HZ BOOKS 华章IT
数据分析与决策技术丛书
[PACKT] PUBLISHING

Mastering Python Data Visualization

Python数据可视化

[印度] 科斯·拉曼（Kirthi Raman） 著
程豪 译

全面讲解Python在不同应用领域的可视化方法
涵盖Python的各种绘图选项，包含大量实际应用案例

机械工业出版社
China Machine Press

数据分析与决策
技术丛书

Mastering Python Data Visualization

Python数据可视化

[印度] 科斯·拉曼（Kirthi Raman） 著

程豪 译

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Python 数据可视化 /（印度）科斯·拉曼（Kirthi Raman）著；程豪译 . —北京：机械工业出版社，2017.3
（数据分析与决策技术丛书）
书名原文：Mastering Python Data Visualization

ISBN 978-7-111-56090-6

I. P… II. ① 科… ② 程… III. 软件工具 – 程序设计 IV. TP311.56

中国版本图书馆 CIP 数据核字（2017）第 032016 号

本书版权登记号：图字：01-2016-1889

Python 数据可视化

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：缪　杰　　责任校对：李秋荣
印　　刷：三河市宏图印务有限公司　　版　　次：2017 年 3 月第 1 版第 1 次印刷
开　　本：186mm × 240mm　1/16　　印　　张：17.75
书　　号：ISBN 978-7-111-56090-6　　定　　价：69.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzit@hzbook.com

The Translator's Words 译者序

海量信息的不断增长，不断刺激着读者对数据可视化的渴望与诉求。作为一种功能强大的开源编程语言，Python 包含了丰富的软件包和绘图技术，从而帮助用户完成数据分析、构建统计模型并展现研究结果。

本书尤其关注 Python 在众多应用领域中的可视化功能，全面覆盖 Python 的各种绘图选项，配合丰富的实际案例，为 Python 初学者和资深人士提供了一本实用指南。对于 Python，我不敢自称有丰富的实战经验，但却有过自学和运用的经历。在承担本书翻译工作的同时，我自己也重温了一次 Python 可视化之旅，收益颇多。故劝荐诸位，不妨深读此书，系统体验 Python 在数据可视化方面的贡献。与音乐一样，知识的传播没有国界。因此，翻译不仅是知识表达语言的转换，更是一次学习和交流的机会。与原作者对话，高山仰止，受益匪浅；与读者对话，高山流水，闻过则喜。

在此，感谢我的朋友钟琰在整个翻译过程中提供的帮助。感谢我的至爱刘钰洁在译稿校对阶段给出的建议。我要感谢我的博士生导师——中国人民大学的易丹辉教授。感谢我在美国联合培养期间的导师——美国哥伦比亚大学的韦颖副教授。特别感谢我父母和家人，是他们给予我前行的动力和勇气。最后，非常感谢机械工业出版社华章公司的编辑让我接触到这本书，并给予中肯建议。感谢身边所有的良师益友。

鉴于个人时间与水平有限，如有纰漏，还望各位读者予以反馈，不吝赐教！

程豪

2016 年 12 月 15 日

前　言 *Preface*

数据可视化旨在清楚明了地提供信息，帮助读者定性理解这些信息。俗话说，一图胜千字（百闻不如一见）。这里，可以换个说法，“一幅图讲述了一个故事，如同万语千言。”因此，可视化是一个宝贵的工具，有助于读者快速理解相应的概念。然而，与其说数据可视化是一种技能，还不如说它是一门艺术。这是因为，如过度使用数据可视化会适得其反。

当前，有太多数据需要处理。这些数据包含着许多见解，这些见解是成功的关键。能够发现数据、清洗数据，并使用正确的工具实现可视化至关重要。本书讲解了用 Python 软件包实现数据可视化的不同方法，并给出很多不同领域的案例，比如，数值计算、金融模型、统计和机器学习，以及遗传学与网络。

本书提供在 Mac OS X 10.10.5 系统上运行的案例程序，具体用到 Python 2.7、IPython 0.13.2、matplotlib 1.4.3、NumPy 1.9.2、SciPy 0.16.0 和 conda 构建 1.14.1 版本。

本书主要内容

第 1 章阐述了数据可视化确实应该被称为“用于知识推断的数据可视化”。本章包含框架，讲解数据 / 信息如何转换为知识，以及有意义的呈现方式（通过取对数、颜色映射、散点图、相关性以及其他）如何能够帮助我们更容易地掌握知识。

第 2 章讲述可视化的重要性，展示可视化过程中的一些步骤，包括可选择的几种工具选项。可视化方法由来已久，很早之前我们就接触过这些方法；比如，连年幼的小孩都能解释条形图。交互式可视化有很多优点，本章将举例说明。

第 3 章解释了从 Continuum Analytics 使用 Anaconda 时，不必安装每个 Python 库的原因。Anaconda 有简化的打包和部署方法，这些方法使得 IPython notebook 与其他库的并行运算变得更加容易。

第 4 章包括交互式绘图方法及在计算物理和应用数学中的实践案例。一些著名的案例

包括用 SciPy 实现插值方法、近似、聚类、抽样、相关关系和凸优化。

第 5 章探索金融工程，该领域有很多数值计算和图表绘制的方法，是探索 Python 的一个有趣的案例。本章通过举例讲述股票报价、回归分析、蒙特卡洛算法和模拟方法。

第 6 章包含了用 NumPy、SciPy、matplotlib 和 scikit-learn 等工具进行处理的统计方法，比如，线性、非线性回归、聚类和分类。

第 7 章包含了有趣的案例，比如社交网络以及现实生活中的有向图举例，适用于这些问题的数据结构，以及网络分析。本章会用到一些具体的库，比如 graph-tool、NetworkX、matplotlib、scipy 和 numpy。

第 8 章包含模拟方法和信号处理案例，用以展示一些可视化方法。这里，我们也给出了其他高级工具的对比，比如 Julia 和 D3.js。

附录给出了 conda 概述，并列出多种 Python 库。

学习本书的准备工作

本书要求用户在操作系统上安装 2.7.6 或以上版本的 Python。对于书中的案例，可以使用 Mac OS X 10.10.5 的 Python 默认版本（2.7.6）来实现。其他会用到的软件包是 IPython——一个交互式 Python 环境。新版的 IPython 叫 Jupyter，该版本现在有 50 种不同语言的内核函数。

安装提前打包好的用于科学计算的 Python 发行版，如果可能的话，可以从 Continuum 安装 Anaconda，或安装 Enthought Python Distribution。Anaconda 一般自带 300 多个 Python 软件包。你可以用 pip 或 conda 安装不在自带软件包列表中的 Python 软件包。有一些案例可见附录。

本书适用对象

目前已有很多 Python 和数据可视化方面的书。然而，对于有一定 Python 知识储备的人来说，几乎很少有把两者内容结合在一起的书值得推荐。有关简化代码、重复使用的小生境（niche）技术的讨论更是少之又少。对于有强烈学习兴趣的 Python 开发人员，本书将提供一系列获得分析结果和产生惊人可视化效果的方法。

本书提供了解决实际问题的一系列分析方法。虽然本书并不是面向初学者的，但是如果有需要，你可以搜索书中推荐阅读的文献资料。如果这是你初次体验 Python 编程或数据可视化，提前阅读一些入门教材会有很大帮助。我最喜欢的书有 John Guttag 教授的《Introduction to Computer Science and Programming》（可从 MIT OpenCourseWare 上免费下载）和来自 UCLA 的 Nathan Yau 的《Visualize This》。

目　录 *Contents*

第 1 章 Chapter 1

数据可视化概念框架

当代，网络和社交媒体的兴起，产生了大量数据，而且数据量的增长已超乎想象。这种现象是怎么发生的？又是何时发生的？

十年前，一种处理问题的新方法演变为：跨企业的从数据源收集、整合大量数据，并进行运算的研究工作。他们这样做的目标是用海量数据改善决策过程。在此期间，促使 Amazon、Yahoo 和 Google 这样的公司在处理大量数据方面取得了显著进展。这些里程碑式的成就促使一些大数据分析技术的诞生。当然，我们不会追究大数据的细节问题，但是我们将尝试探索，为什么很多机构改变了他们以往的模式，用类似的想法获得更好的决策。

到底如何用这些海量数据做出更好的决策？这是我们的终极目标，但首先让我们理解数据、信息和知识间的差异，以及它们与数据可视化之间的关系。或许会有这样一个疑问，为什么要讨论数据、信息和知识。我们将就下面的脉络具体展开：怎样开始、用什么开始、这些内容如何有益于问题解决，以及可视化的作用。我们将通过简要回顾涉及的程序步骤，确定数据可视化所需的概念框架。

本章将包括以下主题：

- ❑ 数据、信息、知识和观点之间的差异
- ❑ 信息转化为知识，进而转化为观点
- ❑ 收集、处理和组织数据
- ❑ 数据可视化的历史
- ❑ 数据可视化如何帮助决策
- ❑ 可视化图像

1.1 数据、信息、知识和观点

数据、信息和知识被广泛用于计算机科学领域。通常，这些术语有很多种充满争议且不相一致的定义。在深入研究这些定义之前，我们先理解这些术语与可视化之间的关系。数据可视化的主要目标是从数据或信息中得出观点（隐含的真理）。本书有关数据、知识和观点的整个讨论属于计算机科学的范畴，而非心理学或认知科学。认知科学方面的文献请参见：https://www.ucsf.edu/news/2014/05/114321/converting-data-knowledge-insight-and-action。

1.1.1 数据

数据是得出结论的前提。尽管在一些特定的背景下，数据和信息看起来相关联。但实际上，数据是离散、客观事实的数字表示。作为后续工作的基础，数据会有不同的组织和安排形式，以方便得到回答实际问题的有用信息。

数据可以是非常简单却庞大冗杂的。离散数据本身不能用于决策。这是因为它没有意义，而且更重要的是，它们之间没有结构或关系。数据收集、转换和储存的过程因数据类型和储存方法的不同而有很多变化。数据有很多形式，一些常见形式如下：

- CSV 文件
- 数据库表格
- 文件格式（Excel、PDF、Word 等）
- HTML 文件
- JSON 文件
- 文本文件
- XML 文件

1.1.2 信息

信息是处理后的数据，为实际问题提供答案。当我们增加一种关系或一个关联时，数据就成为信息。这种关联通过提供数据背景来完成。这个背景有助于我们回答数据相关的问题。

比如，我们假定一名篮球员的数据包含身高、体重、位置、大学、出生日期、应招入队，选拔轮数，NBA- 登场和新成员排名。问题“哪位球员是首位应征入队、身高在 6 英尺[⊖]以上且担任控球后卫？”的回答是一条信息。

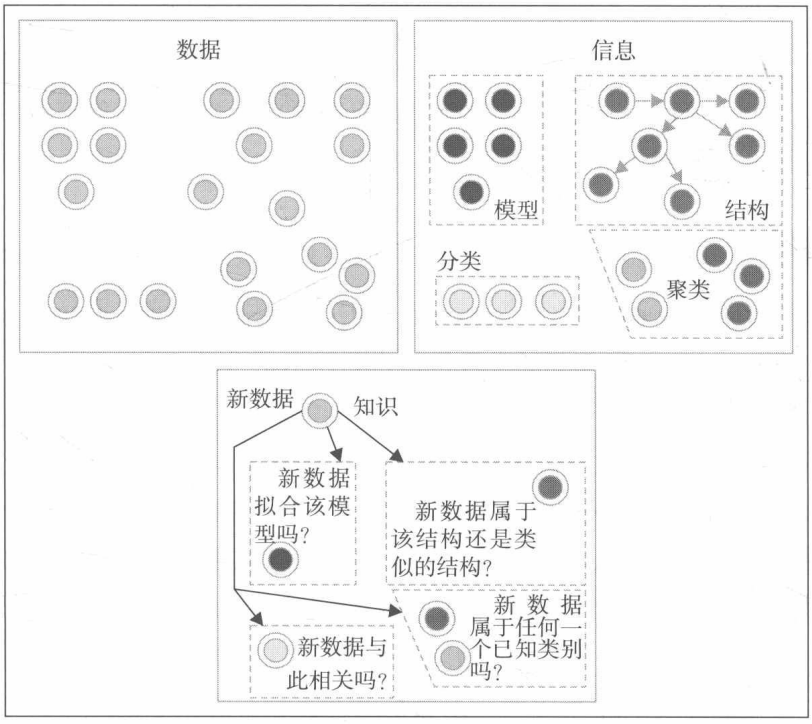
类似地，每个球员的得分也是一条数据。问题“今年每次比赛得分最高的选手是谁？分数是多少？”的回答“LeBron James，27.47”同样也是一条信息。

⊖ 1 英尺 =0.3048 米。

1.1.3　知识

当人类解释和组织信息，并用以决策时，知识便应运而生。知识是数据、信息和通过经验获得的技能。知识包括做出适当决策的能力和执行时所需的技能。

作为必不可少的部分（连接数据）允许我们理解每条信息的相对重要性。通过比较过去的结果和识别模式，我们不必从头开始寻找问题的解决方法。下图总结了数据、信息和知识的概念。



知识以不断增长的方式发生变化，特别是当信息被重新安排或被重新组织，或在一些计算算法发生变化时。知识像箭一样直击算法的结果，该算法与来自数据的过去信息有关。在许多情况下，可以通过与结果的视觉交互获得知识。另一方面，观点开启了通向未来的途径。

1.1.4　数据分析和观点

在我们深入研究观点的定义及其如何与实际问题相关联之前，我们不妨先看看如何获取观点。十年间，组织机构已尽力弄懂他们拥有的所有数据和信息，特别是探索数据量的大小。为了基于已有数据信息得到最佳或现实的决策，他们发现了数据分析的重要性（也就是**数据分析学**或**分析学**）。

分析学依赖数学算法来确定产生观点的数据间的关系。一种简单的方式是通过打比方来理解观点：当数据没有结构且与实际问题相对应时，通过将数据结构化，使其更接近实际目标，这有助于人们更清晰、更深刻地理解数据。观点是“我发现了”的那个时刻，得到突破性的结果。一个人不应该困惑于术语分析学和商务智能。当商务智能提供基于历史数据的分析结果时，分析学就具备了预测能力。

分析学通常用于更广泛的数据，为此，数据内外之间的协作时常发生。在一些实际问题的范式中，这种协作仅发生在海量数据的内部，但在大多数情况下，加入外界信息有助于链接点或完成拼图。最常见的两个外部数据链接源是社交媒体和用户群体。

在本章，我们应用分析法理论得出观点、驱动商业价值，以及改善决策和更好地理解用户，我们得出真实生活故事中有价值的结论。

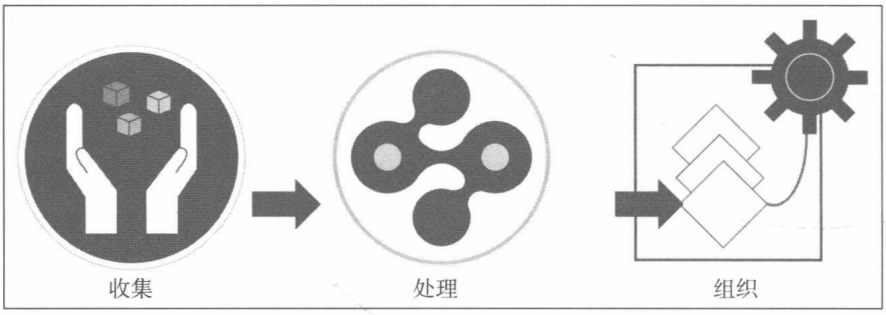
1.2 数据转换

现在，我们了解了数据的定义，但问题是：为什么要收集数据？数据对于描述物质或社会现象以及进一步回答这些问题非常有用。出于这个原因，确保数据的无误、精确和完整是很重要的；否则，错误、不精确和不完整的数据将导致响应结果的不精确或不完整。

数据有不同种类，其中包括*过去表现数据*、*实验数据*和*基准数据*。过去表现数据和实验数据当然很容易理解。另一方面，基准数据是用一个测度标准来比较两种不同项目或产品的特征。数据被转换为信息，得到进一步处理，然后用来解答问题。因此，很明显下一步就是转换的实现。

1.2.1 数据转换为信息

根据数据的内容和重要性，数据收集和储存有一些不同的方式。例如，如果数据是关于篮球季后赛的，那么这些数据将储存为文本和视频格式。另一个例子是一个国家所有城市的温度记录，这些数据通过不同形式收集得到。从数据转换为信息包含数据的收集、处理和组织，如下图所示：



收集来的数据需要处理和组织过程，这些数据后续可能没有结构、没有模型或没有模式。然而，该处理过程至少给我们一种从数据中发现问题答案的组织方式。这种处理可以是一种基于篮球员总得分的简单分类，或者根据城市和州名的分类。

从数据到信息的转换也可以不仅仅是分类，比如统计建模或计算算法。将数据转换为信息确实很重要，这样数据可以被查询、访问和操作。海量数据的转换可能包括这样几种处理方法：过滤、聚集、应用相关性、归一化和分类。

1. 数据收集

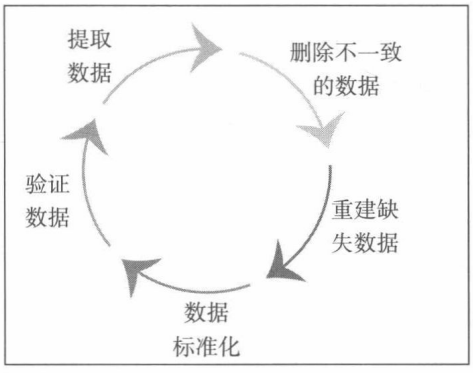
数据收集是一个耗时的过程。因此，人们正在寻找更好的自动数据采集方法。然而，人工数据收集仍然很常见。如今，数据的自动收集过程用到输入设备，比如传感器。例如，通过传感器检测水下珊瑚礁；农业上用传感器检测土壤性质、控制灌溉和施肥方法是另一个应用领域。

另一种自动收集数据的方法是通过扫描文档和日志文件，这也是一种服务器端数据收集的形式。人工处理包括基于网络且储存于数据库的数据收集方法，这些数据可以转换为信息。现在，基于网络的协作环境正受益于交流改善和数据分享。

传统的可视化和可视化分析工具专门为单个用户、单机可视化应用而设计。将这些工具的功能拓展到支持协作的层面需要一个漫长的过程，才能扩大真实世界中可视化的适用范围和应用领域。

2. 数据预处理

如今，基于数据量、数据来源的多重异质性和数据类型的不同，数据很容易受到噪音和不一致的影响。现有一些数据预处理技术，比如*数据清洗*、*数据集成*、*数据压缩*和*数据转换*。数据清洗用于数据中的噪音清理和矛盾修正。数据集成将多个数据源的数据合并起来，通常被称为数据仓库。数据压缩可以通过诸如合并、聚集和消除冗余特征等方法减少数据量。数据转换将数据缩放到一个较小的区间，从而提高处理和可视化的精确性和效率。数据的转换周期如下图所示：



异常值检测是非常规数据的识别，这些数据可能不会落入收集数据的预期行为或模式。异常值也称为离群点或噪音；比如信号数据，一个非常规的特别信号被视为噪音。交易数据中的一个离群点是欺诈交易。准确的数据收集对于保持数据完整性必不可少。然而，从另一角度考虑，异常值也非常重要，比如寻找诈骗保险理赔。

3. 数据处理

数据处理是转换过程中的重要一步。当务之急是关注数据质量。依存模型和聚类有助于准备分析数据和更好地理解处理步骤。虽然也有其他处理技术，但是我们在这不做过多赘述，仅以两种最受欢迎的处理方法为例。

依存模型是建模数据以确定表现方式性质和结构的基本原则。该过程寻找数据元素间的关系；比如，百货公司可能收集顾客购买习惯的数据。该过程有助于百货公司减掉频繁购买的信息。

聚类是在数据中发现群组，从某种方式上看，“相似性模式”没有用数据中已知的结构。

4. 组织数据

数据库管理系统允许用户以结构化的形式存储数据。然而，数据库太大而不能存入内存。有以下两种结构化数据的方法：

- 以结构化的形式将大量数据储存到磁盘中，比如，表、树或图表
- 为了快速访问，以结构化的形式将数据储存到内存中

数据结构由将数据结构化为可被储存和访问的一系列不同格式构成。常用的数据结构类型有数组、文件、表、数、列表、映射等。任何数据结构都是为特定目的而设计的，通过组织数据来进行数据储存、访问和操作。一种数据结构可能被选择或设计来储存数据，以实现用不同算法更快访问的目的。

经过高效收集、处理和组织所存储的数据，使数据更容易被理解，这也有助于更好地理解数据中蕴含的信息。

5. 获取数据集

针对接触不到组织数据的读者，下面列举出一些丰富的数据集资源：

- http://grouplens.org（来自明尼苏达大学）
- http://ichart.finance.yahoo.com/table.csv?s=YHOO&c=1962
- http://datawrangling.com/some-datasets-available-on-the-web
- http://weather-warehouse.com（天气数据）
- http://www.bjs.gov/developer/ncvs/（Justice 统计局）
- http://census.ire.org/data/bulkdata.html（人口普查数据）

- http://www.pro-football-reference.com（足球参考）
- http://www.basketball-reference.com（篮球参考）
- http://www.baseball-reference.com（棒球参考）
- http://archive.ics.uci.edu/ml/datasets.html（机器学习）
- http://www.pewresearch.org/data/download-datasets/
- http://archive.ics.uci.edu/ml/datasets/Heart+Disease（心脏病）

1.2.2　信息转换为知识

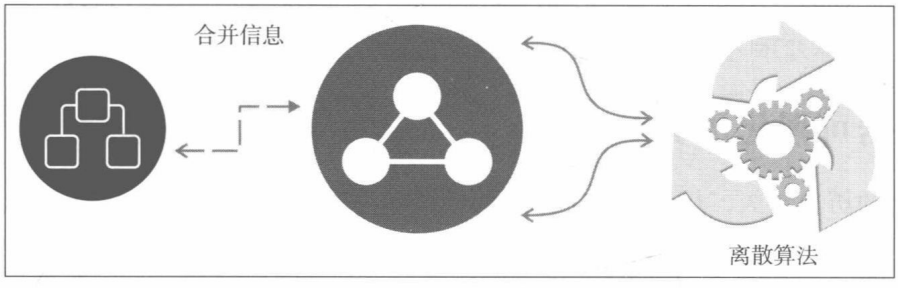
信息是可量化的、可测度的、有形式的，可以被访问、生成、存储、分发、搜索、压缩和复制。信息可以通过数量或信息量进行量化。

通过应用离散算法，信息可转换为知识，知识要比信息更可量化。在某些领域，知识持续经历了一个不断发展的周期。当数据发生实时变化时，这种演变过程随之发生。

知识就像是帮助你做面包的面粉和酵母成分的烹饪配方。另一个看待知识的方法是数据和信息的结合，并加入经验和专家意见，以帮助决策。知识不仅仅是过滤或算法的结果。

转换中包括哪些步骤？这种变化如何发生？当然，它本身是不能发生的。尽管信息这个词是基于定义的不同阐释，但是，我们将在计算的范围内进一步探索。

有一个简单的类比用以说明信息和知识之间的区别：一门特定课程的课程材料为你提供有关概念的重要信息，随后老师引导学生通过讨论来理解概念。这有助于学生获得课程知识。类似地，信息转换为知识也需要完成一些工作。下图展示了信息转换为知识的过程：



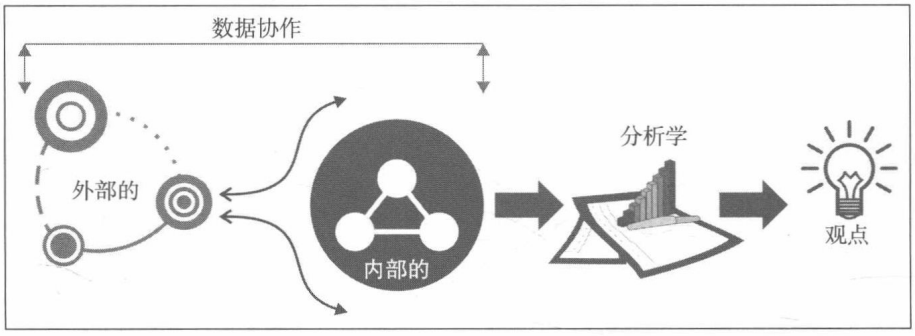
正如上图所示，信息通过一些离散算法进行合并和运行后，就能转换为知识。需要通过整合信息得到更多的知识。通过这种转换获得的知识有助于回答有关数据或信息的问题，比如，公司在哪个季度销售收益最高？广告拉动销售的贡献有多大？今年发布了多少新产品？

1.2.3　知识转换为观点

在传统的系统中，信息经处理、分析并形成报告。自因特网诞生以来，我们可以获取经过处理的信息，而且社交媒体融合成为一种处理实际问题的新方式。

一些组织机构已开始分析外部数据来获得观点。比如，通过 Twitter 上消费者的推文完成对用户情绪的测度，以此来追踪他们对产品品牌的意见。在某些情况下，较高比例的用户会在社交媒体上发布新产品的好评，比如一台 iPhone 或平板电脑。分析工具能够提供该情绪的数据化证据，这就是数据可视化扮演的重要角色。

下面是知识转化为观点的另一个例子。2009 年 Netflix 公司宣布了一场比赛，该比赛基于已有的电影分级，评选用来预测用户对电影评级的最佳协同过滤算法。比赛的获胜者用语用学理论，在预测用户分级方面提高 10.05% 的正确率，增加了 Netflix 公司的商业价值。



知识转换为观点是通过如上图所示的协作和分析来实现的。观点意味着看到解决方案，并发现需要做的事情。得到数据和信息很容易，一些组织机构已经知道获取方法，但是得到观点却很难。观点的得出需要新的创造性思维和连点成线的能力。除了应用创造性思维，数据分析和数据可视化在观点得出的过程中也发挥着很大作用。数据可视化被视为艺术和科学的结合。

1.3 数据可视化历史

可视化的历史悠久，最早用墙上的原始绘图和图像，表中的数字以及黏土上的图像来呈现信息。然而，它们并没有被称为可视化或数据的可视化。数据可视化是一个新术语；它传达出可视化不仅仅是以图表的形式展示数据。数据背后的信息应该用效果良好的图表直观揭示出来；图表本身应该帮助读者看到数据结构。

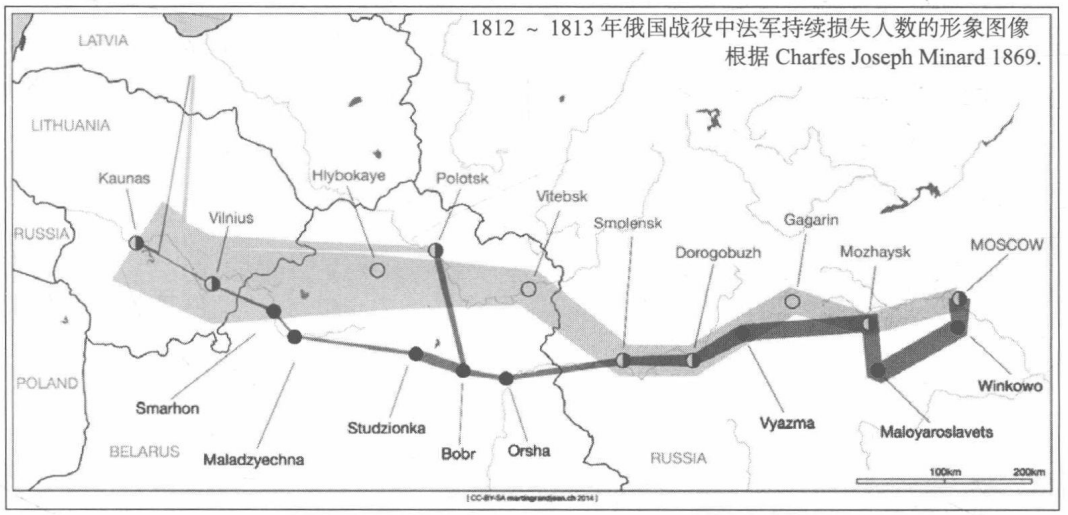
计算机出现前的可视化

在巴比伦时代早期，图片被绘制在黏土上，随后被渲染在纸草上。那些图的目标是给人们提供对信息的定性理解。众所周知，作为一种信息的可视化展示，我们对图片的理解是一种本能，因此理解过程非常轻松。本节只包括可视化历史的部分细节。关于精心设计的细节和例子，我们推荐两个有趣的资源：

- 数据可视化（http://euclid.psych.yorku.ca/datavis/）
- The Work of Edward Tufte and Graphics Press（www.edwardtufte.com/tufte）

1. Minard 的俄国战役 (1812)

Charles Minard 是一位在巴黎工作的土木工程师。他用形象化的图像总结了 1812 年拿破仑进军莫斯科的行军路线。这幅图很简单，包括可视的时间线、描绘军队大小和方向的地理地图、温度以及地标和地点。Edward Tufte 教授曾将这幅图描述为有史以来最好的统计图。



一开始，该楔形的左侧较厚，我们看到战争在波兰边境开始时，军队有 422 000 名战士。当军队深入俄国，气温变低时，该楔形变窄。这种可视化成功地将许多不同数字和地理汇集为一张图：军队何时变少，为什么变少，以及他们的撤退。

2. 伦敦的霍乱疫情（1831 ~ 1855）

在 1831 年 10 月，首例霍乱发生在英国，死亡人数超过 52 000 人。随后，在 1848 ~ 1849 年和 1853 ~ 1854 年间，更多的霍乱疫情造成大量人员死亡。

1855 年，John Snow 博士绘制出伦敦 Broad Street 水泵周围的霍乱死亡情况分布图。John Snow 博士绘制的这幅图具有里程碑意义，但不幸的是，该图在那段时期末期才设计出来。他的地图上显示了每位死者的位置，并得出一个结论，即霍乱可能源自 Broad Street 水泵中被污染的水。那个时期前后，图表的使用在经济和国家规划中变得重要起来。

3. 统计图表（1850 ~ 1915）

在 19 世纪中期，可视化的迅速增长已经在整个欧洲得以建立。在 1863 年，欧洲

Galton 多元天气图表的其中一页显示了 1861 年 12 月的气压、风向、雨水和温度。(来源：The life, letters and labors of Francis Galton, 剑桥大学出版社。)

在这段时期，统计图表成为主流，与此同时，有很多相关的教科书。这些教科书包含图表绘制方法的详细描述，讨论频率和销量选择的影响，以及差异和比率可视化估计的基线。它们也包含在一个图上绘制两个或两个以上时间序列曲线，以实现对历史记录的比较。

4. 数据可视化后期发展

在 1962 年，John W. Tukey 发布了数据分析认知的诉求，作为统计的一个合法分支。不久之后，他开始在**探索性数据分析**（Exploratory Data Analysis, EDA）专栏下，发明各种各样崭新、简单且和有效的图表，随后是**探索性空间数据分析**（Exploratory Spatial Data Analysis, ESDA)。后来，Tukey 在 1977 年写了一本书 *Exploratory Data Analysis*。有很多对 EDA 绘图技术有用的工具，具体如下：

- ❑ 盒形一虚线图（箱线图）
- ❑ 直方图
- ❑ 多元图（源自 K 线图）
- ❑ 运行序列图
- ❑ Pareto 图（在 Vilfredo Pareto 后命名）
- ❑ 散点图
- ❑ 多维量表
- ❑ 目标投影追踪

科学计算的可视化作为一个基于计算的重要领域，旨在提高对数据的理解并迅速做出实时决策。今天，医生诊断疾病的能力与视觉有关。例如，现在做髋关节置换手术可以在外科手术前完成髋关节的定制。通过使用非侵入性的三维成像，在手术前完成精确的测量，从而减少手术后身体排异反应的数量（从 30% 减到仅有 5%)。(来源：http://bonesmart.org/hip/hip-implants-specialized-and-custom-fitted-options/。)

作为研究前沿，人类大脑结构和功能的三维可视化具有深远意义。几乎没有其他研究进展改变神经科学和大脑成像技术领域，比如能够看到大脑内部，读取人类活体的大脑。为了在大脑研究中不断取得进展，很多抽象的集成结构和功能信息将很有必要。

硬件性能功耗比例的持续上升表明：我们已经有能力分析 DNA 序列，并完成视觉上的展示。未来在计算方面的研究进展有望给医学和其他科学领域带来很大的进步。

1.4 可视化如何帮助决策

数据有多种视觉展示的方式。然而，其中仅有少数方式能够用人们视觉上看得懂且观

察到的新模式来刻画数据。数据可视化并不像看起来那么简单；它是一门艺术，且需要很多实践经验。（就像画一幅画，一个人不可能一天之内成为绘画大师，它需要很多实践经验。）

人类感知在数据可视化领域扮演着重要角色。健康人的眼睛具有水平方向大约 200 度的视野范围（两只眼睛共享大约 120 度的视野）。大概人类大脑的四分之一涉及可视化处理，这比其他任何感官都要多。在听觉、视觉和嗅觉中，人的视觉占据最多（约为 60%）（http://contemplatingmadness.tumblr.com/post/27478393311/10-limits-to-human-perception-and-how-they-shape）。

有效的可视化有助于我们分析理解数据。作者 Stephen Few 列举出如下 8 种定量信息（通过可视化），这有助于理解或交流数据（来源：https://www.perceptualedge.com/articles/ie/the_right_ graph.pdf）：

- ❑ 时间序列
- ❑ 排序
- ❑ 局部到整体
- ❑ 偏差
- ❑ 频率分布
- ❑ 相关
- ❑ 名义比较
- ❑ 地理或地理空间

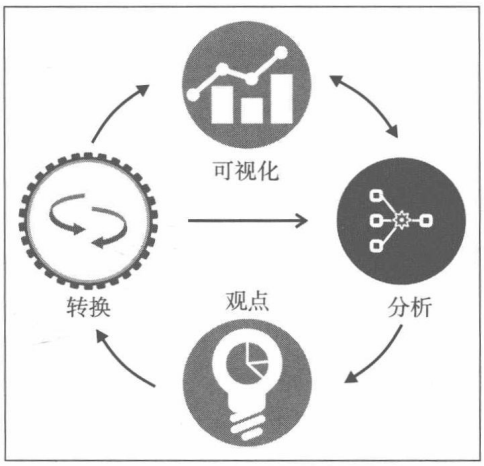
科学家已经绘制出人类基因组，这是我们面临将知识转换为可视化以求更好理解的挑战。换句话说，我们可能不得不找到从视觉上呈现人类基因组的新方法，使得普通人也能理解。

1.4.1　可视化适用于哪里

需要强调的是，数据可视化不是科学的可视化。科学可视化处理的数据本身固有一种物理结构，比如流过飞机机翼的空气分子。另一方面，信息可视化处理抽象数据，帮助解决大量数据集问题。挑战之一是确保数据是干净的，而且通过降维提出不必要的冗余信息。

可视化可以用于知识或数据价值增加的任何情况。通过做更多的数据分析和运行更多的算法即可完成。数据分析的形式可能由最简单变得更复杂。

有时，仅仅观察均值、中位数或总和无法获得真正的价值。这是因为这些测度指标仅仅测度了显而易见的东西。有时，一个区域的并集或数值隐藏着需要特别关注的有趣细节。一个经典的例子（Anscombe 四重奏）包括简单的统计性质几乎相同的四个数据集，但在图像中却截然不同。如果想了解更多，请见链接：https://en.wikipedia.org/wiki/Anscombe%27s_quartet。



在大多数情况下，数据集的可视化可能有不同的形式，但是总会有一些人能够画出比其他人更清晰的图片来帮助理解。在一些情况下，必须通过多次分析来得到可视化的更好理解，如上图所示。

良好的可视化不仅仅能看到如博物馆展览一样的静态图。我们还可以深入挖掘数据，发现更多变化。(通过循序渐进地查看，收缩和过滤，改变展示标尺，再得到可视化结果)。如由 Ben Shneiderman 提供的资料（http://www.mat.ucsb.edu/~g.legrady/academic/courses/11w259/schneiderman.pdf）所示，得到有时在同一图中，以同一个标尺展示所有信息非常有难度，而且用户可以通过个人经验更好地理解这些可视化方法。进一步总结，特别是当数据充足时，可视化在组织和提炼数据方面很有用处。

交互式可视化成为一种新的沟通交流形式，它允许用户分析信息以便创建自己对数据的新理解。

1.4.2 如今的数据可视化

虽然很多计算领域旨在用自动化取代人工判断，但是可视化系统是独一无二的，而且明确设计为无法取代人类。事实上，可视化系统是为确保人类在整个参与过程中的积极性而设计的，这是为什么呢?

