id)
37     if pos[0] <= 0: # 移动时是否碰到画布左边
38         self.x = 0
39     elif pos[2] >= winW: # 移动时是否碰到画布右边
40         self.x = 0
```

主程序也将新增球拍移动调用。


```

61 while True:
62     ball.ballMove()
63     racket.racketMove()
64     tk.update()
65     time.sleep(speed) # 可以控制移动速度

```

程序实例ch32_24.py： 扩充ch32_23.py的功能，增加设计让球拍左右可以移动，下列程序第31行是设定程序开始时，球拍位移是0。下列是球拍类别内容。

```

26 class Racket:
27     def __init__(self, canvas, color):
28         self.canvas = canvas
29         self.id = canvas.create_rectangle(0,0,100,15, fill=color) # 球拍对象
30         self.canvas.move(self.id, 270, 400) # 球拍位置
31         self.x = 0
32         self.canvas.bind_all('<KeyPress-Right>', self.moveRight) # 绑定按往右键
33         self.canvas.bind_all('<KeyPress-Left>', self.moveLeft) # 绑定按往左键
34     def racketMove(self): # 设计球拍移动
35         self.canvas.move(self.id, self.x, 0)
36         pos = self.canvas.coords(self.id)
37         if pos[0] <= 0: # 移动时是否碰到画布左边
38             self.x = 0
39         elif pos[2] >= winW: # 移动时是否碰到画布右边
40             self.x = 0
41     def moveLeft(self, event): # 球拍每次向左移动的单位数
42         self.x = -3
43     def moveRight(self, event): # 球拍每次向右移动的单位数
44         self.x = 3

```

下列是主程序内容。

```

58 racket = Racket(canvas, 'purple') # 定义紫色球拍
59 ball = Ball(canvas, 'yellow', winW, winH) # 定义球对象
60
61 while True:
62     ball.ballMove()
63     racket.racketMove()
64     tk.update()
65     time.sleep(speed) # 可以控制移动速度

```

执行结果 读者可以观察屏幕，球拍已经可以左右移动。

32-6-6 球拍与球碰撞的处理

在上述程序的执行结果中，球碰到球拍基本上是可以穿透过去，这一节将讲解碰撞的处理，首先我们可以增加将Racket类别传给Ball类别，如下所示：

```

6 class Ball:
7     def __init__(self, canvas, color, winW, winH, racket):
8         self.canvas = canvas
9         self.racket = racket

```

当然在主程序建立Ball类别对象时需修改调用如下：

```
67 racket = Racket(canvas, 'purple') # 定义紫色球拍
68 ball = Ball(canvas, 'yellow', winW, winH, racket) # 定义球对象
```

在Ball类别需增加是否球碰到球拍的方法，如果碰到就让球沿路径往上反弹。

```
33         if self.hitRacket(ballPos) == True: # 侦测是否撞到球拍
34             self.y = -step
```

在Ball类别ballMove()方法上方需增加下列hitRacket()方法，检测球是否碰撞球拍，如果碰撞了会返回True，否则返回False。

```
16     def hitRacket(self, ballPos):
17         racketPos = self.canvas.coords(self.racket.id)
18         if ballPos[2] >= racketPos[0] and ballPos[0] <= racketPos[2]:
19             if ballPos[3] >= racketPos[1] and ballPos[3] <= racketPos[3]:
20                 return True
21         return False
```

上述侦测球是否撞到球拍必须判断2个条件：

1. 球的右侧x轴坐标ballPos[2]大于球拍左侧x坐标racketPos[0]，同时球的左侧x坐标ballPos[0]小于球拍右侧x坐标racketPos[2]。



2. 球的下方y坐标ballPos[3]大于球拍上方的y坐标racketPos[1]，同时必须小于球拍下方的y坐标reacketPos[3]。读者可能奇怪为何不是侦测碰到球拍上方即可，主要是球不是一次移动1像素，如果移动3像素，很可能会跳过球拍上方。



下列是球的可能移动方式图。



程序实例ch32_25.py： 扩充ch32_24.py，当球碰撞到球拍时会反弹，下列是完整的Ball类别设计。

```
6 class Ball:
7     def __init__(self, canvas, color, winW, winH, racket):
8         self.canvas = canvas
9         self.racket = racket
10        self.id = canvas.create_oval(0, 0, 20, 20, fill=color) # 建立球对象
11        self.canvas.move(self.id, winW/2, winH/2) # 设定球最初位置
12        startPos = [-4, -3, -2, -1, 1, 2, 3, 4] # 球最初x轴位移的随机数
13        shuffle(startPos) # 打乱序列
14        self.x = startPos[0] # 球最初水平移动单位
15        self.y = step # 垂直移动单位
16    def hitRacket(self, ballPos):
17        racketPos = self.canvas.coords(self.racket.id)
18        if ballPos[2] >= racketPos[0] and ballPos[0] <= racketPos[2]:
19            if ballPos[3] >= racketPos[1] and ballPos[3] <= racketPos[3]:
20                return True
21    def ballMove(self):
22        self.canvas.move(self.id, self.x, self.y) # step是正值表示往下移动
23        ballPos = self.canvas.coords(self.id)
24        if ballPos[0] <= 0: # 侦测球是否超过画布左方
25            self.x = step
26        if ballPos[1] <= 0: # 侦测球是否超过画布上方
27            self.y = step
28        if ballPos[2] >= winW: # 侦测球是否超过画布右方
29            self.x = -step
30        if ballPos[3] >= winH: # 侦测球是否超过画布下方
31            self.y = -step
32        if self.hitRacket(ballPos) == True: # 侦测是否碰到球拍
33            self.y = -step
34
```

执行结果 读者可以观察屏幕，球碰撞到球拍时会反弹。

32-6-7 完整的游戏

在实际的游戏中，若是球碰触画布底端应该让游戏结束，此时首先我们在第16行Ball类别的__init__()函数中先声明notTouchBottom为True，为了让玩家可以缓冲，笔者此时也设定球局开始时球是往上移动（第15行），如下所示：

```
15         self.y = -step # 球先往上垂直移动单位
16         self.notTouchBottom = True # 未接触画布底端
```

我们修改主程序的循环如下：

```
73 while ball.notTouchBottom: # 如果球未接触画布底端
74     ball.ballMove()
75     racket.racketMove()
76     tk.update()
77     time.sleep(speed) # 可以控制移动速度
```

最后我们在Ball类别的ballMove()方法中侦测球是否接触画布底端，如果是则将notTouchBottom设为False，这个False将让主程序的循环中止执行。同时如果关闭Bouncing Ball窗口，不再有错误信息产生了。

程序实例ch32_26.py: 完整的弹球设计。

```
1 # ch32_26.py
2 from tkinter import *
3 from random import *
4 import time
5
6 class Ball:
7     def __init__(self, canvas, color, winW, winH, racket):
8         self.canvas = canvas
9         self.racket = racket
10        self.id = canvas.create_oval(0, 0, 20, 20, fill=color) # 建立球对象
11        self.canvas.move(self.id, winW/2, winH/2) # 设定球最初位置
12        startPos = [-4, -3, -2, -1, 1, 2, 3, 4] # 球最初x轴位移的随机数
13        shuffle(startPos) # 打乱排列
14        self.x = startPos[0] # 球最初水平移动单位
15        self.y = -step # 球先往上垂直移动单位
16        self.notTouchBottom = True # 未接触画布底端
17    def hitRacket(self, ballPos):
18        racketPos = self.canvas.coords(self.racket.id)
19        if ballPos[2] >= racketPos[0] and ballPos[0] <= racketPos[2]:
20            if ballPos[3] >= racketPos[1] and ballPos[3] <= racketPos[3]:
21                return True
22        return False
23    def ballMove(self):
24        self.canvas.move(self.id, self.x, self.y) # step是正值表示往下移动
25        ballPos = self.canvas.coords(self.id)
26        if ballPos[0] <= 0: # 侦测球是否超过画布左方
27            self.x = step
28        if ballPos[1] <= 0: # 侦测球是否超过画布上方
29            self.y = step
30        if ballPos[2] >= winW: # 侦测球是否超过画布右方
31            self.x = -step
32        if ballPos[3] >= winH: # 侦测球是否超过画布下方
33            self.y = -step
34        if self.hitRacket(ballPos) == True: # 侦测是否撞到球拍
35            self.y = -step
36        if ballPos[3] >= winH: # 如果球接触到画布底端
37            self.notTouchBottom = False
38
39 class Racket:
40     def __init__(self, canvas, color):
41         self.canvas = canvas
42         self.id = canvas.create_rectangle(0,0,100,15, fill=color) # 球拍对象
43         self.canvas.move(self.id, 270, 400) # 球拍位置
44
45     self.x = 0
46     self.canvas.bind_all('<KeyPress-Right>', self.moveRight) # 绑定按往右键
47     self.canvas.bind_all('<KeyPress-Left>', self.moveLeft) # 绑定按往左键
48
49     def racketMove(self): # 设计球拍移动
50         self.canvas.move(self.id, self.x, 0)
51         racketPos = self.canvas.coords(self.id)
52         if racketPos[0] <= 0: # 移动时是否碰到画布左边
53             self.x = 0
54         elif racketPos[2] >= winW: # 移动时是否碰到画布右边
55             self.x = 0
56
57     def moveLeft(self, event): # 球拍每次向左移动的单位数
58         self.x = -3
59
60     def moveRight(self, event): # 球拍每次向右移动的单位数
61         self.x = 3
62
63 winW = 640 # 定义画布宽度
64 winH = 480 # 定义画布高度
65 step = 3 # 定义速度可想象成位移步伐
66 speed = 0.01 # 设定移动速度
67
68 tk = Tk()
69 tk.title("Bouncing Ball") # 游戏窗口标题
70 tk.wm_attributes('-topmost', 1) # 确保游戏窗口在屏幕最上层
71 canvas = Canvas(tk, width=winW, height=winH)
72 canvas.pack()
73 tk.update()
74
75 racket = Racket(canvas, 'purple') # 定义紫色球拍
76 ball = Ball(canvas, 'yellow', winW, winH, racket) # 定义球对象
77
78 while ball.notTouchBottom: # 如果球未接触画布底端
79     ball.ballMove()
80     racket.racketMove()
81     tk.update()
82     time.sleep(speed) # 可以控制移动速度
```

执行结果



习题

1. 重新设计程序实例ch32_17.py，输出字符串让玩家由屏幕输入猜哪一个球跑得快，每次皆让计算机有60%赢的机率。
2. 扩充功能，记录每次球拍碰触球的次数，显示在Python Shell窗口。
3. 扩充功能，游戏失败时询问是否再玩一次。
4. 扩充功能，当球拍碰触球次数超过100次时，球拍移动单位数改为2，同时球移动一次改为5。

第33章 声音的控制

这一章将讲解如何使用Python控制声音，将使用Pygame模块为例，其实Pygame模块的功能有许多，也可以使用此模块绘图或设计游戏，撰写本书除了想让各位学得Python的应用，另外也期待读者多认识不同的模块，所以笔者尽量用不同模块解说。

本章所使用的2个声音文件punch.wav和house_lo.mp3，皆是Pygame模块内附的示范文件，本书下载包没有附这2个文件，读者可以至下列文件夹复制至ch33文件夹就可以执行本章范例。注意，~是Python安装目录。