数据可视化是在各种计算工具帮助下受数据驱动并由人类创造的一门艺术。一位艺术家用工具和材料（像刷子和颜料）绘制一幅画。同样，另一位艺术家尝试在计算工具的帮助下创建数据可视化。可视化可以是美观的，并有助于使事情更清晰；根据不同的创建者，有时会缺乏上述一个或两个特点。

如今，数据的可视化展示方式已超过 30 种，每一种特定的方式都有它的用处。正因为可视化方法不断发展进步，我们已经不局限于柱状图和饼状图。数据可视化有很多好处，但它们往往因为缺乏理解而有所不足。在一些情形下，同一个图上聚集太多东西往往会使

得整个构图纷繁复杂。

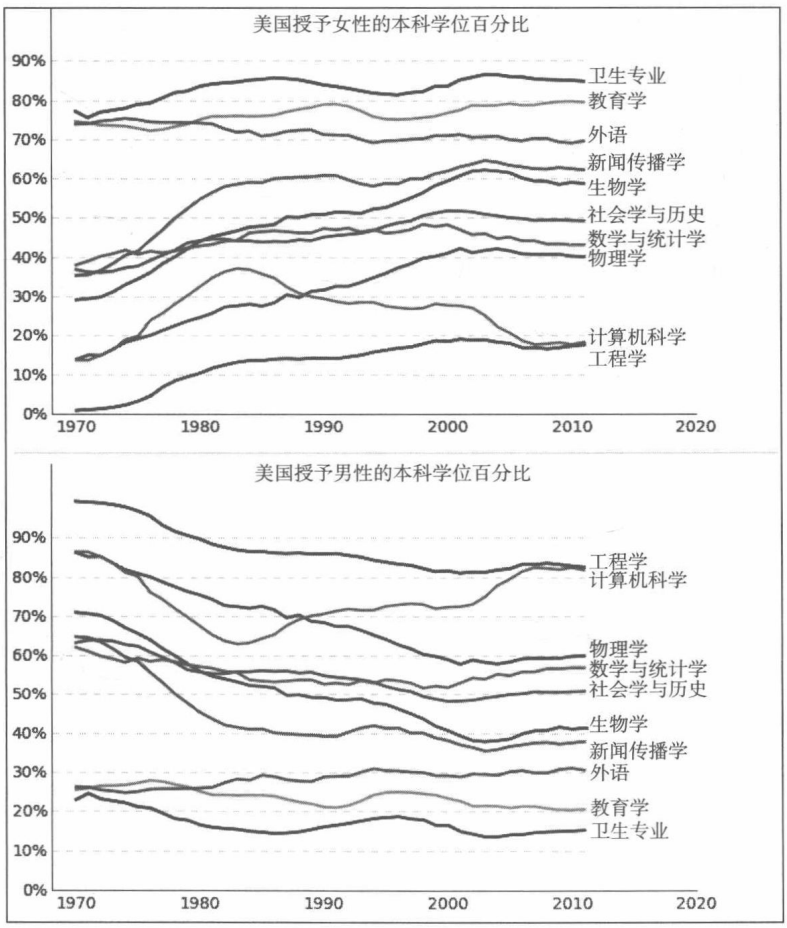
有很多呈现数据的方法，但仅有一小撮适用于大多数情况，这将在本章后面的部分详细讲述。在讨论之前，不妨先看看得到好的可视化效果有哪些重要的注意事项。

什么是好的可视化

好的可视化结果有助于用户探索和理解数据，提供价值和深刻的观点。它是有效的、具有视觉吸引力的、可伸缩的而且容易理解的（好的可视化不必太复杂）。通过开展研究和分析工作，可视化是发现数据模式和趋势的核心工具，我们使用其中的任何一种方法都能够回答数据问题。

有效的可视化背后的主要原则是能够突出你想表现的主要问题，根据观众的层次和背景，精确呈现数据并创造出能够清晰传达信息的可视化结果。

举例：下图来自小样本数据源，该数据展示了1970~2012年10门学科中女性和男性被授予学位的百分比（womens-undergrad-degrees.csv和mens-undergrad-degrees.csv来自http://www.knapdata.com/python/）：



所有数据来源可见 http://nces.ed.gov/programs/digest/d11/ tables/dt11_290.asp，包含全部数据集。

尽管不同学科间授予学位的数量彼此没有关系，但有一种简单的方法，用同一个标尺展现所有学科。让我们分析和观察一下，这种展示方式是否可行，如果不可行，我们需要做什么？还有其他展示方式吗？

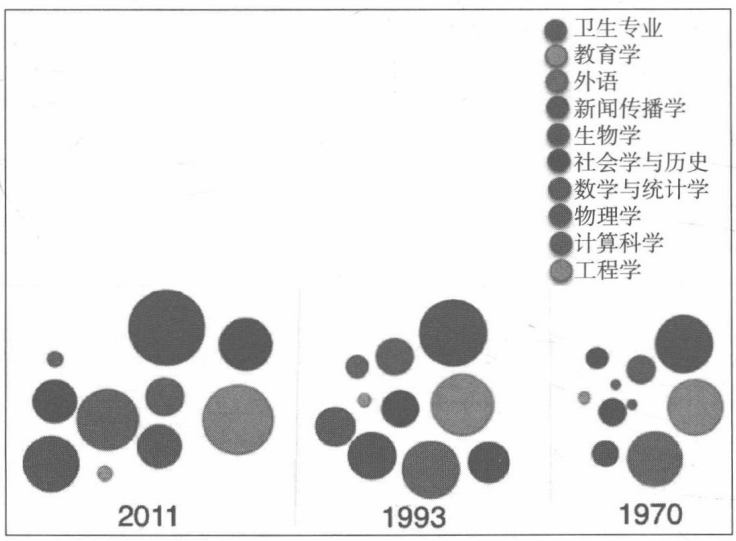
一方面，所有学科数据展示在同一幅图上形成了一个很好的对比。然而，我们不能直接得到 2000 年的信息。除非有一种类似于金融股票图的交互式展示模式，否则没有一种简单的方法来确定 2000 年各学科的学位授予信息。另一方面，同一学科授予男性和女性学位的百分比共占 100%；比如，男性在卫生专业被授予学位的百分比占 15.2%，女性占 84.8%。

我们还有其他可视化方法吗？可以考虑就每年都创建一个气泡图，加入年份因素，通过设置按钮自助切换不同年份的气泡图，完成交互式的可视化图像。

这种可视化方式更适用于该数据。我们也可以用与原始图相同的滑块，通过强调所选年份的数据实现交互。用不同的方法来观察一些图是否比其他图更好，这是一种好习惯。如果数据的数值区间很大（比如，20 ~ 200 000），那么我们可能不得不将数值进行对数处理。

我们可以用 Python 编程绘制气泡图。此外，也可考虑用 D3.js 的 JavaScript 语言和 RStudio 的 R 语言。读者可以进一步探索其他可视化的选择。

可以用 Google Motion 图来可视化，在 developers.google.com/chart/interactive/docs/gallery/motionchart?csw=1#Example 呈现的交互式图表，这里展示了与棋牌图类似的一个工作示例。下面的气泡图仅展示三年的情况，但是你可以创建另一个图展示所有年份的情况。



数据可视化是数据分析后的一个过程。在前面，我们也注意到数据转换、数据分析和数据可视化已被多次尝试。为什么会这样？我们都知道有这样一句名言，“有知识的人给出正确的答案，聪明的人提出正确的问题。”数据分析有助于我们更好地理解数据，因此，数据分析应用于回答有关数据的问题。然而，当数据用不同方式进行视觉展示时，一些新问题就会出现，而且这也是要重复分析和可视化的原因之一。

数据可视化是数据探索的主要途径，而且几乎总是先于或引发数据分析。有很多数据的视觉展示工具，但是用于分析的工具却少之又少。像 Julia、R 和 Python 这些编程语言在表现数据分析方面排名靠前，但是就可视化而言，基于 D3.js 的 JavaScript 在生成交互式数据可视化方面具有更大的潜力。

与 Python 相比，学习 R 语言相对较难。关于这点，Quora 上也有一些争论；你可以在网站（https://www.quora.com/Which-is-better-for-data-analysis-R-or-Python）上验证其有效性。现在，Python 有很多统计建模和数据分析工具，因此，成为了研究数据科学的一种颇具吸引力的选择。

1.5　可视化图像

我们实现可视化是想证实我们对数据的认识。然而，如果数据不容易理解，你可能不会设计出正确的问题。

创建可视化的第一步是弄清楚需要回答的问题。换言之，该可视化的作用如何？另一个挑战是学习正确的绘图方法。下面列出一些可视化方法：

- 条形图和饼图
- 箱线图
- 气泡图
- 直方图
- 核密度估计（Kernel Density Estimation，KDE）图
- 线面图
- 网络图
- 散点图
- 树状图
- 小提琴图

在识别可视化应该传达信息的过程中，关注下面的问题才有意义：

- 要处理多少变量？我们试图画出什么样的图像？
- x 轴和 y 轴指代什么？（三维图中还有 z 轴）

- 数据大小被标准化了吗？数据点的大小意味着做？
- 我们的选色正确吗？
- 对于时间序列数据，我们是否试图识别趋势或相关性？

如果变量太多，可以在同一个图上画多个不同组数据实例。这个技术叫作点阵或网格绘图。它允许观众快速提取复杂数据的大量信息。

考虑一组学生数据，这是（gender、sleep、tv、exercise、computer、gpa）和（height、momheight、dadheight）信息的不寻常混合。computer、tv、sleep 和 exercise 的单位是小时，身高以英寸[⊖]为单位，gpa 的测度标尺为 4.0。

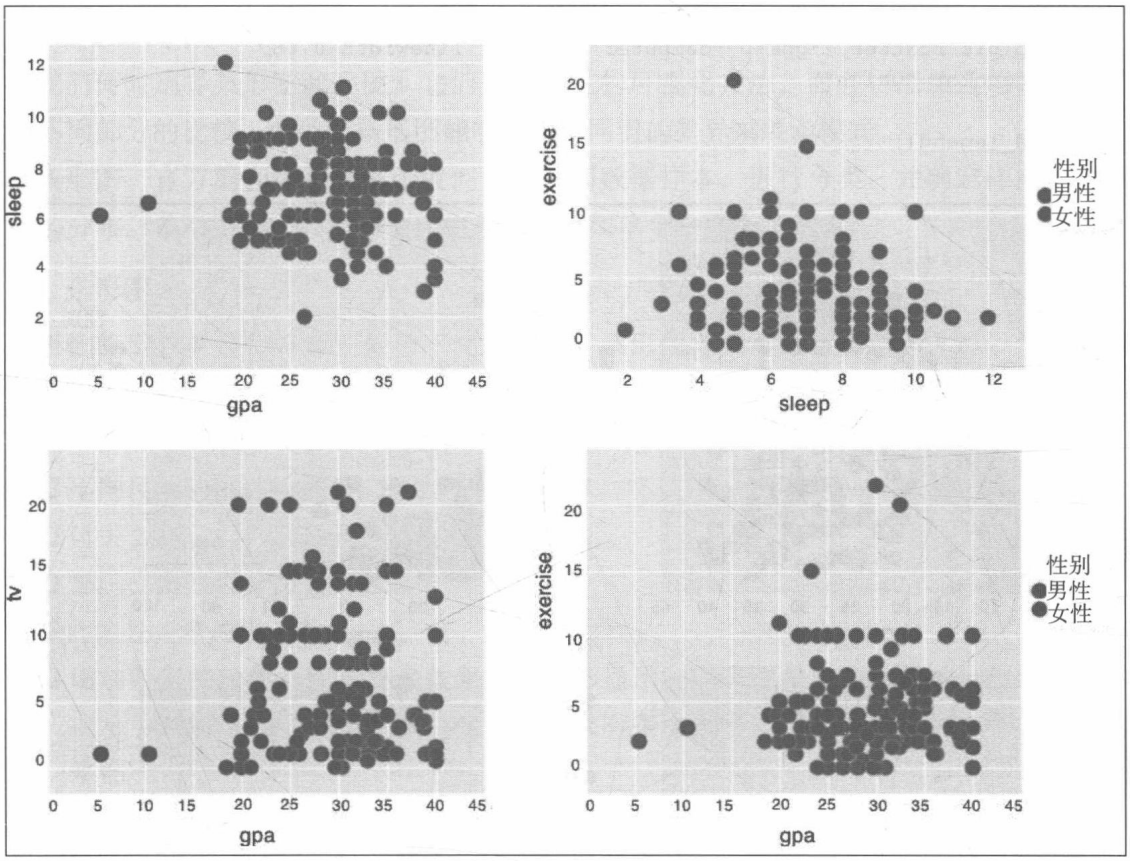
gender	tv	computer	sleep	height	momheight	dadheight	exercise	gpa
Female	13.0	10.0	3.50	66.0	66.0	71.0	10.0	4.000
Male	20.0	7.0	9.00	72.0	64.0	65.0	2.0	2.300
Male	15.0	15.0	6.00	68.0	62.0	74.0	3.0	2.600
Male	8.0	20.0	6.00	68.0	59.0	70.0	6.0	2.800
Female	2.5	10.0	5.00	64.0	65.0	70.0	6.5	2.620
Male	2.0	14.0	9.00	68.5	60.0	68.0	2.0	2.200
Female	4.0	28.0	8.50	69.0	66.0	76.0	3.0	3.780
Female	8.0	10.0	7.00	66.0	63.0	70.0	4.5	3.200
Male	1.0	15.0	8.00	70.0	68.0	71.0	3.0	3.310
Male	8.0	25.0	4.50	67.0	63.0	66.0	6.0	3.390
Male	3.5	9.0	8.00	68.0	62.0	64.0	8.0	3.000
Female	11.0	20.0	5.00	68.0	64.0	69.0	0.0	2.500
Male	10.0	14.0	8.00	68.0	61.0	72.0	10.0	2.800
Male	1.0	84.0	5.00	61.0	62.0	62.0	3.0	2.340
Female	10.0	11.0	9.00	65.0	62.0	66.0	5.0	2.000

上述数据是变量个数多于平常的案例。因此，用网格绘图进行可视化，来展示这些变量间的关系会更有意义。

我们进行可视化的一个原因是为了验证数据的知识。然而，如果没有很好地理解数据，就不可能提炼出正确的问题。

数据中包含两种性别，因此作为第一个变量集有 10 组变量，它们可能是：(sleep, tv)、(sleep, exercise)、(sleep, computer)、(sleep, gpa)、(tv, exercise)、(tv, computer)、(tv, gpa)、(exercise, computer)、(exercise, gpa) 以及（computer, gpa）；另外两个（height，momheight）和（height，dadheight）作为第二个变量集。下面是除 (sleep, tv)、(tv, exercise) 以外的所有组合。

⊖ 1 英寸 =2.54 厘米。



我们的目标是发现哪些变量组合能用来理解数据，或者这些变量中是否有一些存在有意义的影响。因为是关于学生的数据，gpa 可能是影响其他变量的关键所在。前面的图像刻画了这样一幅散点图：有很多女生的 gpa 比男生高，在同一个 gpa 范围内，有很多男生花在计算机上的时间更多。尽管这里给出所有散点图，不过我们的目的是：发现更有用的数据，以及从该数据中得出有用的结论。

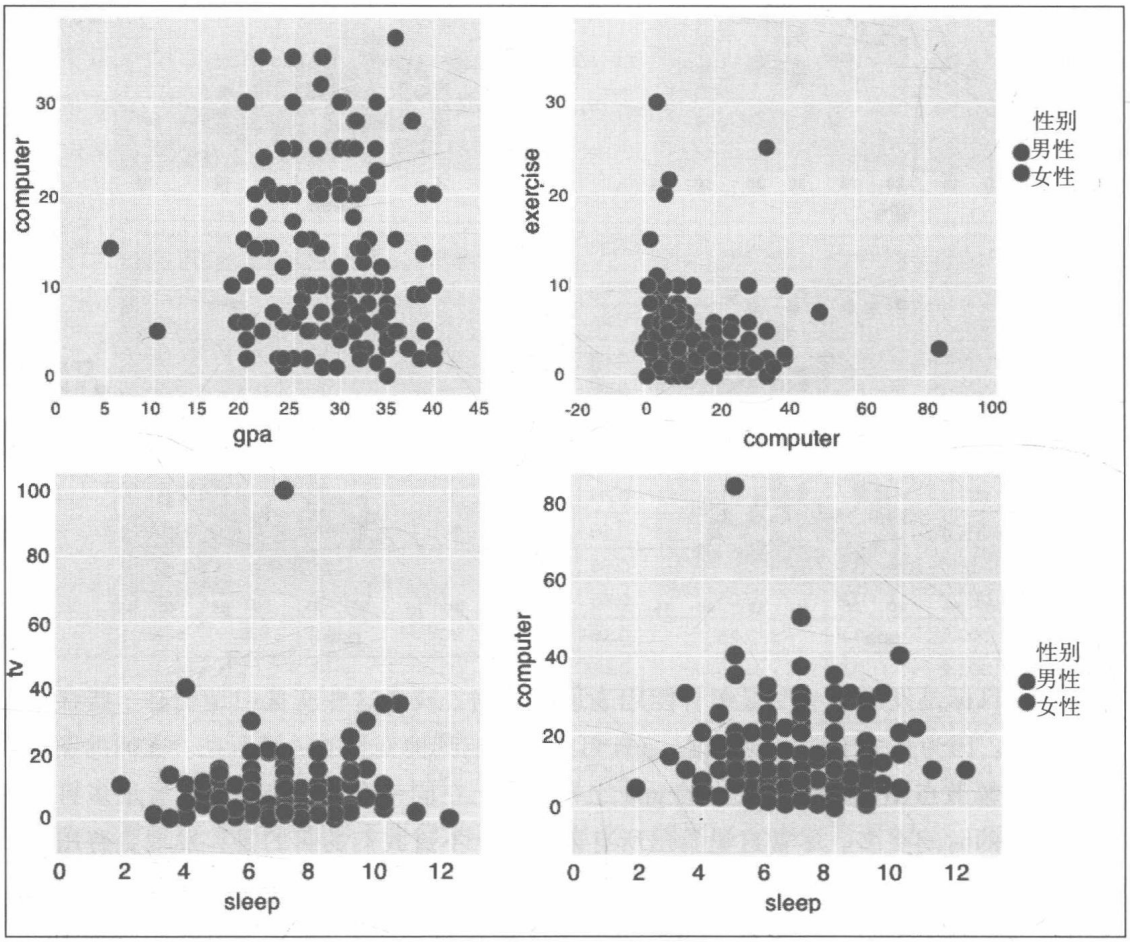
很多蓝色点处于较高的位置（y 轴上的 gpa）说明更多女生 gpa 更高（该数据由 UCSD 收集而来）。

数据可从 http://www.knapdata.com/python/ucdavis.csv 下载。

你可以用 seaborn 软件包编写很少的代码绘制散点图，下面的例子说明与学生花费在计算机上的时间相比，沿 x 轴的 gpa 的散点图：

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

students = pd.read_csv("/Users/kvenkatr/Downloads/ucdavis.csv")
```

```
g = sns.FacetGrid(students, hue="gender", palette="Set1", size=6)
g.map(plt.scatter, "gpa", "computer", s=250, linewidth=0.65,
  edgecolor="white")

g.add_legend()
```



这些图用 matplotlib、pandas 和 seaborn 库软件包生成。seaborn 是一种基于 matplotlib 的统计数据可视化库，由斯坦福大学的 Michael Waskom 创建。这些库的细节将在后面几章讨论。

seaborn 库有很多有用的类。尤其是当我们需要对数据子集中一个变量的分布或者多个变量间关系进行可视化时，FacetGrid 类就派上用场了。FacetGrid 可以刻画三个维度：行、列和色调。这些库软件包和它们的功能将在后面章节中进行描述。

当创建可视化时，第一步是清楚需要回答的问题。换句话说，可视化起什么作用？另一个挑战是选择正确的绘图方法。

1.5.1 条形图和饼图

我们何时选择条形图和饼图？它们是最古老的可视化方法，而且饼图最适用于一个整体中不同部分的比较。此外，条形图能够比较不同组的差异来展示模式。

条形图、直方图和饼图有助于我们比较不同数据样本，进行分类，并确定样本中数据取值的分布。条形图有一些不同的风格形式，比如单个、多个和堆叠。

1. 条形图

当数值型数据被很好地划分为不同类别时，条形图尤其有效。因此，你可以快速地从数据中看到趋势。

当比较不同类别的数据时，条形图很有用。下面是一些著名的案例：

- 不同尺寸的牛仔裤数量
- 过去 20 年世界人口的变化
- 部门支出的百分比

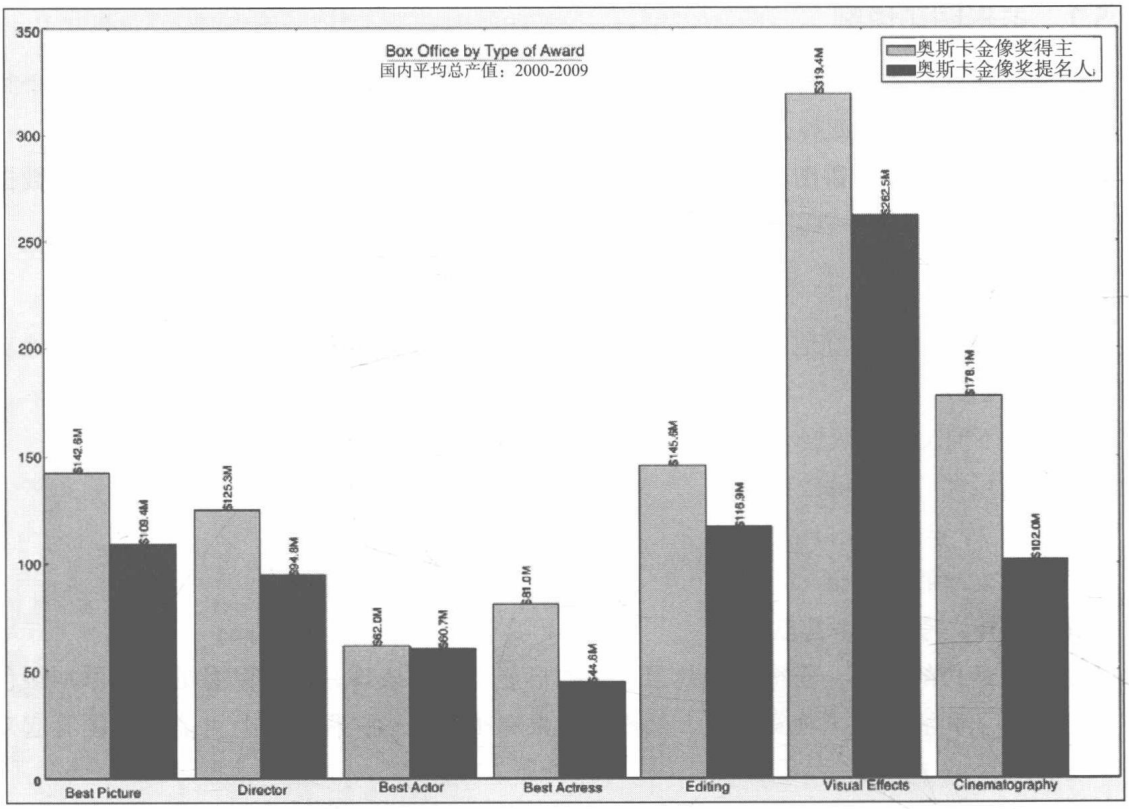
除此之外，考虑以下几点：

- **增加条形的颜色，获得更多效果：**用条形显示利润表现是有信息量的，但添加颜色揭示了利润增加了视觉观点。然而，如果有很多条形，很多颜色可能使得图看起来笨拙。
- **包括仪表盘上的多个条形图：**这有助于观众快速比较相关信息，而不是通过一堆表格或幻灯片来回答问题。
- **在轴的两侧加条形：**沿连续数轴画正反两面的数据是一种发现趋势的有效方法。
- **用层叠或并排的条形：**在彼此上部或旁边展示相关数据，给出你的深入分析，马上处理多个问题。

这些图通过不到 12 行的 Python 代码来实现，后面几章将讨论更多的案例。

条形图中，每一列代表由特定分类定义的一个组；直方图中，每一列代表由定量变量定义的一组。在条形图中，x 轴没有最小值和最大值，这是因为 x 轴上的标签是分类的，而不是定量的。另一方面，在直方图中有一个区间的取值。下面的条形图展示了 2002 ~ 2009 年美国奥斯卡金像奖得主和提名人的统计特征：

下面的 Python 代码用 matplotlib 展示了电影小数据样本的条形图（这可能不一定是一个真实的案例，但却给出了绘图板和比较的想法）：




```
[5]: import numpy as np
     import matplotlib.pyplot as plt

     N = 7
     winnersplot = (142.6, 125.3, 62.0, 81.0, 145.6, 319.4, 178.1)

     ind = np.arange(N)  # the x locations for the groups
width = 0.35        # the width of the bars

fig, ax = plt.subplots()
winners = ax.bar(ind, winnersplot, width, color='#ffad00')

nomineesplot = (109.4, 94.8, 60.7, 44.6, 116.9, 262.5, 102.0)
nominees = ax.bar(ind+width, nomineesplot, width,
  color='#9b3c38')

# add some text for labels, title and axes ticks
ax.set_xticks(ind+width)
ax.set_xticklabels( ('Best Picture', 'Director', 'Best
Actor',
  'Best Actress','Editing', 'Visual Effects',
  'Cinematography'))
```

```
ax.legend( (winners[0], nominees[0]), ('Academy Award
Winners',
  'Academy Award Nominees') )

def autolabel(rects):
  # attach some text labels
  for rect in rects:
    height = rect.get_height()
    hcap = "$"+str(height)+"M"
    ax.text(rect.get_x()+rect.get_width()/2., height, hcap,
      ha='center', va='bottom', rotation="vertical")

autolabel(winners)
autolabel(nominees)

plt.show()
```

2. 饼图

说到饼图，需要考虑的问题是："这几部分能组成一个有意义的整体吗？"以及"你是否有充足的内容用圆形代表？"。使用饼图一直备受争议，主要是因为当类别种类过多时，使用饼图很难比较不同类别的比例来得出结论（来源：https://www.quora.com/How-and-why-are-pie-charts-considered-evil-by-data-visualization-experts）。

饼图适用于在一个空间或图上展示比例。一些著名的案例如下：

- 调查中的响应分类
- 一种特定技术的前五家公司的市场份额（这种情况下，我们可以很快知道哪家公司占据主要的市场份额）

除此之外，还需考虑以下几点：

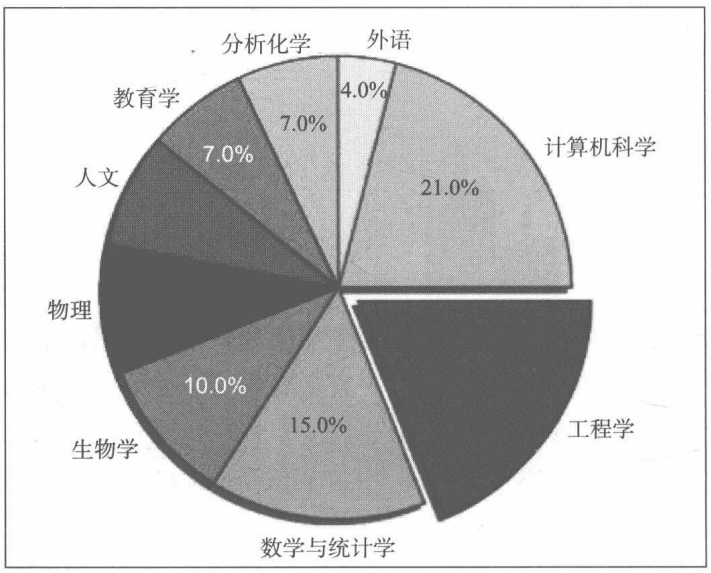
- **限制饼图的扇形数不超过 8 个**：如果有 8 个以上比例需要展示，请考虑条形图。出于对内容的限制，饼图很难有意义地展示和阐释过多的部分。
- **在图上叠加饼图**：在同一幅图中，饼图更容易拓展，凸显地域发展趋势。（此处扇形个数仍有限制。）

考虑下面的代码，绘制一个简单的饼图，比较不同学科的录取情况分布：

```
[6]: import matplotlib.pyplot as plt

     labels = 'Computer Science', 'Foreign Languages',
       'Analytical Chemistry', 'Education', 'Humanities',
       'Physics', 'Biology', 'Math and Statistics', 'Engineering'

     sizes = [21, 4, 7, 7, 8, 9, 10, 15, 19]
     colors = ['yellowgreen', 'gold', 'lightskyblue',
     'lightcoral',
```

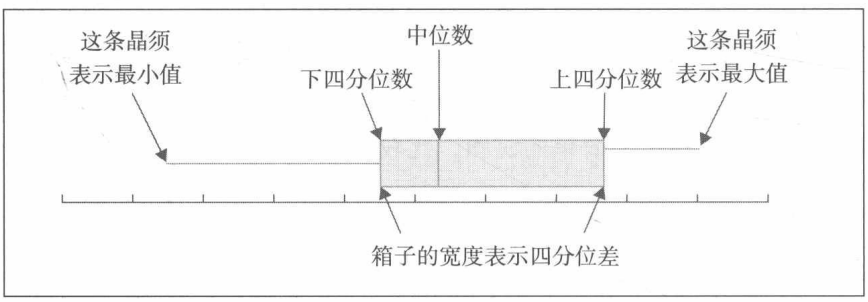
```
  'red', 'purple', '#f280de', 'orange', 'green']
explode = (0,0,0,0,0,0,0,0,0.1)
plt.pie(sizes, explode=explode, labels=labels,
  autopct='%1.1f%%', colors=colors)
plt.axis('equal')
plt.show()
```

下面的饼图展示了大学里一些热门专业的录取情况：



1.5.2 箱线图

箱线图也叫作箱须图。这是一种基于最小值、上四分位数、中位数、下四分位数和最大值 5 个数值特征展示数据分布的标准方式。下图完成了对箱线图的解读：



箱线图是一种从图形上检查一组或多组数据的快捷方式。它们占用了较少的空间，在同一时间定义五种特征。关于箱线图的用途，我们能想到的一个例子是：如果两个以上班级参加同一次考试，则箱线图能够告诉我们哪个班的大多数同学比其他班做得好。另一个

例子是：如果有更多的人吃汉堡，那么中位数的位置会上移或表示最大值处的上晶须比表示最小值处的下晶须更长。在这样的情况下，我们给出了数据分布的一个好的可视化工具。

在理解箱线图适用范围之前，我们首先给出一个定义。离群点是一组数据值中与其他值有异常距离的观测值。

箱线图适用于展示一组数据的分布。一些著名的案例如下：

- 识别数据中的离群点
- 确定数据是左偏还是右偏

除此之外，还有以下问题需要考虑：

- **箱子里隐藏了一些点**：关注离群点
- **跨分布比较**：箱线图适用于快速比较不同数据集的分布

1.5.3 散点图和气泡图

散点图是一种用来展示两个变量之间二维关系的可视化方法。这种交叉数对的模式能从图像上展示不同变量间的关系。散点图是同一组研究对象的两个变量间关系的可视化。另一方面，气泡图展示了数据的三个维度。每个数据点有三重维度 (a, b, c)。其中，xy 轴的坐标表示两个维度变量，气泡的大小表示第三个维度的定量测度结果。

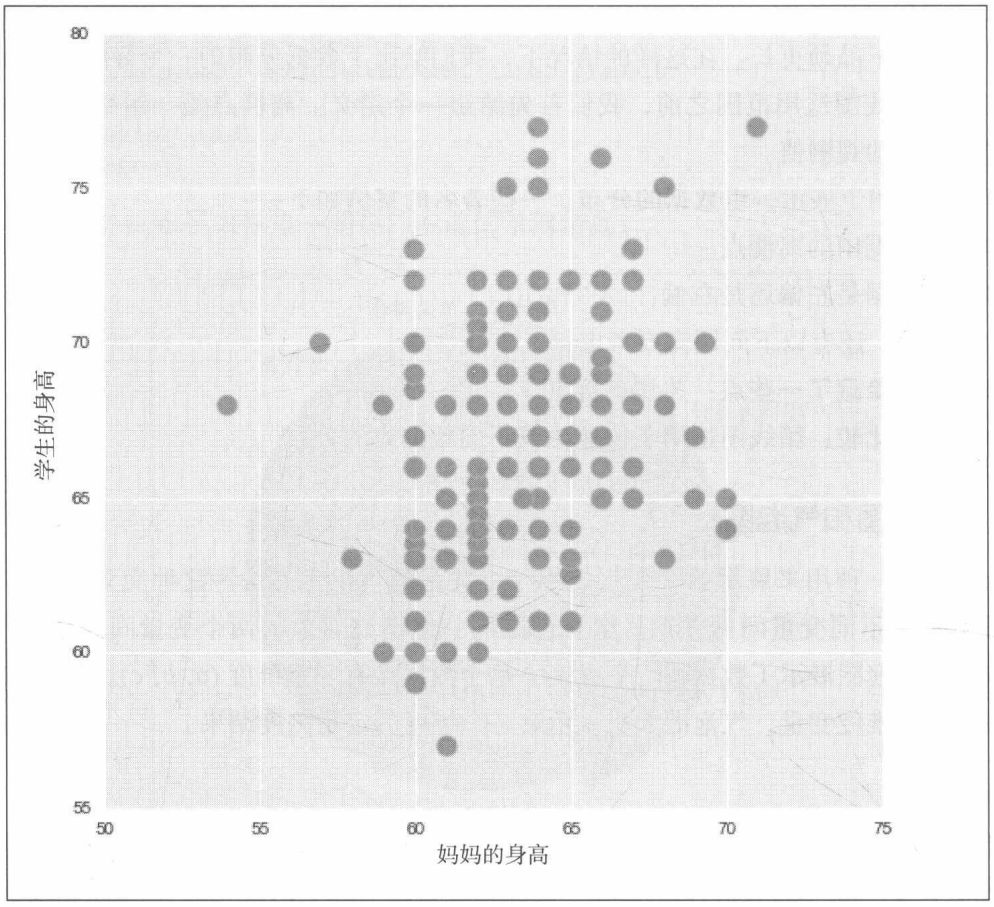
1. 散点图

数据通常是一组点的集合，并且常常用来绘制各种相关性。比如，正相关表示：当一组数据递增时，另一组数据也会增加。前面给出的学生记录数据通过各种散点图展示出它们间的相关性。

在下面的例子中，我们用学生妈妈的身高与学生的身高进行比较，来确定是否他们之间存在正相关关系。数据下载网址为：http://www.knapdata.com/python/ucdavis.csv。

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
students = pd.read_csv("/Users/Macbook/python/data/ucdavis.csv")
g = sns.FacetGrid(students, palette="Set1", size=7)
g.map(plt.scatter, "momheight", "height", s=140, linewidth=.7,
edgecolor="#ffad40", color="#ff8000")
g.set_axis_labels("Mothers Height", "Students Height")
```

我们用 seaborn 软件包处理这个案例，但也可以通过下节所示的 matplotlib 完成。上述代码绘制的散点图如下所示：



散点图最适用于研究不同变量间的关系。一些著名的案例如下：

- 男性与女性人群中不同年龄阶段得皮肤病的可能性
- IQ 测试得分和 GPA 之间的相关性

除此之外，还有以下问题需要考虑：

- **增加一条趋势线或最佳拟合线（如果关系是线性的）**：增加一条趋势线可以展示数据之间的相关性。
- **使用信息标记类型**：信息标记类型适用于通过形状和颜色提高视觉效果来解读数据的情况。

2. 气泡图

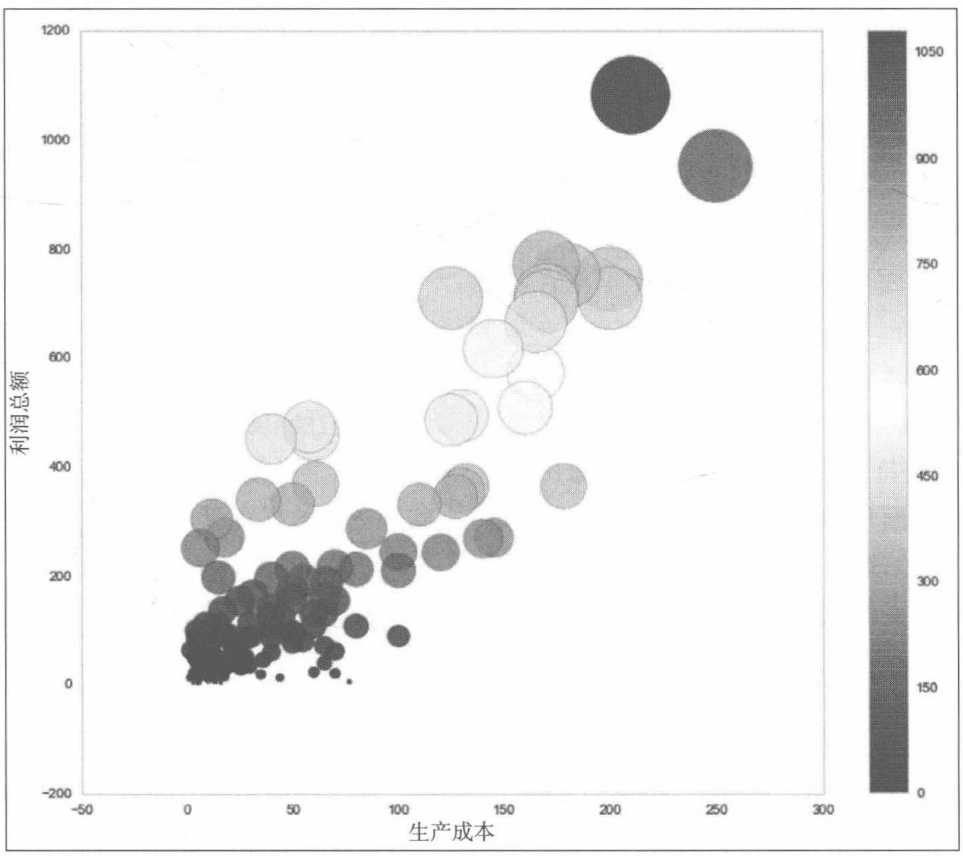
下面的例子展示了如何用颜色作为第三个维度，揭示销量或任何其他驱动收益的指标：

```
[7]: import numpy as np
     import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
```

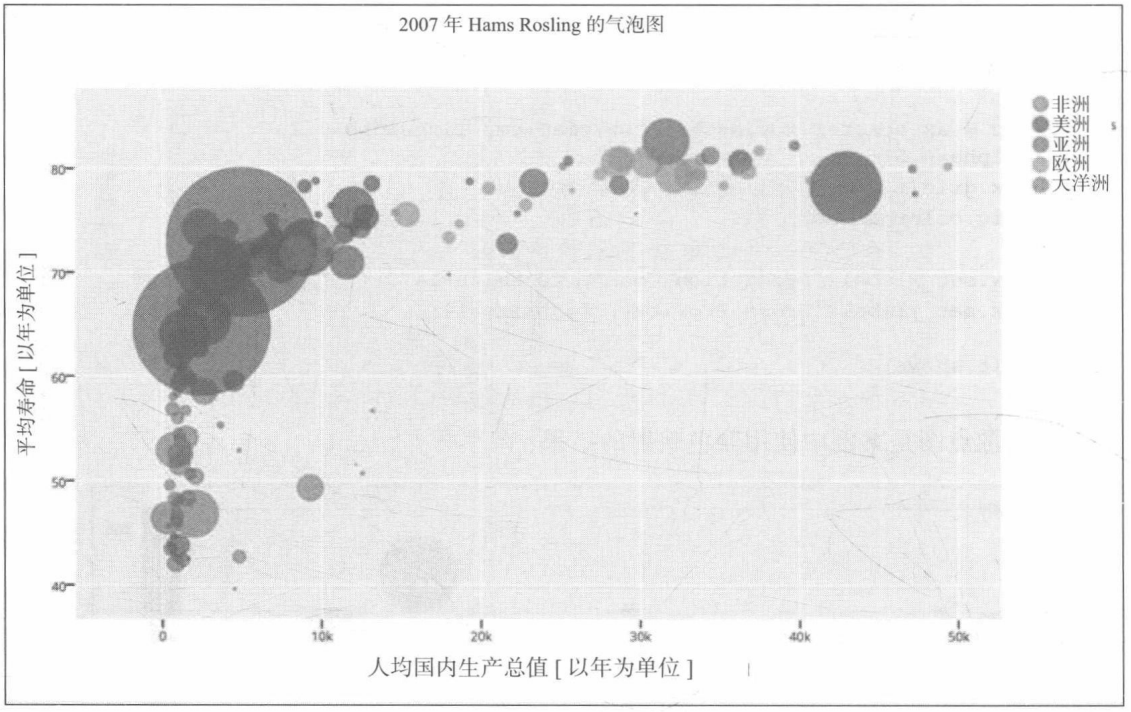
```
sns.set(style="whitegrid")
mov =
pd.read_csv("/Users/MacBook/python/data/2014_gross.csv")

x=mov.ProductionCost
y=mov.WorldGross
z=mov.WorldGross

cm = plt.cm.get_cmap('RdYlBu')
fig, ax = plt.subplots(figsize=(12,10))

sc = ax.scatter(x,y,s=z*3, c=z,cmap=cm, linewidth=0.2,
alpha=0.5)
ax.grid()
fig.colorbar(sc)

ax.set_xlabel('Production Cost', fontsize=14)
ax.set_ylabel('Gross Profits', fontsize=14)

plt.show()
```

...