```
~python\python36-32\Lib\site-packages\pygame\examples\data
```

或是可以参考33-2节在文件夹内搜寻*.wav和*.mp3就可以找到这2个文件，另外网络也有许多免费声音文件可以下载使用，可以用下列关键词搜寻。

```
free wav file  
free mp3 file
```

33-1 安装与导入

使用这个模块前，读者需要使用下列语法安装此模块。

```
pip install pygame
```

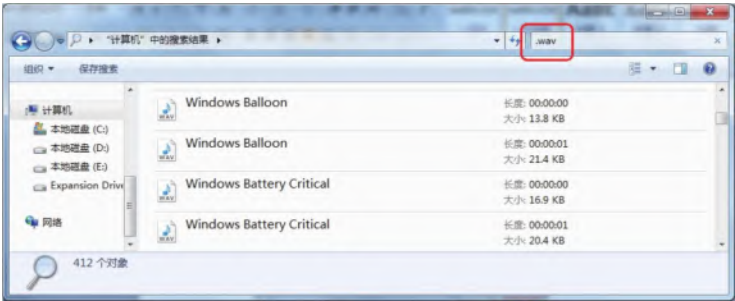
然后使用下列方式导入模块。

```
import pygame  
pygame.mixer.init( ) # 最初化
```


上述相当于最初化mixer对象，使用mixer对象可以执行2类声音的播放，一种是一般音效 ， 另一种是音乐文件 ， 下面将分别说明。

33-2 一般音效的播放Sound()

一般音效通常是指波形的声音文件，扩展名是.wav，在Windows操作系统内可以在搜寻 字段输入*.wav，就可以看到一系列计算机内有的波形声音文件。



我们可以使用Sound()方法先建立一般声音的Sound对象，然后用play()方法执行播放，下列是Sound对象常用的与声音播放有关的方法。

方法	说明
play(n)	n=-1 表示重复播放，0 表示播一次，1 表示 2 次，……
get_volumn()	取得目前播放音量
set_volumn(val)	设定目前播放音量，val 值的范围为 0.0 ~ 1.0
stop()	结束播放

程序实例ch33_1.py： 先播放一次punch.wav，经过3秒后播放3次。这个程序使用了pygame模块的time.delay()方法，参数3000代表3秒。

```
1 # ch33_1.py
2 import pygame
3 pygame.mixer.init()
4
5 soundObj = pygame.mixer.Sound('punch.wav') # 建立Sound对象
6 soundObj.play() # 播放一次
7 pygame.time.delay(3000) # 休息3秒
8 soundObj.play(2) # 播放3次
```

执行结果 请读者执行与测试。本书ch33文件夹没有附上punch.wav声音文件，请读者先搜寻此文件，再将此文件复制至计算机桌面，然后复制到ch33文件夹即可播放此声音。punch.wav是pygame模块的示范声音文件。

有时候声音在初始化时可能需一点时间，所以也可以使用time.delay(1000)延迟1秒，再执行程序。

程序实例ch33_2.py： 让初始化多一秒（第5行），然后再执行程序，休息3秒后将音量调低（第10行）。

```
1 # ch33_2.py
2 import pygame
3 pygame.mixer.init()
4
5 pygame.time.delay(1000)           # 先给声音初始化工作
6
7 soundObj = pygame.mixer.Sound('punch.wav') # 建立Sound对象
8 soundObj.play()                     # 按放一次
9 pygame.time.delay(3000)             # 休息3秒
10 soundObj.set_volume(0.1)           # 声音变小
11 soundObj.play(2)                   # 播放3次
```

执行结果 请读者执行与测试。

程序实例ch33_3.py： 将声音功能应用在ch32_26.py的游戏上，基本上当球撞到球拍时，就会产生音效。这个程序增加下列声明部分。

```
4 import pygame
```

球撞击球拍时产生声音。

```
35         if self.hitRacket(ballPos) == True: # 侦测是否撞到球拍
36             soundObj = pygame.mixer.Sound('punch.wav') # 建立声音对象
37             soundObj.play() # 发出声音
38             self.y = -step
61 pygame.mixer.init() # 初始化声音
```

执行结果 请读者执行与测试。

33-3 播放音乐文件music()

music()除了可以播放wav声音文件外，也可以播放MP3的音乐文件或是以ogg为扩展名的声音文件。

我们可以使用music()方法的load()方法下载音乐，然后用play()方法执行播放，下列是music对象常用的与音乐播放有关的方法。

方法	说明
load(音乐文件)	下载音乐文件
play(n)	n=-1 表示重复播放，0 表示播一次，1 表示 2 次，……
pause()	暂停播放
unpause()	恢复播放
get_busy()	是否播放中，是则传回 True，否则传回 False
set_volumn(val)	设定目前播放音量，val 值的范围为 0.0 ~ 1.0
stop()	结束播放

程序实例ch33_4.py： 播放mp3音乐文件。

```
1 # ch33_4.py
2 import pygame
3 pygame.mixer.init()
4
5 pygame.time.delay(1000)           # 先给声音初始化工作
6 pygame.mixer.music.load('house_lo.mp3') # 下载mp3音乐文件
7 pygame.mixer.music.play()          # 播放mp3音乐文件
```

执行结果 请读者执行与测试。本书ch33文件夹没有附上house_lo.mp3声音文件，请读者先搜寻此文件，再将此文件复制至计算机桌面，然后复制到ch33文件夹即可播放此声音。house_lo.mp3是pygame模块的示范音乐文件。

程序实例ch33_5.py： 这是一个音乐切换程序设计，首先会设定循环永远播放house_lo.mp3音乐文件（第7行），第8行是处理播放3秒，然后第9行询问是否在播放中，如果是第10行先暂停播放，第11行暂停播放3秒，第12~13行播放声音文件punch.wav，第14行暂停播放3秒，第15行恢复播放house_lo.mp3。

```

1 # ch33_5.py
2 import pygame
3 pygame.mixer.init()
4
5 pygame.time.delay(1000) # 先给声音初始化工作
6 pygame.mixer.music.load('house_lo.mp3') # 下载mp3音乐文件
7 pygame.mixer.music.play(-1) # 永远播放mp3音乐文件
8 pygame.time.delay(3000) # 暂停3秒,mp3音乐继续播放
9 if pygame.mixer.music.get_busy():
10     pygame.mixer.music.pause() # 暂停播放
11     pygame.time.delay(3000) # 暂停3秒
12     soundObj = pygame.mixer.Sound('punch.wav') # 建立Sound对象
13     soundObj.play() # 播放Sound对象
14     pygame.time.delay(3000) # 暂停3秒
15     pygame.mixer.music.unpause() # 恢复播放

```

执行结果 请读者执行与测试，当关闭程序时此音乐才会停止。

33-4 背景音乐

如果读者仔细观察可以发现在播放音乐时不会干扰程序的进行，其实我们可以将这个特性应用在设计游戏时，当作背景音乐。然后需要特殊效果的音乐时，可以将背景音乐先暂停(pause)，播放完特殊效果音乐时，再恢复(unpause)播放背景音乐即可。

程序实例ch33_6.py： 扩充ch33_3.py，游戏开始时或进行中将持续播放背景音乐house_lo.mp3，但是当球碰撞到球拍时，会暂停背景音乐改播punch.wav声音文件，声音文件播放结束后，会恢复播放背景音乐。下列是在Ball类别的__init__函数增加的内容。

```

18 pygame.mixer.music.load('house_lo.mp3') # 下载mp3音乐文件
19 pygame.mixer.music.play(-1) # 永远播放mp3音乐文件

```

下列是侦测球是否碰到球拍，响应True时的设计内容。

```

37 if self.hitRacket(ballPos) == True: # 侦测是否碰到球拍
38     pygame.mixer.music.pause() # 暂停播放背景音乐
39     soundObj = pygame.mixer.Sound('punch.wav') # 建立碰撞声音对象
40     soundObj.play() # 发出碰撞声音
41     pygame.mixer.music.unpause() # 恢复播放背景音乐
42     self.y = -step

```

执行结果 请读者执行与测试，当关闭程序时此音乐才会停止。

33-5 mp3音乐播放器

这一节将介绍一个简单的mp3音乐播放器的制作，在这个音乐播放器中笔者的选单列表有3首mp3，但是笔者在计算机内只找到2首mp3文件，所以程序第14行重复使用house_lo.mp3文件，读者可以至网络上搜寻免费mp3文件取代此音乐。第11行的NotifyPopup.mp3是Windows操作系统内附的mp3文件。

程序实例ch33_7.py： 建立一个mp3播放器，本程序执行时默认音乐选单是第一首歌，可以用选项按钮更改所选的音乐，按**播放** 按钮可以循环播放，按**结束** 按钮可以停止播放。

```
1 # ch33_7.py
2 from tkinter import *
3 import pygame
4
5 def playmusic():
6     selection = var.get()
7     if selection == '1':
8         pygame.mixer.music.load('house_lo.mp3')
9         pygame.mixer.music.play(-1)
10    if selection == '2':
11        pygame.mixer.music.load('NotifyPopup.mp3')
12        pygame.mixer.music.play(-1)
13    if selection == '3':
14        pygame.mixer.music.load('house_lo.mp3')
15        pygame.mixer.music.play(-1)
16 def stopmusic():
17     pygame.mixer.music.stop()
18
19 # 建立mp3音乐选项按钮内容的串行
20 musics = [('house_lo.mp3', 1),
21           ('NofityPopup.mp3', 2),
22           ('happy.mp3', 3)]
23
24 pygame.mixer.init()
25
26 tk = Tk()
27 tk.geometry('480x220')
28 tk.title('Mp3 Player')
29 mp3label = Label(tk, text='\n我的Mp3 拨放程序')
30 mp3label.pack()
31 # 建立选项按钮Radio button
32 var = StringVar()
33 var.set('1')
34 for music, num in musics:
35     radioB = Radiobutton(tk, text=music, variable=var, value=num)
36     radioB.pack()
37 # 建立按钮Button
38 button1 = Button(tk, text='播放', width=10, command=playmusic)
39 button1.pack()
40 button2 = Button(tk, text='结束', width=10, command=stopmusic)
41 button2.pack()
42 mainloop()
```

执行结果



这个程序几个重要观念如下：

①程序第27行`geometry()`方法，是另一种使用tkinter模块建立窗口的方式。

②第29和30行在窗口内使用`Label()`建立标题(label)，同时安置(pack)。有的程序设计师喜欢在`pack()`方法内加上`anchor=W`表示安置时锚点靠左对齐。

③第32～36行是建立选项按钮，这些相同系列的选项按钮必须使用相同的变量`variable`，至于选项值则由`value`设定。

④第32行表面意义是设定字符串对象，真实内涵是设定选单用字符串表示，如果想用整数可以将`StringVar()`改成`IntVar()`。

⑤第33行`set()`是设定默认选项是1。

⑥第34～36行循环主要是使用`Radiobutton()`方法建立音乐选项按钮，音乐选单的来源是第20～22行的列表，此列表元素是元组(tuple)，相当于将元组的第一个元素以`music`变量放入`text`，第二个元素以`num`变量放入`value`。

⑦第38行当按播放按钮时执行`playmusic()`方法。

⑧第5～15行是`playmusic()`播放方法，最重要是第7行`get()`方法，可以获得目前选项按钮的选项，然后可以根据选项播放音乐。

⑨第40行是当按播放按钮时执行`stopmusic()`方法。

⑩第16、17行是`stopmusic()`方法，主要是停止播放mp3音乐。

习题

1. 扩充设计`ch33_6.py`：请选择更能代表这个游戏的音效当背景音乐和碰撞音乐。

2. 扩充上一个习题，当球碰撞球拍5次以上时，请选用一个更刺激的碰撞音效。
3. 请扩充mp3音乐为5首歌。
4. 请扩充上一个习题，增加暂停、恢复按钮、放大声、放小声按钮。

第34章 人脸识别系统设计

人脸识别是一个非常复杂的学问，所考虑的包含CPU的密集运算、3D显示和光线追踪。

以个人能力要完成上述工作非常困难，1999年美国Intel公司主导开发了OpenCV (Open Source Computer Vision Library) 计划，这是一个跨平台的计算机视觉数据库，可以将它应用在人脸识别、人机互动、机器人视觉、动作识别等，本章的重点则是使用OpenCV将它应用在人脸识别系统设计。2000年这个版本的第一个预览版本在IEEE on Computer Vision and Pattern Recognition公开，经过5个测试版本后，2006年OpenCV 1.0版正式上市，2009年10月OpenCV 2.0版上市，2015年6月OpenCV 3.0版上市。从2012年起，OpenCV的非营利组织成立 (OpenCV.org)，目前由这个组织协助支持与维护同时授权可以免费在教育研究和商业上使用。

34-1 安装OpenCV

一般我们安装OpenCV时，会同时安装Numpy，因为有些人脸识别的运算需要使用Numpy的数学函数库的数据类型。

34-1-1 安装OpenCV

首先请至下列网站，下载一个whl文件：

<http://www.lfd.uci.edu/~gohlke/pythonlibs/#opencv>

OpenCV, a real time computer vision library.

- [opencv_python-2.4.13.2-cp27-cp27m-win32.whl](#)
- [opencv_python-2.4.13.2-cp27-cp27m-win_amd64.whl](#)
- [opencv_python-3.1.0-cp27-cp27m-win32.whl](#)
- [opencv_python-3.1.0-cp27-cp27m-win_amd64.whl](#)
- [opencv_python-3.1.0-cp34-cp34m-win32.whl](#)
- [opencv_python-3.1.0-cp34-cp34m-win_amd64.whl](#)
- [opencv_python-3.3.1+contrib-cp35-cp35m-win32.whl](#)
- [opencv_python-3.3.1+contrib-cp35-cp35m-win_amd64.whl](#)
- [opencv_python-3.3.1+contrib-cp36-cp36m-win32.whl](#)
- [opencv_python-3.3.1+contrib-cp36-cp36m-win_amd64.whl](#)
- [opencv_python-3.3.1-cp35-cp35m-win32.whl](#)
- [opencv_python-3.3.1-cp35-cp35m-win_amd64.whl](#)
- [opencv_python-3.3.1-cp36-cp36m-win32.whl](#)
- [opencv_python-3.3.1-cp36-cp36m-win_amd64.whl](#)

将上述文件下载后，可以存入任意文件夹内，笔者将它存入C:\opencvy文件夹。接着请进入此文件夹，然后使用下列方式安装。

```
C:\opencv>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\Scripts\pip install opencv_python-3.3.1-cp36-cp36m-win32.whl
Processing c:\opencv\opencv_python-3.3.1-cp36-cp36m-win32.whl
Installing collected packages: opencv-python
Successfully installed opencv-python-3.3.1

C:\opencv>
```

这时如果在Python Shell窗口输入import cv2，没有错误信息就代表安装成功了。

```
>>> import cv2
>>>
```

34-1-2 安装Numpy

这个安装相对单纯，可以直接使用pip install numpy安装。

34-2 读取和显示图像

34-2-1 建立OpenCV图像窗口

可以使用namedWindow()建立未来要显示图像的窗口，它的语法如下：

```
cv2.namedWindow(窗口名称[, 窗口旗标参数])
```

窗口旗标参数flag 可能值如下：

WINDOW_NORMAL：如果设定，用户可以自行调整窗口大小。

WINDOW_AUTOSIZE：系统将依图像调整窗口大小，用户无法调整窗口大小，这是预设。

WINDOW_OPENGL：将以OpenGL支持方式打开窗口。

实例： 可以使用cv2.namedWindow(“Face”)建立标题为Face的窗口。

34-2-2 读取图像

可以使用cv2.imread()读取图像，读完后将图像放在图像对象内，OpenCV支持大部分图像格式，例如，*.jpg、*.jpeg、*.png、*.bmp、*.tiff等。

```
image  
  
= cv2.imread(图像文件, 图像旗标  
) # image
```

是图像对象可以自行命名

图像旗标参数的可能值如下：

cv2.IMREAD_COLOR：这是默认，以彩色图像读取，值是1。

cv2.IMREAD_GRAYSCALE：以灰色图像读取，值是0。

cv2.IMREAD_UNCHANGED：以彩色读取包含alpha值的图像，值是-1。

实例： 下列分别以彩色和黑白读取图像picture.jpg。

```
img = cv2.imread('picture.jpg', 1) # 彩色图像
```


读取

```
img = cv2.imread('picture.jpg', 0) # 灰色图像
```

读取

34-2-3 使用OpenCV窗口显示图像

可以使用`cv2.imshow()`将前一节读取的图像对象显示在OpenCV窗口内，此方法的使用格式如下：

```
cv2.imshow(窗口名称, 图像对象)
```

34-2-4 关闭OpenCV窗口

将图像显示在OpenCV窗口后，若是想删除窗口可以使用下列方法。

```
cv2.destroyAllWindows(窗口名称  
) # 删除单一所指定
```

的窗口

```
cv2.destroyAllWindows() # 删除所有OpenCV的图像窗口
```

34-2-5 时间等待

可以使用`cv2.waitKey(n)`运行时间等待，`n`单位是毫秒，若是`n=0`，代表无限期等待。若是设为`cv2.waitKey(1000)`相当于`time.sleep(1)`，有等待1秒的效果。其实这是一个键盘绑定函数，在34-4-4小节将会做另一种应用的解说。

程序实例ch34_1.py：以彩色和黑白显示图像的应用，其中彩色的OpenCV窗口无法调整窗口大小，黑白的OpenCV窗口则可以调整窗口大小。

```

1 # ch34_1.py
2 import cv2
3 cv2.namedWindow("MyPicture1")
4 cv2.namedWindow("MyPicture2", cv2.WINDOW_NORMAL) # 使用预设
5 img1 = cv2.imread("jk.jpg") # 彩色读取
6 img2 = cv2.imread("jk.jpg", 0) # 灰色读取
7 cv2.imshow("MyPicture1", img1) # 显示img1
8 cv2.imshow("MyPicture2", img2) # 显示img2
9 cv2.waitKey(3000) # 等待3秒
10 cv2.destroyWindow("MyPicture1") # 删除MyPicture1
11 cv2.waitKey(3000) # 等待3秒
12 cv2.destroyAllWindows() # 删除所有窗口

```

执行结果 下列右边窗口可以重设大小。



34-2-6 存储图像

可以使用`cv2.imwrite()`存储图像，它的使用格式如下：

`cv2.imwrite(文件路径, 图像对象)`

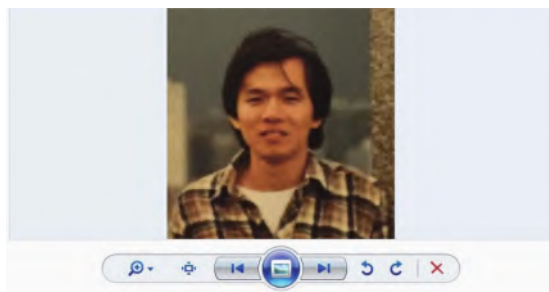
程序实例ch34_2.py： 打开图像，使用OpenCV窗口存储，然后存入out34_2.jpg。