下面的散点图是案例中使用颜色映射的结果：



在用三维数值空间比较数据之间的关系时，气泡图大有帮助，它包括：x 轴数据、y 轴数据，并用气泡大小表示数据。气泡图有些像 XY 散点图，但是散点图上每个点都追加了一个额外的数据取值信息，用以 XY 点为中心的圆圈或气泡的大小表示。气泡图的另一个例子如下所示（没有 Python 代码，只是用来说明一种可视化风格）：



在上面的研究中，气泡图展示了不同大洲的平均寿命和人均国内生产总值。

气泡图最适用于展示沿两个坐标轴的数据信息，以及第三个显著测度的数据元素。一些著名的案例如下：

- 电影的制作成本和毛利润，以及沿递增标尺的显著性测度

除此之外，还有下面问题需要考虑：

- **增加颜色和形状的显著性**：通过比较大小和颜色，数据点可以转换为问题答案的可视化结果
- **变为交互式**：如果数据点过多，则气泡图会变得很复杂，因此需要在时间轴或类别上，对这些数据进行分组，完成交互式的可视化

1.5.4 核密度估计图

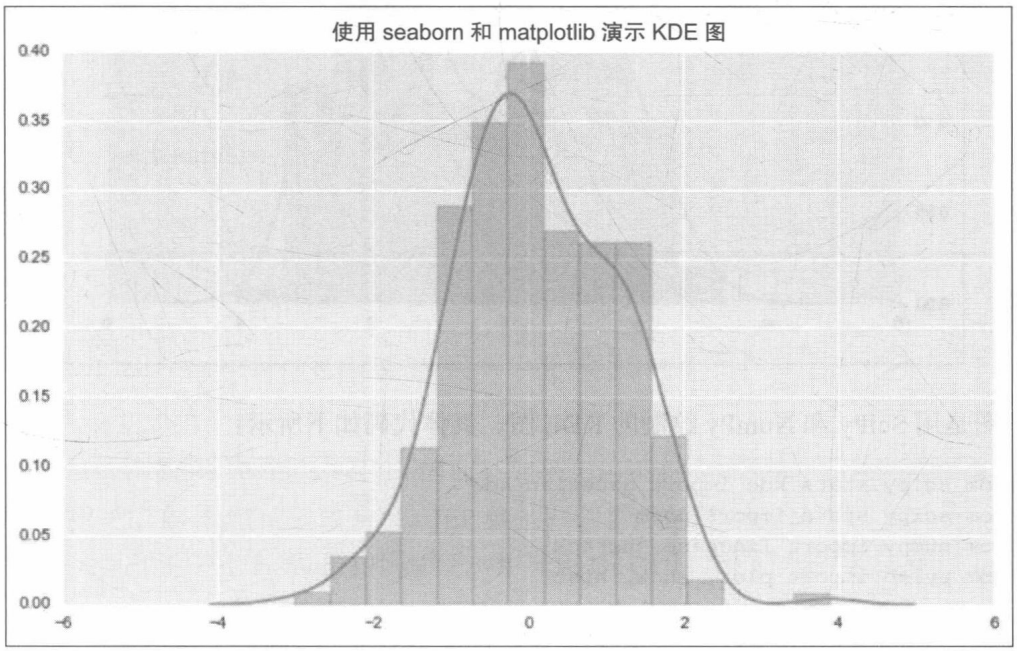
核密度估计（Kernel Density Estimation，KDE）是一种用来估计概率密度函数的非参数方法。可以通过对观测到的数据点取平均实现平滑逼近。核密度函数与直方图密切相关，

但有时能够通过核概念用平滑性或连续性赋予实际含义。

概率密度函数（Probability Density Function，PDF）的核是 PDF 的形式。这种形式不考虑非变量函数因素。本书仅关注概率密度函数的可视化，如果想了解更多理论，可以参考统计学方面的书。

Python 包含能够用来在各种深度和层次完成一个 KDE 图的库，包括 matplotlib、Scipy、scikit-learn 和 seaborn。下面是 KDE 绘图的两个例子。后面几章会有更多的例子展示 KDE 图的各种其他方式。

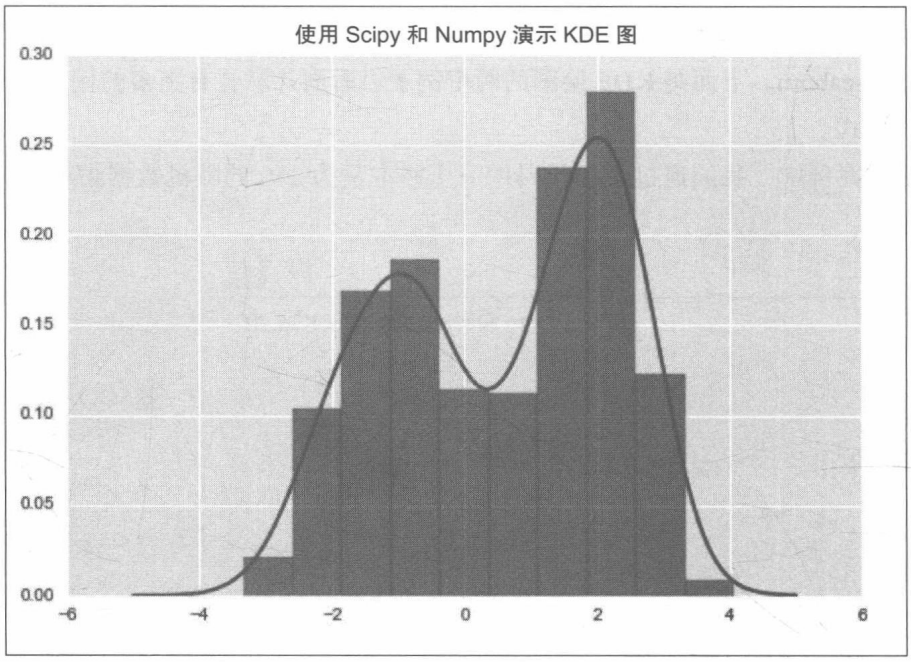
在下面的案例中，我们通过少量代码用一个样本量为 250 的随机数据集和 seaborn 软件包展示分布图：



通过 numpy.random 生成一个随机样本，可以用 seaborn 展示数据图的简单分布：

```
from numpy.random import randn
import matplotlib as mpl
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_palette("hls")
mpl.rc("figure", figsize=(10,6))
data = randn(250)
plt.title("KDE Demonstration using Seaborn and Matplotlib",
fontsize=20)
sns.distplot(data, color='#ff8000')
```

在第二个例子中，我们用 SciPy 和 NumPy 表明概率密度函数。首先我们用 SciPy 中的 norm() 创建正态分布样本，而后用 NumPy 中的 hstack() 进行水平方向上的堆叠，并应用 SciPy 中的 gaussian_kde()。



上图是用 SciPy 和 NumPy 绘制的 KDE 图，具体代码如下所示：

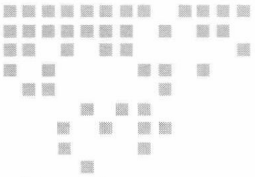
```
from scipy.stats.kde import gaussian_kde
from scipy.stats import norm
from numpy import linspace, hstack
from pylab import plot, show, hist

sample1 = norm.rvs(loc=-1.0, scale=1, size=320)
sample2 = norm.rvs(loc=2.0, scale=0.6, size=320)
sample = hstack([sample1, sample2])
probDensityFun = gaussian_kde(sample)
plt.title("KDE Demonstration using Scipy and Numpy", fontsize=20)
x = linspace(-5,5,200)
plot(x, probDensityFun(x), 'r')
hist(sample, normed=1, alpha=0.45, color='purple')
show()
```

其他可视化方法，比如线面图、网络图、树状图、热力图、雷达图或蜘蛛图，以及小提琴图，将在后续章节进行讨论。

1.6 总结

迄今为止，这些案例仅仅给你一个在可视化前应该如何思考和规划的想法。最重要的阶段是为可视化开展的数据熟悉和准备过程。尝试后的结局会影响能否率先得到数据或者形成意欲表达的故事。就像“鸡和鸡蛋”的情况——数据是先来还是后到？最初，可能不清楚需要什么数据，但在大多数情况下，只要数据没有错误，经过几次反复总能使问题变得清晰。

通过清理或降低维度（如果需要），可以转换数据质量填补空白。除非数据质量有保证，否则一个人投入可视化的努力就白费了。正确理解数据后，就可以决定选择哪种可视化更合适。在某些情况下，用不同的方式更有助于清晰地讲述数据的故事。



Chapter 2 第 2 章

数据分析与可视化

大多数可视化故事是围绕问题或话题展开的：数据探索或收集的起源。这问题包含了整个故事的起因，构成整个故事。这样的数据征程以一个问题开始，比如，2014 年，报道的伊波拉病毒死亡人数是多少？回答这个问题需要一个彼此协作的团队完成。数据传播者的作用应该是创造一种转变观众看法的经历。

故事的关键在于有意义的可视化过程。这些可视化的内容回答了下面几个问题：

- 数据充足吗？
- 有这个数据存在的时间窗吗？
- 全球哪些相关的事件会影响数据？

需要重申的是，在理解数据的基础上识别出我们试图回答的问题非常重要。有时，在确定最终的问题前，可以先开始挖掘数据。在这种情况下，提炼对数据的理解可能会得到一个改进后更清晰的问题。

在有现成的获取、分析和收集所需信息方法的前提下，这个过程就从输入数据开始。还有一些情况，最好能够通过可视化收集来的信息来消除噪音，而在另外一些情况下，可以在可视化之前先进行数据过滤和数据分析。本章我们将学习不同的数据探索方法，为可视化做准备。下面是我们需要回顾的一些有趣的故事和相关的概念：

- 获取、解析和过滤数据、探测离群点和异常值、数据挖掘和提炼、可视化呈现以及交互
- 用数据讲述有趣的故事
- 感知、演示方法和可视化的最佳实践

❑ 交互式可视化——探索事件的听众和布局

2.1　为什么可视化需要规划

可视化的整个过程需要具有不同技能和专业领域知识的人。数据工人努力收集数据并完成分析。数学家和统计学家理解可视化设计原则，并用这些原则完成数据交流。设计师或艺术家在一些情况下，称为开发先驱者具备可视化所需的技能，而业务分析员在寻找顾客行为模式、离群点或突发趋势等。然而，这往往从获取或收集数据开始，步骤如下：

❑ **获得或收集数据**　这些数据来自外部资源、网站或磁盘上的文件

❑ **解析和过滤数据**　用编程方法进行解析、清洗和减少数据

❑ **分析和提炼数据**　删除噪音和一些不必要的维度，发现模式

❑ **呈现和交互**　用更容易得到和理解的方法展示数据

处理过程中需要做的工作因不同问题而异。在一些情况下，分析比过滤数据要做更多工作。上一章已讨论过，一些案例需要进行反复分析和可视化。换句话说，这些步骤的分布不总是可预测的和一致的。

2.2　Ebola 案例

为了说明上述步骤如何得到容易理解的可视化结果，不妨考虑一下我们先前提出的问题：2014 年，报道的伊波拉病毒（Ebola）死亡人数是多少？数据来自世界卫生组织（http://www.who.int/en/）或者人道主义数据交换中心（https://hdx.rwlabs.org）。虽然该数据的原始来源是世界卫生组织（World Health Organization，WHO），但人道主义数据交换中心（Humanitarian Data Exchange，HDX）是贡献者。然而，请注意，我们将在同一个地方得到本书的所有数据，以及 Python 资源代码。

该数据包括 Ebola 疾病在几内亚、利比里亚、马里、尼日利亚、塞内加尔、塞拉利昂、西班牙、英国和美国的传播信息。

该信息由以下网址提供：https://data.hdx.rwlabs.org/dataset/ebola-cases-2014/。

CSV 格式的数据文件的内容包括指标、国家名称、日期和与指标相关的死亡数量或感染数量。一共有 36 个指标，前 10 个指标如下（其他指标可见附录）：

❑ 过去 7 天可能的 Ebola 案例数

❑ 过去 21 天可能的 Ebola 死亡数

❑ 过去 21 天疑似 Ebola 案例数

❑ 过去 7 天疑似 Ebola 案例数

- 过去 21 天疑似 Ebola 死亡数
- 过去 21 天 Ebola 案例的确认比例
- 过去 7 天 Ebola 案例的确认比例
- 过去 21 天 Ebola 死亡的确认比例
- 过去 7 天疑似 Ebola 案例比例
- 过去 21 天疑似 Ebola 死亡比例

了解完这些指标后，本章一开始提出的问题“2014 年，报道的 Ebola 死亡人数是多少?”可以转换为多组问题。为了简单起见，我们持续关注一个问题，并观察如何才能够进一步分析这些数据，得到一个可视化方法。首先，让我们看一些数据文件的读取方法。

对于任何一种编程语言，读取文件的方法不止一种，有一种是用 Python 中的 pandas 库。这种方法有高性能的数据结构和数据分析工具。另一个选择是用 csv 库读取 CSV 格式的数据文件。这两种方法有什么不同呢? 它们都可以读取数据。在旧版的 pandas 中，会遇到大数据的记忆映射问题（如果 CSV 格式的数据文件很大），但是现在已完成优化。代码如下：

```
[1]:  with open('("/Users/kvenkatr/python/ebola.csv ', 'rt') as f:
        filtereddata = [row for row in csv.reader(f) if row[3] !=
"0.0" and
        row[3] != "0" and "deaths" in row[0]]

[2]:    len(filtereddata)
Out[2]: 1194
```

上面的过滤也可以用 pandas 完成，具体如下：

```
import pandas as pd
eboladata = pd.read_csv("/Users/kvenkatr/python/ebola.csv")
filtered = eboladata[eboladata["value"]>0]
filtered = filtered[filtered["Indicator"].str.contains("deaths")]
len(filtered)
```

数据可从 http://www.knapdata.com/python/ebola.csv 下载。下一步是用 read text (rt) 格式打开数据文件。因为指标字符串中有 deaths 这个单词，所以在读取每一行后，可以过滤死亡人数为 0 的那行。这是一种非常直接的过滤，被用于忽略没有报道的案例或死亡情况。下面只给出过滤后数据的前五行：

```
[3]:  filtereddata[:5]
Out[3]:
[['Cumulative number of confirmed Ebola deaths',
'Guinea','2014-08-29', '287.0'],
 ['Cumulative number of probable Ebola deaths','Guinea','2014-08-29',
```

```
  '141.0'],
 ['Cumulative number of suspected Ebola deaths','Guinea','2014-08-29',
  '2.0'],
 ['Cumulative number of confirmed, probable and suspected Ebola
deaths',
  'Guinea','2014-08-29','430.0'],
 ['Cumulative number of confirmed Ebola deaths',
  'Liberia','2014-08-29','225.0']]
```

如果每个国家报道的 Ebola 案例的所有数据是分散的，我们怎样进一步过滤？其实可以按国家列将它们进行分类。该数据文件有四行：indicator、country、date 和 number value。代码如下：

```
[4]:  import operator
      sorteddata = sort(filtereddata, key=operator.itemgetter(1))
[5]:  sorteddata[:5]
Out[5]:
[['Cumulative number of confirmed Ebola deaths',
'Guinea','2014-08-29', '287.0'],
 ['Cumulative number of probable Ebola deaths','Guinea','2014-08-29',
  '141.0'],
 ['Cumulative number of suspected Ebola deaths','Guinea','2014-08-29',
  '2.0'],
 ['Cumulative number of confirmed, probable and suspected Ebola
deaths',
  'Guinea','2014-08-29','430.0'],
 ['Number of confirmed Ebola deaths in the last 21 days', 'Guinea',
  '2014-08-29','8.0']]
```

看完数据后，有两个指标是我们感兴趣的：

- 确认的 Ebola 死亡累积数
- 确认的、可能的和疑似的 Ebola 死亡累积数

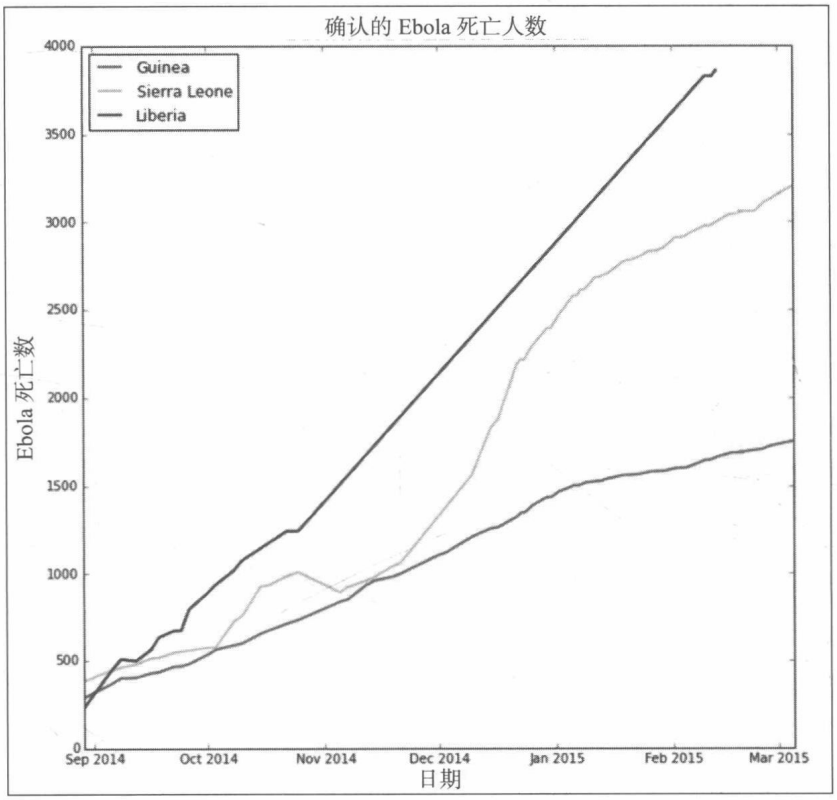
通过多次可视化应用，我们也注意到其中一些国家，几内亚、利比里亚和塞拉利昂比其他国家有更多被确认的死亡人数。现在，我们想看看应该怎样对报道中这三个国家的死亡人数进行绘图：

```
import matplotlib.pyplot as plt
import csv
import operator
import datetime as dt

with open('/Users/kvenkatr/python/ebola.csv', 'rt') as f:
  filtereddata = [row for row in csv.reader(f) if row[3] != "0.0" and
  row[3] != "0" and "deaths" in row[0]]
```

```
sorteddata = sorted(filtereddata, key=operator.itemgetter(1))
guineadata  = [row for row in sorteddata if row[1] == "Guinea" and
  row[0] == "Cumulative number of confirmed Ebola deaths"]
sierradata  = [row for row in sorteddata if row[1] == "Sierra Leone"
and
  row[0] == "Cumulative number of confirmed Ebola deaths"]
liberiadata = [row for row in sorteddata if row[1] == "Liberia" and
  row[0] == "Cumulative number of confirmed Ebola deaths"]

g_x = [dt.datetime.strptime(row[2], '%Y-%m-%d').date() for
  row in guineadata]
g_y = [row[3] for row in guineadata]

s_x = [dt.datetime.strptime(row[2], '%Y-%m-%d').date() for
  row in sierradata]
s_y = [row[3] for row in sierradata]

l_x = [dt.datetime.strptime(row[2], '%Y-%m-%d').date() for
  row in liberiadata]
l_y = [row[3] for row in liberiadata]

plt.figure(figsize=(10,10))
plt.plot(g_x,g_y, color='red', linewidth=2, label="Guinea")
plt.plot(s_x,s_y, color='orange', linewidth=2, label="Sierra Leone")

plt.plot(l_x,l_y, color='blue', linewidth=2, label="Liberia")
plt.xlabel('Date', fontsize=18)

plt.ylabel('Number of Ebola Deaths', fontsize=18)

plt.title("Confirmed Ebola Deaths", fontsize=20)

plt.legend(loc=2)


plt.show()
```

结果看起来如下图所示。

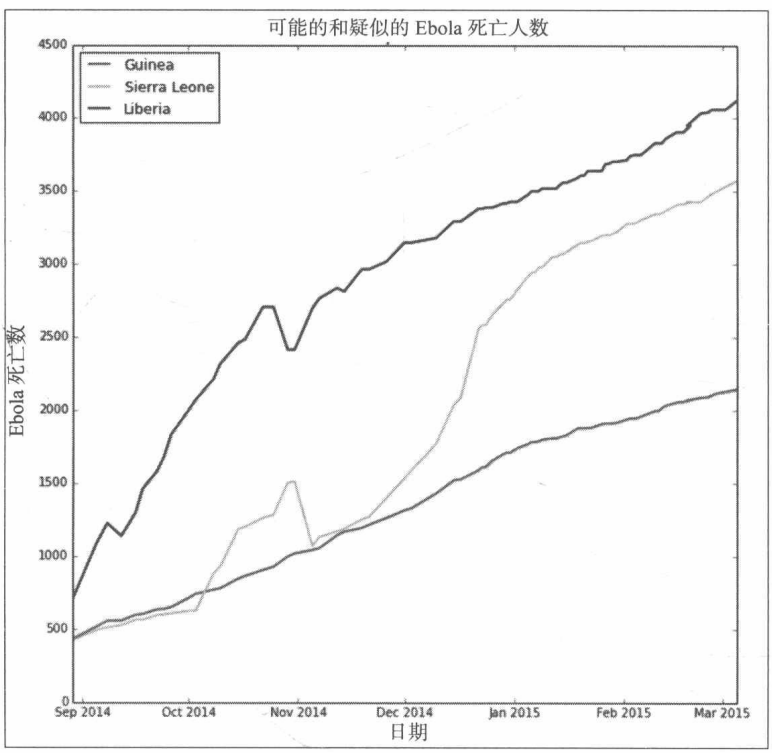
我们可以对另一个指标绘制类似的图，即：确认的、可能的、疑似的 Ebola 死亡累积数。(虽然这不是最好的方法，但可以通过绘制一个类似的图来展示更多国家的数据信息。)




```
import matplotlib.pyplot as plt
import csv
import operator
import datetime as dt

with open('/Users/kvenkatr/python/ebola.csv', 'rt') as f:
  filtereddata = [row for row in csv.reader(f) if row[3] != "0.0" and
  row[3] != "0" and "deaths" in row[0]]

sorteddata = sorted(filtereddata, key=operator.itemgetter(1))

guineadata  = [row for row in sorteddata if row[1] == "Guinea" and
  row[0] == "Cumulative number of confirmed, probable and suspected
Ebola deaths"]
sierradata  = [row for row in sorteddata if row[1] == "Sierra Leone"
and
  row[0] == " Cumulative number of confirmed, probable and suspected
Ebola deaths "]
liberiadata = [row for row in sorteddata if row[1] == "Liberia" and
  row[0] == " Cumulative number of confirmed, probable and suspected
Ebola deaths "]

g_x = [dt.datetime.strptime(row[2], '%Y-%m-%d').date() for
  row in guineadata]
```

```
g_y = [row[3] for row in guineadata]

s_x = [dt.datetime.strptime(row[2], '%Y-%m-%d').date() for
  row in sierradata]
s_y = [row[3] for row in sierradata]

l_x = [dt.datetime.strptime(row[2], '%Y-%m-%d').date() for
  row in liberiadata]
l_y = [row[3] for row in liberiadata]

plt.figure(figsize=(10,10))
plt.plot(g_x,g_y, color='red', linewidth=2, label="Guinea")
plt.plot(s_x,s_y, color='orange', linewidth=2, label="Sierra Leone")

plt.plot(l_x,l_y, color='blue', linewidth=2, label="Liberia")
plt.xlabel('Date', fontsize=18)

plt.ylabel('Number of Ebola Deaths', fontsize=18)

plt.title("Probable and Suspected Ebola Deaths", fontsize=20)

plt.legend(loc=2)


plt.show()
```

结果应该如下图所示：



2.3 体育案例

为了说明另一个案例以及某种可视化方法的优势，我们不妨考虑一个不同的问题：2015年2月美国足球四分卫球员的前五个最高纪录是哪几个？原始数据源为Len Dawson NFL和AFL Statistics。（数据来源：http://www.pro-football-reference.com/players/D/DawsLe00.htm。）

数据包括前22名四分卫球员的信息，他们是Peyton Manning、Brett Favre、Dan Marino、Drew Brees、Tom Brady、Frank Tarkenton、John Elway、Warren Moon、John Unitas、Vinny Testaverda、Joe Montana、Dave Krieg、Eli Manning、Sonny Jurgensen、Dan Fouts、Philip Rivers、Ben Roethlisberger、Drew Bledsoe、Boomer Esiason、John Hadle、Tittle和Tony Romo：

Name	Year	Age	Cmp	Att	Yds	TD	Teams
Peyton Manning	1998	22	326	575	3739	26	Multi
Peyton Manning	1999	23	331	533	4135	26	Multi
Peyton Manning	2000	24	357	571	4413	33	Multi
Peyton Manning	2001	25	343	547	4131	26	Multi
Peyton Manning	2002	26	392	591	4200	27	Multi
Peyton Manning	2003	27	379	566	4267	29	Multi
Peyton Manning	2004	28	336	497	4557	49	Multi
Peyton Manning	2005	29	305	453	3747	28	Multi
Peyton Manning	2006	30	362	557	4397	31	Multi
Peyton Manning	2007	31	337	515	4040	31	Multi
Peyton Manning	2008	32	371	555	4002	27	Multi
Peyton Manning	2009	33	393	571	4500	33	Multi
Peyton Manning	2010	34	450	679	4700	33	Multi
Peyton Manning	2011	35	0	0	0	0	Multi
Peyton Manning	2012	36	400	583	4659	37	Multi
Peyton Manning	2013	37	450	659	5477	55	Multi

在我们考虑可视化之前，需要做一些分析工作。这些四分位球员参加比赛的时期不同。例如，Brett Favre参赛时间从1991年到2010年，Dan Marino参赛时间从1983年到1999年。挑战在于：如果使用条形图或气泡图，我们将只能得到一维的可视化结果。

进行可视化的第一步是解析CSV文件，这有几种选择，我们可以用pandas中的read_csv函数或csv模式，这些都有一些诸如DictReader的便利的函数：

```
import csv
import matplotlib.pyplot as plt

# csv has Name, Year, Age, Cmp, Att, Yds, TD, Teams
```

```
with open('/Users/MacBook/java/qb_data.csv') as csvfile:
   reader = csv.DictReader(csvfile)
   for row in reader:
     name  = row['Name']
     tds = row['TD']
```

四分卫数据可以从前面的数据源下载；过滤数据也同样可从 http://www.knapdata.com/python/qb_ data.csv 上下载。csv 模式包括像字典一样用行表示类别，因此可以命名域。DictReader 和 DictWriter 类将行翻译成字典，而不是列表。字典的关键要点能被输入或从输入的第一行进行推断（该行包括抬头标题）。通过 DictReader 可读取 CSV 文件的内容，其中的列输入值被视为字符串：

```
#ways to call DictReader

# if fieldnames are Name, Year, Age, Cmp, Att, Yds, TD, Teams
fieldnames = ['Name', 'Year', 'Age', 'Cmp', 'Att', 'Yds', 'TD',
'Teams']

reader = csv.DictReader(csvfile, fieldNames=fieldnames)
# If csv file has first row as Name, Year, Cmp, Att, Yds, TD, Teams
#   we don't need to define fieldnames, the reader automatically
recognizes
#   them.
```

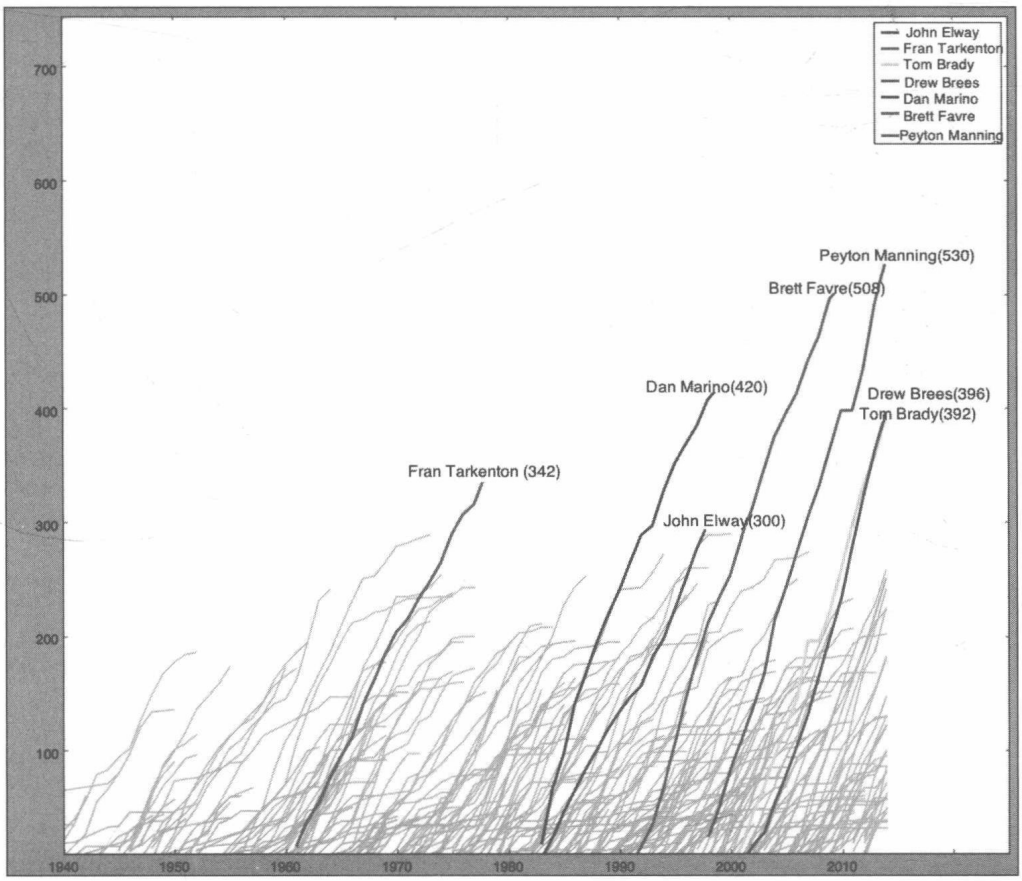
为了完成数据的数值型转换，我们可能需要一个转换和返回数值型数值的函数。我们也有额外的函数，如 prepare.py 中的 getcolors() 和 num()，这些将在下面的案例中使用：

```
# num(s) and getcolors() functions
def num(s):
  try:
    return int(s)
  except ValueError:
    return 0

def getcolors():
  colors = [(31, 119, 180), (255,0,0), (0,255,0), (148, 103, 189),
(140, 86, 75), (218, 73, 174), (127, 127, 127), (140,140,26), (23,
190, 207), (65,200,100), (200, 65,100), (125,255,32), (32,32,198),
(255,191,201), (172,191,201), (0,128,0), (244,130,150), (255,
127, 14), (128,128,0), (10,10,10), (44, 160, 44), (214, 39, 40),
(206,206,216)]
for i in range(len(colors)):
  r, g, b = colors[i]
  colors[i] = (r / 255. , g / 255. , b / 255.)
return colors
```

呈现可视化结果

基于输入数据的域名，每一个四分卫球员的触地得分统计量和传球码数统计量可以画在同一个时间线上。知道绘图内容后，我们应弄清楚绘制方式。

用（年份，触地得分）和（触地得分，年份）域绘制简单的 *X-Y* 图应该是一个好的开始。然而，输入数据文件中的 252 位四分位球员绝大多数彼此无关。因此用不同颜色来展示没有意义。（没有意义的原因在于我们没有 252 种不同的颜色。）我们试图画出前 7 或前 10 种结果，如下图所示：



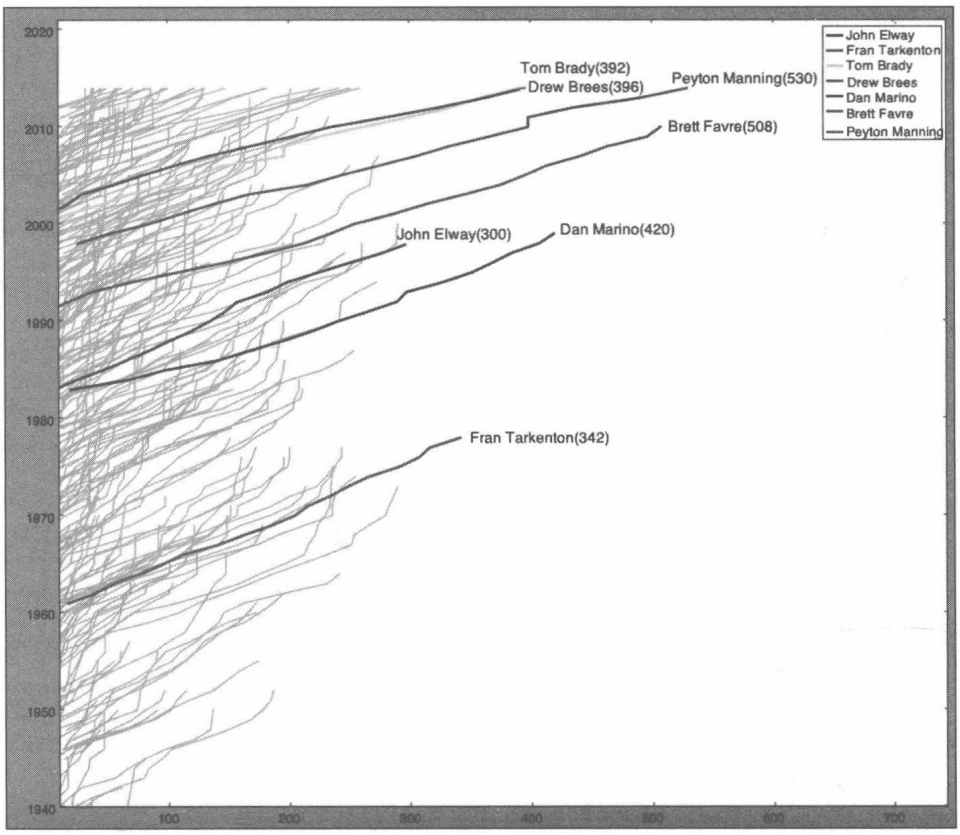
下面的 Python 程序表明怎样用 matplotlib 展示前 10 位四分卫球员的触地得分数。该程序的运行结果如上图所示：

```
import csv
import matplotlib.pyplot as plt

# The following functions can be in separate file
#  (If it does, you need to import)
```

```
def num(s):
  try:
    return int(s)
  except ValueError:
    return 0

def getcolors():
  colors = [(31, 119, 180), (255,0,0), (0,255,0), (148, 103, 189),
(140, 86, 75), (218, 73, 174), (127, 127, 127), (140,140,26), (23,
190, 207), (65,200,100), (200, 65,100), (125,255,32), (32,32,198),
(255,191,201), (172,191,201), (0,128,0), (244,130,150), (255,
127, 14), (128,128,0), (10,10,10), (44, 160, 44), (214, 39, 40),
(206,206,216)]

  for i in range(len(colors)):
    r, g, b = colors[i]
    colors[i] = (r / 255. , g / 255. , b / 255.)
  return colors

def getQbNames():
  qbnames = ['Peyton Manning']
  name=''
  i=0
  with open('/Users/MacBook/java/qb_data.csv') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
      if ( row['Name'] != name and qbnames[i] != row['Name']):
        qbnames.append(row['Name'])
        i = i+1
  return qbnames

def readQbdata():
  resultdata = []
  with open('/Users/MacBook/java/qb_data.csv') as csvfile:
    reader = csv.DictReader(csvfile)
    resultdata = [row for row in reader]
  return resultdata

fdata=[]
prevysum=0

#    -- functions End --

qbnames = getQbNames()
fdata = readQbdata()

i=0
rank=0
prevysum=0
lastyr=0
highrank=244
```

```
colorsdata = getcolors()

fig = plt.figure(figsize=(15,13))
ax=fig.add_subplot(111,axisbg='white')

# limits for TD
plt.ylim(10, 744)
plt.xlim(1940, 2021)

colindex=0
lastage=20

for qbn in qbnames:
  x=[]
  y=[]
  prevysum=0
  for row in fdata:
    if ( row['Name'] == qbn and row['Year'] != 'Career'):
      yrval = num(row['Year'])
      lastage = num(row['Age'])
      prevysum += num(row['TD'])
      lastyr = yrval
      x += [yrval]
      y += [prevysum]

  if ( rank > highrank):
    plt.plot(x,y, color=colorsdata[colindex], label=qbn,
linewidth=2.5)
    plt.legend(loc=0, prop={'size':10})
    colindex = (colindex+1)%22
    plt.text(lastyr-1, prevysum+2, qbn+"("+str(prevysum)+"):"
+str(lastage), fontsize=9)

  else:
    plt.plot(x,y, color=colorsdata[22], linewidth=1.5)
    rank = rank +1
plt.xlabel('Year', fontsize=18)

plt.ylabel('Cumulative Touch Downs', fontsize=18)

plt.title("Cumulative Touch Downs by Quarter Backs", fontsize=20)
plt.show()
```

将图 (X, Y) 转换为 (Y, X)，有足够的空间展示四分卫球员的名字。在上面的代码片段中，我们可能要做出如下改变：



如果我们转换 x 轴和 y 轴，就会有更多的空间展示四分卫球员的名字和触地得分总和，如上图所示。为了完成这项任务，我们需要转换 x 轴和 y 轴，根据新的 x 轴和 y 轴，在合适的位置添加标签。

```
plt.xlim(10, 744)
plt.ylim(1940, 2021)

# remaining code all un-changed except

y += [num(row['Year'])]
x += [prevysum]

# Don't forget to switch the x,y co-ordinates of text display

plt.text(prevysum+2, lastyr-1, qbn+"("+str(prevysum)+"):"
str(lastage), fontsize=9)
```

乍一看，我们只能发现职业生涯中触地得分占领先位置的四分卫球员（2014 ~ 2015 年足球赛季）。基于这种可视化，你可以进一步尝试分析和理解还能从数据中推断出什么结论。这些发现围绕下述问题的答案展开：

❑ 哪位四分卫球员职业生涯最长？

❑ 现在还有能超越 Peyton Manning 触地得分记录的四分卫球员吗？

在读取的输入文件中，Age 正好是我们得到的其中一个字段值。用 Age 起始值绘制 Age 与 Touchdown 统计量有很多种试验方法。为了回答第一个问题，我们不得不跟踪 Age 而不是 Year。下面的片段既可以用于一个单独的函数（如果常常被用到），也可以包含在主要脚本中：

```
maxage = 30

with open('/Users/MacBook/java/qb_data.csv') as csvfile:
  reader = csv.DictReader(csvfile)
    for row in reader:
      if ( num(row['Age']) > maxage ):
        maxage = num(row['Age'])

print maxage
```