```

1 # ch34_2.py
2 import cv2
3 cv2.namedWindow("MyPicture")
4 img = cv2.imread("jk.jpg")
5 cv2.imshow("MyPicture", img)
6 cv2.imwrite("out34_2.jpg", img) # 将文件写入out34_2.jpg
7 cv2.waitKey(3000) # 等待3秒
8 cv2.destroyAllWindows() # 删除所有窗口

```

执行结果 可以在ch34文件夹看到下列out34_2.jpg图像。



34-3 OpenCV的绘图功能

OpenCV也像大多数的图像模块一样可以执行绘图，当然这不是学习OpenCV的目的，因为有其他好用的绘图模块可以使用。

直线

`cv2.line`(绘图对象, (x1, y1), (x2, y2), 颜色, 宽度)

绘图对象可想成是画布，(x1, y1)是线条的起点，(x2, y2)是线条的终点，画布左上角是(0, 0)，往右x轴增加，往下y轴增加，单位为像素。颜色是3个RGB值(Blue, Green, Red)，介于0~255间，预设是黑色。线条宽度预设是1。

实例： 下列是从x1=50, y1=100, 绘一条线至x2=300, y2=350，蓝色，线宽是2。

```
cv2.line(img, (50,100), (300,350), (255,0,0), 2)
```

矩形

`cv2.rectangle`(绘图对象, (x1, y1), (x2, y2), 颜色, 宽度)

(x1, y1)是矩形左上角坐标，(x2, y2)是矩形右下角坐标，颜色使用与线条相同，线宽是矩形宽，如果线宽是负值代表实心矩形。

实例： 下列是建立一个绿色线条，宽度是3，左上角是x1=50, y1=100，右下角是x2=300, y2=350的矩形。

```
cv2.rectangle(img, (50,100), (300,350), (0,255,0), 3)
```

圆形

```
cv2.circle(绘图对象, (x,y), radius, 颜色, 宽度)
```

(x,y)是圆中心，radius是圆半径。

实例： 下列是在(100,100)为圆中心，绘半径50，红色的圆，宽度为1。

```
cv2.circle(img, (100,100), 50, (0,0,255), 2)
```

输出文字

```
cv2.putText(绘图对象, 文字, 位置
```

, 字体

, 字体大小, 颜色, 文字宽度)

其中字体 格式有下列选项：

FONT_HERSHEY_SIMPLEX: sans-serif字体正常大小。

FONT_HERSHEY_PLAIN: sans-serif字体较小字体。

FONT_HERSHEY_COMPLEX: serif字体正常大小。

FONT_ITALIC: italic字体。

上述位置 是指第一个字的左下角坐标。

程序实例ch34_3.py： 在绘图对象输出线条、矩形与文字的应用。

```
1 # ch34_3.py
2 import cv2
3 cv2.namedWindow("MyPicture")
4 img = cv2.imread("antarctica3.jpg")
5 cv2.line(img, (100,100), (1200,100), (255,0,0), 2)
6 cv2.rectangle(img, (100,200), (1200,400), (0,0,255), 2)
7 cv2.putText(img, "I Like Python", (400,350),
8             cv2.FONT_ITALIC, 3, (255,0,0), 8)
9 cv2.imshow("MyPicture", img)
10 cv2.waitKey(3000)
11 cv2.destroyAllWindows()
```

```
# 使用预设
# 彩色读取
# 输出线条
# 输出矩阵
# 输出文字
# 显示img
# 等待3秒
# 删除所有窗口
```

执行结果



34-4 人脸识别

人脸识别是计算机技术的一种，这个技术可以测出人脸在图像中的位置，同时也可以找出多个人脸，在检测过程中基本上会忽略背景或其他物体，例如，身体、建筑物或树木等。当然在检测过程中，很重要的是与图像数据库互相匹配比对，所用的技术是哈尔(Harr)特征，OpenCV已经将许多已经训练测试过的面部、表情、笑脸等特征分类文件存储在`~\opencv\sources\data\haarcascades`文件夹内。

haarcascade_eye	2017/1/31 下午 1...	XML Document	334 KB
haarcascade_eye_tree_eyeglasses	2017/1/31 下午 1...	XML Document	588 KB
haarcascade_frontalcatface	2017/6/30 上午 0...	XML Document	402 KB
haarcascade_frontalcatface_extended	2017/6/30 上午 0...	XML Document	374 KB
haarcascade_frontalface_alt	2017/1/31 下午 1...	XML Document	661 KB
haarcascade_frontalface_alt_tree	2017/1/31 下午 1...	XML Document	2,627 KB
haarcascade_frontalface_alt2	2017/1/31 下午 1...	XML Document	528 KB
haarcascade_frontalface_default	2017/1/31 下午 1...	XML Document	909 KB
haarcascade_fullbody	2017/1/31 下午 1...	XML Document	466 KB
haarcascade_lefteye_2splits	2017/1/31 下午 1...	XML Document	191 KB
haarcascade_licence_plate_rus_16st...	2017/1/31 下午 1...	XML Document	47 KB
haarcascade_lowerbody	2017/1/31 下午 1...	XML Document	387 KB
haarcascade_profileface	2017/1/31 下午 1...	XML Document	810 KB
haarcascade_righteye_2splits	2017/1/31 下午 1...	XML Document	192 KB
haarcascade_russian_plate_number	2017/1/31 下午 1...	XML Document	74 KB
haarcascade_smile	2017/6/30 上午 0...	XML Document	185 KB
haarcascade_upperbody	2017/1/31 下午 1...	XML Document	768 KB

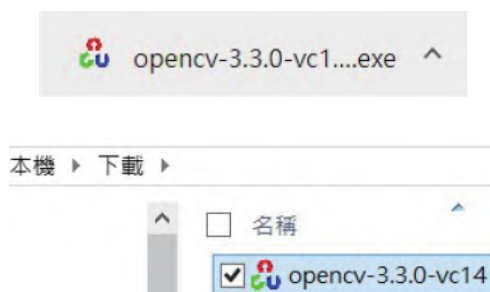
未来我们可以加载上述文件，再利用OpenCV所提供的API应用方法，即可执行人脸检测识别。

34-4-1 下载人脸识别特征文件

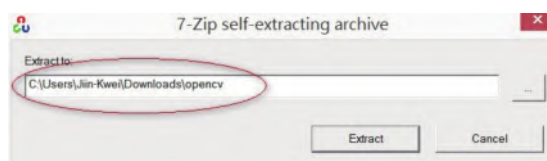
在34-1-1小节安装OpenCV时，并没有下载人脸识别特征文件，可以进入下列网址下载这些文件。



下载完成后可以在窗口下方或硬盘C:的下载区看到opencv-3.3.0-vc14文件。



连点2下可以执行与解压缩，将看到下列画面，笔者设定下载区的opencv文件夹。



然后按Extract 按钮就可以执行解压缩，最后笔者将上述所有解压缩的文件，复制到34-1-1小节的C:\opencv文件夹，这样就大功告成了。

34-4-2 脸部识别

设计人脸识别系统第一步是可以让程序使用OpenCV将图像文件的人脸标记出来，下列是常用的人脸识别特征文件，我们可以使用CascadeClassifier()类别执行脸部识别。

```
face_cascade
=
cv2.CascadeClassifier('~haarcascade_frontalface_default.xml')
```

~ 是指文件路径，face_cascade 是识别对象，当然你可以自行取名。接着需要使用识别对象启动detectMultiScale()方法，语法如下：

```
faces = face_cascade
.detectMultiScale(img, 参数1, 参数2, ...)
```

上述参数意义如下：

scaleFactor：如果没有指定一般是1.1，主要是指在特征比对中，图像比例的缩小倍数。

minNeighbors：每个区块的特征皆会比对，设定多少个特征数达到才算比对成功，默认值是3。

minSize：最小识别区块。

maxSize：最大识别区块。

笔者研究许多文件发现，最常见的是设定前3个参数，例如，下列表示图像对象是img，scaleFactor是1.3，minNeighbors是5。

```
faces = face_cascade.detectMultiScale(img, 1.3, 5)
```

上述执行成功后的返回值是faces列表，列表的元素是元组(tuple)，每个元组内有4组数字分别代表脸部左上角的x轴坐标、y轴坐标、脸部的宽w和脸部的高h。有了这些数据就可以在图像中标出人脸，或是将人脸存储。我们可以用len(faces)获得找到几张脸。

程序实例ch34_4.py： 使用第4行所载明的人脸特征文件，标示图像中的人脸，并用蓝色框框着人脸，以及在图像右下方标注所发现的人脸数量。下列程序可以应用在发现很多人脸的场所，主要是程序第17和18行，笔者将返回的列表（元素是元组），依次绘制矩形将脸部框起。

```
1 # ch34_4.py
2 import cv2
3
4 pictPath = r'C:\opencv\sources\data\haarcascades\haarcascade_frontalface_default.xml'
5 face_cascade = cv2.CascadeClassifier(pictPath) # 建立识别文件对象
6 img = cv2.imread("jk.jpg") # 读取图像文件建立图像文件对象
7 faces = face_cascade.detectMultiScale(img, scaleFactor=1.1,
8 minNeighbors = 3, minSize=(20,20))
9 # 标注右下角底色是黄色
10 cv2.rectangle(img, (img.shape[1]-140, img.shape[0]-20),
11 (img.shape[1],img.shape[0]), (0,255,255), -1)
12 # 标注找到多少的人脸
13 cv2.putText(img, "Finding " + str(len(faces)) + " face",
14 (img.shape[1]-135, img.shape[0]-5),
15 cv2.FONT_HERSHEY_COMPLEX, 0.5, (255,0,0), 1)
16 # 将人脸框起来，由于有可能找到好几个脸所以用循环绘出来
17 for (x,y,w,h) in faces:
18     cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2) # 蓝色框住人脸
19 cv2.namedWindow("Face", cv2.WINDOW_NORMAL) # 建立图像对象
20 cv2.imshow("Face", img) # 显示图像
```

执行结果



上述右边是程序实例ch34_5.py，第6行使用g2.jpg的执行结果。下列是程序实例ch34_6.py第6行使用g5.jpg更多人脸的识别结果。



当然使用上偶尔也会出现识别错误的情况，读者可以自行体会。

34-4-3 将脸部存档

我们已经成功识别脸部了，下一步是将脸部存储，就像我们进入海关，要享受便利的人脸识别通关，首先海关人员会先为我们拍照，然后将我们的脸形存档，未来我们每次出入海关都会拍照，主要是将我们的脸形与计算机所存的脸形文件进行比对。

要完成本节工作，我们可以使用27章的Pillow模块，这个模块有下列方法可以使用。

使用Image.open()打开文件，可参考27-3-1小节。

使用crop()依据人脸识别矩形框裁切图片，可参考27-5-1小节。

使用resize()更改图像大小，可参考27-4-1小节。

使用save()存储图像，可参考27-3-5小节。

程序实例ch34_7.py： 扩充ch34_6.py，将所识别的人脸分别存入face1.jpg，…，face5.jpg。

```

17 # 将人脸框起来，由于有可能找到好几个脸所以用循环绘出来
18 num = 1
19 for (x,y,w,h) in faces:
20     cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
21     filename = "face" + str(num) + ".jpg"
22     image = Image.open("g5.jpg")
23     imageCrop = image.crop((x, y, x+w, y+h))
24     imageResize = imageCrop.resize((150,150),Image.ANTIALIAS)
25     imageResize.save(filename)
26     num += 1

```

文件名编号
蓝色框住人脸
建立文件名
PIL模块开启
裁切
高质量重制大小
存储大小
文件编号

执行结果 重点是在ch34文件夹由左到右有face1.jpg，…，face5.jpg文件。



34-4-4 读取摄像头所拍的画面

OpenCV有提供功能可以让我们读取一般影片，也可以读取摄像头所拍画面，当然可以提取所拍画面的脸形。控制摄像头的语法如下：

```
cap
= VideoCapture(n
) # 笔记本电脑上内置摄像头，n
是0
```

上述cap 是摄像头对象，可自行取名。可以由cap.isOpened() 判断摄像头是否打开，如果打开则返回True，否则返回False。当摄像头打开时，可以使用下列方法读取摄像头所拍的图像。

```
ret, img = cap.read( )
```

ret是布尔值，如果是True则表示拍摄成功，如果是False则表示拍摄失败。img是摄像头所拍的图像对象。拍摄结束可以使用cap.release()关闭摄像头。

34-2-5小节有介绍cv2.waitKey()，这个方法除了可以作为一般等待，也可以等待用户的按键，如下所示：

```
key = cv2.waitKey(n) # n是等待时间，key是用户的按键
```

当用户有按键发生时所按的键会传给key，这个key是一个ASCII码值。

程序实例ch34_8.py： 将摄像头所拍摄的图像存储至photo.jpg，可参考第8~11行，同时将这个图像做识别处理，框出脸形，同时将所框的脸形存入faceout.jpg。

```
1 # ch34_8.py
2 import cv2
3 from PIL import Image
4
5 pictPath = r'C:\opencv\sources\data\haarcascades\haarcascade_frontalface_default.xml'
6 face_cascade = cv2.CascadeClassifier(pictPath) # 建立识别文件对象
7 cv2.namedWindow("Photo")
8 cap = cv2.VideoCapture(0) # 开启摄像头
9 while(cap.isOpened()): # 如果摄像头开启就执行循环
10     ret, img = cap.read() # 读取图像
11     cv2.imshow("Photo", img) # 显示图像在OpenCV窗口
12     if ret == True: # 如果读取图像成功
13         key = cv2.waitKey(200) # 0.2秒检查一次
14         if key == ord("a") or key == ord("A"): # 如果按A或a
15             cv2.imwrite("photo.jpg", img) # 将图像写入photo.jpg
16             break
17 cap.release() # 关闭摄像头
18
19 faces = face_cascade.detectMultiScale(img, scaleFactor=1.1,
20 minNeighbors = 3, minSize=(20,20))
21 # 标注右下角底色是黄色
22 cv2.rectangle(img, (img.shape[1]-120, img.shape[0]-20),
23 (img.shape[1],img.shape[0]), (0,255,255), -1)
24 # 标注找到多少的人脸
25 cv2.putText(img, "Find " + str(len(faces)) + " face",
26 (img.shape[1]-110, img.shape[0]-5),
27 cv2.FONT_HERSHEY_COMPLEX, 0.5, (255,0,0), 1)
28 # 将人脸框起来
29 for (x,y,w,h) in faces:
30     cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2) # 蓝色框住人脸
31     myimg = Image.open("photo.jpg") # PIL模块开启
32     imgCrop = myimg.crop((x, y, x+w, y+h)) # 裁切
33     imgResize = imgCrop.resize((150,150), Image.ANTIALIAS)
34     imgResize.save("faceout.jpg") # 存储文件
35
36 cv2.namedWindow("FaceRecognition", cv2.WINDOW_NORMAL)
37 cv2.imshow("FaceRecognition", img)
```

执行结果 下方右图是faceout.jpg的输出。



如果得到上述结果，恭喜成功，因为若和机场的人脸识别系统相比较，目前只剩比对数据库的脸形了。

34-4-5 脸形比对

其实脸形比对的算法也是相对复杂，对我们而言只要使用前人所开法的算法即可。此节笔者使用的是`histogram()`方法，它的基本观念是取出2个脸形的颜色(RGB)分布的直方图，对2个颜色做RMS(`root-mean-square`)，如果2个图一样所得的RMS为0，RMS结果值越大代表图差异越大。

程序实例ch34_9.py： 计算2张相同图的RMS值，这个程序需要导入许多模块。

```
1 # ch34_9.py
2 from functools import reduce
3 from PIL import Image
4 import math, operator
5 h1 = Image.open("face1.jpg").histogram()
6 h2 = Image.open("face1.jpg").histogram()
7 RMS = math.sqrt(reduce(operator.add, list(map(lambda a,b:
8 (a-b)**2, h1, h2)))/len(h1))
9 print("RMS = ", RMS)
```

执行结果

```
===== RESTART: D:/Python/ch34/ch34_9.py =====
RMS = 0.0
>>>
```

程序实例ch34_10.py： 比较ch34文件夹的`face1.jpg`(这是笔者2017年9月28日拍的照片)和`faceout.jpg`(这是2017年11月29日拍的照片)的结果。

```
5 h1 = Image.open("face1.jpg").histogram()
6 h2 = Image.open("faceout.jpg").histogram()
```

执行结果

```
===== RESTART: D:/Python/ch34/ch34_10.py =====
RMS = 67.85402914222068
>>>
```

2张脸形比对的结果是67.8x，其实这在识别领域可以归做同一个脸形了，一般若是所得的结果是在100左右算是临界值，读者可以自行测试，这是一个有趣的应用。了解了以上观念，相信读者也可以设计机场的脸形通关系统了。

34-5 设计桃园国际机场的出入境人脸识别系统

方式与观念如下：

①填写个人资料，拍照建立个人脸形，读者可以使用身份证号码当作个人的脸形文件。所以只要在执行ch34_8.py前增加输入个人身份证号码就可以了，可以将这个程序称为faceSave.py。下列是增加以及修改的内容。

```
5 ID = input("请输入身份证号码 = ")          # 读取所输入的身份证号码
6 print("脸形文件将储存在 ", ID + ".jpg")      # 未来的脸形文件
7 faceFile = ID + ".jpg"
```

下列是将脸形文件存储。

```
38 imgResize.save(faceFile)                    # 存储文件
```

执行结果 下列是程序执行时Python Shell窗口的画面。



```
===== RESTART: D:\Python\ch34\faceSave.py =====
请输入身份证号码 = J111111111
脸形文件将储存在 J111111111.jpg
>>>
```

②未来每次出入海关，皆会先扫描护照，主要目的是先将个人的图片文件调出来当作比对依据，我们暂时没有这个设备，可以要求用户屏幕输入身份证号码，有了身份证号码就可以将数据库的个人脸形图库叫出。然后使用ch34_8.py程序拍照存盘，再利用ch34_10.py将现在所拍的脸形和原先数据库的脸形做比对就可以了，如果比对结果的RMS值小于100则比对成功，否则比对失败，可以将这个程序称为faceCheck.py。

```
4 from functools import reduce
5 import math, operator
6
7 ID = input("请输入身份证号码 = ")          # 读取所输入的身份证号码
8 face = ID + ".jpg"                        # 未来的脸形文件
```

下列第39行是将脸形文件存储，第41-44行是执行比对。

```

39     imgResize.save("newface.jpg")                # 存储文件
40
41     h1 = Image.open(face).histogram()
42     h2 = Image.open("newface.jpg").histogram()
43     RMS = math.sqrt(reduce(operator.add, list(map(lambda a,b:
44         (a-b)**2, h1, h2)))/len(h1))
45     if RMS <= 100:
46         print("欢迎出入境")
47     else:
48         print("比对失败")

```

执行结果 下列是程序执行时Python Shell窗口的画面。