运行上面的代码可知，44 是四分位球员的最大年龄（在联赛中发挥积极作用时，有 3 位四分位球员：Warren Moon、Vinny Testaverde 和 Steve DeBerg。名义上，George Blanda 在 48 岁前一直参赛（年龄最大的球员），但他一开始是四分位球员，也曾经当过几年球员）。

为了回答其他问题，我们用图表示触地得分统计量和四分位球员的年龄，代码如下：

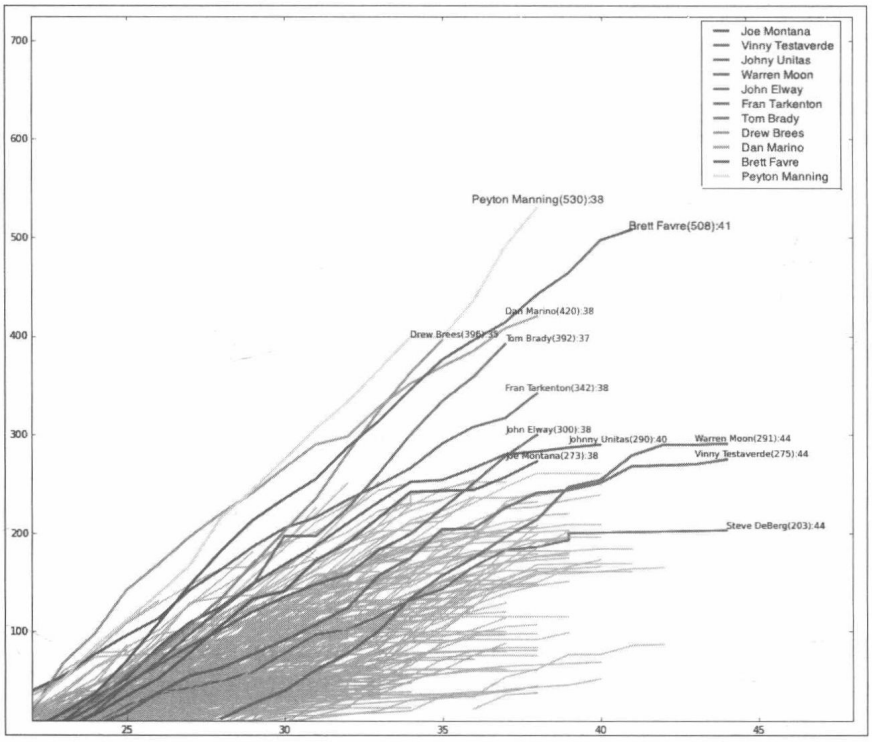
```
import csv
import matplotlib.pyplot as plt

# The following functions can be in a separate file
#    -- functions Begin --
def num(s):
  try:
    return int(s)
  except ValueError:
    return 0

def getcolors():
  colors = [(31, 119, 180), (255,0,0), (0,255,0), (148, 103, 189),
(140, 86, 75), (218, 73, 174), (127, 127, 127), (140,140,26), (23,
190, 207), (65,200,100), (200, 65,100), (125,255,32), (32,32,198),
(255,191,201), (172,191,201), (0,128,0), (244,130,150), (255,
127, 14), (128,128,0), (10,10,10), (44, 160, 44), (214, 39, 40),
(206,206,216)]

  for i in range(len(colors)):
    r, g, b = colors[i]
    colors[i] = (r / 255. , g / 255. , b / 255.)
  return colors
```

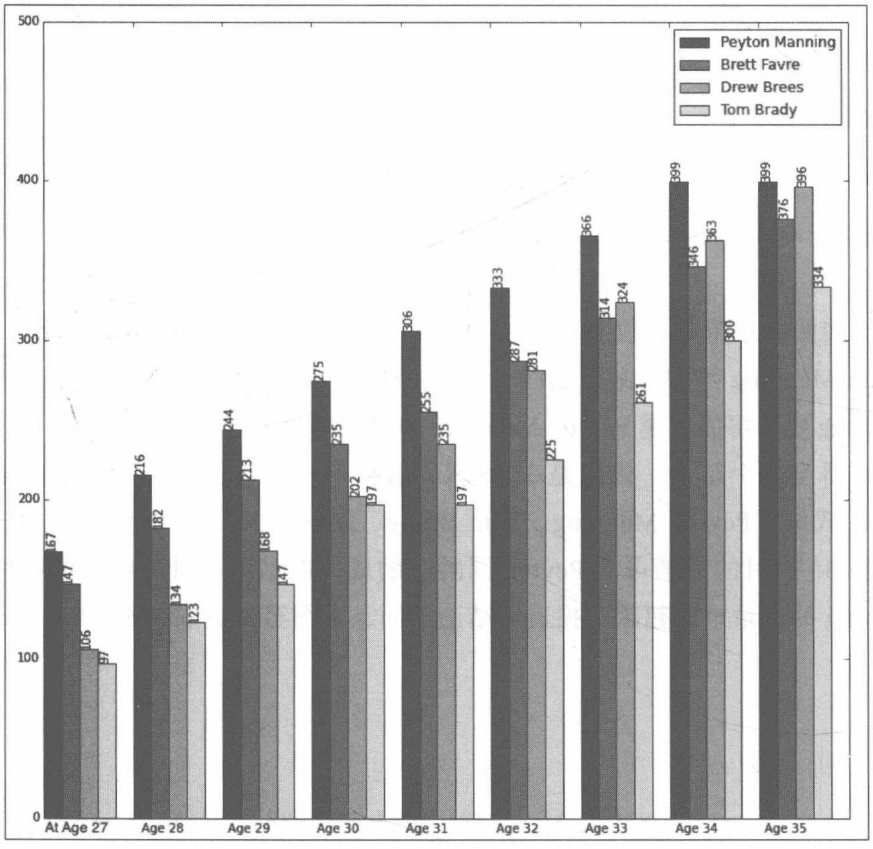
```
def getQbNames():
  qbnames = ['Peyton Manning']
  name=''
  i=0
  with open('/Users/MacBook/java/qb_data.csv') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
      if ( row['Name'] != name and qbnames[i] != row['Name']):
        qbnames.append(row['Name'])
         i = i+1
  return qbnames

def readQbdata():
  resultdata = []
  with open('/Users/MacBook/java/qb_data.csv') as csvfile:
    reader = csv.DictReader(csvfile)
    resultdata = [row for row in reader]
  return resultdata

fdata=[]
prevysum=0

#    -- functions End --

qbnames = getQbNames()
fdata = readQbdata()

i=0
rank=0
prevysum=0
lastyr=0
highrank=244
colorsdata = getcolors()

fig = plt.figure(figsize=(15,13))
ax=fig.add_subplot(111,axisbg='white')

# limits for TD
plt.ylim(10, 744)
#change xlimit to have age ranges
plt.xlim(20, 50)

colindex=0
lastage=20

for qbn in qbnames:
  x=[]
  y=[]
  prevysum=0
  for row in fdata:
    if ( row['Name'] == qbn and row['Year'] != 'Career'):
```

```
        yrval = num(row['Year'])
        lastage = num(row['Age'])
        prevysum += num(row['TD'])
        lastyr = yrval
        x += [lastage]
        y += [prevysum]

    if ( rank > highrank):
      if ( lastage == 44):
        plt.plot(x,y, color='red', label=qbn, linewidth=3.5)
      else:
        plt.plot(x,y, color=colorsdata[colindex], label=qbn,
linewidth=2.5)
        plt.legend(loc=0, prop={'size':10})

      colindex = (colindex+1)%22
      plt.text(lastage-1, prevysum+2, qbn+"("+str(prevysum)+"):"
+str(lastage), fontsize=9)

    else:
      if ( lastage == 44):
        plt.plot(x,y, color='red', label=qbn, linewidth=3.5)
        plt.text(lastage-1, prevysum+2, qbn+"("+str(prevysum)+"):"
+str(lastage), fontsize=9)
      else:
        plt.plot(x,y, color=colorsdata[22], linewidth=1.5)
      rank = rank +1

plt.xlabel('Age', fontsize=18)

plt.ylabel('Number of Touch Downs', fontsize=18)

plt.title("Touch Downs by Quarter Backs by Age", fontsize=20)
plt.show()
```

当你看这些图的结果时，仅有两位四分位球员（Drew Brees 和 Tom Brady）在他们 35 岁时与 Peyton Manning 势均力敌。然而，考虑到 Tom Brady 的年龄和他目前的成就，似乎只有 Drew Brees 更有可能超越 Peyton Manning 的触地得分记录。

由下图可得出这个结论。该图是 35 岁球员数据的一个较简单的图。比较排名前四的四分卫球员的成绩（Peyton Manning、Tom Brady、Drew Brees 和 Brett Favre），我们发现 Drew Brees 在 35 岁时的成绩可与 Peyton 同龄时的成绩匹敌。尽管与纽约时报中“为什么 Peyton Manning 的记录难以超越”报道的结论不同，下图至少更倾向于 Drew Brees 有可能会打破纪录：


Joe Montana
Vinny Testaverde
Johny Unitas
Warren Moon
John Elway
Fran Tarkenton
Tom Brady
Drew Brees
Dan Marino
Brett Favre
Peyton Manning
Peyton Manning(530):38
Brett Favre(508):41
Dan Marino(420):38
Drew Brees(396):37
Tom Brady(392):37
Fran Tarkenton(342):38
John Elway(300):38
Johnny Unitas(290):40
Warren Moon(291):44
Joe Montana(273):38
Vinny Testaverde(275):44
Steve DeBerg(203):44
700
600
500
400
300
200
100
25
30
35
40
45


Peyton Manning
Brett Favre
Drew Brees
Tom Brady
500
400
300
200
100
0
167
147
106
97
216
182
134
123
244
213
168
147
275
235
202
197
306
255
235
197
333
287
281
225
366
314
324
261
399
346
363
300
399
376
396
334
At Age 27
Age 28
Age 29
Age 30
Age 31
Age 32
Age 33
Age 34
Age 35

2.4　用数据编写有趣的故事

数据可视化通常改善用数据讲述故事的能力，而且在某些情况下，需要视觉上没有那么琐碎。不久前，新闻记者已经更倾向于将可视化结果整合到叙述中，这样通常有助于读者理解故事。在商界，很少有案例成功掌握数据与故事相关联的方法。对于观众来说，这些有意义的故事应从感性和理性角度都具有吸引力。正如 Rudyard Kipling 所写，“如果历史以故事的形式串联，那么它将不会被遗忘。”类似的想法用在数据方面。因此，我们应该会理解：如果数据展示方式正确，那么读者会更好地理解和记住这些数据。

2.4.1　为什么故事如此重要

现有很多可视化的工具和方法：条形图和饼图、表格、线图、气泡图、散点图等等，不胜枚举。然而，这些工具关注于数据探索而非故事创作。尽管可视化案例有助于讲述故事，但这种情况很少发生。这主要是因为找到一则故事比处理数据难度大很多。当然，也有以**读者驱动**为导向的故事和以**作者驱动**为导向的故事。

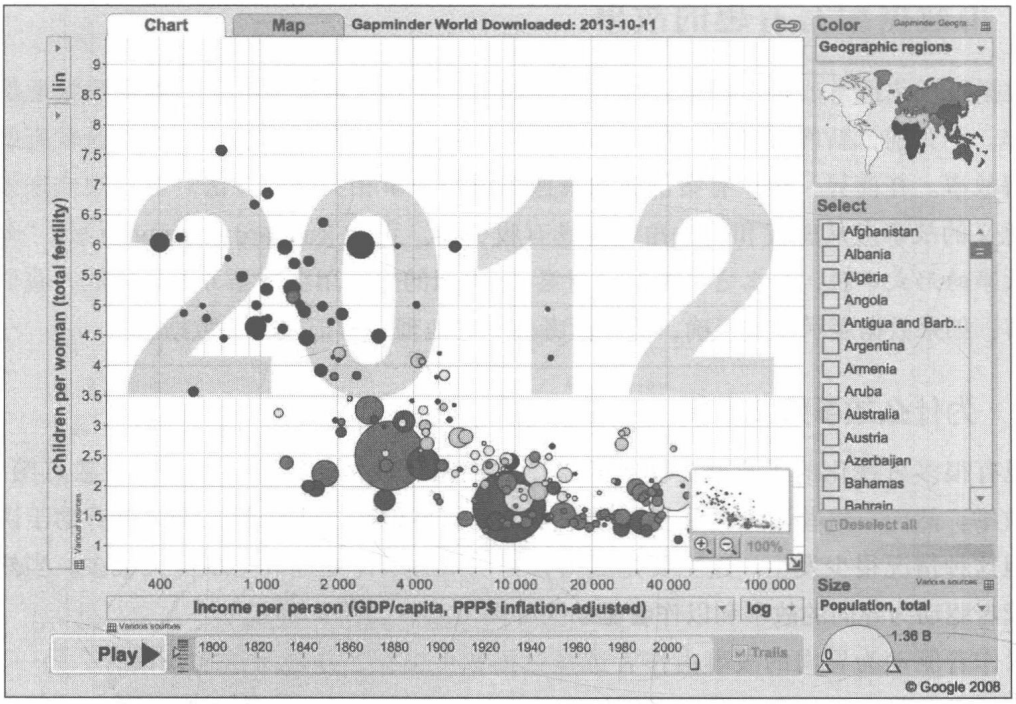
以作者驱动为导向的故事由作者选择数据和可视化，并展示给大众读者。另一方面，以读者驱动为导向的故事向读者提供处理数据的工具和方法，使读者有更多的灵活性和选择来进行分析和理解可视化。

2.4.2　以读者驱动为导向的故事

2010 年，斯坦福大学的学者对故事产生的重要性进行研究和评论，并且提出一些叙事可视化的设计策略。根据他们的研究，一种纯粹的以作者驱动为导向的方法有一条完成可视化的严格路线，依赖信息而没有与读者的互动，而以读者驱动为导向的故事没有固定顺序的图像，不依赖信息而高度交互。以作者驱动为导向的方法有这样一个案例，其中列出的 7 种可视化故事包括：杂志风格、著名图表、分区海报、流程图、连环画、幻灯片和电影 / 视频 / 动画。

1. Gapminder

以读者驱动为导向的故事与以数据驱动为导向的故事相结合的一个经典案例是 Gapminder World（http://gapminder.org/world）。该网站有超过 600 个国际经济、环境、健康、技术等方面的数据指标。同时也向学生提供真实世界问题的研究工具，发现模式、趋势和相关性。该网站由瑞典 Hans Rosling 基地研发的 Trendalyzer 软件创建的，在 2007 年 3 月被 Google 收购。



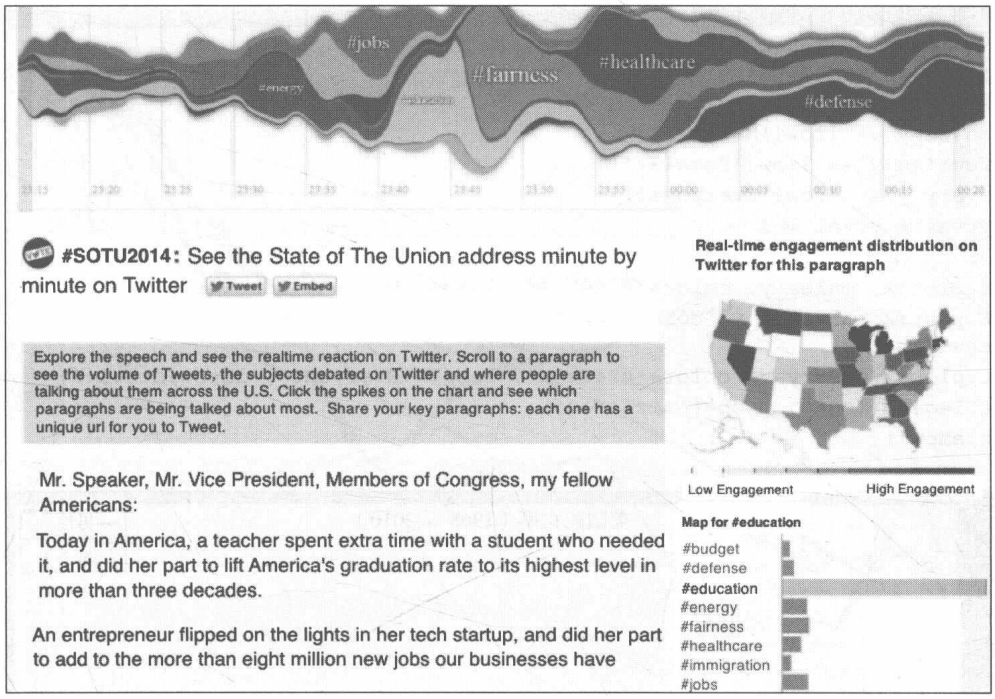
Gapminder 使用的信息可视化技术是一个自带 5 个默认设置变量的交互式气泡图。这 5 个变量是：*X*、*Y*、气泡大小、颜色和被滑块控制的时间变量。滑块控制和沿 *X*、*Y* 轴的选择使该图变得可交互。但即便使用这样的工具创作一个故事也不一定容易。讲故事是一门手艺，也可以是一种有效的知识共享技术。因为它传达了背景和情感丰富的内容，比大多数其他的交流方式更有效。

大多数有吸引力的故事讲述者都能抓住理解观众的重要性。他们可能会给小孩和大人讲述相同的故事，但是故事的起承转合将有所不同。同样，一则以数据驱动或以读者驱动为导向的故事应该根据聆听和研究的对象不同而作调整。例如，对于一名管理者，统计可能是关键，但是一位商业智能管理者很可能对方法和技术感兴趣。

现在有很多可用的 JavaScript 框架用来创建互动式的可视化，最受欢迎的是 D3.js。目前，用 Python 只有少量创建互动式可视化的方法（不用 Flash）。一种方法是生成 D3.js 能用来作图的 JSON 格式的数据，第二种选择是用 Plotly（http://www.plot.ly）。我们将在本章深入讲解更多 Plotly 的细节。

2. 国情咨文

在 Obama 总统演讲期间，Twitter 根据推文的位置和话题创建了可视化图像。该可视化之所以有趣，是因为它在一个位置捕捉了很多细节。演讲期间，该图的滚动播出展示出 Twitter 的反应；它被贴在下面的网址：http://twitter.github.io/interactive/sotu2015/#p1。



3. 美国死亡率

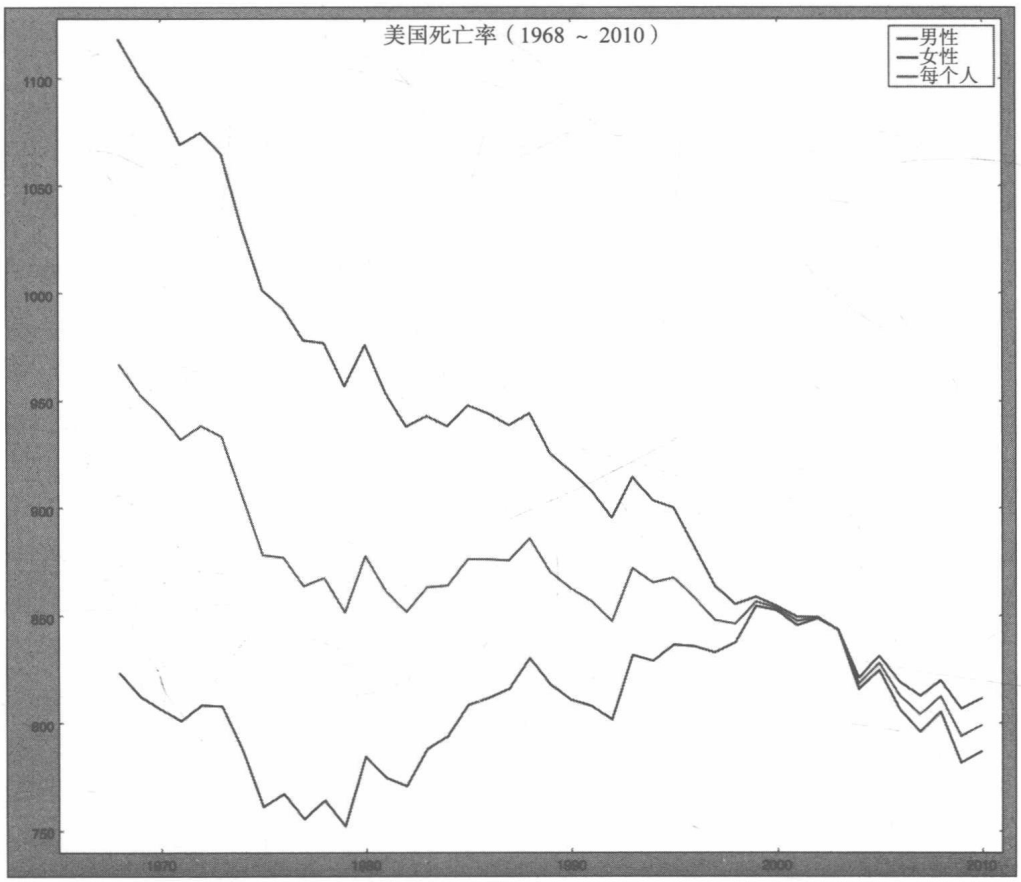
从1968年到2010年，美国的死亡率下降17%（详细数据来自http://www.who.int/healthinfo/mortality_data/en/）。几乎所有的改善都可以归因于男性生存前景的改善。它看起来像20世纪90年代中期前的进步，但原因可能是从那以后人口寿命增加了很多。可以从Bloomberg读取这种现象的一个完整描述，但我们试图展示两个可视化结果：

- 1968 ~ 2010年间，男性、女性和全部人口的死亡率
- 7个年龄组死亡率展示了一些有趣的结果

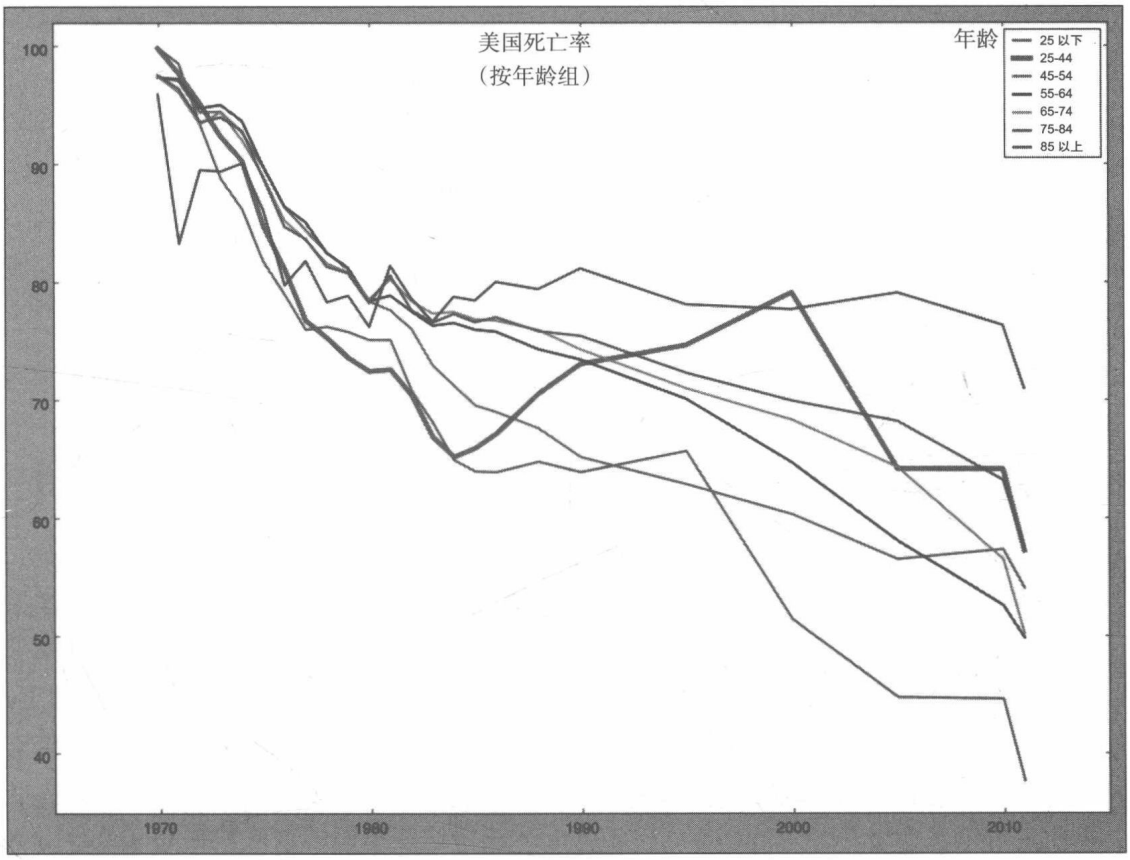
该案例的代码如下所示：

```
import csv
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(15,13))
plt.ylim(740,1128)
plt.xlim(1965,2011)
# Data from http://www.who.int/healthinfo/mortality_data/en/
with open('/Users/MacBook/Downloads/mortality1.csv') as csvfile:
  mortdata = [row for row in csv.DictReader(csvfile)]

x=[]
males_y=[]
females_y=[]
```

```
every_y=[]
yrval=1968
for row in mortdata:
  x += [yrval]
  males_y += [row['Males']]
  females_y += [row['Females']]
  every_y += [row['Everyone']]
  yrval = yrval + 1

plt.plot(x, males_y, color='#1a61c3', label='Males', linewidth=1.8)
plt.plot(x, females_y, color='#bc108d', label='Females',
linewidth=1.8)
plt.plot(x, every_y, color='#747e8a', label='Everyone', linewidth=1.8)
plt.legend(loc=0, prop={'size':10})
plt.show()
```



死亡率是以 100 000 人为单位进行衡量的。通过将人口划分为不同的年龄层，可以看出寿命一直在增加，尤其是 25 岁以下年龄组有最大的改善。为什么 25~44 岁以下年龄组的人口在下跌（用粗线显示）？ Bloomberg 上的故事通过关联到另一个事实很好得给出了原因：在那段时间，AIDS 导致的死亡数会影响那个年龄组。



AIDS 杀死了 40 000 多个美国人，而且他们中有 75% 属于 25 ~ 44 岁的年龄组。因此，可从那段时间内看到特别结果。

```
import csv
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(15,13))
plt.ylim(35,102)
plt.xlim(1965,2015)

colorsdata = ['#168cf8', '#ff0000', '#009f00', '#1d437c', '#eb912b',
'#8663ec', '#38762b']
labeldata = ['Below 25', '25-44', '45-54', '55-64', '65-74', '75-84',
'Over 85']

# using reader() instead of DictReader() so that we could loop to
# build y-values in list
with open('/Users/MacBook/Downloads/mortality2.csv') as csvfile:
  mortdata = [row for row in csv.reader(csvfile)]

x=[]
```

```
for row in mortdata:
  yrval = int(row[0])
  if ( yrval == 1969 ):
    y = [[row[1]],[row[2]],[row[3]],[row[4]],[row[5]],[row[6]],[r
ow[7]]]
  else:
   for col in range(0,7):
     y[col] += [row[col+1]]
  x += [yrval]

for col in range(0,7):
  if ( col == 1 ):
    plt.plot(x, y[col], color=colorsdata[col], label=labeldata[col],
linewidth=3.8)
  else:
    plt.plot(x, y[col], color=colorsdata[col], label=labeldata[col],
linewidth=2)

plt.legend(loc=0, prop={'size':10})
plt.show()
```

csv.reader() 和 csv.DictReader() 的不同之处在于：输入 CSV 文件何时有字段名（或列名），DictReader() 用字段名作为键，用列中的实际数值作为数据取值。在上面的案例中，我们用到 reader()，这是因为用循环（y[col] = [row[col+1]]）方便。而且，如果 CSV 文件中有列名，用 reader() 应该会忽略第一行。

我们也可以从 http://www.knapdata.com/python 获得这些案例的过滤数据 mortality1.csv 和 mortality2.csv。

对于 mortdata[:4]，不同的读取方法，结果会不同。换句话说，当我们用 reader() 后，mortdata[:4] 的结果如下：

```
[['1969', '100', '99.92', '97.51', '97.47', '97.54', '97.65',
'96.04'],  ['1970', '98.63', '97.78', '97.16', '97.32', '96.2',
'96.51', '83.4'],  ['1971', '93.53', '95.26', '94.52', '94.89',
'93.53', '93.73', '89.63'],  ['1972', '88.86', '92.45', '94.58',
'95.14', '94.55', '94.1', '89.51']]
```

用 DictReader()，假定 CSV 文件有字段名，四行数据会如下展示：

```
[{'25-44': '99.92', '45-54': '97.51', '55-64': '97.47', '65-74':
'97.54', '75-84': '97.65', 'Below 25': '100', 'Over 85': '96.04',
'Year': '1969'},
 {'25-44': '97.78', '45-54': '97.16', '55-64': '97.32', '65-74':
'96.2', '75-84': '96.51', 'Below 25': '98.63', 'Over 85': '83.4',
'Year': '1970'},
 {'25-44': '95.26', '45-54': '94.52', '55-64': '94.89', '65-74':
'93.53', '75-84': '93.73', 'Below 25': '93.53', 'Over 85': '89.63',
'Year': '1971'},
 {'25-44': '92.45', '45-54': '94.58', '55-64': '95.14', '65-74':
```

```
'94.55', '75-84': '94.1', 'Below 25': '88.86', 'Over 85': '89.51',
'Year': '1972'}]
```

4. 其他的案例故事

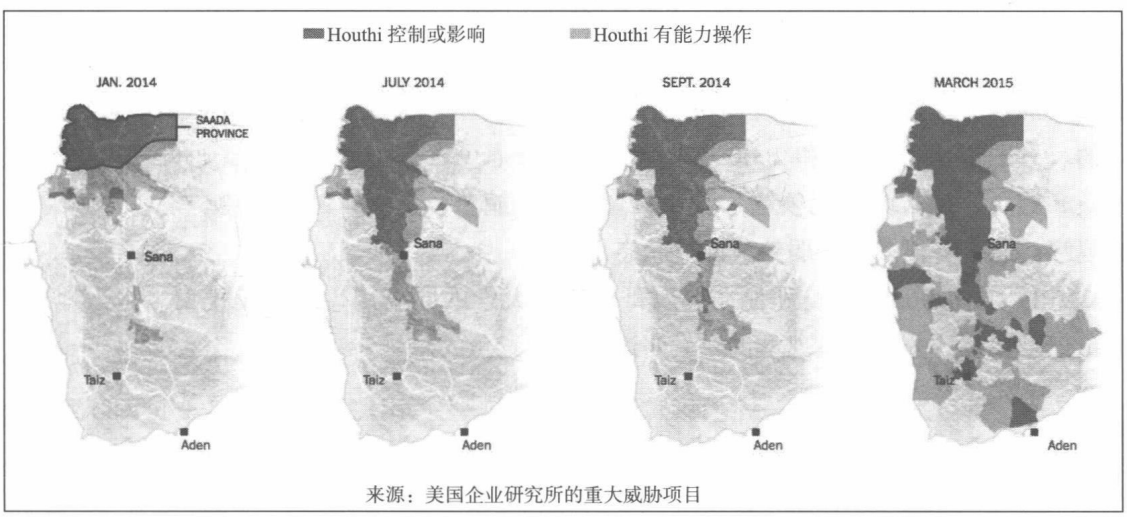
我们有很多探索、可视化、交互和尝试的案例。一些著名的案例如下：

- 用 255 张图展示经济衰退如何重塑经济（《纽约时报》）：这个故事说明大萧条结束之后的 5 年，经济如何重新获得 900 万个岗位，强调哪些行业比其他行业恢复速度快。（来源：http://tinyurl.com/nwdp3pp。）
- 2013 年华盛顿奇才中射击明星（《华盛顿邮报》）：该交互式图创建于几年前，基于 2013 年华盛顿奇才的表现为基础。该图尝试分析和观察 Paul Pierce 的中档射击水平如何得以有很大提高。（来源：http://www.washingtonpost.com/wp-srv/special/sports/wizards-shooting-stars/。）

2.4.3 以作者驱动为导向的故事

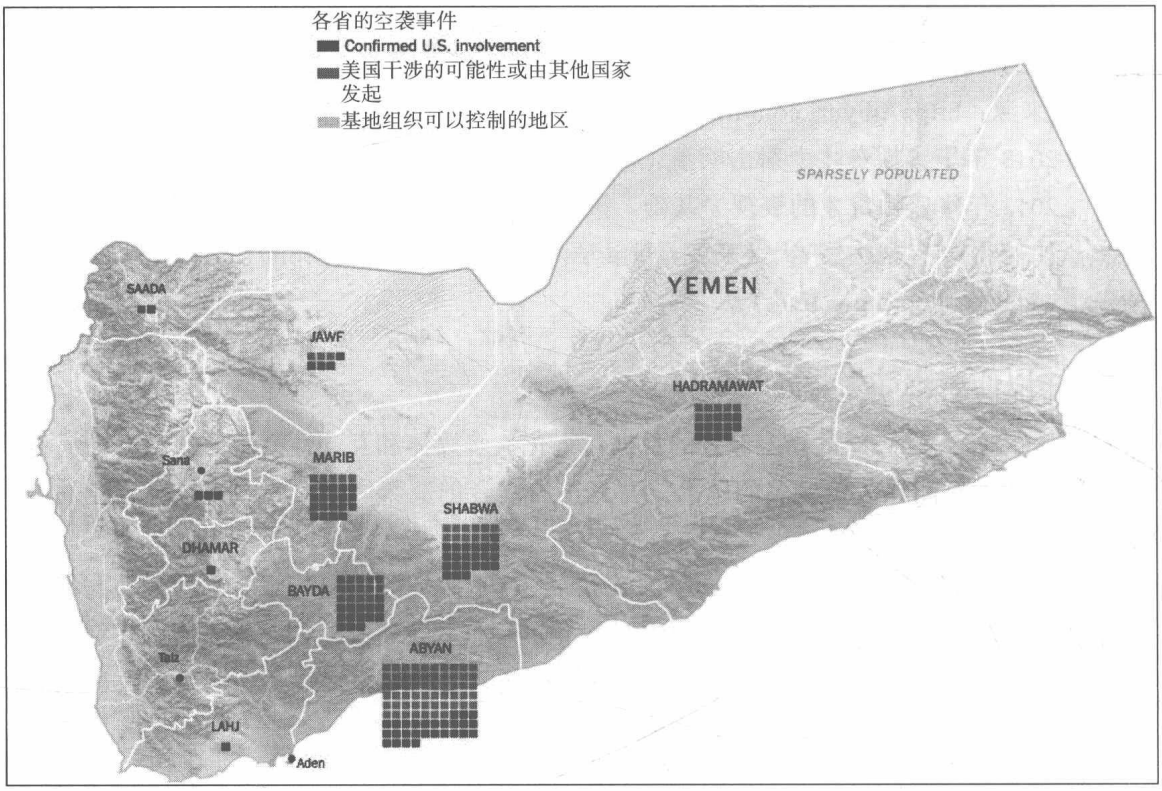
《纽约时报》有一些世界上最好的数据可视化、多媒体和互动性故事。该报社的终极目标始终是达到新闻界久负盛名的标准，并真正为读者创造新体验。其中，故事文化的讲述是工作背后的能量来源之一。

例如，“*The Geography of Chaos in Yemen*”就是一个以数据与作者驱动为导向的故事相结合的案例。2015 年 3 月 26 日，沙特阿拉伯的喷气式飞机在也门与侯塞因团体完成驱动器方面的对抗。也门在诸如沙特阿拉伯、伊朗和美国这几个关键成员中起着重要作用。过去几年，纽约时报的作者发现 Houthi 的影响力已有所扩大。



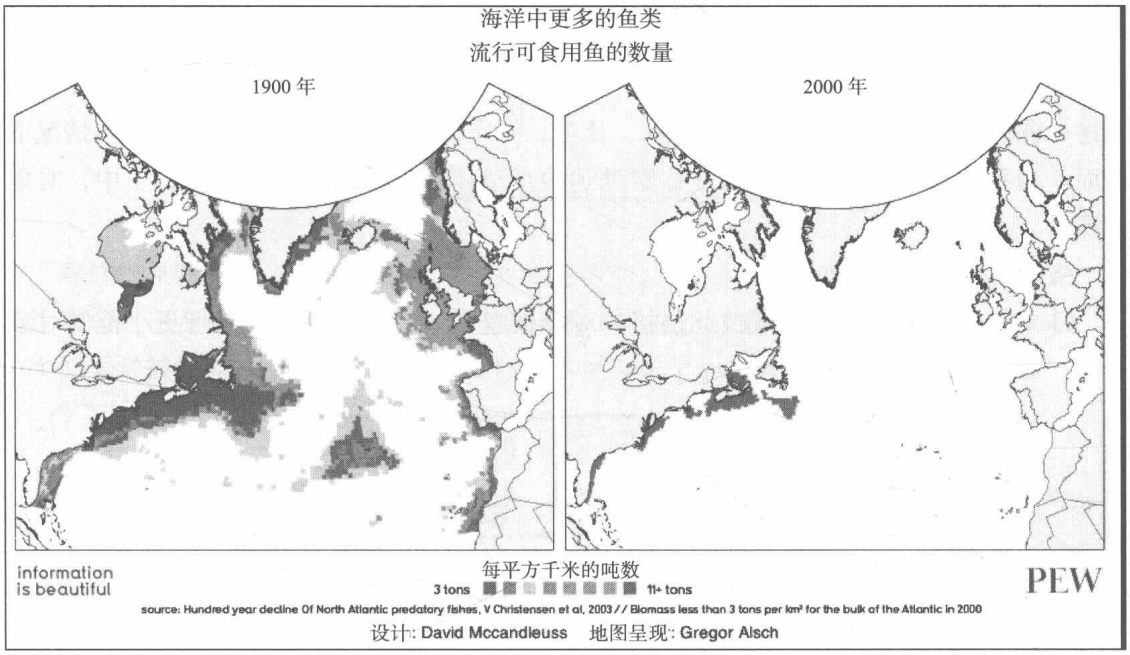
来源：美国企业研究所的重大威胁项目

在阿拉伯半岛，也门是“基地”组织最有活力的一个分支。自 2009 年以来，美国已经在也门开展了至少 100 次空袭。除了基地组织的侵占，伊斯兰国也在那个区域有活动。最近，他们宣称对萨那两个什叶派清真寺 135 人遇难的轰炸事件负责。下图来自新闻调查局（Bureau of Luvestigative Journalism）的美国企业研究所的重大威胁项目：



另一个不错的案例是 David McCandless 对大西洋的过去进行可视化，展示了过度捕鱼前海洋的样子。难以想象过度捕鱼给海洋造成的破坏。这种效果在海洋中是不可见的、隐藏的。下图说明 1900 年和 2000 年北大西洋的普通可食用鱼的总量。普通可食用鱼包括金枪鱼、鳕鱼、黑线鳕、无须鳕、大比目鱼、鲱、鲭鱼、狭鳕、鲑、海鳟、条纹鲈、鲟鱼和大菱鲆，现在有很多处于弱势或濒危状态。

Villy Christensen 博士和他在英属哥伦比亚大学的同事用生态系统模型、水下地形图，捕鱼记录和统计分析来呈现本世纪大西洋不同位置鱼的总量。



2.5　感知与表达方法

以前，数据大小和种类不会带来很大的挑战；因此，感知和分析数据是直接的。如今，在众多领域都存在着大量数据，用数据可视化可以完成与数据的感知和交互，这能够为人类提供有价值的帮助。人为因素显著帮助整个可视化过程，来完成理解数据和帮助决策的任务。

可视化技术能被分为两块：

- **科学可视化**：这涉及科学数据有一个固有的物质实体
- **信息可视化**：这涉及抽象数据（空间或非空间）

大多数可视化系统被设计为可以人机合作，每一个都能完成下面的任务：

- 视觉展示数据来提高数据分析
- 视觉展示模型、数据的阐释、想法、假设和观点
- 通过发现他们假设的支持性或对立性证据，帮助用户改善模型
- 帮助用户组织和分享他们的想法

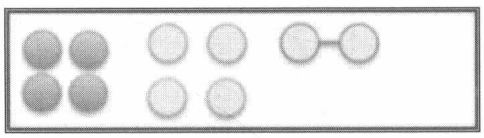
融入视觉感知的新见解是从除信息可视化以外其他各种学科的研究中提出的，比如人为因素和人机交互。视觉可视化的优势是我们处理可视化信息比口头信息更快。在 20 世纪 20 年代的德国，第一批研究过知觉组织的心理学家创立了格式塔学派。

感知格式塔原理

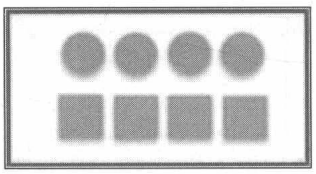
格式塔一词是指“整体组织”或逐一识别总体中各部分的不同特征。例如，可以通过描述一棵树的不同部分来诠释树的含义，比如，树干、叶子、树枝和果实（在一些情况下）。然而，当我们观察一整棵树时，往往察觉到的是整体而不是局部。在这个例子中，对象是整棵树。

格式塔感知的原则如下：

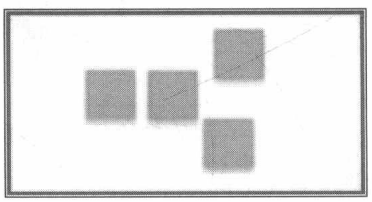
- **接近性**：聚集在一起或彼此连接的对象被视为一组，逐一减少处理更小范围对象的需要。



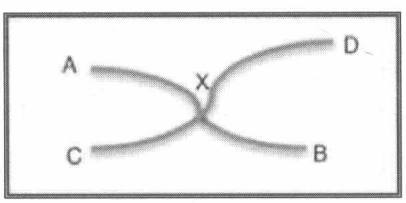
- **类似性**：共享相似属性、颜色或形状的对象被视为一组。



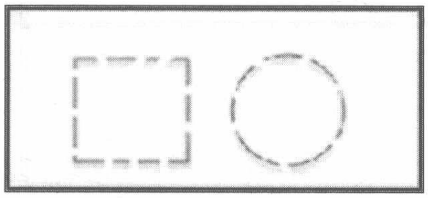
- **共性**：当接近性和相似性都到位时，会发生一个变化。然后他们会改变组别。



- **好的连续性**：一部分和整体一种重要，这意味着如果有中断，就会改变感知阅读。下图中，我们感知到两条线而不是四条线在中间相交：

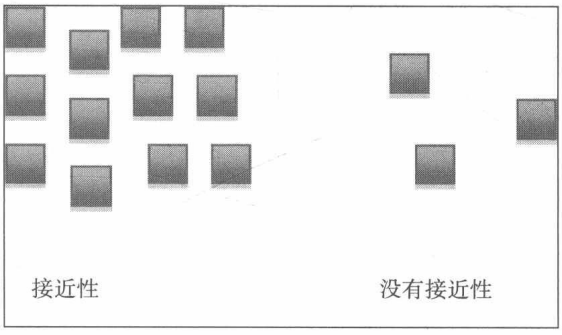


- **闭包**：即使一个形状的部分边界缺失，我们仍能看到被边界完整包围的形状并忽略这个空白。



了解这些原则对创建任何可视化方法非常有帮助。

让我们进一步举例阐述。接近性是指将看起来相似的形状分到一组的可视化方法。这样一个组通常被视为一个单独的单元。例如，下图展示了如何区分接近性：



2.6 一些最好的可视化实践

我们完成一个好的可视化的最首要一步是了解努力背后的目标。如何了解可视化是否有其意图？同样重要的是了解观众群以及这种方式是否有帮助。

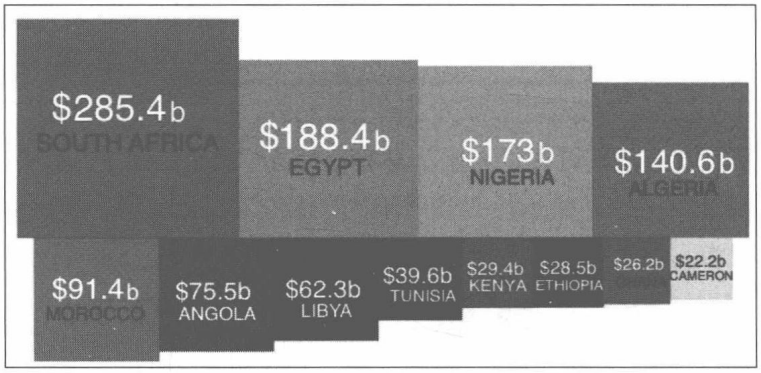
一旦回答了这些问题并透彻理解可视化的目标，那么下一个挑战就是选择正确的呈现方法。最常用的可视化类型可根据下述内容进一步分类：

- 比较和排名
- 相关性
- 分布
- 位置定位或地理数据
- 局部到整体的关系
- 随时间的变化趋势

2.6.1 比较和排名

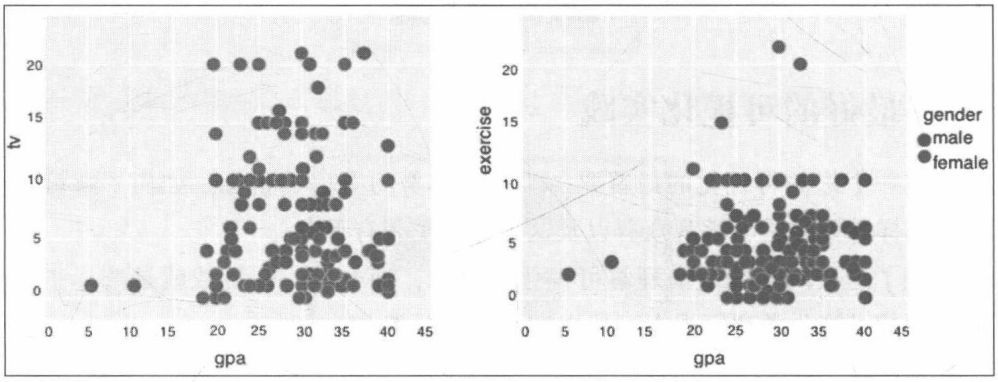
比较和排名的方法不止一种，传统的方法使用条形图。条形图是在相同基线上以编码数值为长度的图形。但它并不总是比较和排名的最佳方法。例如，为了展示非洲 GDP 排名

前 12 名的国家，下面是一种创造性的可视化方式（来源：Stats Legend，Andrew Gelman and Antony Unwin）：

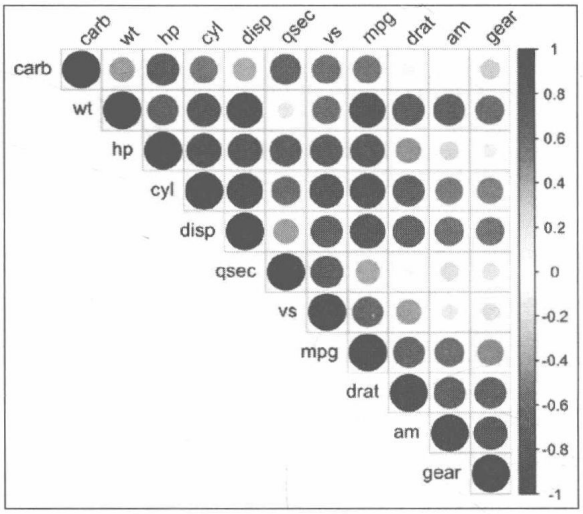


2.6.2 相关性

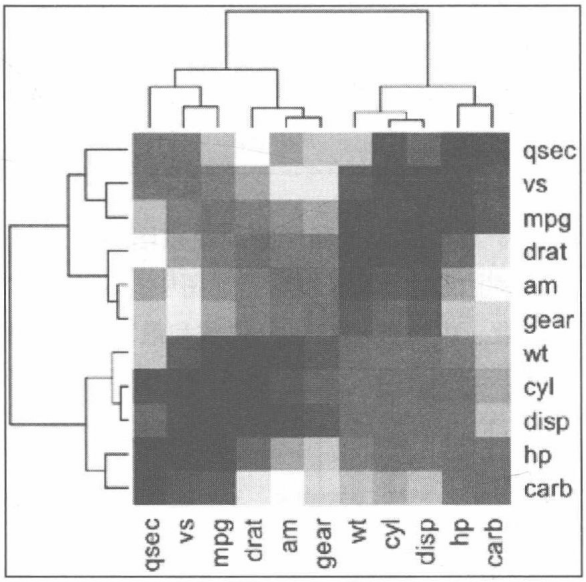
相关性不能保证一种关系，但一种简单的相关性分析是识别不同测度间关系的良好开端。通常需要一种统计方法来确保关系的真实存在。下面的例子是通过构建一个简单的散点图来检测两个因子间关系，比如同一所大学中学生的 gpa 和 tv，gpa 和 exercise：



我们也可以用其他方法展示相关矩阵。比如，可以用散点图、热力图，或一些特定的案例来展示 S&P 100 中股票的影响力网。（下面两个图来自 Statistical Tools for High Throughput Analysis，网址是：http://www.sthda.com。）需要进一步强调，一个相关矩阵包括数据的矩阵形式。如下面的例子所示，数据通过一个缩放后的颜色图计算相关性。为了解更多细节，我们建议您参考下面的网站：http://www.sthda.com。

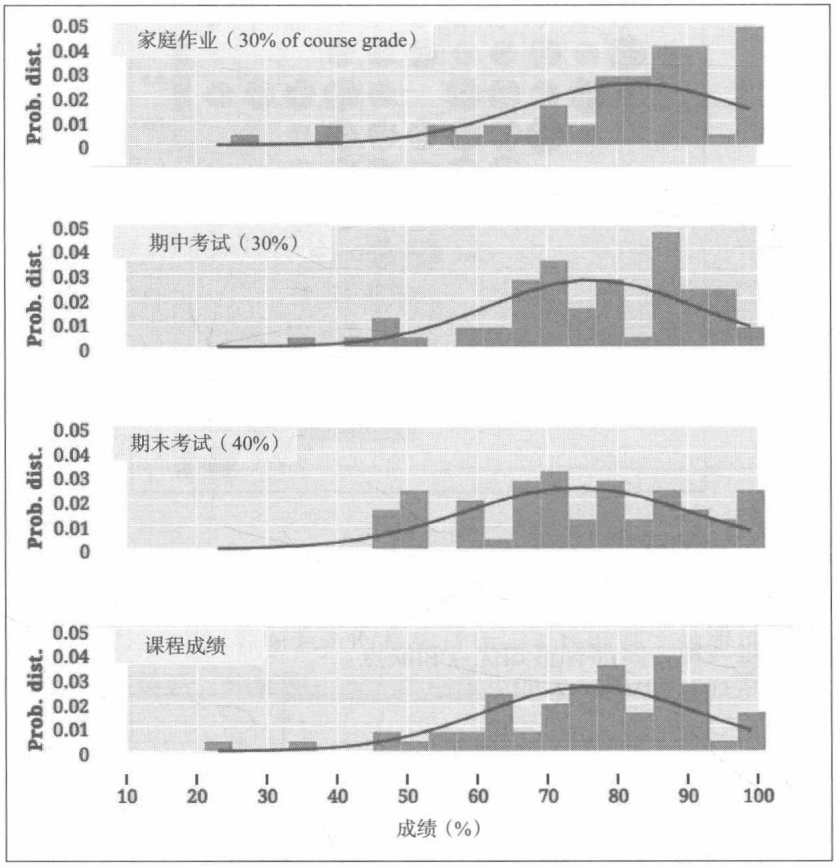


相关矩阵被用来研究同一时间多个变量间的依赖性。结果是一张表，包括每一个变量与其他变量的相关系数。热力图源于数据矩阵形式的二维展示。通过很多不同的颜色方案来说图解热力图，每一种方案都有感知优点和缺点。

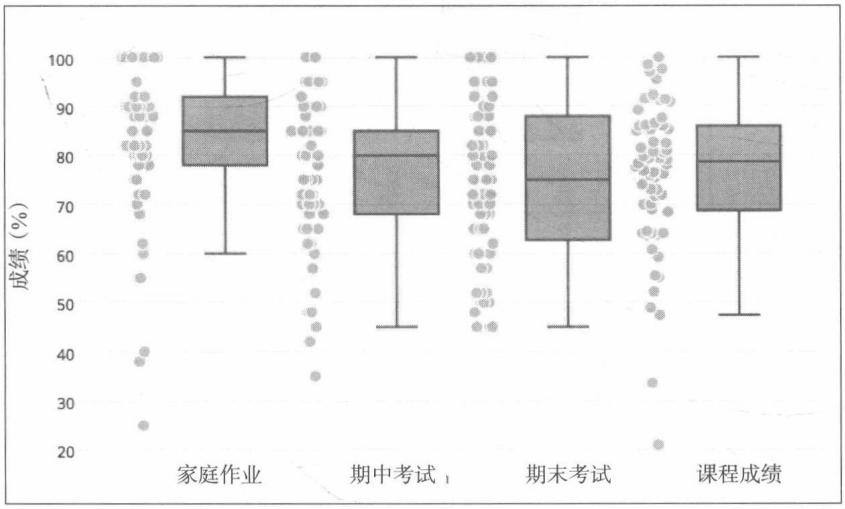


2.6.3　分布

分布分析展示了数值在区间范围内的分布，因此，在数据分析中非常有用。比如，比较同一个班的学生在家庭作业、期中考试、期末考试和全部课程成绩的得分分布。在这个例子中，我们将讨论使用最普遍的两种图表类型来实现这个目的。一种是**直方图**（如下图所示），另一种是**箱线图**或**箱须图**。



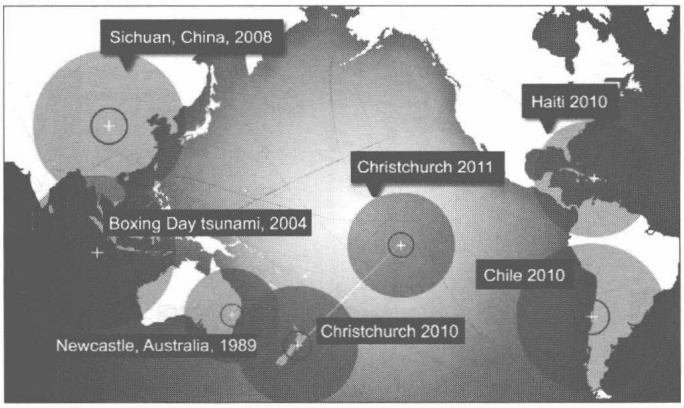
直方图的形状非常依赖于箱子的大小和位置。箱须图特别适用于多元分布。它们将所有数据点（在这个例子中是学生的得分）打包入箱须图中。现在，你可以很容易识别出所有类别中的最小值、25% 分位数、中位数、75% 分位数和最大值，这些都能同时进行。



Python 中有一种便捷的画图方法 Plotly 它是一种在线分析和可视化工具。Plotly 提供在线画图、分析和统计工具，还有 Python、R、Julia 和 JavaScript 的科学绘图库。直方图和箱线图的案例请见：https://plot.ly/python/histograms-and-box-plots-tutorial。

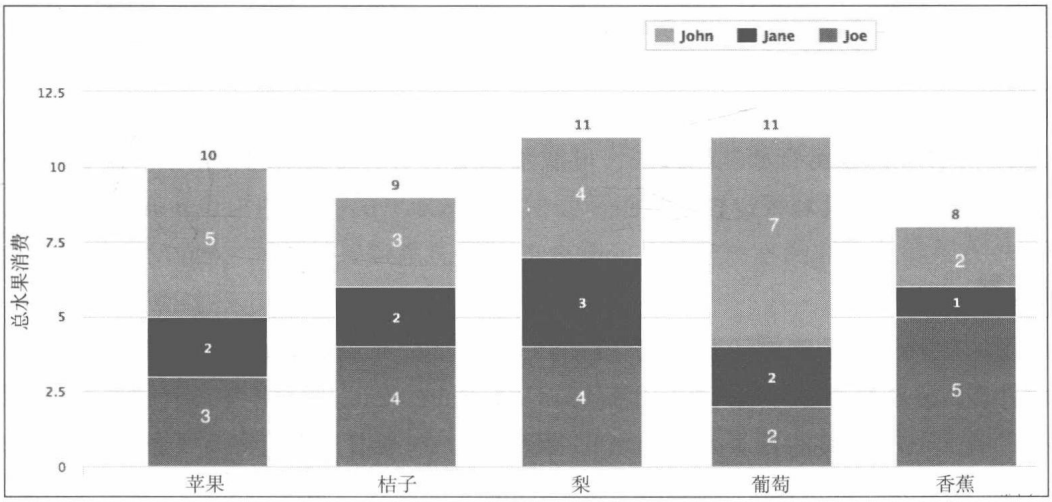
2.6.4　位置定位或地理数据

地图是展示位置定位数据的最佳方法。地图最适用于与另外一个图配对的情况，该图详细说明了地图的展示内容（比如条形图按从大到小排序，线图展示趋势，等等）。比如，下面的地图展示了不同大洲之间地震的强度：



2.6.5　局部到整体的关系

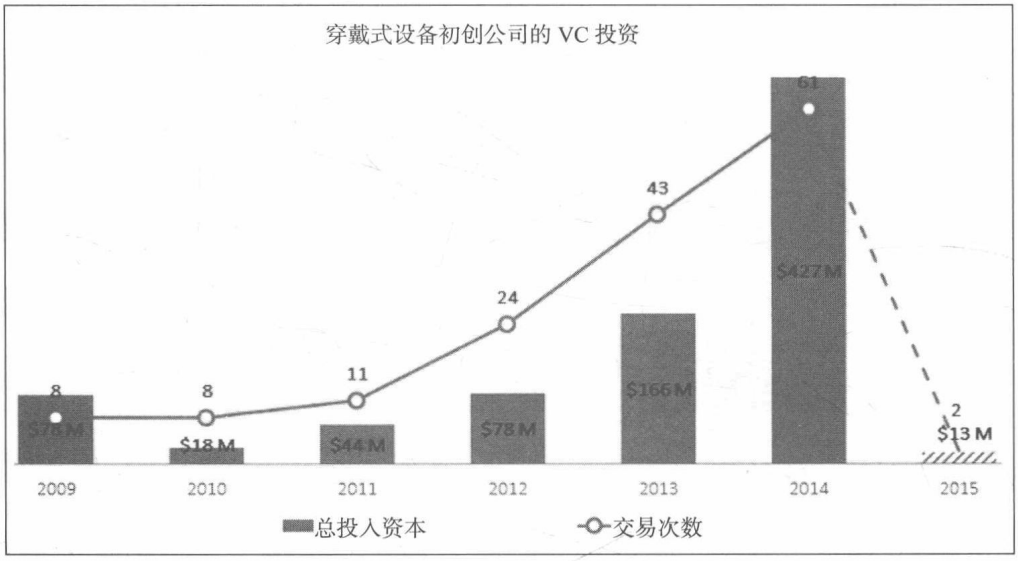
饼图适用于展示局部到整体的关系，但还有其他方式。分组的条形图适用于比较类别中每个元素与其他元素，以及所有类别的元素。然而，分组使区分每个组的整体差异变得更难。下图是累积柱形图。



累积柱形图适用于展示整体差异，这是因为它们在视觉上聚集同一个组中的所有类别。不足之处是比较单个类别大小变得更难。这种累积方式也展示出一种局部到整体的关系。

2.6.6 随时间的变化趋势

最常用类进行数据分析的可视化方法是展示一段时间的变化趋势。在下面的例子中，2009 ~ 2015 年穿戴式设备的初创公司投资情况被画成图。该图表明，该项投资在几年中持续增长；而在 2013 年，只有 43 起总价值 $166 百万的交易，所有年份中最高点是 2014 年 61 起总价值 $427 百万的交易，相比仅一年前。



通过观察，我们很乐意看到未来几年市场的演变。

2.7 Python 中的可视化工具

数据分析和可视化需要一些软件工具：用一个文本编辑器来写代码（最好语法高亮），用 Python 和其他库来运行和测试代码，可能还要用一个工具展示这些结果。现有两种软件工具：通用的软件工具和特定软件组件。

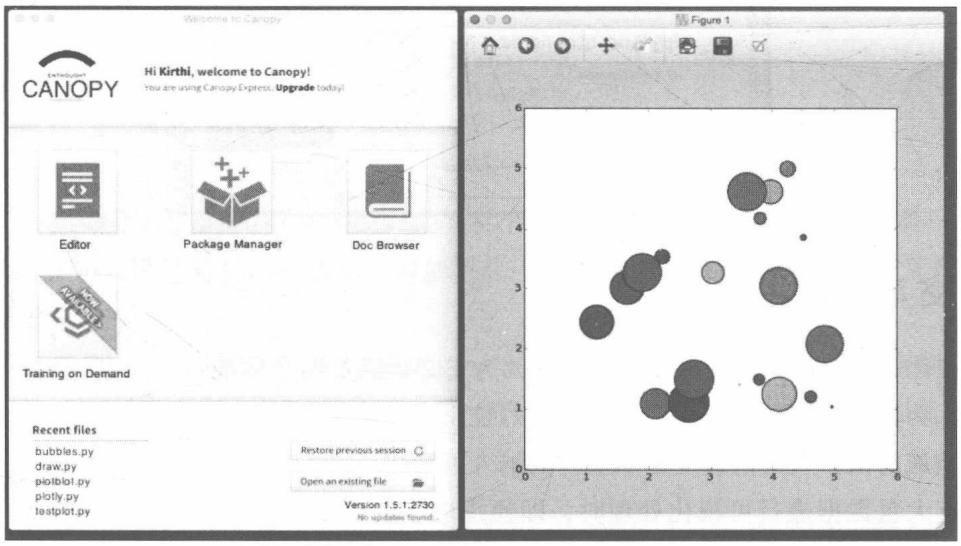
开发工具

通用的软件工具是**集成开发环境**（integrated development environment，IDE），这是一种同一软件包内囊括所有生产工具的应用程序。从处理 Python 库的角度来看，这些 IDE 通常非常方便。有关 IDE 工具的更多细节将在下一章讨论。本章中，我们将简要介绍 Enthought 中的 Canopy 和 Continuum Analytics 中的 Anaconda。

特定的软件绘图组件是 Python 绘图库，比如 Bokeh、IPython、matplotlib、NetworkX、SciPy 和 NumPy、Scikit-learn 以及 Seaborn。IDE 都能非常方便地处理增加、删除和更新这些绘图库的版本。

1. Enthought 中的 Canopy

在一些其他的库中，Enthought Canopy 有一个在 BSD- 风格执照下发布的免费版本，而且以 GraphCanvas、SciMath 和 Chaco 为绘图工具。它有高级文本编辑器、集成 IPython 控制台、图表软件包管理器和在线文件链接。Canopy 分析环节简化了数据分析、可视化、算法设计和面向科学家、工程师和分析师的应用开发环境。



2. Continuum Analytics 中的 Anaconda

Anaconda IDE 以 conda 应用为基础。Conda 是发现和访问软件包的一种应用。conda 软件包是一个包括 Python 模块、可执行程序，或其他组件的二进制压缩包。Conda 跟踪了软件包与平台细节间的依赖性，使得从其他组软件包中创建工作环境变得简单。

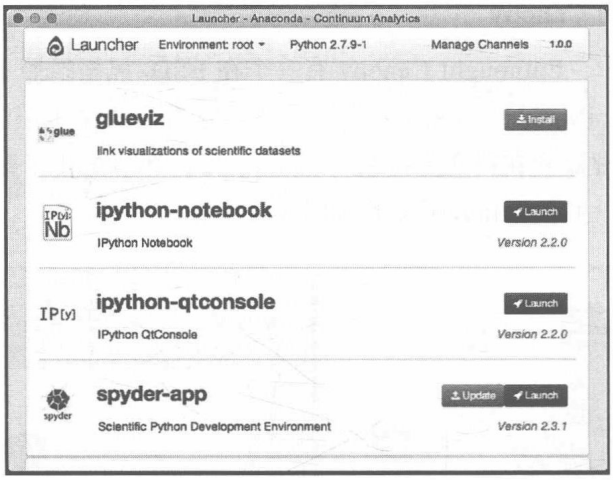
Anaconda 带有 sypder-app，一种科学的 Python 开发环境，该环境也有 IPython 查看器。除此之外，IPython 可以由一个 GUI 或基于网络的笔记本发起。最方便的地方在于：你可以在主目录访问 Python，而不用接触系统安装的 Python。并非所有的软件包能用在 Python 3 中；因此，最好将这些 IDE 用在 Python 2。

IPython（http://ipython.scipy.org/）提供了一个提高的、互动的 Python shell，而且由于数据科学和可视化在本质上的互动性，它被极度推荐。大多数平台支持 IPython。IPython 还有其他一些特征：

- **标签完成**：这涉及变量、函数、方法、属性和文件名的完成。标签完成是通过 GNU

Readline 库（http://tiswww.case.edu/php/chet/readline/rltop.html）完成的。用完 GNU Readline 之后，很难回到常规的命令行界面。

- **命令历史功能**：这是为充分考虑以前命令而发布的命令历史。



2.8 交互式可视化

可视化之所以被认为是可交互的，是因为它必须满足两个标准：

- **人为输入**：某些信息视觉呈现方面的控制必须可为人所用
- **响应时间**：人为的改变必须及时的纳入可视化

当有大量数据进行可视化处理时，即便用现在的技术也很不容易，有时甚至不可能实现；因此，“交互式可视化”通常被应用到系统中。该系统在完成输入的几秒内就可以向用户提供反馈。很多交互式可视化系统支持一种类导航，类似于穿越物质世界的导航。

交互式的好处在于人们在较短时间内可以探索较大的信息空间。这一点可以通过一个平台得到很好的理解。而不足之处在于这种交互性需要大量时间详细检查每一种可能性以测试可视化系统。而且，设计及时反馈用户行为的系统需要关注显著有效的算法。

任何可视化方法都需要好的布局设计方案。一些设计方法自动生成对称图形；另外一些画图方法从数据对称性开始。交互式可视化通过事件监听器来实现。对于一些事件监听器，交互式可视化可能是一种常识，而对于其他事件监听器来说，未必如此。下节描述了所有相关内容。

2.8.1 事件监听器

事件监听器是移动或点击鼠标即可实现的处理过程。技术上，有很多种事件，但纯粹的交互式可视化，我们仅需要了解当用户用鼠标浏览可视化时发生的情况。互动的等待时

间，即系统对鼠标操作的响应时间，会非常有影响。

最显著的原则是：用户应该确定已完成一些操作，而不是悬而未决该操作是否仍在进行中。因此，诸如突出显示一个选项的反馈是一种成功确认操作完成的好方法。视觉反馈通常应该在大概一秒的立即响应滞后时间内发生。下面是 Google 图表中一个 JavaScript 事件监听器的案例：

```
chart = new google.visualization.PieChart(document.getElementById(
'chart_div'));
google.visualization.events.addListener(chart, 'select',
selectHandler);
chart.draw(data, options);

function selectHandler() {
  var selectedItem = chart.getSelection()[0];
  var value = data.getValue(selectedItem.row, 0);
  alert('The user selected ' + value);
}
```

另一个原则是执行时间是否比用户的自然预期长很多，一些进度指标应显示给用户。在 JavaScript 中写事件监听器更容易，但如果想用 Python 绘图方法创建一个交互式可视化，应该选择 Plotly。

另一种模块，graph-tool（https://graph-tool.skewed.de），能够以一种直接的方式完成动画。在一个交互式窗口或画面外的文件，用 GTK+ 来展示动画。这种想法很容易实现可视化，在网站上加以呈现和嵌入。

2.8.2 布局设计

为了从视觉上有效展示数据，了解布局设计方法就变得很重要。美学是测度布局算法的优势与不足的标准之一。如果可能的话，为了增加布局结果的可读性，所述结构需要层次或对称；一条关键的因素是对空间的利用。

对解析和理解任何图表来说，良好的布局是必要的。通常，为了便于理解，每一种布局唯一适用于不同的数据可视化。一些著名的布局设计方法如下：

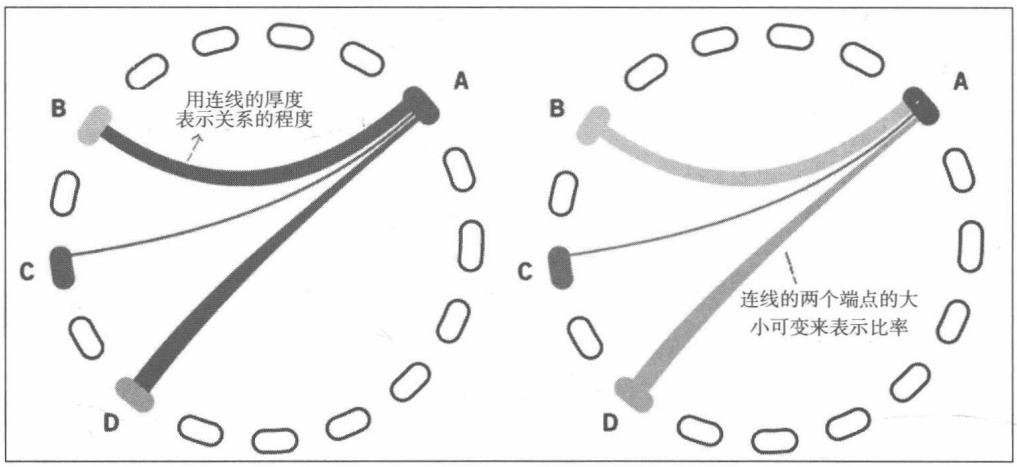
- 圆形布局
- 径向布局
- 球状布局

1. 圆形布局

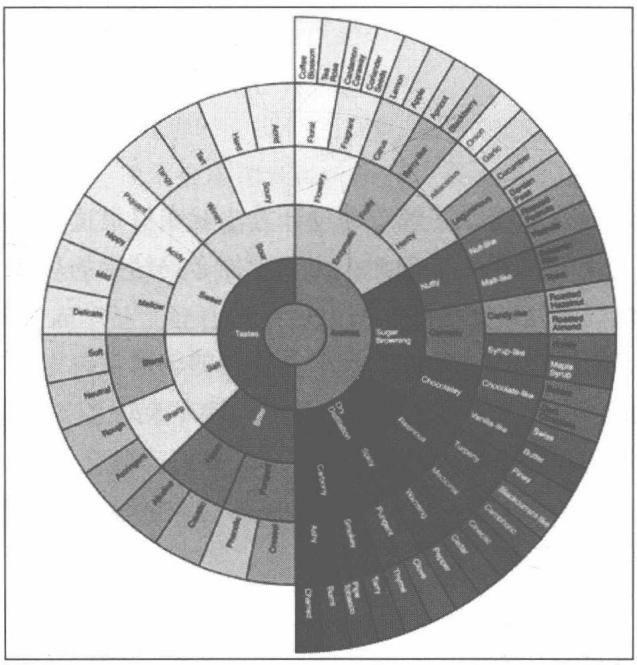
表格是数据的自然容器。无论信息何时呈现，用一个表的均值来展现表格的几率很高。然而，在很多情况下，当信息复杂（表格很大）时，表格很难用肉眼分析，表格数据模型仍

然不够透明。

换句话说，一种有用的容器不总是呈现方法的有用方法。该表格很好得展示出单个数据，但是它们间的关系以及它们组成的模式仍然难以看到。圆形布局会用一些不同的结合（定性和定量）来实现可视化，如下图所示：



例如，如上图一样直观，用有限的空间展示复杂关系。



上图给出一个用圆形布局展示复杂分层关系的案例。

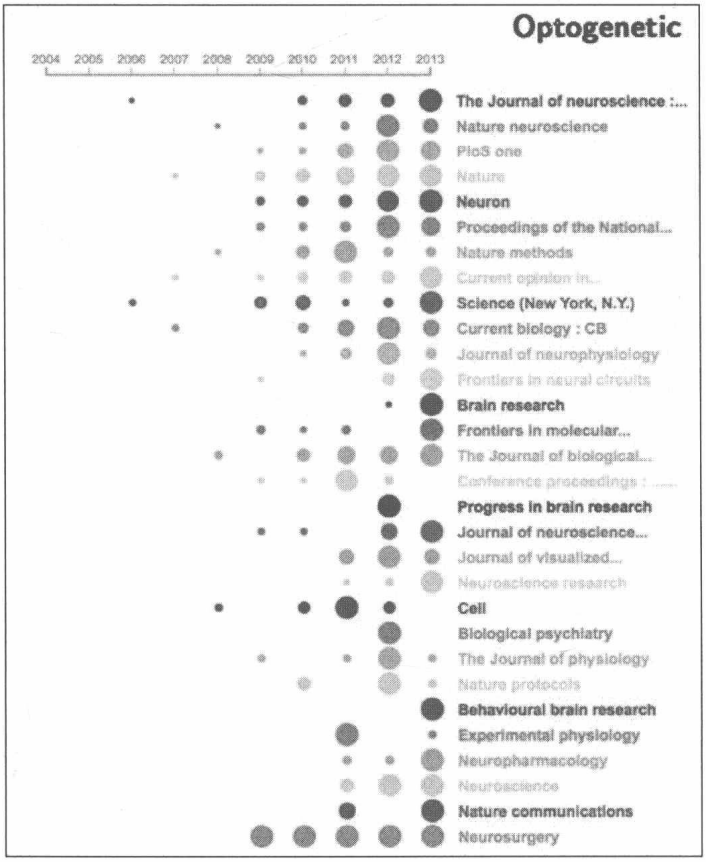
2. 径向布局

旭日可视化是一种径向空间填充的可视化技术，来展示树状结构（如上图所示）。还有其他的空间填充方法，即用其他视觉编码来描述层次。比如，树形图是空间填充的可视化，用到“容量”来展示“父子”关系。有一些微弱的变化，可以改善这种可视化的信息交流方式。

由于每个轨道的长度随半径增加，这趋向于有更多空间放置节点。随着水平的增加，径向树将在更大面积上扩散更多数量的节点。

3. 球状布局

球状布局存在不同的变化，而且其中一种可能是气泡形式。然而，如果我们用不同颜色和大小的气球（或者圆圈 / 气泡），则可视化结果能展示更多信息，如下图所示：



2.9　总结

创建一个有效的故事应遵守可视化方法的原则。本章中的故事给了有关出美学方面的

想法和方法的巨大变化。

数据可视化的目标是通过所选方法的视觉展示、清晰有效得与用户交流信息。有效的可视化有助于分析和推理数据和证据。这使得复杂数据更容易接触、理解和使用。用户可能有特殊的分析任务，比如进行比较或理解因果关系，以及该任务的图表设计原则。

表通常被用于用户查找变量的具体测度，而各种类别的图表用于展示数据中变量的模式或关系。

数据可视化既是一项艺术又是一门科学，同时也像解一道数学难题。不存在唯一正确的解决方法。类似地，也没有唯一正确的可视化方法。有很多可视化工具，而且我们了解到一些支持 Python 的工具。下面的章节将更详细地讨论这些工具。

第 3 章 Chapter 3

开始使用 Python IDE

作为一种广泛使用的编程语言，Python 已经有大概 20 多年的历史。在很多语言中，Python 因其简单性和动态类型而广受欢迎。Type(datum) 动态决定了数据对象的类型。它有一种语法，允许程序员编写极少的代码。Python 支持多种编程模式，包括函数型、面向对象型和程序型。

Python 解释器在绝大多数正在使用的操作系统中能获取到。其内置数据结构结合了动态捆绑，这使得 Python 变得很具吸引力，并作为一种高性能语言来快速连接现有的操纵组件。即使在分布式应用中，Python 也能像胶水一样用来关联 Hive（NoSQL），以快速有效地完成任务。软件开发社区中强大流行的 Python 需要一个可互动的环境来创建、编辑、测试、调试和运行程序。

集成开发环境（integrated development environment，IDE）是一个软件应用，提供了一种全面强大的工具集，为运行 Windows、Linux 或 Mac OS 操作系统的目标系统构建应用程序。这些工具提供了一个单一的和一致的集成环境，可用来实现生产率最大限度的提高。Python 编程有多种 IDE 选择。本章将具体讨论。此外，我们将讨论以下主题：

- Python 中的 IDE 工具
- 安装指南——下载和安装工具说明
- conda 的命令行界面（command-line interface，CLI）和 Spyder
- 在对可视化有用的库中，IDE 工具中的数据可视化工具
- 交互式可视化软件包
- 一些使用 IDE 工具的绘图案例

3.1 Python 中的 IDE 工具

分析和可视化数据需要一些软件工具：一个编写代码的文本编辑器（语法突出显示），其他运行和测试代码的工具与库，或许还有一组展示结果的工具。IDE 有很多优点，下面是一些显著的优点：

- 语法突出显示（马上显示错误或警告）
- 在调试模式中逐步读代码
- 可交互的控制台
- 集成交互的图形 notebook（比如 IPython）

3.1.1 Python 3.x 和 Python 2.7

Python 3.x 与 Python 2.x 版本不兼容。这就是为什么我们还在使用 Python 2.7。本书，我们使用 Python 2.7 版本。版本的选择问题不在本书的讨论范围内，我们建议你检索如何用不同版本的 Python 编写代码。一些 IDE 工具包含一些具体的适合所有版本的使用说明。在一些情况下，这些代码不可避免会存在差异。

3.1.2 交互式工具类型

在进一步讨论 Python IDE 之前，考虑不同的交互数据可视化方法非常重要。尽管存在很多创建交互式数据可视化的选择，但这里我们仅考虑两种受欢迎的工具：

- IPython
- Plotly

1. IPython

在 2001 年，Fernando Perez 开始用 IPython 工作，这是一种经改进交互式得以强化的 Python shell，比如历史缓冲、配置文件、对象信息和会话日志。Python 从最初关注交互式计算拓展到后来包含 Julia、R、Ruby 等项目。附加了一些时间成本低且可用性效率高的功能选项，比如自动加括弧和 Tab 自动补全。标准的 Python 需要导入新模块实现 Tab 自动补全，而 IPython 默认提供了 Tab 自动补全。