```

===== RESTART: D:\Python\ch34\faceSave.py =====
请输入身份证号码 = J1111111111
欢迎出入境
>>>

```

习题

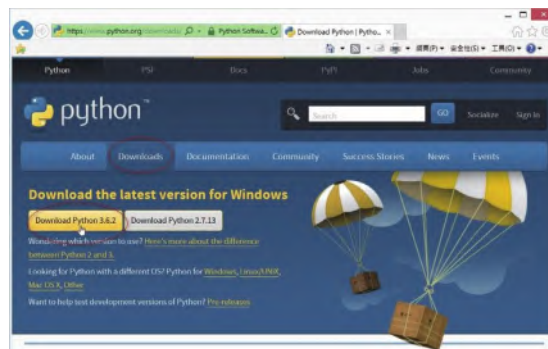
1. 为程序实例ch33_3.py建立自己的脸形，每次程序启动要做人脸识别，如果通过则可以正式玩这个游戏，没有通过则不能玩这个游戏，把RMS值小于100当作通过标准。
2. 请在桃园机场脸形比对前增加检查脸形文件是否存在。
3. 将faceSave.py和faceCheck.py写成一个文件，请自行美化设计。

附录A 安装Python

Python安装程序在安装前会先侦测你的计算机使用环境，然后自动协助选择安装程序。请先进入下列网页：

www.python.org

然后选择Downloads标签，接着可以看到2个按钮，笔者撰写此书时分别是下载3.6.2版与下载2.7.12版的按钮。



A-1 Windows操作系统的安装Python版

此时读者可以选择下载哪一个版本，此例笔者选择下载3.6.2版。

未来可以单击，就可以直接安装Python 3.6.2，然后将看到下列安装画面：



注 如果单击Add Python 3.6 to PATH，不论是在哪一个文件夹都可以执行python可执行文件，非常方便。上述画面是未勾选状态，建议勾选。

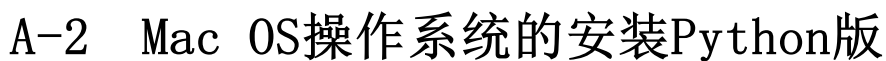
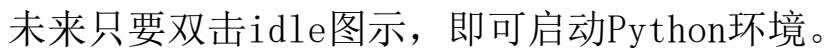
注 上述默认安装路径是C:\文件夹路径，如果想安装其他路径，建议可以单击Customize installation，然后再选择路径，例如：选择C:\即可。

下列是笔者采用默认安装路径的画面，上述如果单击Install Now 选项可以进行安装，下方可以看到，未来安装Python的所在的文件夹。安装完成后将看到下列画面。

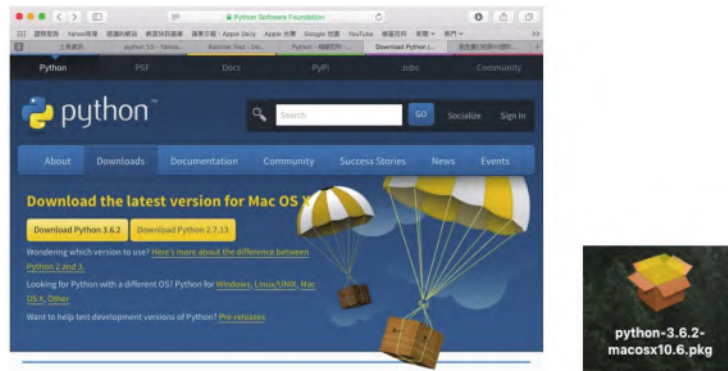


安装完成后，请进入所安装的文件夹，找寻idle 文件，这是Python 3.x版的整合环境程序，未来可以使用它编辑与执行Python。如果你可以顺利进入此文件夹，恭喜你，如果找不到，可以搜索C盘，搜索字符串可以是“Python3”。

当搜索到了以后，可以将此Python整合环境的idle程序拖曳至桌面。



然后可以下载安装程序如下所示:

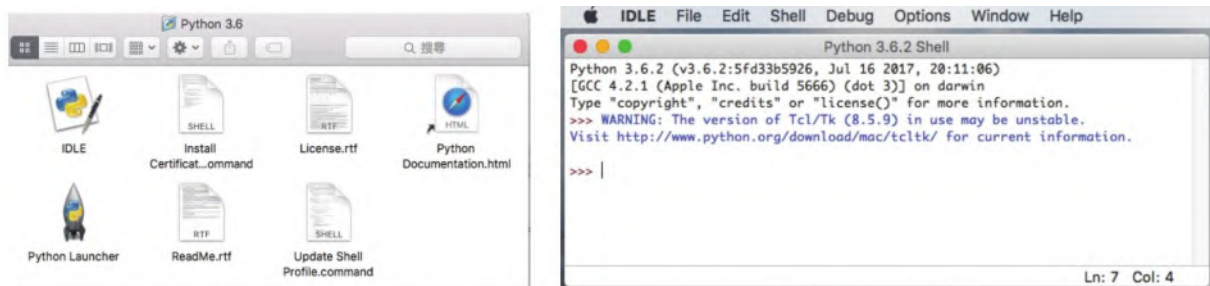


然后可以进入安装画面。

请单击继续 按钮，接着请遵照指示，一步一步即可以安装Python。安装完成后，可以在应用程序文件夹，看到所安装的Python程序。



不论是安装3.6x版本或2.7x版本，皆可以在上述文件夹看到所安装的Python程序。左下图是Python 3.6x版本的实例。请单击IDLE即可以进入右下图Python的IDLE环境。



现在可以正式使用Python了。

附录B 安装第三方模块

Python是一个免费的软件，因此吸引许多公司以它作为公司的官方语言，同时也吸引了很多公司或个人将所开发的模块放到网页上供其他人下载使用。通常我们将这些放在网络上可以下载使用的模块称为第三方模块。

B-1 pip工具

安装第三方模块在Windows操作系统需使用pip工具，如果是Mac OS或Linux则使用pip3工具。安装Python完成后，这些工具放在Scripts目录内。

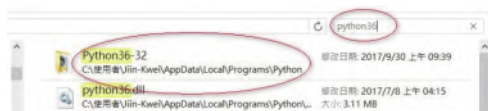
B-1-1 Windows系统Python 3.6.2安装在C:\

如果你的Python 3.6.2版是建立在C:\Python36-32，则pip工具是在下列位置。

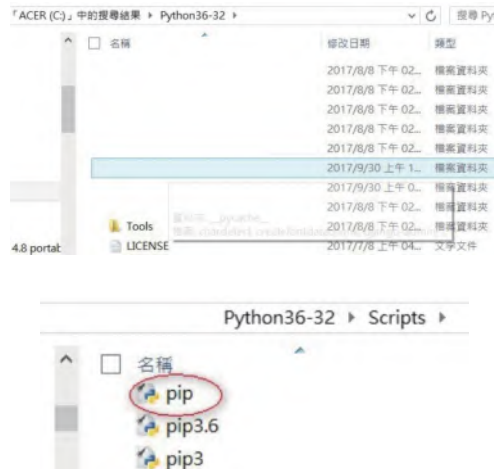
```
C:\Python36-32\Scripts\pip.exe
```

B-1-2 Python 3.6.2安装在硬盘更深层

如果你的Python不是安装在C:\，例如，笔者计算机是Windows 8，安装Python时，在默认安装模式下Python是安装在下列文件夹：



整个系统有一点复杂，需搜索python36，找寻此数据字符串，找到后请单击Python36-32进入此文件夹，双击Scripts文件夹可以进入此文件夹，然后可以看到pip.exe文件。



由于我们未来需进入DOS模式安装第三方模块，此时最好是用复制路径方式将pip.exe文件路径复制到DOS提示行，不会有错误产生。首先按住Shift键右击pip.exe文件，然后选择复制为路径 命令，如下所示。



这时路径已经复制了，路径复制时此路径的字符串前后有双引号，如下所示。

```
"C:\Users\ .... Scripts\pip.exe"
```

未来在DOS贴上此路径时，须将前后双引号删除，如下所示。

```
C:\Users\ ... Scripts\pip.exe
```

B-2 启动DOS与安装模块

B-2-1 DOS环境

B-2-1-1安装Python时没有设定Add Python x.x to PATH

在Windows 7系统可以按下键盘的“窗口键+r键”开启DOS环境。接着将看到DOS的运行窗口。

这时必须将启动安装第三方模块的指令输入到打开字段，首先读者可以将鼠标光标移至打开 字段，单击鼠标右键 打开快捷菜单，执行粘贴，就可以将pip.exe的路径粘贴进来。



此时各位请先贴上路径，记住需将双引号删除，然后在 \Scripts\ 右边，输入下列指令。

```
pip install send2trash          # Windows系统
```

```
sudo pip3 install send2trash    # Mac OS或Linux
```

在上图中单击确定 按钮，就可以看到Windows系统会另外开启DOS窗口执行下载安装第三方模块的画面，这个窗口会在安装完成后自

动关闭。

B-2-1-2安装Python时设定Add Python x.x to PATH

若是设定Add Python x.x(版本信息) to PATH, 可以直接输入下列指令安装相同的模块。

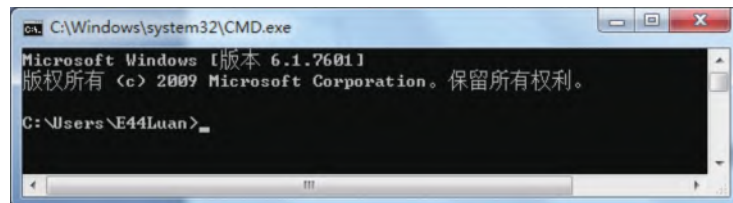
```
pip install send2trash    # Windows系统
```

B-2-2 DOS命令提示字符

单击Windows开始菜单，在搜索框中输入“cmd”。



按下回车键打开命令行窗口。



可参考B-2-1小节将pip.exe的路径复制，再执行pip install send2trash。

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\Scripts\pip install send2trash
```

即可进行安装第三方模块。

B-3 导入模块安装更新版模块

模块安装完成后，未来可以在程序前面执行import指令导入模块，同时可以测试是否安装成功，如果没有错误信息就表示安装成功了。

```
import 模块名称  
import send2trash # 导入send2trash为实例
```