IPython 为 Python 脚本提供了下面一组丰富的工具：

- 方便的终端命令和基于 Qt 的工具
- 一个交互式环境，纯粹是基于网络的 notebook，它们和独立的 notebook 有相同的核心功能；它也支持代码、文本、数学表达式和内嵌图
- 一种方便的交互式数据可视化；这种性能已经是很多种集成支持 IPython IDE 的原因
- 多进程计算中，提供了容易使用和高性能的工具

用简短的描述，给出 IPython 中四种最有用的命令：

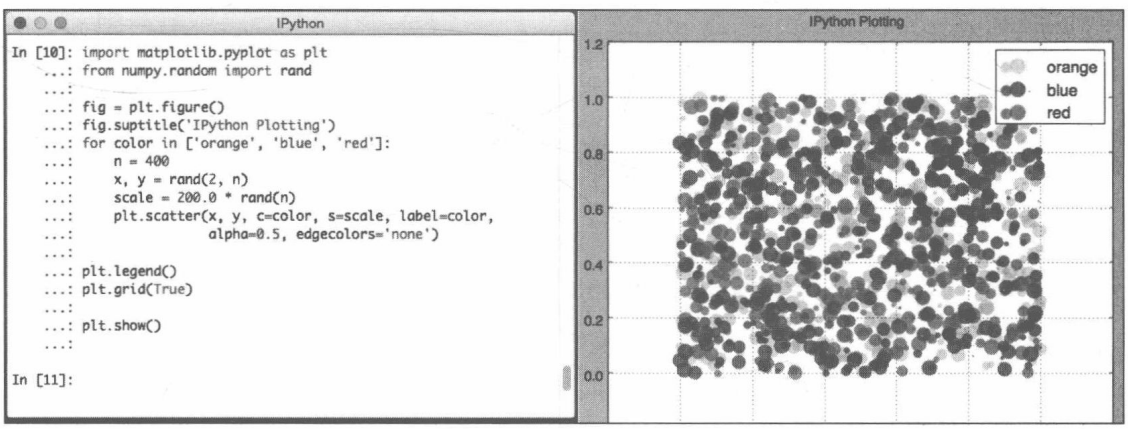
命　令	说　明
?	阐述了 IPython 的功能介绍和概述
%quickref	表示快速参考
--help-all	阐述了 Python 的帮助
%who/%whos	给出有关标识符的信息

IPython notebook 是一种基于网络的交互式计算环境。你可以用它将代码、数学和绘图合并到一个文件中。

IPython（http://ipython.scipy.org/）提供了一种交互式得以强化的 Python shell。之所以强烈推荐 IPython，主要是因为数据分析和可视化存在本质上的交互性。大多数平台都支持 IPython。IPython 的额外功能有：

- **Tab 自动补全**：这涉及变量、函数、方法、属性和文件名的完成。Tab 自动补全是通过 GNU Readline 库（http://tiswww.case.edu/php/chet/readline/rltop.html）实现的。用完 GNU Readline 之后，很难回到常规的命令行界面。
- **命令历史功能**：这是为充分考虑以前的命令而发布的命令历史。

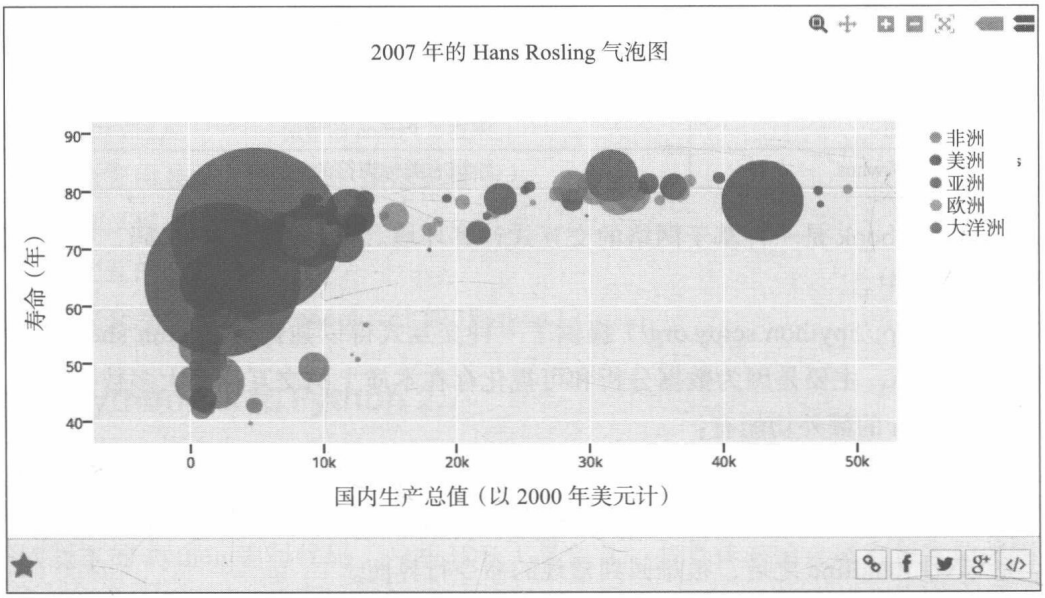
下面的截图展示了 IPython 上运行的一个案例。想要详细了解 IPython 和 IPython notebook，请参见 http://nbviewer.ipython.org。



2. Plotly

Plotly 是一种在线分析和数据可视化工具。为更好地协作，Plotly 提供了在线画图、分析学和统计工具。这种工具通过将 Python 与使用 JavaScript 的用户界面及和使用 D3.js、HTML 和 CSS 创建的可视化库一起使用来构建。Plotly 包括多语言兼容的科学绘图库，比如 Arduino、Julia、MATLAB、Perl、Python 和 R。Plotly 的案例请参见：https://plot.ly/~etpinard/84/fig-31a-hans-roslings-bubble-chart-for-the-year-2007/。

下面的气泡图展示了全球人均 GDP。



Plotly 提供了一种从 matplotlib 转换到 Plotly 的便捷画图方法，如下面的代码所示（假定你有一个 Plotly 账户，并且用凭据登录）：

```
import plotly.plotly as py
import matplotlib.pyplot as plt
#auto sign-in with credentials or use py.sign_in()
mpl_fig_obj = plt.figure()
#code for creating matplotlib plot
py.plot_mpl(mpl_fig_obj)
```

3.1.3 Python IDE 类型

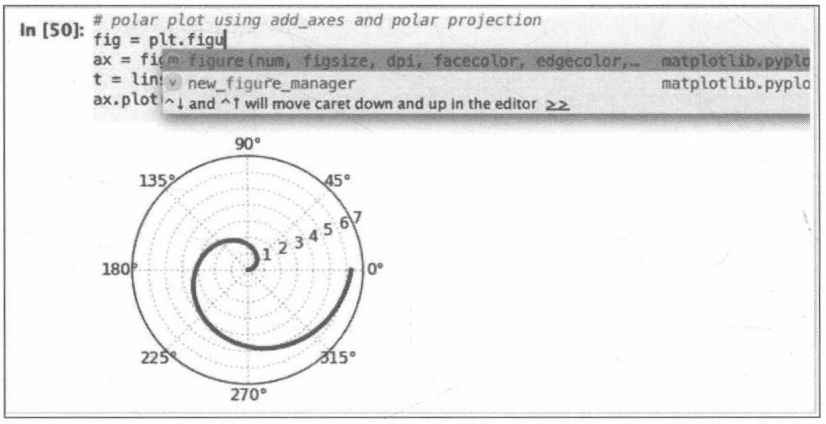
下面是目前可用的一些受欢迎的 Python IDE。

- PyCharm：这指定了基于 Java Swing 的用户界面
- PyDev：这是指基于 SWT 的用户界面（适用于 Eclipse）
- Python 的交互式编辑器（Interactive Editor for Python，IEP）
- Enthought 中的 Canopy：以 PyQt 为基础
- Continuum Analytics 中 Spyder 的 Anaconda 发行版：也以 PyQt 为基础

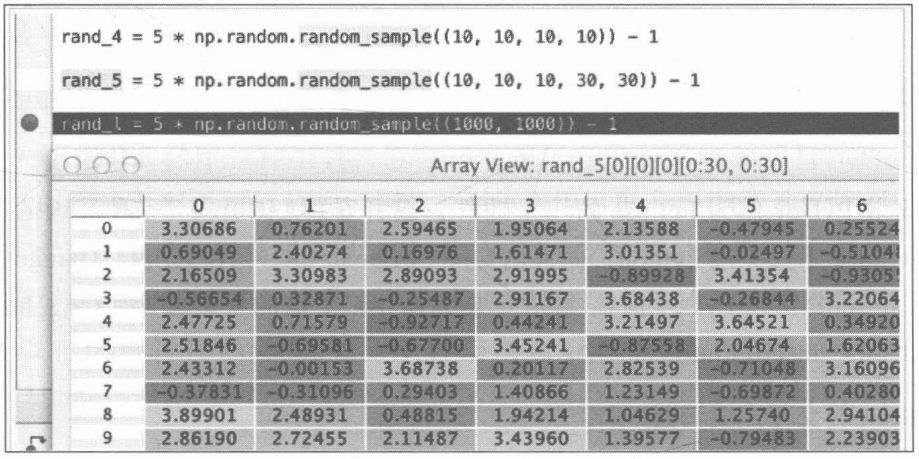
1. PyCharm

PyCharm 是一种少数受欢迎且功能强大的 IDE，而且社区版是免费的。PyCharm 4.0.6 社区版是本书写作时的最新版本，可从下面的网址免费下载：https://www.jetbrains.com/pycharm/download。它们有适用于 Mac、Linux 和 Windows 的快捷参考卡。Pedro Kroger

博士已在PyCharm上做了详尽的描述，网址如下：http://pedrokroger.net/getting-started-pycharm-python-ide/。你可以参考这个链接了解更多细节。在许多有趣的功能中，代码向导和NumPy的阵列查看器如下面的屏幕截图所示：



正如上面的屏幕截图所示，Polar项目可以很快完成，而且随机样本阵列的创建如下面的屏幕截图所示：



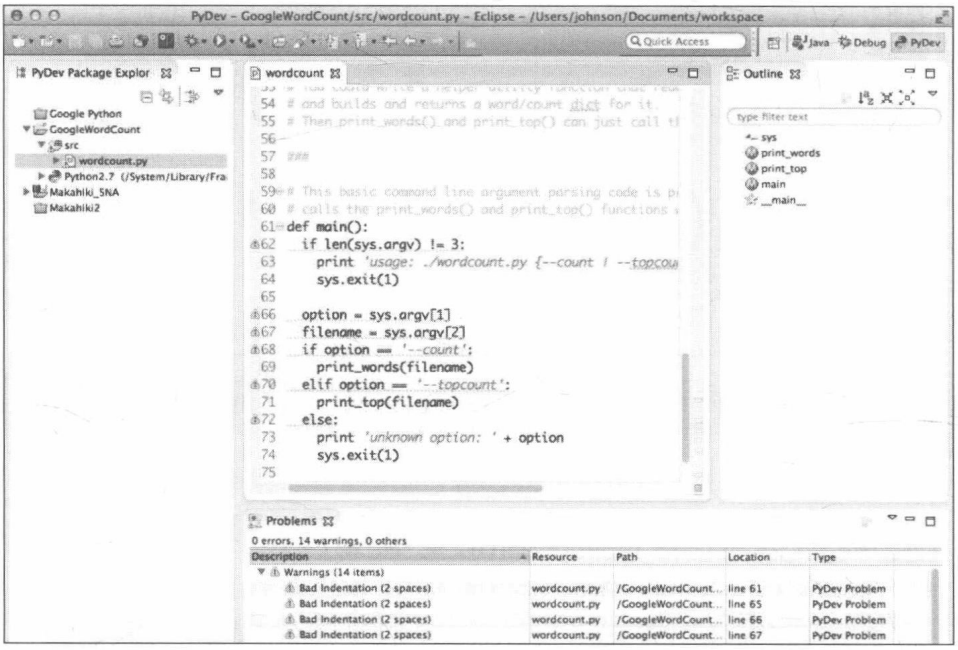
	0	1	2	3	4	5	6
0	3.30686	0.76201	2.59465	1.95064	2.13588	-0.47945	0.25524
1	0.69049	2.40274	0.16976	1.61471	3.01351	-0.02497	-0.5104
2	2.16509	3.30983	2.89093	2.91995	-0.89928	3.41354	-0.9305
3	-0.56654	0.32871	-0.25487	2.91167	3.68438	-0.26844	3.22064
4	2.47725	0.71579	-0.92717	0.44241	3.21497	3.64521	0.34920
5	2.51846	-0.69581	-0.67700	3.45241	-0.87558	2.04674	1.62063
6	2.43312	-0.00153	3.68738	0.20117	2.82539	-0.71048	3.16096
7	-0.37831	-0.31096	0.29403	1.40866	1.23149	-0.69872	0.40280
8	3.89901	2.48931	0.48815	1.94214	1.04629	1.25740	2.94104
9	2.86190	2.72455	2.11487	3.43960	1.39577	-0.79483	2.23903

在不同IDE（比如Spyder）中创建一个简单的随机样本，这里给出一个案例：

```
rand_4 = np.random.random_sample((2,2,2,2))-1
array([[[[-0.6565232 , -0.2920045 ],
[-0.45976502, -0.70469325]],
[[-0.80218558, -0.77538009],
[-0.34687551, -0.42498698]]],
[[[-0.60869175, -0.9553122 ],
[-0.05888953, -0.70585856]],
[[-0.69856656, -0.21664848],
[-0.29017137, -0.61972867]]]])
```

2. PyDev

PyDev 是 Eclipse IDE 的一个插件。换句话说，Eclipse 的一个插件足以利用 IDE 可能有的其他默认功能，而不是创建一个新的 IDE。PyDev 支持代码重构、图形化调试、交互式控制台、代码分析和代码折叠。



可以安装 PyDev 作为 Eclipse 的一个插件，或者安装 LiClipse（一种高级的 Eclipse 发行版）。LiClipse 不仅支持 Python，而且也支持 CoffeeScript、JavaScript、Django 模板等。

在 LiClipse 中安装 PyDev 前需要先安装 Java 7。完整的安装步骤请见：http://pydev.org/manual_101_ install.html。

3. Python 的交互式编辑器

IEP 是另一种与 IDE 有类似可用工具的 Python IDE，但看起来像 Microsoft Windows 中的工具一样。

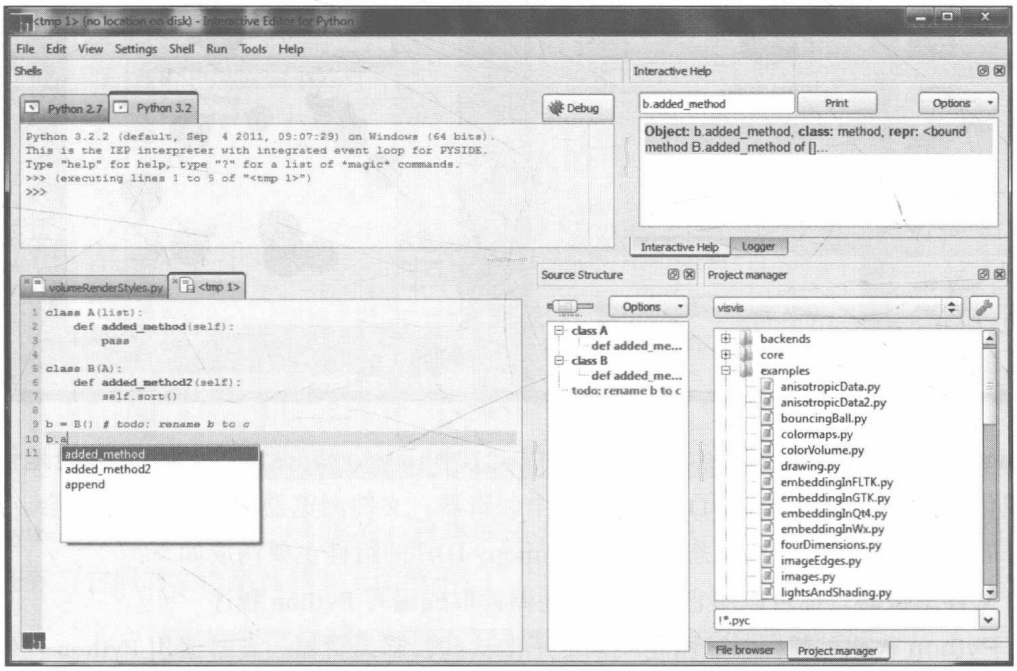
IEP 是以交互性和自检功能为目的的跨平台的 Python IDE，这使得它非常适合用于科学计算。出于简单和效率的考虑，IEP 的设计符合实用性原则。

IEP 包括编辑器和 shell 两个主要组件，通过一组插件工具尽可能为程序员提供帮助。一些案例工具是：源结构、项目管理器、交互式帮助和工作区。一些重要的功能如下：

- ❑ 与任何现代 IDE 类似的代码自检功能
- ❑ 既可以从命令行运行 Python 脚本，也可以通过文件或 IPython 界面完成交互

- 作为后台处理进程运行 shell
- 多 shell 可以用不同的 Python 版本（从 v2.4 到 3.x）

下面的屏幕截图展示出如何在同一个 IDE 中用两个不同版本的 Python：

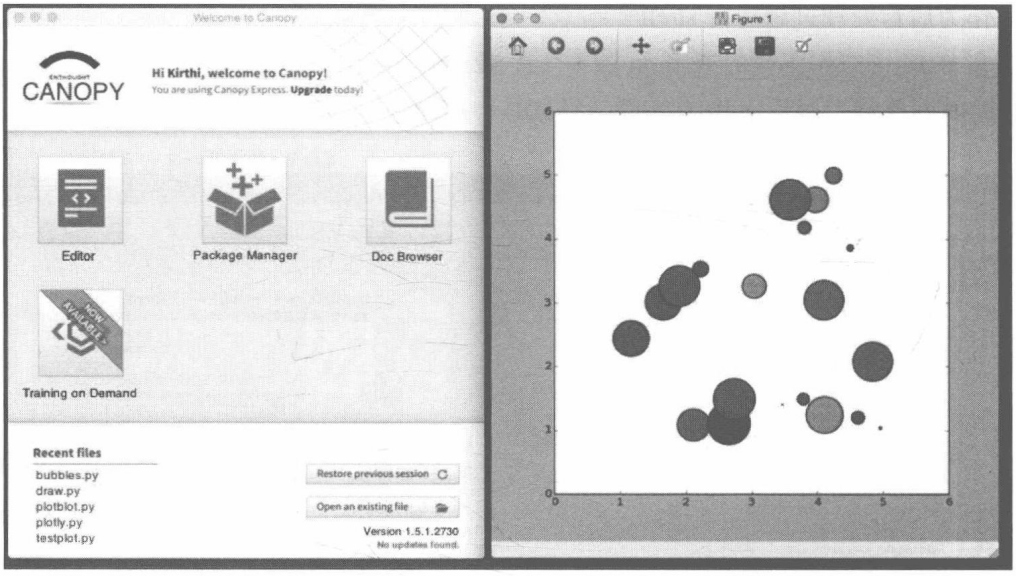


一些人不把 IEP 看作一种工具，但它却实现了开发、编辑和运行 Python 程序的目的，而且同时支持多个 Python shell。因此，如果想用不止一个 GUI 工具包（比如 PySide、PyQt4、GTK 和 Tk）交互地完成编程，IEP 将是一个非常高产的工具。

IEP 可（纯粹地）在 Python 3 中编写，并且用 Qt GUI 工具包，但是它可以在任何一个 Python 版本中执行代码。IEP 可从下面的网址进行下载：http://www.iep-project.org/downloads.html。

4. Enthought 中的 Canopy

在其他库中，Enthought Canopy 有一个在 BSD 风格执照下发布的免费版本，而且以 GraphCanvas、SciMath 和 Chaco 为绘图工具。像所有 IDE 一样，它有一个文本编辑器。它也有 IPython 控制台，这对运行和可视化结果非常有用。此外，它还有一个图形软件包管理器。启动 Canopy，将给出一个编辑器、软件包管理器和文档浏览器的选项以供选择。也可以尝试使用它们的训练材料，如下面的屏幕截图所示：



Canopy 除了其他开发代码，同时也集成了 IPython notebook 和能够用来便捷地创建数据可视化的函数。像大多数 IDE，它有一个编辑器、文件浏览器和 IPython 控制台。此外，有一个展示当前编辑状态的状态显示器。Canopy IDE 的组件主要构成如下：

- **文件浏览器**：你可以用它从硬盘驱动器读取或编写 Python 程序
- **Python 代码编辑器**：这指定突出显示语法的代码编辑器，自带适用 Python 代码的额外功能
- **Python 窗格**：这是一个集成的 IPython（交互式 Python）提示，能用来交互地运行 Python 程序，而不是从一个文件运行
- **编辑器状态条**：可以用来显示行号、列号、文件类型和文件路径

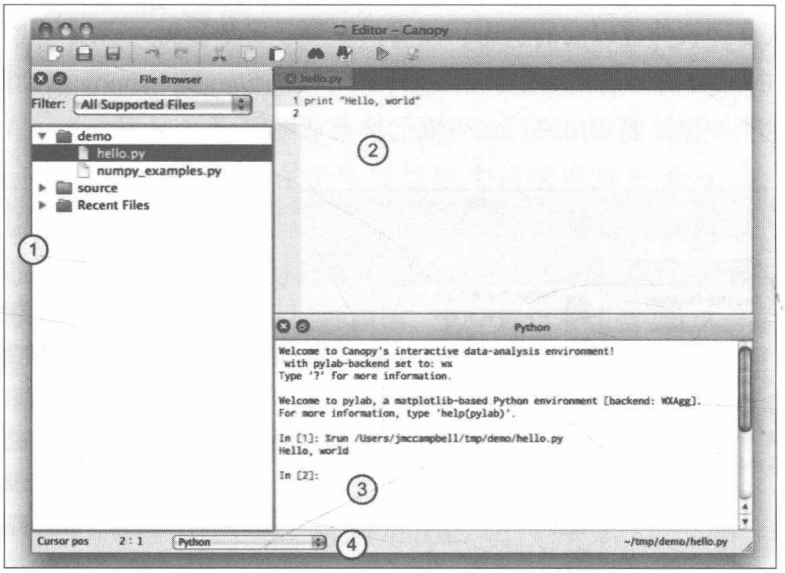
下面的屏幕截图突出显示了数字，代表了前面描述过的 IDE 组件。文件浏览器和 Python 窗格可以在代码编辑器窗口内外被拖放到不同位置。当窗格被拖放时，所在的位置突出显示，如下面的屏幕截图所示：

该文档通过 Canopy 文档浏览器完成组织，具体可见**帮助**菜单。这包括一些常用 Python 软件包的文档链接。

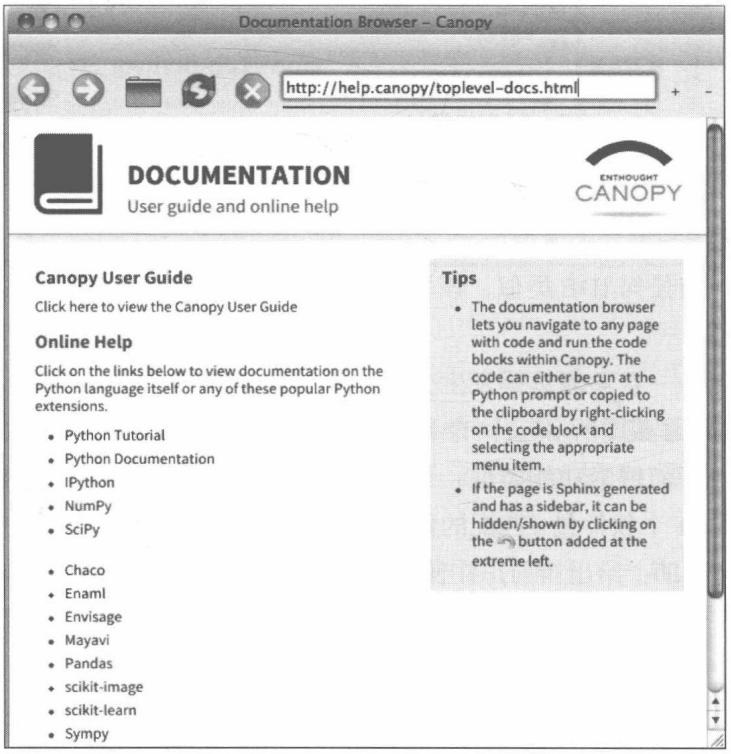
文档浏览器的一个显著特征是容易得到文档形式的样本代码。当用户鼠标右击样本代码箱时，可以显示上下文菜单。进而，你可以选择**复制代码**选项，将代码的内容复制到编辑器中的 Canopy 复制 – 粘贴缓冲区。

Canopy 有几款不同的产品，其免费版本叫做 Canopy Express。它有近 100 个核心软件包。该免费版本是一种用 Python 开发便于科学和分析计算的有用工具。在 Windows、Linux 或 Mac OS 中选择其中一个作为目标操作系统后，你可以从下面的网址下载这个版

本：https://store.enthought.com/downloads/。

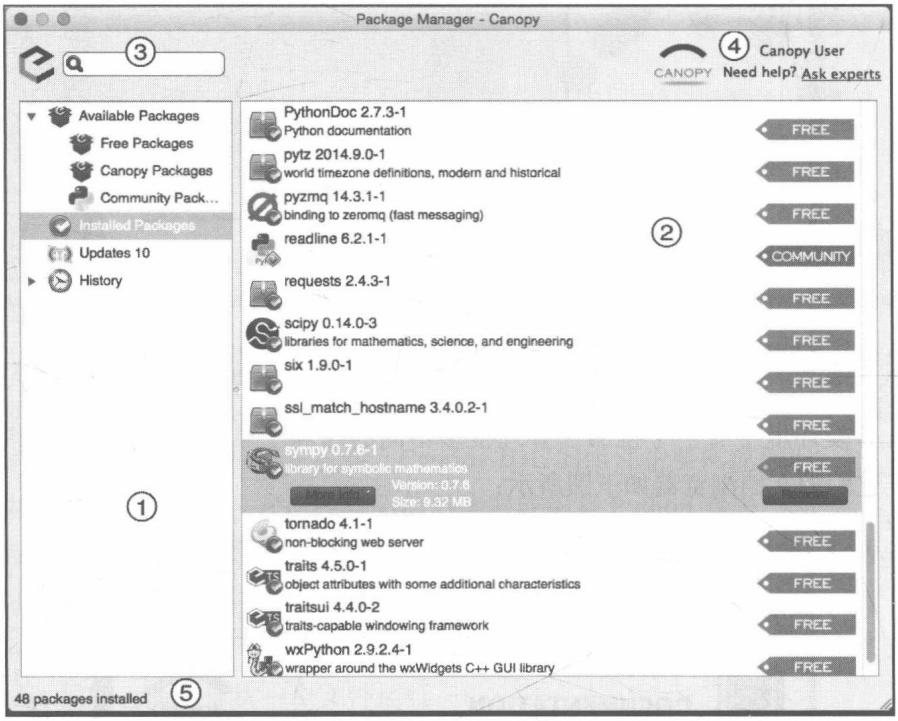


Python开发环境中有一个挑战：管理很多不同的库和工具软件包是一项非常耗时且艰巨的任务。它们的文档浏览器如下图所示：



Canopy 有一个软件包管理器，可以用来发现 Canopy 中可用的 Python 软件包，并决定安装和删除哪些额外的软件包。此外，还有一个便捷的搜索界面用来发现和安装任何可用的软件包，并恢复到软件包以前的状态。

Canopy 用 Python 的功能来确定可用的 Python 软件包。当 Canopy 启动时，它首先在可视化环境中查找软件包并展示出来，如下面的屏幕截图所示：



IDE 中突出显示的编号区域为：

1. **导航面板**：与任何 IDE 类似，该导航有一种树状列表的结构，用以选择软件包管理器的组件。

2. **主视图区域**：一旦左侧选择发生变化，右侧面板将展示可选项目，以及有关的软件包列表（如上面的屏幕截图所示），一个名为 More Info 的特定软件包信息按钮，等等。

3. **搜索栏**：与任何搜索功能类似，帮助快速搜索软件包的名称和描述。例如，打字机过滤列表，删减到 11 个软件包（匹配的数量可以根据操作系统而发生变化）。

4. **订阅状态和帮助**：给出当前正在使用的订阅和账户名称的链接。

5. **状态栏**：对于每一个用户导航，状态栏将展示当前导航变化结果的细节。

5. Continuum Analytics 中的 Anaconda

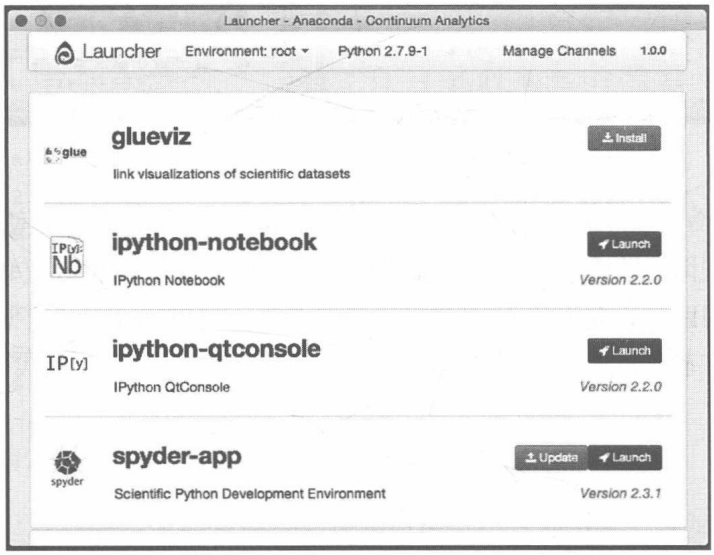
Anaconda 是社群最受欢迎的 IDE。它包括一列集成的软件包清单。IDE 以名为 conda

的核心组件为基础（后面会有详细解释），而且可用 conda 或 pip 安装或更新 Python 软件包。

Anaconda 是一组免费而强大的 Python 软件包，用以实现大规模数据管理、分析以及商务智能、科学分析、工程、机器学习等更多内容的可视化。

Anaconda 有一个科学 Python 研发环境（Scientific Python Development Environment，Spyder），后者有一个 IPython 查看器。此外，IPython 可以被启动为一个 GUI，或一个基于网络的 notebook。最便捷之处在于你可以在主目录安装 Python，而不用接触安装 Python 的系统。Python 3 并没有囊括所有软件包，因此，最好在 Python 2 中使用这些 IDE。Anaconda IDE 有两个重要的组件（conda 和 spyder），它以 conda 软件包管理器为基础。

下面的屏幕截图是启动 Anaconda 时的画面，其中展示了包括 IPython 控制台、IPython notebook、Spyder IDE 和 glueviz 的一些选项：

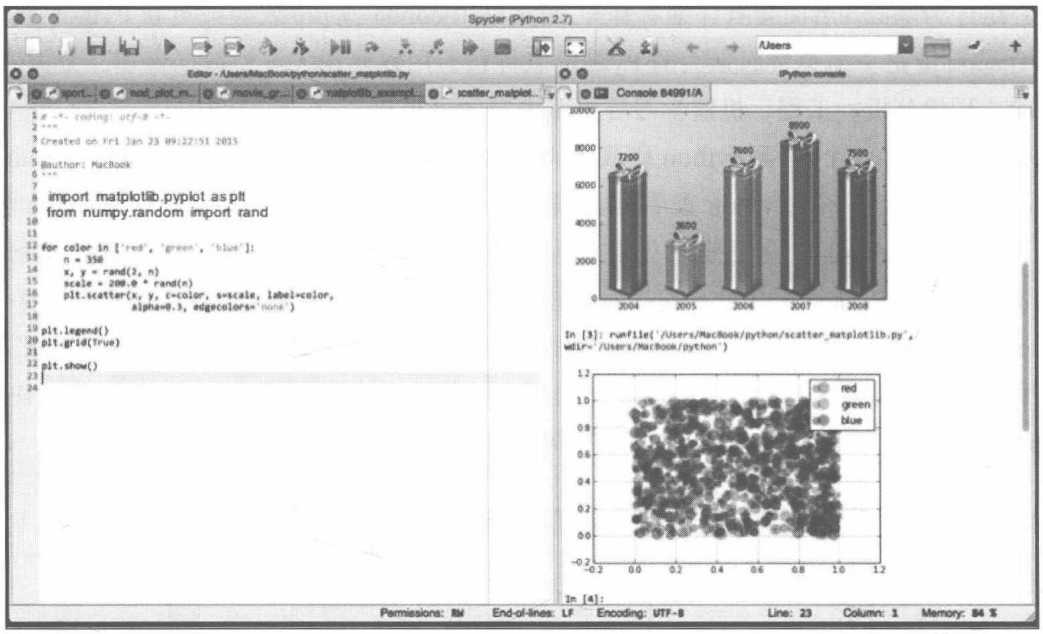


Spyder 综述

Spyder 是具备以下组件的 Python 开发环境：

- **Python 代码编辑器**：这配备有一个单独的函数浏览器，而且类编辑器支持 Pylint 代码分析。如今，代码完成已经变为一种规范，而且在所有的 IDE 中都很方便，因此，它也支持代码完成功能。
- **交互式控制台**：Python 语言最适用于交互式工作；因此，当务之急是控制台具备所有支持即时评价编辑器中已写代码的必要工具。
- **探索变量**：在任何交互式执行中，探索变量有助于提高整体生产率。编辑变量也是可能的，比如字典，有时是阵列。

代码编辑器和 IPython 控制台如下面的屏幕截图所示：



conda 综述

conda 是一种命令行工具，用于管理环境和 Python 软件包，而不必使用 pip。现有几种方法，可以用来查询和搜索软件包，必要时创建新环境，或安装和更新 Python 软件包。这个命令行工具也跟踪软件包和平台细节间的依赖关系，这有助于结合不同软件包来创建新的工作环境。为了验证运行的是哪个 conda 版本，你可以输入下面的代码（本书使用的是 3.10.1 版本）：

```
Conda -v
3.10.1
```

conda 环境是一个文件系统目录，包括一组具体的 conda 软件包。作为一个具体的例子，你可能需要一个提供 NumPy 1.7 的环境和另一个为传统测试提供 NumPy 1.6 的环境；conda 使得这种混合和匹配变得容易。为了开始使用环境，仅需要设置 PATH 变量来指向它的 bin 目录。

让我们看看这样一个例子，如何用 conda 安装 SciPy 软件包。假设你已经正确安装 Anaconda，而且 conda 在运行路径中可以正常使用，那么你需要输入下面的代码来安装 SciPy：

```
$ conda install scipy

Fetching package metadata: ....
Solving package specifications: .
Package plan for installation in environment /Users/MacBook/anaconda:
```

```
The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    flask-0.10.1               |           py27_1         129 KB
    itsdangerous-0.23          |           py27_0          16 KB
    jinja2-2.7.1               |           py27_0         307 KB
    markupsafe-0.18            |           py27_0          19 KB
    werkzeug-0.9.3             |           py27_0         385 KB

The following packages will be linked:

    package                    |            build
    ---------------------------|-----------------
    flask-0.10.1               |           py27_1
    itsdangerous-0.23          |           py27_0
    jinja2-2.7.1               |           py27_0
    markupsafe-0.18            |           py27_0
    python-2.7.5               |                2
    readline-6.2               |                1
    sqlite-3.7.13              |                1
    tk-8.5.13                  |                1
    werkzeug-0.9.3             |           py27_0
    zlib-1.2.7                 |                1

Proceed ([y]/n)?
```

应该注意的是，任何需要安装的软件包都应该可以自动识别、下载和连接。如果任何软件包需要安装或更新，你必须用下面的代码：

```
conda install <package name>  or conda update <package name>
```

下面是一个使用 conda 从命令行更新软件包的例子（更新 matplotlib）：

```
conda update matplotlib

Fetching package metadata: ....
Solving package specifications: .
Package plan for installation in environment /Users/MacBook/anaconda:

The following packages will be downloaded:
```

```
package                    |            build
---------------------------|-----------------
freetype-2.5.2             |                0         691 KB
conda-env-2.1.4            |          py27_0          15 KB
numpy-1.9.2                |          py27_0         2.9 MB
pyparsing-2.0.3            |          py27_0          63 KB
pytz-2015.2                |          py27_0         175 KB
setuptools-15.0            |          py27_0         436 KB
conda-3.10.1               |          py27_0         164 KB
python-dateutil-2.4.2      |          py27_0         219 KB
matplotlib-1.4.3           |       np19py27_1        40.9 MB
------------------------------------------------------------
                                       Total:        45.5 MB

The following NEW packages will be INSTALLED:

    python-dateutil: 2.4.2-py27_0

The following packages will be UPDATED:

    conda:           3.10.0-py27_0     --> 3.10.1-py27_0
    conda-env:       2.1.3-py27_0      --> 2.1.4-py27_0
    freetype:        2.4.10-1          --> 2.5.2-0
    matplotlib:      1.4.2-np19py27_0 --> 1.4.3-np19py27_1
    numpy:           1.9.1-py27_0      --> 1.9.2-py27_0
    pyparsing:       2.0.1-py27_0      --> 2.0.3-py27_0
    pytz:            2014.9-py27_0     --> 2015.2-py27_0
    setuptools:      14.3-py27_0       --> 15.0-py27_0

Proceed ([y]/n)?
```

为了检验用 Anaconda 安装的软件包，导航至命令行，输入下面的命令快速展示默认环境下安装的所有软件包列表：

```
conda list
```

此外，还有一些常用的方法安装一软件包，比如 pip install，或者用 setup.py 源文件安装。尽管 conda 是软件包安装工具的首选，但与 pip 相比，也没有特别优势。

虽然 IPython 不是必需的，但还是强烈推荐使用。IPython 应该在 Python、GNU Readline 和 PyReadline 之后安装。Anaconda 和 Canopy 默认完成这些工作。本书包括所有

案例中使用的 Python 软件包。我们在下节会更新这个软件包列表。

3.2　Anaconda 可视化绘图

从数据的获取，操作和处理，再到数据可视化和交流研究成果，Python 和 Anaconda 都支持科学数据流程中的各种处理过程。Python 能用于各种各样的应用程序（不限于科学计算）；用户能够快速采用这种语言，而且不需要学习新的软件或编程语言。Python 开源的可用性强化了研究成果，使用户可以连接全球众多的科学家和工程师。