B-4 安装更新版模块

未来如果有更新版，可用下列方式更新至最新版模块。

```
pip install -U 模块名称 # 更新至最新版模块
```

附录C 函数或方法索引表

(记录所出现章节)

`__init__( )`...12-1-3

`abs( )`...3-2-6

`acquire( )`...30-2-9

`add( )`...10-3-1

`add( )`...24-6-1

`add_heading( )`...17-4-1

`add_page_break( )`...17-4-4

`add_paragraph( )`...17-4-2

`add_picture( )`...17-4-5

`add_row( )`...17-5-2

`add_run( )`...17-4-3

`add_table( )`...17-5

`addPage( )`...18-6

`Alignment( )`...19-7

`append( )`...19-3-7

`append( )`...6-4-1, 6-7-1

`append( )`...8-7

`arange( )`...23-3-1
`AreaChart( )`...19-9
`AreaChart3D( )`...19-9
`asctime( )`...13-6-3
`autofmt_xdate( )`...23-7-8
`axes( )get_xaxis( )`...23-4-3
`axes( )get_yaxis( )`...23-4-3
`axis( )`...23-1-2, 23-7-11
`back( )`...22-8
`back( )`...31-3
`backward( )`...31-3
`bar( )`...23-6
`BarChart( )`...19-9-1
`BarChart3D( )`...19-9-2
`Barrier( )`...30-2-14
`bbox_to_anchor( )`...23-1-10
`BeautifulSoup( )`...21-4-1
`begin_fill( )`...31-7
`bfUpdate( )`...32-4
`bgcolor( )`...31-9
`bgpic( )`...31-9

bin()...3-2-2
bind_all()...32-5-4
bk()...31-3
BoundedSemaphore()...30-2-13
BubbleChart()...19-9
Button()...32-2
Canvas()...32-3-1
cell()...19-2-6
choice()...13-5-2
chr()...3-4-6
Chrome()...22-1-4-2
circle()...31-5
clear()...10-3-6
clear()...22-4
clear()...30-2-15
clear()...9-1-6
clearstamp()...31-6
clearstamps()...31-6
click()...22-5
click()...28-1-8
Client()...25-3

close()...20-5-1
color()...31-4
column_index_from_string()...19-2-8
config()...32-4
coords()...32-6-2
copy()...10-3-2
copy()...27-5-2
copy()...9-1-9
count()...6-6-2
create_arc()...32-3-4
create_image()...32-3-8
create_line()...32-3-2
create_oval()...32-3-5
create_polygon()...32-3-6
create_rectangle()...32-3-3
create_sheet()...19-3-4
create_text()...32-3-7
crop()...27-5-1
csvDictReader()...20-4-5
csvDictWriter()...20-5-5
csvreader()...20-4-2

csvwriter()...20-5-2
csvwriter()...20-5-4
currentThread()getName()...30-2-3
cv2CascadeClassifier()...33-4-2
cv2circle()...33-3
cv2destroyAllWindows()...34-2-4
cv2destroyWindow()...34-2-4
cv2imread()...34-2-2
cv2imshow()...34-2-3
cv2imwrite()...34-2-6
cv2line()...33-3
cv2putText()...33-3
cv2rectangle()...33-3
cv2waitKey()...34-2-5
datetime()...30-1-2
decrypt()...18-5
detectMutilScale()...33-4-2
difference()...10-2-3
difference_update()...10-3-12
dir()...4-5
dir()...6-2-3

discard()...10-3-4
Document()...17-2-1
down()...31-4
dragRel()...28-1-10
dragTo()...28-1-10
dump()...24-3-1
dumps()...24-2-1
ehlo()...26-1-5
ellipse()...27-4-3
encrypt()...18-8
end()...16-6-2
end_fill()...31-7
enumerate()...10-4-4
enumerate()...6-12, 8-10
Event()...30-2-15
extend()...6-7-2
extractall()...14-5-3
fd()...31-3
figure()...23-5-1
figure()...23-7-5
fill_between()...23-7-10

fillcolor()...31-7

filling()...31-7

filter()...11-7-2

find()...14-2-8

find()...21-4-5

find_all()...21-4-6

find_element_by_class_name()...22-4

find_element_by_css_selector()...22-4

find_element_by_id()...22-4

find_element_by_link_text()...22-4

find_element_by_name()...22-4

find_element_by_partial_link_text()...22-4

find_element_by_tag_name()...22-4

find_elements_by_class_name()...22-4

find_elements_by_css_selector()...22-4

find_elements_by_id()...22-4

find_elements_by_link_text()...22-4

find_elements_by_name()...22-4

find_elements_by_partial_link_text()...22-4

find_elements_by_tag_name()...22-4

findall()...16-2-4

Firefox()...22-1-4-1
float()...3-2-5
Font()...19-4-1
font()...31-10
format()...4-2-4
forward()...22-8
forward()...31-3
fromkeys()...9-7-2
frozenset()...10-5
geometry()...33-5
get()...21-2-1
get()...30-2-12
get()...32-4
get()...33-5
get()...9-7-3
get_active_sheet()...19-2-2
get_attribute()...22-4
get_busy()...33-3
get_column_letter()...19-2-8
get_sheet_by_name()...19-2-3
get_sheet_names()...19-2-2

get_volumn()...33-2
getcolor()...27-1-2
getPage()...18-3
getpixel()...27-4-4
getrgb()...27-1-1
getshapes()...31-6-2
getText()...21-4-7
globglob()...14-1-11
goto()...31-3
grid()...32-4
group()...16-2-3
groups()...16-3-2
help()...4-1
help()...6-2-3
hex()...3-2-4
hideturtle()...31-6-1
histogram()...33-4-5
hotkey()...28-3-4
ht()...31-3
id()...6-11-1
image_to_string()...29-3

index()...6-6-1
infolist()...14-5-2
input()...4-4
insert()...6-4-2
insert_paragraph_before()...17-4-2
int()...3-2-5
intersection()...10-2-1
intersection_update()...10-3-10
IntVar()...33-5
is_displayed()...22-4
is_enabled()...22-4
is_selected()...22-4
isAlive()...30-2-6
isdecimal()...16-2-1
isdisjoint()...10-3-7
isdwon()...31-4
isinstance()...12-6-2
iskeyword()...13-8-2
islower()...6-2-3
isOpened()...33-4-4
issubset()...10-3-8

issuperset()...10-3-9
isvisible()...31-6-1
items()...9-2-1
iter_content()...21-2-7
join()...30-2-5
keyDown()...28-3-3
keyPress()...28-3-3
keys()...9-2-2
keyUp()...28-3-3
Label()...33-5
left()...31-3
legend()...23-1-10
len()...10-4-2
len()...17-5-3
len()...6-1-6
len()...6-9-3
len()...8-7
len()...9-1-10
len()...9-7-1
line()...27-6-2
LineChart()...19-9

LineChart3D()...19-9

linspace()...23-3-1

list()...6-9-4

list()...8-8

load()...24-3-2

load()...33-3

load_workbook()...19-2-1

load_workbook()...19-2-6

loads()...4-2-2

localtime()...13-6-4

loggingbasicConfig()...15-7-2, 15-7-3

loggingcritical()...15-7-2

loggingdebug()...15-7-2

loggingdisable()...15-7-11

loggingerror()...15-7-2

logginginfo()...15-7-2

loggingwarning()...15-7-2

login()...26-1-7

lower()...6-2-1

lstrip()...6-2-1

lt()...31-3

mainloop()...31-11-1
mainloop()...31-11-1
map()...11-7-3
max()...10-4-1
max()...6-1-5
max()...6-9-3, 8-9
merge()...18-9
merge_cells()...19-8-1
messagecreate()...25-3
MIMEText()...26-2-6
min()...10-4-1
min()...6-1-5
min()...6-9-3, 8-9
mouseDown()2...8-1-9
mouseUp()...28-1-9
move()...32-5-1
moveRel()...28-1-5
moveTo()...28-1-4
music()...33-3
namedWindow()...34-2-1
namelist()...14-5-2

new()...27-3-6
notify()...30-2-11
notifyAll()...30-2-11
now()...30-1-1
oct()...3-2-3
onclick()...31-11-1
onkey()...31-11-2
open()...14-2-2
open()...14-3-1
open()...14-6
open()...18-1
open()...20-4-1
open()...21-1-1
open()...27-3-1
open()...4-3-1
ord()...3-4-6
osgetcwd()...14-1-3
oslistdir()...14-1-10
ospathabspath()...14-1-4
ospathbasename()...21-5
ospathchdir()...14-1-7

ospathexist()...14-1-6
ospathgetsize()...14-1-9
ospathisabs()...14-1-6
ospathisdir()...14-1-6
ospathisfile()...14-1-6
ospathjoin()...14-1-8
ospathmkdir()...14-1-7
ospathrelpath()...14-1-5
ospathremove()...14-1-7
ospathrmdir()...14-1-7
oswalk()...14-1-12
pack()...32-2
paste()...27-5-3
pause()...33-3
pd()...31-4
PdfFileReader()...18-2
Pen()...31-2
pencolor()...31-4
pendown()...31-5
pensize()...31-4
penup()...31-5

PhotoImage()...32-3-8
PieChart()...19-9-3
PieChart3D()...19-3-4
pixelMatchesColor()...28-2-4
play()...33-2
play()...33-3
plot()...23-1-2
plot()...23-1-7, 23-1-8
point()...27-6-1
poll()...30-3-2
polygon()...27-6-5
pop()...10-3-5
pop()...6-4-3
pop()...9-7-5
Popen()...30-3-1
position()...28-1-3
pow()...3-2-6
print()...4-2
pu()...31-4
put()...30-2-12
putpixel()...27-4-4

pyperclipcopy()...14-7
pyperclippaste()...14-7
quit()...22-8
RadarChart()...19-9
Radiobutton()...33-5
raise_for_status()...21-2-4
randint()...13-5-1
random()...23-4
range()...7-2
recompile()...16-2-2
refindall()...16-2-5
rematch()...16-6-1
research()...16-2-5
read()...14-2-1
read()...33-4-4
readline()...13-7-2
readlines()...14-2-4
rectangle()...27-6-4
Reference()...19-9-1
refresh()...22-8
register_shape()...31-6-2

release()...30-2-9
remove()...10-3-3
remove()...6-4-4
remove_sheet()...19-3-5
render_fo_file()...24-6-1
replace()...14-2-6
resize()...27-4-1
reverse()...6-5-1
rgb()...27-1-1
right()...31-3
rjust()...28-1-7
rotate()...27-4-2
rotateClockwise()...18-7
rotateCounterClockwise()...18-7
round()...3-2-6
rstrip()...6-2-1
rt()...31-3
run()...30-2-8
run()...30-3-6
save()...19-3-2
save()...27-3-5

savefig()...23-1-11
Scale()...32-4
scatter()...23-2
screenshot()...28-2-1
scroll()...28-1-11
search()...16-2-3
Select()...21-4-8
Semaphore()...30-2-13
send_keys()...22-6
send2trashsend2trash()...14-4-8
sendmail()...26-1-8
set()...10-1-2
set()...30-2-15
set()...32-4
set()...33-5
set_categories()...19-9-1
set_visible()...23-4-3
set_volumn()...33-2
setDaemon()...30-2-4
setdefault()...9-7-4
seth()...31-5

setheading()...31-5
setpos()...31-3
setposition()...31-3
setup()...31-9
setworldcoordinates()...31-9
shape()...31-6
show()...23-1-1
shuffle()...13-5-3
shutilcopy()...14-4-1
shutilcopytree()...14-4-2
shutilmove()...14-4-3, 14-4-4, 14-4-5, 14-4-6
shutilrmtree()...14-4-7
size()...28-1-2
sleep()...13-6-2
SMTP()...26-1-4
SMTP_SSL()...26-1-4
sort()...6-5-2
sorted()...10-4-3
sorted()...6-5-3
sorted()...9-2-3
Sound()...33-2

span()...16-6-2
speed()...31-3
split()...6-9-6
st()...31-3
stamp()...31-6
start()...16-6-2
start()...30-2-2
start()...30-2-2
starttls()...26-1-6
StockChart()...19-9
stop()...33-2
stop()...33-3
str()...3-4-4
strftime()...30-1-5
StringVar()...33-5
StringVar()...33-5
strip()...6-2-1
strptime()...23-7-6
sub()...16-7-1
submit()...22-6
subplot()...23-5-2

sum()...6-1-5
super()...12-3-5
symmetric_difference()...10-2-4
symmetric_difference_update()...10-3-12
text()...27-6
Thread()...30-2-2
threadingactive_count()...30-2-7
threadingcurrent_thread()...30-2-7
threadingenumerate()...30-2-7
threadWork()...30-2-2
tick_params()...23-1-5
tight_layout()...23-1-10
time()...13-6-1
timedelay()...33-2
timedelta()...30-1-3
title()...23-1-4
title()...31-9
title()...32-3-9
title()...6-2-1
Tk()...32-1
total_second()...30-1-3

`tracebackformat_exc( )`...15-4

`transpose( )`...27-4-3

`tuple( )`...8-8

`type( )`...3-1

`typewrite( )`...28-3-1

`union( )`...10-2-2

`unmerge_cells( )`...19-8-2

`unpause( )`...33-3

`up( )`...31-4

`update( )`...10-3-11

`update( )`...32-5-1

`upper( )`...6-2-1

`values( )`...9-2-4

`VideoCapture( )`...33-4-4

`wait( )`...30-2-11

`wait( )`...30-3-3

`width( )`...31-4

`window_height( )`...31-9

`window_width( )`...31-9

`Workbook( )`...19-3-1

`World( )`...24-6-1

`write( )`...13-7-3

`write( )`...14-3-1

`write( )`...18-6

`write( )`...31-10

`writeheader( )`...20-5-5

`writerow( )`...20-5-3

`xlabel( )`...23-1-4

`xticks( )`...23-1-9

`ylabel( )`...23-1-4

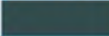
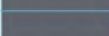
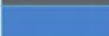
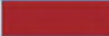
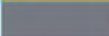
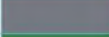
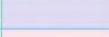
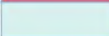
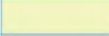
`yticks( )`...23-1-9

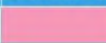
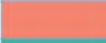
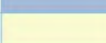
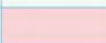
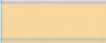
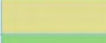
`zip( )`...8-11

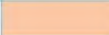
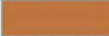
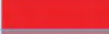
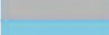
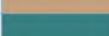
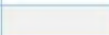
`zipfileZipFile( )`...14-5-1

附录D RGB色彩表

色彩名称	16 进位	色彩样式
AliceBlue	#F0F8FF	
AntiqueWhite	#FAEBD7	
Aqua	#00FFFF	
Aquamarine	#7FFFD4	
Azure	#F0FFFF	
Beige	#F5F5DC	
Bisque	#FFE4C4	
Black	#000000	
BlanchedAlmond	#FFEBCD	
Blue	#0000FF	
BlueViolet	#8A2BE2	
Brown	#A52A2A	
BurlyWood	#DEB887	
CadetBlue	#5F9EA0	
Chartreuse	#7FFF00	
Chocolate	#D2691E	
Coral	#FF7F50	
CornflowerBlue	#6495ED	
Cornsilk	#FFF8DC	
Crimson	#DC143C	
Cyan	#00FFFF	
DarkBlue	#00008B	
DarkCyan	#008B8B	
DarkGoldenRod	#B8860B	
DarkGray	#A9A9A9	
DarkGrey	#A9A9A9	
DarkGreen	#006400	
DarkKhaki	#BDB76B	
DarkMagenta	#8B008B	
DarkOliveGreen	#556B2F	
DarkOrange	#FF8C00	
DarkOrchid	#9932CC	
DarkRed	#8B0000	
DarkSalmon	#E9967A	
DarkSeaGreen	#8FBC8F	
DarkSlateBlue	#483D8B	
DarkSlateGray	#2F4F4F	

色彩名称	16 进位	色彩样式
DarkSlateGrey	#2F4F4F	
DarkTurquoise	#00CED1	
DarkViolet	#9400D3	
DeepPink	#FF1493	
DeepSkyBlue	#00BFFF	
DimGray	#696969	
DimGrey	#696969	
DodgerBlue	#1E90FF	
FireBrick	#B22222	
FloralWhite	#FFFAF0	
ForestGreen	#228B22	
Fuchsia	#FF00FF	
Gainsboro	#DCDCDC	
GhostWhite	#F8F8FF	
Gold	#FFD700	
GoldenRod	#DAA520	
Gray	#808080	
Grey	#808080	
Green	#008000	
GreenYellow	#ADFF2F	
HoneyDew	#F0FFF0	
HotPink	#FF69B4	
IndianRed	#CD5C5C	
Indigo	#4B0082	
Ivory	#FFFFFF	
Khaki	#F0E68C	
Lavender	#E6E6FA	
LavenderBlush	#FFF0F5	
LawnGreen	#7CFC00	
LemonChiffon	#FFFACD	
LightBlue	#ADD8E6	
LightCoral	#F08080	
LightCyan	#E0FFFF	
LightGoldenRodYellow	#FAFAD2	
LightGray	#D3D3D3	
LightGrey	#D3D3D3	
LightGreen	#90EE90	

色彩名称	16 进位	色彩样式
LightPink	#FFB6C1	
LightSalmon	#FFA07A	
LightSeaGreen	#20B2AA	
LightSkyBlue	#87CEFA	
LightSlateGray	#778899	
LightSlateGrey	#778899	
LightSteelBlue	#B0C4DE	
LightYellow	#FFFFE0	
Lime	#00FF00	
LimeGreen	#32CD32	
Linen	#FAF0E6	
Magenta	#FF00FF	
Maroon	#800000	
MediumAquaMarine	#66CDAA	
MediumBlue	#0000CD	
MediumOrchid	#BA55D3	
MediumPurple	#9370DB	
MediumSeaGreen	#3CB371	
MediumSlateBlue	#7B68EE	
MediumSpringGreen	#00FA9A	
MediumTurquoise	#48D1CC	
MediumVioletRed	#C71585	
MidnightBlue	#191970	
MintCream	#F5FFFA	
MistyRose	#FFE4E1	
Moccasin	#FFE4B5	
NavajoWhite	#FFDEAD	
Navy	#000080	
OldLace	#FDF5E6	
Olive	#808000	
OliveDrab	#6B8E23	
Orange	#FFA500	
OrangeRed	#FF4500	
Orchid	#DA70D6	
PaleGoldenRod	#EEE8AA	
PaleGreen	#98FB98	
PaleTurquoise	#AFEEEE	

色彩名称	16 进位	色彩样式
PaleVioletRed	#DB7093	
PapayaWhip	#FFEFD5	
PeachPuff	#FFDAB9	
Peru	#CD853F	
Pink	#FFC0CB	
Plum	#DDA0DD	
PowderBlue	#B0E0E6	
Purple	#800080	
RebeccaPurple	#663399	
Red	#FF0000	
RosyBrown	#BC8F8F	
RoyalBlue	#4169E1	
SaddleBrown	#8B4513	
Salmon	#FA8072	
SandyBrown	#F4A460	
SeaGreen	#2E8B57	
SeaShell	#FFF5EE	
Sienna	#A0522D	
Silver	#C0C0C0	
SkyBlue	#87CEEB	
SlateBlue	#6A5ACD	
SlateGray	#708090	
SlateGrey	#708090	
Snow	#FFFAFA	
SpringGreen	#00FF7F	
SteelBlue	#4682B4	
Tan	#D2B48C	
Teal	#008080	
Thistle	#D8BFD8	
Tomato	#FF6347	
Turquoise	#40E0D0	
Violet	#EE82EE	
Wheat	#F5DEB3	
White	#FFFFFF	
WhiteSmoke	#F5F5F5	
Yellow	#FFFF00	
YellowGreen	#9ACD32	