下面是 Anaconda 中常见的一些绘图库：

- matplotlib：这是 Python 中最受欢迎的绘图库。和 NumPy 和 SciPy 一起，是科学 Python 社区中的主要驱动力。IPython 有一种 pylab 模式，这是专门设计用来使用 matplotlib 进行交互式绘图的。
- Plotly：这是一个在浏览器上工作的绘图分析平台。它支持用 IPython notebook 的交互式图表。图是可交互的，而且可以通过修改代码和交互检查结果而改变风格。使用 matplatlib 生成的绘图代码可以容易地移值到 Plotly 上。
- Veusz：这是一个用 Python 和 PyQt 编写的 GPL- 科学绘图软件包。Veusz 也能嵌入到其他 Python 程序中。
- Mayavi：这是一个三维绘图软件包，完全支持 Python 脚本，与用于绘图阵列的简单的 pylab 和类 MATLAB 界面类似。
- NetworkX：这是一个 Python 语言软件包，用来创建、操作和研究结构、动力学，以及复杂网络的函数。
- Pygooglechart：这是一个创建可视化方法的强大的软件包，允许你与 Google 图表 API 连接。

3.2.1　表面三维图

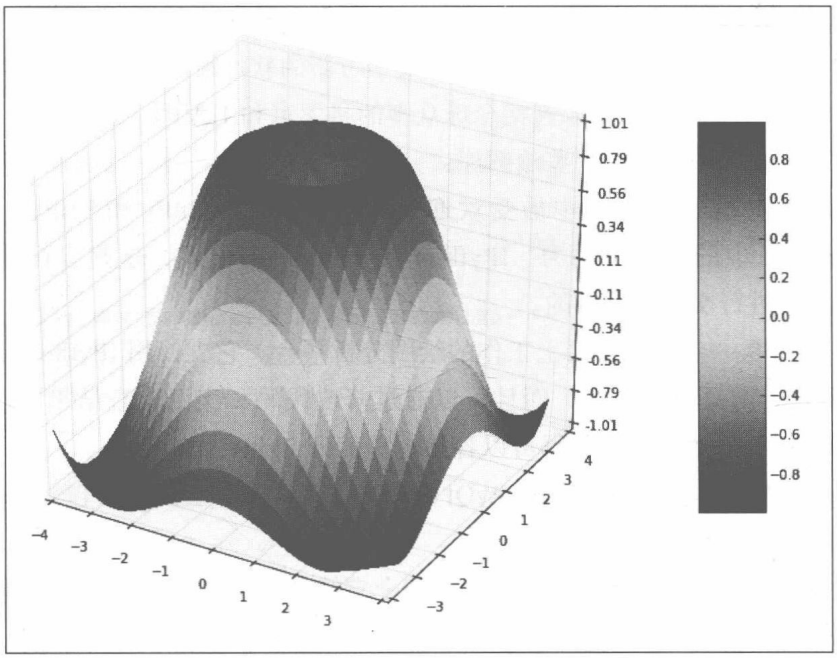
三维图由数据生成，该数据定义为 (X, Y) 的函数 Z。数学表示为 $Z=f(X, Y)$。在这个例子中，我们将绘制 $Z=\sin(\text{sqrt}(X2+Y2))$，这与二维抛物线基本相似。绘图时，需要遵守下述步骤。

1. 首先，用下面的代码产生 X 和 Y 网格：

```
import numpy as np

X = np.arange(-4, 4, 0.25)
Y = np.arange(-4, 4, 0.25)
X, Y = np.meshgrid(X, Y)
```

```
Generate the Z data:
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)
```

这里展示的是用 mpl_toolkits 软件包绘制 sin(sqrt(*X***2+*Y***2)) 的简单三维表面图，下图用条形表示：



2. 然后，通过下面的代码绘图：

```
from mpl_toolkits.mplot3d import Axes3d
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure(figsize=(12,9))
ax = fig.gca(projection='3d')
X = np.arange(-4, 4, 0.25)
Y = np.arange(-4, 4, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)

surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.
coolwarm, linewidth=0, antialiased=False)

ax.set_zlim(-1.01, 1.01)
```

```
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

fig.colorbar(surf, shrink=0.6, aspect=6)

plt.show()
```

为了绘制三维图，你必须确保已安装 matplotlib 和 NumPy。Anaconda 已默认安装了一些软件包。

3.2.2 方形图

在上一章，我们讨论了比较和排名的案例，使用 squarify 算法（和 matplotlib）展示了非洲 GDP 排前 12 名的国家。你可以获得树形图，代码如下所示：

```
# Squarified Treemap Layout : source file (squarify.py)
# Implements algorithm from Bruls, Huizing, van Wijk, "Squarified
Treemaps"
# squarify was created by Uri Laserson
# primarily intended to support d3.js

def normalize_sizes(sizes, dx, dy):
  total_size = sum(sizes)
  total_area = dx * dy
  sizes = map(float, sizes)
  sizes = map(lambda size: size * total_area / total_size, sizes)
  return sizes

def pad_rectangle(rect):
  if rect['dx'] > 2:
    rect['x'] += 1
    rect['dx'] -= 2
  if rect['dy'] > 2:
    rect ['y'] += 1
    rect['dy'] -= 2

def layoutrow(sizes, x, y, dx, dy):
  covered_area = sum(sizes)
  width = covered_area / dy
  rects = []
  for size in sizes:
    rects.append({'x': x, 'y': y, 'dx': width, 'dy': size / width})
    y += size / width
  return rects


def layoutcol(sizes, x, y, dx, dy):
  covered_area = sum(sizes)
  height = covered_area / dx
```

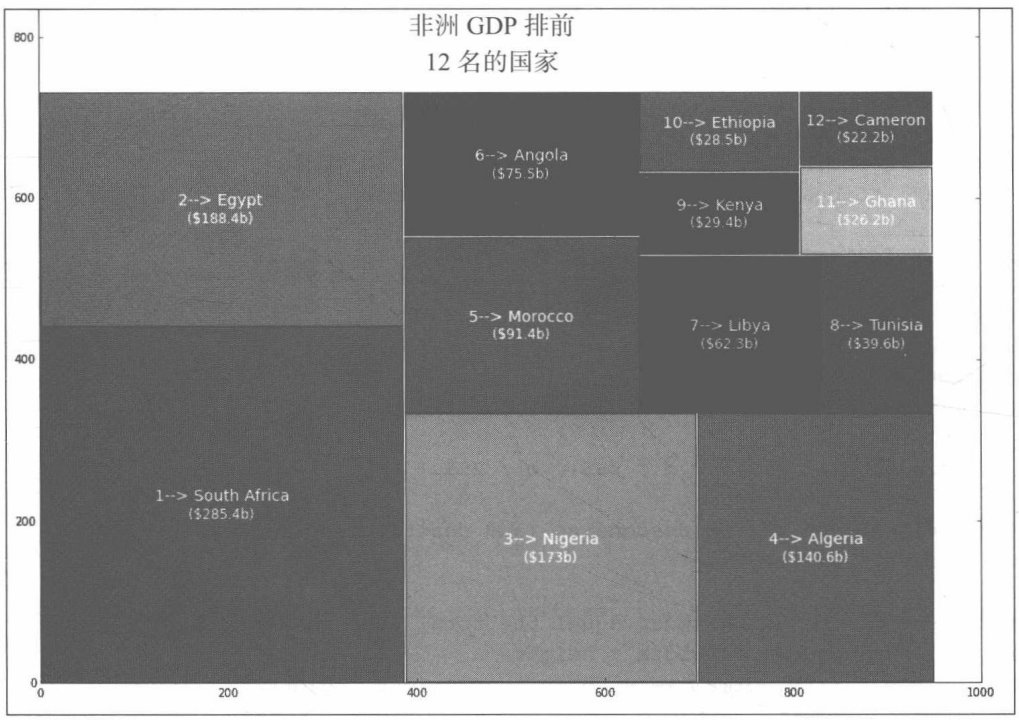
```
  rects = []
  for size in sizes:
    rects.append({'x': x, 'y': y, 'dx': size / height, 'dy': height})
    x += size / height
  return rects

def layout(sizes, x, y, dx, dy):
  return layoutrow(sizes, x, y, dx, dy) if dx >= dy else
layoutcol(sizes, x, y, dx, dy)

def leftoverrow(sizes, x, y, dx, dy):
  covered_area = sum(sizes)
  width = covered_area / dy
  leftover_x = x + width
  leftover_y = y
  leftover_dx = dx - width
  leftover_dy = dy
  return (leftover_x, leftover_y, leftover_dx, leftover_dy)

def leftovercol(sizes, x, y, dx, dy):
  covered_area = sum(sizes)
  height = covered_area / dx
  leftover_x = x
  leftover_y = y + height
  leftover_dx = dx
  leftover_dy = dy - height
  return (leftover_x, leftover_y, leftover_dx, leftover_dy)

def leftover(sizes, x, y, dx, dy):
  return leftoverrow(sizes, x, y, dx, dy) if dx >= dy else
leftovercol(sizes, x, y, dx, dy)

def worst_ratio(sizes, x, y, dx, dy):
  return max([max(rect['dx'] / rect['dy'], rect['dy'] / rect['dx'])
for rect in layout(sizes, x, y, dx, dy)])

def squarify(sizes, x, y, dx, dy):
  sizes = map(float, sizes)
  if len(sizes) == 0:
    return []
  if len(sizes) == 1:
    return layout(sizes, x, y, dx, dy)
  # figure out where 'split' should be
  i = 1
  while i < len(sizes) and worst_ratio(sizes[:i], x, y, dx, dy) >=
worst_ratio(sizes[:(i+1)], x, y, dx, dy):
    i += 1
  current = sizes[:i]
  remaining = sizes[i:]
  (leftover_x, leftover_y, leftover_dx, leftover_dy) =
leftover(current, x, y, dx, dy)
```

```
    return layout(current, x, y, dx, dy) + \
  squarify(remaining, leftover_x, leftover_y, leftover_dx, leftover_dy)

  def padded_squarify(sizes, x, y, dx, dy):
    rects = squarify(sizes, x, y, dx, dy)
    for rect in rects:
      pad_rectangle(rect)
    return rects
```

上面代码中的 squarify 函数能用来展示非洲 GDP 排前 12 名的国家，代码如下所示：

```
import matplotlib.pyplot as plt
import matplotlib.cm
import random
import squarify

x = 0.
y = 0.
width = 950.
height = 733.
norm_x=1000
norm_y=1000

fig = plt.figure(figsize=(15,13))
ax=fig.add_subplot(111,axisbg='white')

initvalues = [285.4,188.4,173,140.6,91.4,75.5,62.3,39.6,29.4,28.5,
26.2, 22.2]
values = initvalues
labels = ["South Africa", "Egypt", "Nigeria", "Algeria", "Morocco",
"Angola", "Libya", "Tunisia", "Kenya", "Ethiopia", "Ghana", "Cameron"]

colors = [(214,27,31),(229,109,0),(109,178,2),(50,155,18),
(41,127,214),(27,70,163),(72,17,121),(209,0,89),
(148,0,26),(223,44,13), (195,215,0)]
# Scale the RGB values to the [0, 1] range, which is the format
matplotlib accepts.
for i in range(len(colors)):
  r, g, b = colors[i]
  colors[i] = (r / 255., g / 255., b / 255.)

# values must be sorted descending (and positive, obviously)
values.sort(reverse=True)

# the sum of the values must equal the total area to be laid out
# i.e., sum(values) == width * height
values = squarify.normalize_sizes(values, width, height)

# padded rectangles will probably visualize better for certain cases
rects = squarify.padded_squarify(values, x, y, width, height)
```

```
cmap = matplotlib.cm.get_cmap()

color = [cmap(random.random()) for i in range(len(values))]
x = [rect['x'] for rect in rects]
y = [rect['y'] for rect in rects]
dx = [rect['dx'] for rect in rects]
dy = [rect['dy'] for rect in rects]

ax.bar(x, dy, width=dx, bottom=y, color=colors, label=labels)

va = 'center'
idx=1

for l, r, v in zip(labels, rects, initvalues):
  x, y, dx, dy = r['x'], r['y'], r['dx'], r['dy']
  ax.text(x + dx / 2, y + dy / 2+10, str(idx)+"--> "+l, va=va,
     ha='center', color='white', fontsize=14)
  ax.text(x + dx / 2, y + dy / 2-12, "($"+str(v)+"b)", va=va,
     ha='center', color='white', fontsize=12)
  idx = idx+1
ax.set_xlim(0, norm_x)
ax.set_ylim(0, norm_y)
plt.show()
```



3.3 交互式可视化软件包

几年前，除了IPython外，其他交互式工具并不多。为了理解如何使可视化变成可交互的，需要用一个现有的工具（比如D3.js）进行比较，这样才有意义。D3.js之所以很强大，是因为基于JavaScript的绘图框架使这些图可以在网上展示。而且，它自带可以轻松配置的所有事件驱动函数。

在一些可用库中，名为Bokeh和VisPy的可视化库很受欢迎。还有另一个工具称为Wakari。就如何创建基于浏览器的可视化而言，Wakari与IPython相似，主要用于数据分析。Ashiba项目是由Clayton Davis在Continuum研发的另一种工具，但由于Continuum的重点转移到Bokeh和Wakari上，因此过去几年在Ashiba上完成的工作非常少。

3.3.1 Bokeh

Bokeh是一个在Python中开发的交互式可视化库，其目标是通过网络浏览器进行工作。Bokeh这一名字来源于哪？这是一个日语单词，描述图片中模糊或失焦的部分。其目标是开发一个美学上与D3.js相似的库；名字Bokeh的选择看起来是匹配的。Bokeh写入HTML5 Canvas库，因此能保证在支持HTML5的浏览器上工作。这一点很有用，能比较基于JavaScript的图与Python绘图的不同。

我们不再赘述这个工具。你可以从以下网址读取和探索更多信息：http://bokeh.pydata.org。然而，重要的是了解Bokeh库的依赖性。在安装Bokeh库前，需要先安装jsonschema，如下所示：

```
conda install jsonschema

Fetching package metadata: ....
Solving package specifications: .
Package plan for installation in environment /Users/MacBook/anaconda:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    jsonschema-2.4.0           |          py27_0          51 KB

The following NEW packages will be INSTALLED:

    jsonschema: 2.4.0-py27_0

Proceed ([y]/n)?
```

用 Bokeh、pandas、SciPy、matplotlib 和 ggplot 实现交互式可视化的案例可见：http://nbviewer.ipython.org/gist/fonnesbeck/ ad091b81bffda28fd657。

3.3.2 VisPy

VisPy 是一个绘制交互式高性能的 2D 或 3D 可视化库。你可以利用 OpenGL 的知识快速创建可视化，其中包括一些不需要深入理解 OpenGL 的方法。更多的信息请阅读 vispy.org 上的文档。

为了安装 VisPy 库，可以尝试用 conda install vispy 命令，但这最有可能用 binstar 搜索 – t conda vispy 建议响应。下面的代码是列表中的其中一项：

```
conda install --channel https://conda.binstar.org/asmeurer vispy
```

用这行命令，你将得到下面的响应：

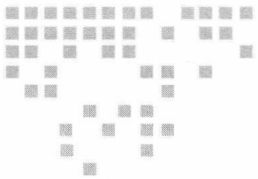
```
Fetching package metadata: ......
Solving package specifications: .
Package plan for installation in environment /Users/MacBook/anaconda:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    numpy-1.8.2                |           py27_0         2.9 MB
    vispy-0.3.0                |       np18py27_0         679 KB
    ------------------------------------------------------------
                                           Total:         3.6 MB

The following NEW packages will be INSTALLED:

    vispy: 0.3.0-np18py27_0

The following packages will be DOWNGRADED:

    numpy: 1.9.2-py27_0 --> 1.8.2-py27_0

Proceed ([y]/n)?
```

在 VisPy 图库中有很多例子。其中一个特别的例子是用 vispy.gloo 命令和 GLS L 图形编程代码实现的可视化，请见 http://vispy.org/gloo.html?highlight=gloo#module-vispy. gloo。

3.4 总结

现在，Python 开发者已有一套可用的好工具和软件包。Python 有一个庞大的标准库，其中的模块可以创建图形用户界面、连接关系数据库、伪随机数生成器、任意精度小数的算术和操作正则表达式。此外，还有绘制 2D 和 3D 图的高性能软件包、机器学习和统计算法，等等。

在很多其他领域中，我们已经看到 IDE 工具（比如 Canopy 和 Anaconda）能高效地完成计算和可视化开发工作。利用这些工具，可以获得很多产生可视化方法的有效途径。在后面的几章，我们将用这些工具和软件包展示有趣的案例。



Chapter 4 第 4 章

数值计算和交互式绘图

高性能的数值计算领域在于众多学科和技能的交叉。当前，为了成功实现高性能的数值计算，人们需要同时掌握编程、数据科学、应用数学等领域的知识。此外，计算问题的有效解决还需要理解处理过程和存储设备。

近年来，科学领域的数值计算已经上升到一个新的水平。以前，编程语言（比如 R 和 MATLAB）在学术研究和科学计算中十分常见。而如今，Python 在科学计算领域扮演着重要角色。原因是 Python 集合了众多有效的工具和软件包，这些工具和软件包不仅被研究群体使用，而且也被著名商业机构使用，如 Yahoo、Google、Facebook 和 Amazon 等公司。

有两大软件包广泛用于科学计算。它们是 Numerical Python Package（NumPy）和 Scientific Python Package（SciPy）。NumPy 之所以深受喜爱是因为它有效的排列功能和简便的索引方式。本章我们将讨论以下内容：

- ❑ NumPy、SciPy 和 MKL 函数
- ❑ 数值索引和逻辑索引
- ❑ 数据结构——栈、队列、元组、集合、检索树和字典
- ❑ 利用 matplotlib 等包进行数据绘图与可视化
- ❑ 举例使用 NumPy 和 Scipy 实现优化和插值
- ❑ 用 Numpy 整合 Cython，以及 Cython 的优点

4.1　NumPy、SciPy 和 MKL 函数

几乎所有的科学计算和数值计算都需要将数据表示为向量或矩阵的形式，而 Numpy 可以使用数组处理所有这些问题。

NumPy 和 SciPy 是 Python 的计算模块，它们用预先编译好的、快速的函数提供了方便的数学和数值方法。NumPy 包提供了处理大型数值数组或矩阵的基本方法。Scipy 包则在 NumPy 的基础上，增加了众多有用的应用数学技术算法。在 Numpy 中，ndarray 是一个数组对象，用来表示已知长度的多维同质数组。

4.1.1　NumPy

NumPy 不仅包含数组对象，同时还包含许多方便用于计算的线性代数函数。NumPy 提供了数组的快速实现方法，以及与数组相关的函数。利用数组对象，我们可以轻松实现包括矩阵乘法，向量和矩阵的转置，方程组求解，向量乘法和向量规一化等众多操作。

1. NumPy 的通用函数

通用函数（universal function，ufunc）是在 ndarray 上对每个元素进行逐一处理、支持数据类型转换，并具有其他的特征的函数。换句话说，ufunc 是一种对具有标量输入、有标量输出的函数的向量化包装。很多内嵌函数都是在编译好的 C 程序中实现的，这一点使得运行速度更快。因为循环是在编译好的程序中实现的，所以 NumPy 的通用函数比 Python 自带的函数执行速度更快。而且，数组已经被分过类，故它们的类型在任何一种计算出现之前都是已知的。

下面是 ufunc 作用于每个元素的一个简单示例：

```
import numpy as np
x = np.random.random(5)
print x
print x + 1    # add 1 to each element of x

[ 0.62229809  0.18010463  0.28126201  0.30701477  0.39013144]
[ 1.62229809  1.18010463  1.28126201  1.30701477  1.39013144]
```

其他例子有 np.add 和 np.subtract。

NumPy 的 ndarray 类似于 Python 中的列表，但 ndarray 要求存储在数组中的元素类型必须一致。换句话说，同一个 Python 列表中可以存储不同类型的元素，如列表中的第一个元素是一个数值型数字，第二个元素是一个列表，第三个元素是另一个列表（或字典）。在下面的例子中，通过比较使用两种类型处理数据的运算时间，我们可以看到对一个大型数组，用 ndarray 处理元素的速度比用列表进行处理的速度要快得多。如果你想知道 Numpy

是如何在C当中实现的，请访问http://docs.scipy.org/doc/numpy/reference/internals.code-explanations.html，该网站中包含一篇说明文档。

```
import numpy as np

arr = np.arange(10000000)
listarr = arr.tolist()

def scalar_multiple(alist, scalar):
    for i, val in enumerate(alist):
        alist[i] = val * scalar
    return alist

# Using IPython's magic timeit command
timeit arr * 2.4
10 loops, best of 3: 31.7 ms per loop
# above result shows 31.7 ms (not seconds)

timeit scalar_multiple(listarr, 2.4)
1 loops, best of 3: 1.39 s per loop
# above result shows 1.39 seconds (not ms)
```

在上面的代码中，每个数组的元素占4个字节。因此，100万个整数数组大约占44MB内存，而列表占711MB内存。然而，对于小量级的数据，数据的运算速度较慢，而对大量级的数据，数组占较少的内存空间，且明显比数组运算速度更快。

NumPy中包含众多有用的函数，这些函数可以大致归类为：三角函数、算术函数、指数和对数函数，以及其他各种函数。在这些函数中，比较受欢迎的有：用于计算线性卷积的convolve()函数和用于线性插值的interp()函数。此外，在大部分包含等距数据的研究工作中，研究员广泛使用linspace()和random.rand()函数，其他类似的函数并不多。

2. 定形和重塑操作

通常情况下，改变现有数组的形状要比从旧数据创建一个新数组形状更高效。在第一个例子中，重塑发生在内存中（数组还没有被存储在一个变量中）。而在下面的代码中，数组先存储在变量a中，而后再对a重塑：

```
import numpy as np

np.dandom.rand(2,4)
array([[ 0.96432148,  0.63192759,  0.12976726,  0.56131001],
    [    0.27086909,  0.92865208,  0.27762891,  0.40429701]])

np.random.rand(8).reshape(2,4)
array([[ 0.39698544,  0.88843637,  0.66260474,  0.61106802],
       [ 0.97622822,  0.47652548,  0.56163488,  0.43602828]])
```

在上面的例子中，先随机生成8个数值，然后将他们重塑到一个有效的选择维度中。代码如下所示：

```
#another example
a = np.array([[11,12,13,14,15,16],[17,18,19,20,21,22]])

print a
[[11, 12, 13, 14, 15, 16], [17, 18, 19, 20, 21, 22]]

# the following shows shape is used to know the dimensions
a.shape
(2,6)

#Now change the shape of the array
a.shape=(3,4)
print a
[[11 12 13]  [14 15 16]  [17 18 19]  [20 21 22]]
```

在编程过程中，常用xrange替代range，因为xrange在进行循环时比range更快，并且能够避免存储整数列表；它只进行逐一生成。定形和重塑的反向操作是ravel()，代码如下所示：

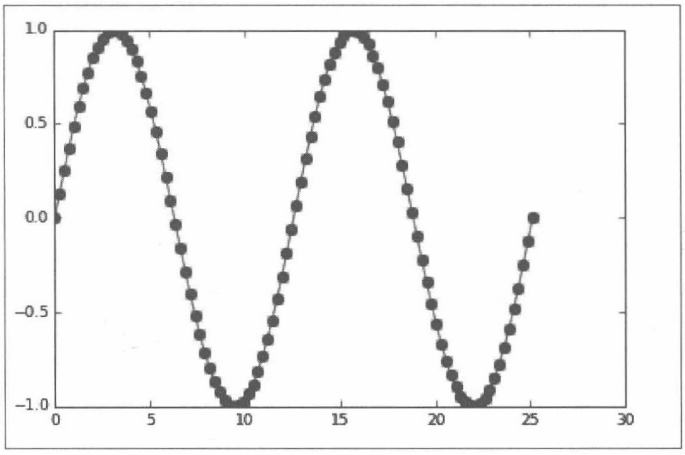
```
#ravel example
a = np.array([[11,12,13,14,15,16],[17,18,19,20,21,22]])

a.ravel()
array([11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22])
```

3. 一个插值的例子

下面是一个用interp()插值的例子：

```
n=30

# create n values of x from 0 to 2*pi
x = np.linspace(0,2*np.pi,n)

y = np.zeros(n)

#for range of x values, evaluate y values
for i in xrange(n):
   y[i] = np.sin(x[i])
```

下图展示了一个简单的正弦曲线插值结果：



下面的代码能绘制出插值前后的曲线：

```
import numpy as np
import matplotlib.pyplot as plt

# create n values of x from 0 to 2*pi
x = np.linspace(0, 8*np.pi, 100)

y = np.sin(x/2)

#interpolate new y-values
yinterp = np.interp(x, x, y)
#plot x,y values using circle marker (line style)
plt.plot(x, y, 'o')

#plot interpolated curve using dash x marker
plt.plot(xvals, yinterp, '-x')

plt.show()
```

4. 向量化函数

通过 NumPy 和 SciPy 中的 vectorize() 能够创建向量化函数，从而使函数变得非常有效。向量化通过对每一个元素执行同样的运算，能将一个输入参数为标量的函数转变为一个输入参数为数组的函数。

在第一个例子中，利用一个以 3 个标量为输入参数的函数生成一个以 3 个数组为输入参数的向量化函数，具体代码如下所示：

```
import numpy as np

def addition(x, y, z):
    return x + y + z
```

```
def addpoly():
    i = np.random.randint(25)
    poly1 = np.arange(i, i+10)
    i = np.random.randint(25)
    poly2 = np.arange(i, i+10)
    poly3 = np.arange(10, 18)
    print poly1
    print poly2
    print poly3
    print '-' * 32
    vecf = np.vectorize(addition)
    print vecf(poly1,poly2,poly3)

addpoly()

[ 4  5  6  7  8  9 10 11 12 13]
[13 14 15 16 17 18 19 20 21 22]
[10 11 12 13 14 15 16 17 18 19]
--------------------------------
[27 30 33 36 39 42 45 48 51 54]
```

其中，arrange 是一个数组版本的 Python 内建 range 函数。

在第二个例子中，将一个有 1 个标量输入参数的函数变为一个有 1 个数组输入参数的向量化函数，具体代码如下所示：

```
import numpy as np

def posquare(x):
  if x >= 0: return x**2
  else: return -x

i = np.random.randint(25)
poly1 = np.arange(i,i+10)

print poly1
vecfunc = vectorize(posquare, otypes=[float])
vecfunc(poly1)

[14 15 16 17 18 19 20 21 22 23]
array([ 196., 225., 256., 289., 324., 361., 400., 441., 484., 529.])
```

还有另外一个有趣的例子，让我们在示例代码的帮助下进行学习。这个例子展示了三种增加数组元素的方法，并通过测量运行时间来考察哪种方法更加快速：

```
import numpy as np
from time import time

def incrembyone(x):
    return x + 1
```

```
dataarray=np.linspace(1,5,1000000)

t1=time()
lendata = len(dataarray)
print "Len = "+str(lendata)
print dataarray[1:7]
for i in range(lendata):
    dataarray[i]+=1
print " time for loop (No vectorization)->" + str(time() - t1)

t2=time()

vecincr = np.vectorize(incrembyone) #1
vecincr(dataarray) #2
print " time for vectorized version-1:" + str(time() - t2)
t3 = time()
# This way to increment array elements with one line
# is pretty powerful, accomplishes same thing as #1 and #2
dataarray+=1  # how does this achieve the results
print dataarray[1:7]
print " time for vectorized version-2:" + str(time() - t3)

Len = 1000000
 [ 1.000004 1.000008 1.000012 1.000016 1.00002 1.000024]
time for loop (No vectorization)->0.473765850067
time for vectorized version-1:0.221153974533 # half the time

[ 3.000004 3.000008 3.000012 3.000016 3.00002 3.000024]
time for vectorized version-2:0.00192213058472 # in fraction time
```

除向量化技术外，还有另外一个提倡程序效率的编程技巧。如果在循环结构中需要调用有前缀符号的函数（如下面例子当中的 math.sin），最好对它新建一个本地别名（如 fastsin），并在循环中使用该本地别名，如下所示：

```
fastsin = math.sin

x = range(1000000)
for i in x:
    x[i] = fastsin(x[i])
```

5. NumPy 线性代数的总结

下面的列表总结了 NumPy 提供的一些著名的线性代数函数。

名　称	描　述
dot(a,b)	两个数组的点乘
linalg.norm(x)	矩阵或向量的模

（续）

名　称	描　述
linalg.cond(x)	指定条件数
linalg.solve(A,b)	解线性方程组 Ax=b
linalg.inv(A)	A 的逆
linalg.pinv(A)	A 的一个伪逆
linalg.eig(A)	A 平方的特征根 / 特征向量
linalg.eigvals(A)	A 的特征根
linalg.svd(A)	奇异值分解

4.1.2　SciPy

NumPy 已经有很多便于计算的函数。那么，我们为什么还需要 SciPy 呢？ SciPy 是 NumPy 的一个扩展，它包含更多数学、科学以及工程学相关的软件包，能够解决线性代数、积分、插值、快速傅立叶变换、大型矩阵操作、统计计算等众多问题。下面的表格是对 SciPy 软件包的一个简要介绍。

子软件包	功能简介
scipy.cluster	包含用于聚类分析的函数，如向量量化和 k- 均值
scipy.fftpack	表示快速傅立叶变换的函数
scipy.integrate	包含用于数值积分的函数，利用了 trapezoidal、Simpson、Romberg 等方法。同时包含常微分方程的积分方法。利用 quad、dblquad 和 tplquad 函数能够分别实现对一个函数对象进行一重、二重和三重积分
scipy.interpolate	包含用于对具有不连续的数值数据的对象进行差值以进行线性和样条插值的函数和类
scipy.linalg	对 NumPy 中的 linalg 包的一个封装。NumPy 中的所有函数都是 scipy.linalg 的一部分
scipy.optimize	包含用于最大化或最小化函数的方法，包括 Neider-Mead Simplex、Powell's、共轭梯度 BFGS 算法、最小二乘法、有约束的优化、模拟退火法、牛顿法、二分法、Broyden Anderson 和一维搜索等
scipy.sparse	包含用于处理大型稀疏矩阵的函数
scipy.special	包含用于计算物理学的特殊函数，如 elliptic、bessel、gamma、beta、hypergeometric、parabolic、cylinder、mathieu 和 spheroidal wave

除上面列举的子软件包以外，SciPy 还有 scipy.io 软件包。该软件包包含读取矩阵的函数 spio.loadmat()，保存矩阵的函数 spio.savemat()，以及读取图片的函数 scio.imread() 等。如果要使用 Python 编写计算程序，那么最好先看看 SciPy 的说明文档，SciPy 中可能已经存在完成任务的函数。

下面我们来看一个运用 scipy.polyId() 的例子：

```
import scipy as sp

# function that multiplies two polynomials
def multiplyPoly():
    #cubic1 has coefficients 3, 4, 5 and 5
    cubic1 = sp.poly1d([3, 4, 5, 5])

    #cubic2 has coefficients 4, 1, -3 and 3
    cubic2 = sp.poly1d([4, 1, -3, 3])

    print cubic1
    print cubic2

    print '-' * 36

    #print results of polynomial multiplication
    print cubic1 * cubic2

multiplyPoly()  # produces the following result

   3     2
3 x + 4 x + 5 x + 5
   3     2
4 x + 1 x - 3 x + 3
------------------------------------
    6      5      4      3      2
12 x + 19 x + 15 x + 22 x + 2 x + 15
```

函数输出的结果与利用传统的逐项相乘的方法（见下面的代码）得到的结果一致：

$$(3x^3 + 4x^2 + 5x + 5)(4x^3 + x^2 - 3x + 3)$$
$$= (12x^6 + 9x^5 + 15x^4 + 22x^3 + 2x^2 + 15)$$

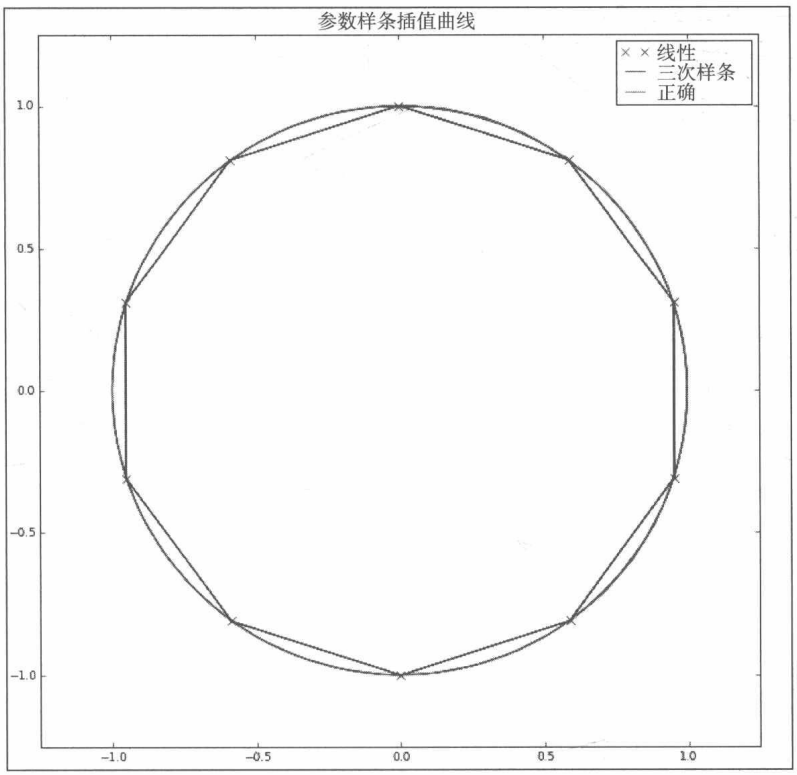
同样，多项式表示还可以用于积分、求导以及其他的计算物理学问题。这些应用与NumPy、SciPy和其他扩展包中的很多函数清楚地展现了Python对MATLAB的替代作用，因此在一些学术研究中可以使用Python进行编程。

SciPy中提供了很多种不同的插值方法。下面的例子使用了interpolate.splev函数，这是一种利用B样条和它的导数进行插值的方法，以及interpolate.splprep函数，它可以展示二维（更一般的情况为N维）曲线的B样条：

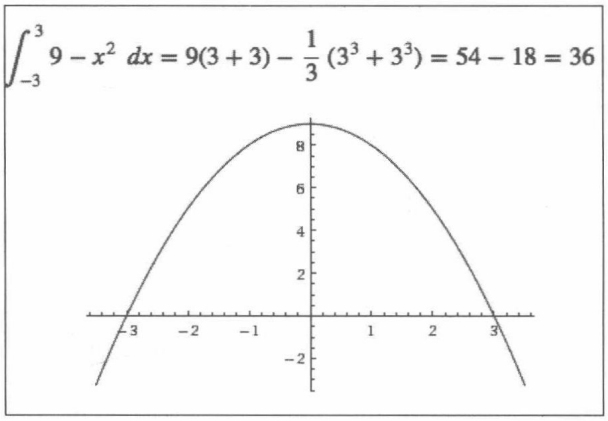
```
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
t = np.arange(0, 2.5, .1)
x = np.sin(2*np.pi*t)
y = np.cos(2*np.pi*t)

tcktuples,uarray = sp.interpolate.splprep([x,y], s=0)
unew = np.arange(0, 1.01, 0.01)
```

```
splinevalues = sp.interpolate.splev(unew, tcktuples)

plt.figure(figsize=(10,10))
plt.plot(x, y, 'x', splinevalues[0], splinevalues[1],
np.sin(2*np.pi*unew), np.cos(2*np.pi*unew), x, y, 'b')

plt.legend(['Linear', 'Cubic Spline', 'True'])
plt.axis([-1.25, 1.25, -1.25, 1.25])
plt.title('Parametric Spline Interpolation Curve')

plt.show()
```

下图展现了利用 SciPy 和 NumPy 进行样条插值的结果：



让我们看一个数值积分的例子。我们用 SciPy 中的一些函数（如 Simpson's 和 Romberg）来解决线性方程，并将其与 NumPy 中的 trapezoidal 函数进行比较。我们知道函数 $f(x) = 9 - x^2$ 从 –3 积分到 3 的结果为 36，如下图所示：



上图展示了函数 $f(x) = 9 - x^2$ 的图，它关于 Y 轴对称，因此在数学上，从 –3 到 3 上的积分等于从 0 到 3 上积分的两倍。我们如何利用 SciPy 进行数值积分？下面的代码展示了使用 SciPy 和 NumPy 中的 trapezoidal 方法进行积分的过程。

```
import numpy as np
from scipy.integrate import simps, romberg

a = -3.0; b = 3.0;
N = 10

x = np.linspace(a, b, N)
y = 9-x*x
yromb = lambda x: (9-x*x)

t = np.trapz(y, x)
s = simps(y, x)
r = romberg(yromb, a, b)

#actual integral value
aiv = (9*b-(b*b*b)/3.0) - (9*a-(a*a*a)/3.0)

print 'trapezoidal = {0} ({1:%} error)'.format(t, (t - aiv)/aiv)
print 'simpsons = {0} ({1:%} error)'.format(s, (s - aiv)/aiv)
print 'romberg  = {0} ({1:%} error)'.format(r, (r - aiv)/aiv)
print 'actual value = {0}'.format(aiv)

trapezoidal = 35.5555555556 (-1.234568% error)
simpsons = 35.950617284 (-0.137174% error)
romberg  = 36.0 (0.000000% error)
actual value = 36.0
```

1. 一个线性方程组的例子

下面让我们来尝试求解一组含有三个变量 (x, y, z) 的线性方程组：

- $x + 2y - z = 2$
- $2x - 3y + 2z = 2$
- $3x + y - z = 2$

NumPy 提供了一种方便求解线性方程组的方法 np.linalg.solve()。然而，该函数输入时需要写成特定的向量形式。下面的代码展示了 NumPy 如何求解线性方程组：

```
import numpy as np

# Matrix A has coefficients of x,y and z
A = np.array([[1, 2, -1],
              [2, -3, 2],
              [3, 1, -1]])
#constant vector
b = np.array([2, 2, 2])

#Solve these equations by calling linalg.solve
v = np.linalg.solve(A, b)

# v is the vector that has solutions
print "The solution vector is "
print v
# Reconstruct Av to see if it produces identical values
print np.dot(A,v) == b

The solution vector is
[ 1.  2.  3.]
[ True  True  True]
```

注意，np.dot(A，v) 是矩阵乘法（而不是 A*v）。程序得到的解 v = [1，2，3] 结果正确。

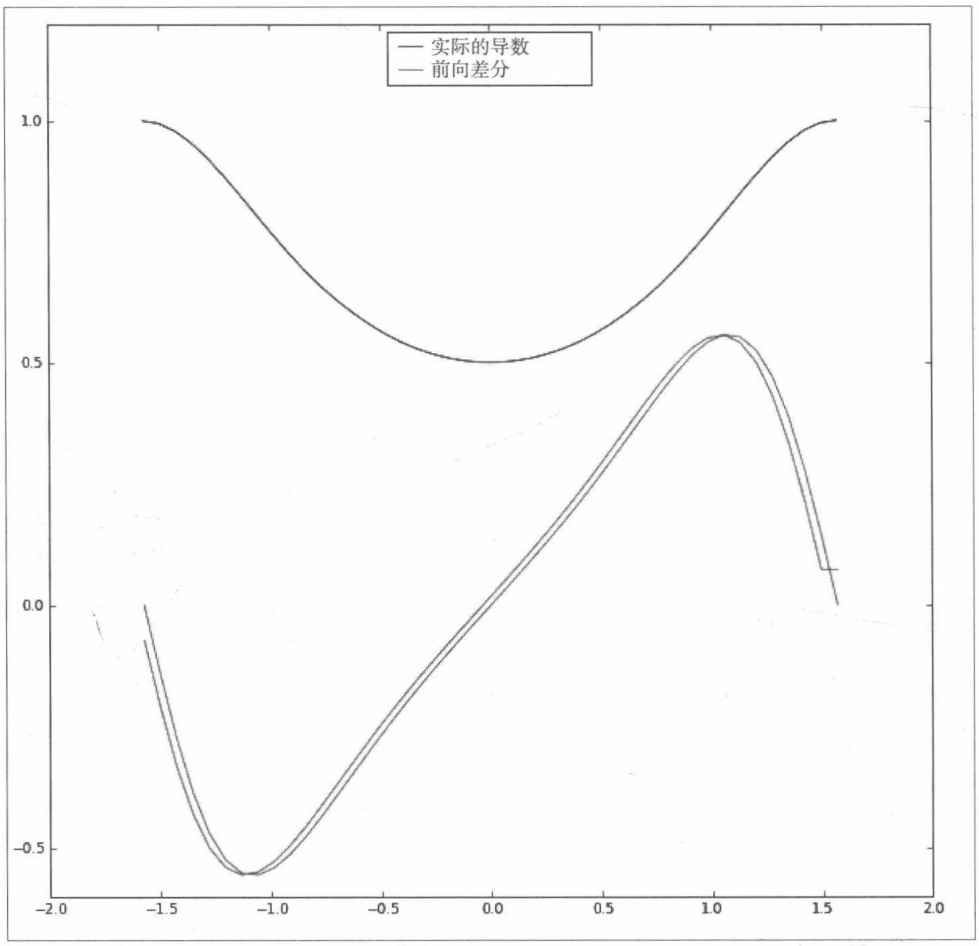
2. 向量化数值求导

下面是本部分的最后一个例子。我们将考察 NumPy 提供的向量化数值求导方法。通常情况下，我们通过应用微分商法则来进行求导 $\frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{1}{1+\cos^2(x)}\right)=\frac{\sin 2x}{(1+\cos^2(x))^2}$。但是，使用 Python 中的向量化方法能够在不使用循环的情况下进行求导，代码如下：

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-np.pi/2, np.pi/2, 44)
y = 1/(1+np.cos(x)*np.cos(x))
dy_actual = np.sin(2*x)/(1+np.cos(x)*np.cos(x))**2

fig = plt.figure(figsize=(10,10))
ax=fig.add_subplot(111,axisbg='white')
```

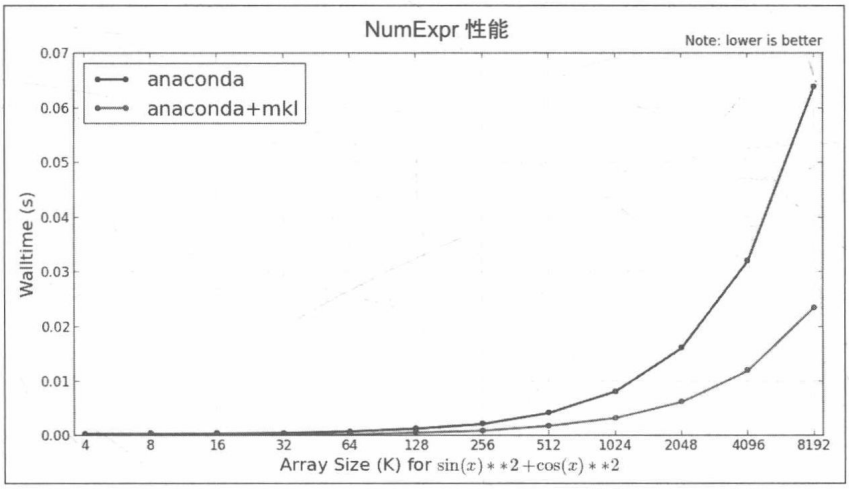
```
# we need to specify the size of dy ahead because diff returns
dy = np.zeros(y.shape, np.float) #we know it will be this size
dy[0:-1] = np.diff(y) / np.diff(x)
dy[-1] = (y[-1] - y[-2]) / (x[-1] - x[-2])

plt.plot(x,y, linewidth=3, color='b', label='actual function')
plt.plot(x,dy_actual,label='actual derivative', linewidth=2,
color='r')
plt.plot(x,dy,label='forward diff', linewidth=2, color='g')
plt.legend(loc='upper center')
plt.show()
```

在下图中，我们将看到如何将原始方程、它的导数，以及前向差分同时绘制到一张图上。实际的导数被整合在 dy_actual 中，而前向差分可以用 NumPy 中的 diff() 函数进行计算。下图是程序运行的结果：



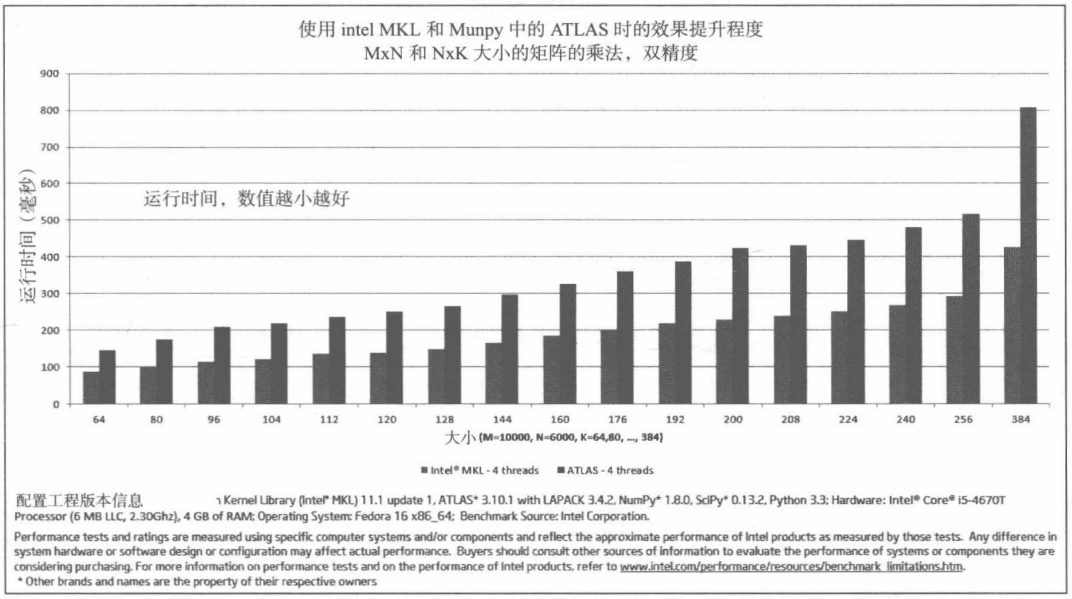
4.1.3　MKL 函数

Intel 的 MKL 函数提供了向量和矩阵的高性能运算方法。它们涵盖了 FFT 函数和向量统计函数。这些函数已得到加强和优化，从而能在 Intel 处理器中快速高效得运行。对于 Anaconda 用户，Continuum 已经把 FFT 函数打包到二进制版本的 Python 库，用来优化 MKL。然而 MKL 优化可以通过额外安装 Anaconda Accelerate 包来实现。下图比较了使用 MKL 前后的处理速度。



此图来自于网站：https://store.continuum.io/cshop/mkl-optimizations/

输入大型数组，MKL 能够在处理效果上得到显著提升，如下图所示：



此图来自于网站：https://software.intel.com/en-us/ articles/numpyscipy-with-intel-mkl

4.1.4 Python 的性能

Python 程序员经常为了提高处理效能而尝试用 C 来重写 Python 内部的循环，或者调用 Python 中用 C 编写的函数。有很多的项目（如 Cython）都旨在简化这种优化过程。然而，更好的方式是在不依赖其他编程语言的情况下提高现有的 Python 代码执行速度。

在 Python 中，还有其他方法能够提高大运算量程序处理性能：

- **使用 Numbapro**：这是一个在 Continuum Analytics 中的 Python 编译器，能够编写基于 CUDA 的 GPU 执行或多核 CPU 执行的 Python 代码。这种代码能够运行本地编译码，且比解释代码的速度快好几倍。Numbapro 能够在运行时启用编译（just-in-time，JIT 编译）。通过 Numbapro 能够实现编写标准的 Python 函数，并将它们放在基于 CUDA 的 CPU 中运行。和广泛使用的 NumPy 库一样，Numbapro 的设计基于数组导向的任务处理方式。Numbapro 是 Numba 的一个强化版本，并且是 Anaconda Accelerate（Continuum Analytics 的一个商业产品）的一部分。
- **使用 Scipy.weave**：这个模块可以在 Python 中插入一些 C 代码的片段，并将 NumPy 中的数组无缝的转换为 C 中的层。它还具有很多十分有效的宏。
- **使用多核方法**：在 Python 2.6 及更高版本中，包含了用于多进程处理的包，它提供了一种相对简单的建立子进程的机制。现如今连台式电脑都具有多核处理器，因此将所有的处理器都用于工作已经变得非常的有意义。这种方式要比使用多线程更加简单。
- **使用进程池 pool**：这是多进程处理包中的另一个类。利用进程池，你可以定义在进程池中运行的进程的数量，并对每个程序传入一个包含参数的可迭代对象。
- **在一个分布式计算包下使用 Python（如 Disco）**：Disco 是一个轻量级的开源框架，它基于 MapReduce 范式实现分布式计算（http://discoproject.org）。其他类似的包有 Hadoop Streaming、mrjob、dumbo、hadoopy 和 pydoop。

4.2 标量选择

标量选择是从一个数组中选取元素的最简单的方法。对于一维数组，它使用 [rowindey] 的方法选择，对于二维数组，它使用 [rowindex, cdumnindex] 的方法进行选择，依此类推。下面是一个展示数组元素引用的例子：

```
import numpy as np
x = np.array([[2.0,4,5,6], [1,3,5,9]])

x[1,2]
5.0
```

纯粹的标量选择通常只返回单个元素，而不是一个数组。选出元素的数据类型与数组中元素的数据类型相同。标量选择也可用于向数组中的某个位置的元素进行赋值，代码如下所示：

```
x[1,2] = 8

x
array([[2, 4, 5, 6],[1, 3, 8, 9]])
```

4.3　切片

和列表、元组一样，数组也可以进行切片操作。数组切片的方法除语法更加简洁外，与列表切片的方法完全一致。数组切片采用 [: , :, ... :] 的语法形式，其中数组的维数决定了切片的数量。如果有一些维度上的切片被省略了，那么该维度上的所有元素都会被选择。例如 b 是一个三维数组，b[0:2] 与 b[0:2,:,:] 表示的含义完全相同。在使用切片的过程中，有一些快速的简写法，常见的简写法如下：

- :（单独一个冒号）：相当于 0:*n*:1，其中 *n* 为数组长度
- m：和 m:n ：相当于 m:*n*:1，其中 *n* 为数组长度
- :n：相当于 0:*n*:1
- ::d：相当于 0:*n*:*d*，其中 *n* 为数组长度

以上所有切片方法都可以在数组和列表中使用。一维数组的切片等同于一个列表的切片（正如一个一维数组可以被等价的看做一个列表一样）。所有切片操作返回的数据类型都与被切片的数组中元素的数据类型一致。下面的例子展现了数组切片的功能：

```
x = array([5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20])

# interpret like this - default start but end index is 2
y = x[:2]
array([5, 6])

# interpretation - default start and end, but steps of 2
y = x[::2]
array([5,7,9,11,13,15,17,19])
```

如果将同一数据类型中的元素插入到不同数据类型的数组中，那么 NumPy 会尝试自动转换该元素的数据类型。例如数组中的数据是整数型，那么在数组中新放入一个浮点型数据，浮点数据会被删除浮点部分，并存储为整数型。这种情况潜藏着极大的风险。因此在大部分情况下，除非有较好的理由表明这里需要使用整型数据类型，数组常被初始化为存储浮点类型的数据。下面的例子将说明，即使在输入元素中只有一个元素为浮点型，而其

他元素均为整数型，数组的类型也会被定义为浮点型以保证数组能够正确的工作。

```
a = [1.0, 2,3,6,7]
b = array(a)

b.dtype
dtype('float64')
```

利用 flat 进行切片

矩阵中的数据是以行编号为主次序进行存放的，即矩阵中的元素在索引过程中先考察行，然后再考察列。例如在下面的一个 3 行 3 列的矩阵中，元素的读取顺序为 4，5，6，7，8，9，1，2，3（对于每一行，依次选取每个列上的元素）。

$$A = \begin{bmatrix} 4 & 5 & 6 \\ 7 & 8 & 9 \\ 1 & 2 & 3 \end{bmatrix}$$

线性切片对数组中的每一个元素都分配一个索引，并通过该索引顺序来读取元素，形成一个一维数组。在二维数组或数列中，线性切片在工作中先按行从小到大开始访问，再按列从小到大开始访问。为了使用线性切片，我们需要使用 flat 函数，其代码如下：

```
a=array([[4,5,6],[7,8,9],[1,2,3]])
b = a.flat[:]

print b
[4, 5, 6, 7, 8, 9, 1, 2, 3]
```

4.4 数组索引

选取 NumPy 数组中的元素有 4 种方法：标量选择、切片、数值索引以及逻辑索引（布尔索引）。前面讨论的标量选择和切片是选取数组中元素最常用的方法。数值索引和逻辑索引具有密切的联系，它们能够更加灵活的选取元素。数值索引利用数列或数组的位置来选取元素，而逻辑索引通过使用带有布尔类型数据的数组来选取元素。

4.4.1 数值索引

数值索引是切片的一种替代方法。数值索引的想法是利用坐标来选择元素，这与切片很相似。利用数值索引得到的数组是对原始数据的复制，而切片索引只查看原始数据，但并没有复制原始数据。基于性能的考虑，我们更应该使用切片方法。切片与一维数组十分类似，但切片的形状是由切片的输入来决定的。

一维数组中的数值索引利用数组中元素的索引数字作为位置坐标，并返回与索引数组

维度一致的数组。索引数组可以是一个列表或者是包含整型数据的 NumPy 数组，代码如下：

```
a = 10 * arange(4.0)
array([0.,10.,20.,30.])

a[[1]]  # arrays index is list with first element
array([ 10.])

a[[0,3,2]] # arrays index are 0-th, 3-rd and 2-nd
array([  0.,  30.,  20.])

sel = array([3,1,4,2,3,3])  # array with repetition
a[sel]
array([ 30.  10.   0.  20.  30.  30.])

sel = array([4,1],[3,2]])
a[sel]
array([[ 30.,10.], [ 0.,20.]])
```

这些例子均展示出数值索引决定了选取的元素的位置，而索引数组的形状决定了输出的形状。

类似于切片，数值索引也可以与 flat 函数结合选取数组中的元素，并把他们按照以行为主次序的方式排列为新的数组。用 flat 的这种数值索引行为与在底层数组的平铺版本上用数值索引一致。

```
a = 10 * arange(8.0)
array([  0.,  10.,  20.,  30.,  40., 50., 60., 70.])

a.flat[[3,4,1]]
array([ 30., 40., 10.])

a.flat[[[3,4,7],[1,5,3]]]
array([[ 30., 40., 70.], [ 10., 50., 30.]])
```

4.4.2　逻辑索引

逻辑索引与切片和数值索引不同，它利用逻辑指标来选择元素、行或列。逻辑索引标如同台灯的开关一样，不是 true 就是 false。纯逻辑索引利用一个与数据数组大小形状一样的逻辑索引数组来进行数据选取，同时返回一个一维数组，其代码如下：

```
x = arange(-4,5)

x < 0
array([True, True, True, True, False, False, False, False, False],
dtype=bool)

x[x>0]
array([1, 2, 3, 4])
```

```
x[abs(x) >= 2]
array([-4, -3, -2,  2,  3,  4])

#Even for 2-dimension it still does the same
x = reshape(arange(-8, 8), (4,4))
x
array([[-8, -7, -6, -5], [-4, -3, -2, -1], [ 0,  1,  2,  3], [ 4,  5,
6,  7]])

x[x<0]
array([-8, -7, -6, -5, -4, -3, -2, -1])
```

下面是另外一个展现逻辑索引的例子：

```
from math import isnan
a = [[3, 4, float('NaN')], [5, 9, 8], [3, 3, 2], [9, -1,
float('NaN')]]

list2 = [3, 4, 5, 6]
list1_valid = [elem for elem in list1 if not any([isnan(element) for
element in elem])]

list1_valid
[[3, 7, 8], [1, 1, 1]]

list2_valid = [list2[index] for index, elem in enumerate(list1) if not
any([isnan(element) for element in elem])]

list2_valid
 [4, 5]
```

4.5 其他数据结构

Python 有多种数据结构，如栈、列表、集合、队列、元组、堆、数组、字典和双队列。我们已经讨论过列表，并且尝试理解数组的含义。而元组是一个典型的比列表记忆性能更高的数据类型，因为元组是不可变的。

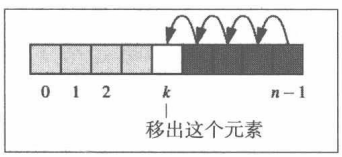
4.5.1 栈

栈是一种抽象的数据类型，栈中的元素采用后进先出的执行方法。利用以下函数，我们可以很方便地将一个列表作为栈进行使用。利用 append() 能实现在列表的尾部添加一个元素，而利用 pop() 能够实现从列表的尾部提取一个元素，从而实现栈的后进先出的功能。同时利用 remove(item-value) 还可以删除栈中特定的对象，代码如下：

```
stack = [5, 6, 8]
stack.append(6)
stack.append(8)
```

```
stack
[5, 6, 8, 6, 8]

stack.remove(8)
stack
[5, 6, 6, 8]

stack.pop()
8

stack.remove(8)
Traceback (most recent call last):
File "<ipython-input-339-61d6322e3bb8>", line 1, in <module>
stack.remove(8)
  ValueError: list.remove(x): x not in list
```

pop() 函数运算起来非常有效（耗费时间为常数），因为 pop 函数在执行过程中，除最后一个元素以外，其他所有元素都保持它们原本的位置。然而，带参数版本的函数 pop(k) 就没有那么有效，它能移出列表中索引数值 $k<n$ 的元素，并将移动所有指定位置后面的元素来填补移出该元素造成的空缺。因此，pop(k) 的效率是线性的，因为移动元素的数量取决于索引 k，从下图可以证明这一点：



4.5.2 元组

元组是一个看起来与列表相似的不可变的对象列。元组是一个异质的数据结构，这意味着它们的元素可以具有不同的含义，而列表是一个同质的元素列。元组具有结构，而数列有顺序。元组的一些例子有：一周中的天数，课程名称和分数级别。代码如下：

```
#days of the week
weekdays = ("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday")

#course names
courses = ("Chemistry", "Physics", "Mathematics", "Digital Logic",
"Circuit Theory")

#grades
grades = ("A+", "A", "B+", "B", "C+", "C", "I")
```

元组具有不可改变的对象，你不可以更改或者删除元组中的对象。但元组可以整体删除，如利用 "del grades" 语句将会删除 gardes 元组。在删除元组之后，如果还尝试去使用这

个元组，程序就会报错。下面列举了一些元组的内建函数：

- cmp(tup1, tup2): 比较两个元组中的元素
- len(tuple): 获取元组的长度
- max(tuple): 获取元组中的最大值
- min(tuple): 获取元组中的最小值
- tuple(lista): 将一个列表转换为一个元组

针对数值型数据，Python 中的 max() 函数会返回它们的最大值，而针对一列字符型数据，max() 会返回长度最长的元素。

```
weekdays = ("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday")
print max(weekdays)
Wednesday
```

min() 函数在针对一组字符型数据时具有相似的结果，会返回长度最短的元素。

```
print min(weekdays)
Friday
```

我们经常需要获取元组或列表中有多少个元素，这个时候利用 len() 函数能轻松实现。

```
len(weekdays)
7
```

4.5.3 集合

集合与列表类似，但主要存在两方面的区别：第一，集合中的元素是无序的，而列表中的元素是有序的（依据位置或索引进行排序）；第二，集合中没有重复的元素（类似于数学定义中的集合），而列表中可以存在重复的元素。下面的程序展现了集合的表示方法：

```
setoftrees = { 'Basswood', 'Red Pine', 'Chestnut', 'Gray Birch',
'Black Cherry'}

newtree = 'Tulip Tree'
if newtree not in setoftrees:  setoftrees.add(newtree)
```

用上述命令，你可以看到 setoftrees 上的内容：

```
setoftrees  # typing this shows list of elements shown below
{'Basswood', 'Black Cherry', 'Chestnut', 'Gray Birch', 'Red Pine',
'Tulip Tree'}
```

下面，我们通过适当的程序创建 charsinmath 和 charsinchem 集合，代码如下：

```
#example of set of operation on letters
charsinmath = set('mathematics')
charsinchem = set('chem')
```

现在，观察一下这些集合中的元素：

```
Charsinmath # typing this shows letters in charsinmath
{'a', 'c', 'e', 'h', 'i', 'm', 's', 't'}

charsinchem # typing this shows letters in charsinchem
{'c', 'e', 'h', 'm'}
```

为了考察两个集合的区别，我们需要展示 charsinmath – charsinchem，如下所示：

```
# take away letters from charsinchem from charsinmath
charsinmath - charsinchem
{'a', 'i', 's', 't'}
```

4.5.4　队列

同栈一样，列表也可以当作一个队列使用。然而，队列与栈的区别主要在于队列从列表的一侧（如头部）添加数据，而从另一侧（如尾部）移出数据。尽管从列表的尾部添加和删除元素的效率很高，但是从列表的头部添加或删除元素的效率却很低，因为需要移动列表中的所有元素。

为了解决这个问题，Python 在集合软件包中提供了 deque 类型，该类型能够有效地利用 append()、pop()、appendleft() 和 popleft() 函数，快速地实现向列表的尾部添加元素，并从列表的任意一端删除元素，代码如下：

```
from collections import deque

queue = deque(["Daniel", "Sid", "Mathew",  "Michael"])
queue.append("Dirk")       # Dirk arrives
queue.append("Monte")    # Monte arrives queue

queue
deque(['Daniel', 'Sid', 'Mathew', 'Michael', 'Dirk', 'Monte'])

queue.popleft()
'Daniel'

queue.pop()
'Monte'

queue.appendleft('William')
queue
deque(['William', 'Sid', 'Mathew', 'Michael', 'Dirk'])
```

```
queue.append('Lastone')
queue
deque(['William', 'Sid', 'Mathew', 'Michael', 'Dirk', 'Lastone'])
```

4.5.5 字典

字典是一个无序的数据集，每个数据都由一个键－值对组成，每一个值有唯一一个索引键进行标识。如果 key 是字符型或其他非整型类型的数据，索引是否能够进行工作，该如何进行工作？答案是字典利用一个哈希函数来实现通过索引键查询值位置的过程，因此键必须是可哈希的数据类型：该哈希函数以输入键值，输出一个整数，之后字典利用这个数（或哈希值）来存储和提取值。下面是字典的几个例子：

```
#example 1: Top 10 GDP of Africa
gdp_dict = { 'South Africa': 285.4, 'Egypt': 188.4, 'Nigeria': 173,
'Algeria': 140.6, 'Morocco': 91.4, 'Angola': 75.5, 'Libya': 62.3,
'Tunisia': 39.6, 'Kenya': 29.4, 'Ethiopia': 28.5, 'Ghana': 26.2,
'Cameron': 22.2}

gdp_dict['Angola']
75.5

#example 2: English to Spanish for numbers one to ten
english2spanish = { 'one' : 'uno', 'two' : 'dos', 'three': 'tres',
'four': 'cuatro', 'five': 'cinvo', 'six': 'seis', 'seven': 'seite',
'eight': 'ocho', 'nine': 'nueve', 'ten': 'diez'}

english2spanish['four']
'cuatro'
```

为了得到预期的哈希数值，键值是不可改变的，否则，键值的改变会导致对应的哈希数值的改变，从而查询到一个与之前不同的位置，并获取一个不可预期的值。默认的字典类型并不具备保存值顺序插入到字典中的功能，因此在通过迭代插入大量键－值对之后，键－值对的排列顺序是任意的。

Python 的集合软件包中具有功能相同的 OrderedDict() 函数，它可以保持键－值对的插入顺序。默认字典与有序字典之间的另外一个区别就是：在默认字典中，只要两个字典具有同样的键－值对集合，那么两个字典是等价的，而在有序字典中，只有两个字典中的键－值对以及它们的排列顺序都完全一致，两个有序字典才是等价的，下面的例子证明了这点：

```
# using default dictionary
dict = {}

dict['cat-ds1'] = 'Introduction to Data Structures'
dict['cat-ds2'] = 'Advanced Data Structures'
dict['cat-la1'] = 'Python Programming'
```

```
dict['cat-la2'] = 'Advanced Python Programming'
dict['cat-pda'] = 'Python for Data Analysis'
dict['cat-ps1'] = 'Data Science in Python'
dict['cat-ps2'] = 'Doing Data Science'

for key, val in dict.items():  print key,val

cat-ps1 Data Science in Python
cat-ps2 Doing Data Science
cat-pda Python for Data Analysis
cat-la2 Advanced Python Programming
cat-la1 Python Programming
cat-ds1 Introduction to Data Structures
cat-ds2 Advanced Data Structures

#using OrderedDict (inserting data the same way as before)
odict = OrderedDict()

odict['cat-ds1'] = 'Introduction to Data Structures'
odict['cat-ds2'] = 'Advanced Data Structures'
odict['cat-la1'] = 'Python Programming'
odict['cat-la2'] = 'Advanced Python Programming'
odict['cat-pda'] = 'Python for Data Analysis'
odict['cat-ps1'] = 'Data Science in Python'
odict['cat-ps2'] = 'Doing Data Science'

for key, val in odict.items():  print key,val

cat-ds1 Introduction to Data Structures
cat-ds2 Advanced Data Structures
cat-la1 Python Programming
cat-la2 Advanced Python Programming
cat-pda Python for Data Analysis
cat-ps1 Data Science in Python
cat-ps2 Doing Data Science
```

想要实现和上面类似的功能，使用 ISBN 作为键比使用图书馆中的目录编号作为键在计算上效率更高。然而，可能有些旧书不具有 ISBN 号，因此就需要使用一个等价的唯一的键 – 值与其他是有 ISBN 号的新书保持一致。一个哈希数通常是一个数字，而相比包含文字与数字的键，一个纯数值型的键能更加方便地被哈希函数使用。

4.5.6　字典的矩阵表示

通常，当存在键 – 值关联时，我们可以应用字典，这样的案例有很多。例如国家的缩写和全名；它们中的任何一个都可以作为键，而另外一个作为值，但是用简写作为键无疑更有效率。其他案例有：单词和单词出现的次数，或城市名和人口。计算领域的一个有趣例子是，字典有效地用来表示一个稀疏矩阵。

1. 稀疏矩阵

首先，让我们考察如下矩阵的空间利用率：用列表表示一个 100 × 100 的矩阵，每个元素占 4 字节，因而存储整个矩阵需要 40 000 字节，大约 40KB 的空间。然而，在这 40 000 字节当中，有 100 个元素是非 0 的，而其他的元素都为 0，因此大部分的空间都浪费了。下面为了简化讨论，我们先考虑一个更小的矩阵，如下图所示：

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 2 \\ 0 & 4 & 0 & 3 & 0 & 0 & 0 & 0 & 1 & 0 \\ 6 & 0 & 1 & 0 & 0 & 7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 3 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

这个矩阵大概有 20% 的元素是非 0 的。因此，我们需要尝试找到一个更好的方法表示矩阵中的非 0 元素。这个矩阵中有 7 个 1，5 个 2，5 个 3，以及 4、6、7 各一个。利用字典可以用以下方法表示它们：

```
A = {1: [(2,2),(6,6), (0,7),(1,8),(7,8),(3,9),(8,9)],
  2: [(5,2),(8,2),(6,3),(0,4),(0,9)],
  3: [(5,0),(8,0),(9,1),(1,3),(5,8)],
  4:[(1,1)], 6:[(2,0)], 7:[(2,5)]}
```

然而，这种表示方法让我们更加难以获得矩阵 A 第 i 行第 j 列位置上的元素，因此一种利用字典更好地表示稀疏矩阵的方法如下：

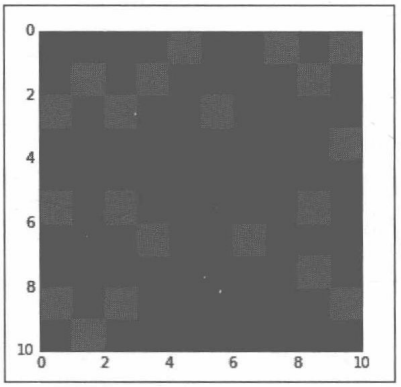
```
def getElement(row, col):
    if (row,col) in A.keys():
       r = A[row,col]
    else:
       r = 0
    return r

A={(0,4): 2, (0,7): 1, (1,1): 4, (1,3):3, (1,8): 1, (2,0): 6, (0,9):
2, (2,2):1, (2,5): 7, (3,9): 1, (5,0): 3, (5,2): 2, (5,8): 3, (6,3):
2, (6,6):1, (7,8): 1, (8,0): 3, (8,2): 2, (8,9): 1, (9,1): 3}
print getElement(1,3)
3

print getElement(1,2)
0
```

为了获得 A 矩阵中 (1, 3) 位置上的元素，我们使用 A[(1, 3)]，但是如果在 (1, 3) 这个位

置元素为 0，即 (1, 3) 这个键值不存在，那么就会抛出一个异常。为了在非 0 元素的时候利用键获取到值，并在键不存在的时候返回 0，我们需要使用 getElement() 函数，使用方法见上面的代码。

可视化稀疏性

利用 SquareBox 图表，我们可以从视觉上看到一个矩阵有多稀疏。下图展现了利用 sparseDisplay() 函数对矩阵进行绘图的结果。矩阵中的每一个元素均用一个带颜色的正方形表示，深色代表着元素为 0，而灰色代表着元素非 0。

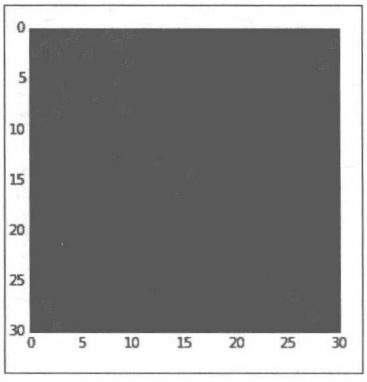


下面的代码展现了如何实现可视化稀疏性：

```
import numpy as np
import matplotlib.pyplot as plt

"""
  SquareBox diagrams are useful for visualizing values of a 2D array,
  Where black color representing sparse areas.
"""
def sparseDisplay(nonzero, squaresize, ax=None):
    ax = ax if ax is not None else plt.gca()

    ax.patch.set_facecolor('black')
    ax.set_aspect('equal', 'box')
    for row in range(0,squaresize):
      for col in range(0,squaresize):
        if (row,col) in nonzero.keys():
           el = nonzero[(row,col)]
           if el == 0:  color='black'
           else:  color = '#008000'
           rect = plt.Rectangle([col,row], 1, 1,
                   facecolor=color, edgecolor=color)
           ax.add_patch(rect)

    ax.autoscale_view()
    ax.invert_yaxis()

if __name__ == '__main__':
```

```
    nonzero={(0,4): 2, (0,7): 1, (1,1): 4, (1,3): 3, (1,8): 1,
(2,0): 6, (0,9): 2, (2,2): 1, (2,5): 7, (3,9): 1, (5,0): 3,
(5,2): 2, (5,8): 3, (6,3): 2, (6,6): 1, (7,8): 1, (8,0): 3, (8,2): 2,
(8,9): 1, (9,1): 3}

    plt.figure(figsize=(4,4))
    sparseDisplay(nonzero, 10)
    plt.show()
```

这只是一个展现稀疏矩阵的简单例子。假设你有一个 30 × 30 的矩阵，矩阵中只有少量的非 0 元素，那么它的可视化结果将会与下面的图像相类似。利用字典存储稀疏矩阵节约了 97% 的使用空间。换句话说，矩阵越大，矩阵的空间利用率就越低，如下图所示：



既然已经找到了一种使用字典来存储稀疏矩阵的方法，在以后面对稀疏矩阵存储的时候，就要记得使用，以免重蹈覆辙。同时，学会使用字典存储系数矩阵会让你更加深入认识到字典的强大。在这里，我们强烈推荐你去阅读 SciPy 和 pandas 软件包中关于稀疏矩阵的内容。本书其他例子还会接触到类似应用。

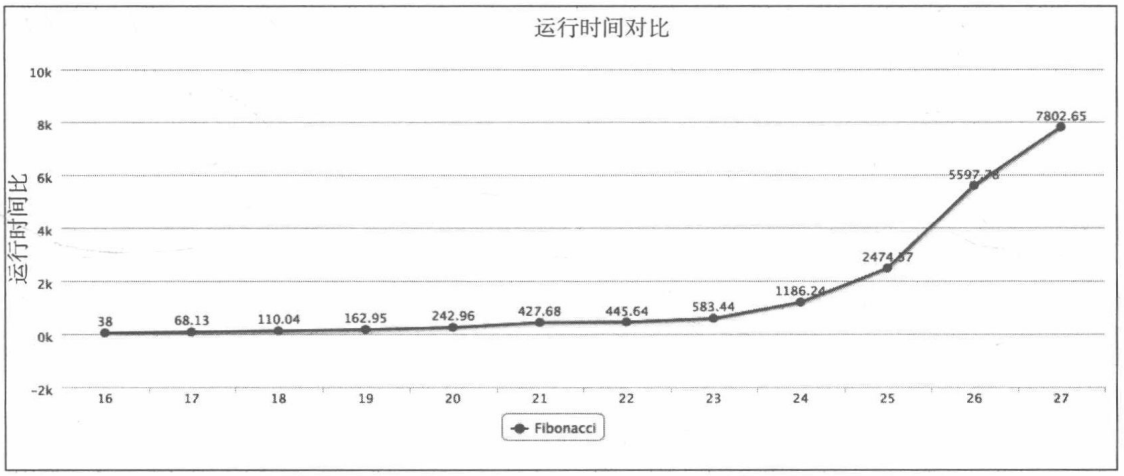
2. 利用字典进行备忘

备忘是计算科学中的一种优化技术，它存储中间结果，避免繁琐地重复计算。不是所有的问题都需要备忘，对于那些有需要利用函数反复计算相同的数值的场景，备忘就非常有用了。例如在计算 Fibonacci 函数的过程中使用字典来存储已经计算好的值，在下一次运算时，你仅需搜索这些值，而不是再计算一次，其代码如下：

```
fibvalues = {0: 0, 1: 1, 2:1, 3:2, 4:3, 5:5}

def fibonacci(n):
    if n not in fibvalues:
        sumvalue = fibonacci(n-1) + fibonacci(n-2)
        fibvalues[n] = sumvalue
    return fibvalues[n]
```

```
fibonacci(40)
102334155


print sorted(fibvalues.values())
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987,
1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393,
196418, 317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887,
9227465, 14930352, 24157817, 39088169, 63245986, 102334155]

#regular fibonacci without using dictionary
def fib(n):
    if n <= 1 : return 1
    sumval = fib(n-1)+fib(n-2)
    return sumval
```

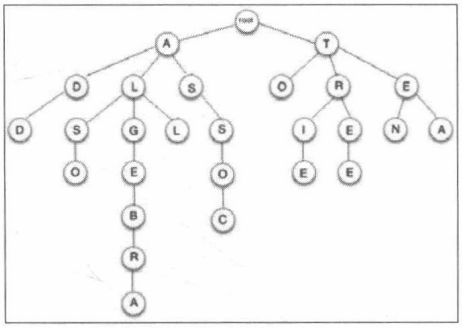
fibvalues 字典非常有用，它用来防止反复计算 Fibonacci 值。这里使用 fibcalled 是为了对比说明在使用字典时，对于一个特定的 n，只需要调用 fibonacci() 1 次。通过比较 fib()（不利用字典来存储结果）和 fibonacci() 的运行时间比，我们可以看到与下图类似的结果：




```
from time import time

for nval in range(16,27):
  fibvalues = {0: 0, 1: 1, 2:1, 3:2, 4:3, 5:5}
  t3 = time()
  fibonacci(nval)
  diftime1 = time()-t3
  t2 = time()
  fib(nval)
  diftime2 = time()-t2
  print "The ratio of time-2/time-1 :"+str(diftime2/diftime1)
```

4.5.7 Trie 树

Trie 树是一种数据结构，它具有不同的名称，如数字树、基数树或前缀树。Trie 树可以非常有效地用于搜索、插入和删除函数。Trie 树是一种非常优秀的存储结构。例如当单词 add、also、algebra、assoc、all、to、trie、tree、tea 和 ten 存储到 trie 中时，trie 树的图形结构如下所示：



在上面的图像中，用大写字母表示字符是为了更清楚地表达意图。在实际的存储过程中，单词会依据它的实际内容进行存储。Trie 树非常适合存储单词计数（word count）的结果。它的搜索功能非常有效，尤其是在模式不匹配的情况下，程序运行速度会更快。换言之，如果想要搜索 are，会因为 trie 树中不存在单词 r 而快速返回搜索失败。

Trie 树的另外一个非常著名的功能是最长前缀匹配。换句话说，如果我们尝试在字典中找到所有与特定搜索字符（如 base）具有最长相同前缀的所有单词。这些结果包括 base、baseline、basement 等其他字典中符合此条件的单词。

Python 具有众多不同 Trie 树的实现：suffix_tree、pytire、trie、datrie 等。J. F. Sebastian 在 https://github.com/zed/trie-benchmark 网页中展示了这些实现的详细对比。

大部分搜寻引擎都有一种名为反向索引的 Trie 树的实现，它的核心是空间优化。在这种存储结构下，能够快速有效地搜索关键词与文本之间的关系。由于 Trie 树能在存储大量数据的同时节省很多存储空间，因此它广泛应用于 IP 路由中。

下面的代码展现了 Python 中 Trie 树的一种实现（虽然不是必要的但是却是最有效的）：

```
_end = '_end_'

# to search if a word is in trie
def in_trie(trie, word):
    current_dict = trie
    for letter in word:
        if letter in current_dict:
            current_dict = current_dict[letter]
```

```
            else:
                return False
        else:
            if _end in current_dict:
                return True
            else:
                return False

#create trie stored with words
def create_trie(*words):
    root = dict()
    for word in words:
        current_dict = root
        for letter in word:
            current_dict = current_dict.setdefault(letter, {})
        current_dict = current_dict.setdefault(_end, _end)
    return root

def insert_word(trie, word):
    if in_trie(trie, word): return

    current_dict = trie
    for letter in word:
            current_dict = current_dict.setdefault(letter, {})
    current_dict = current_dict.setdefault(_end, _end)

def remove_word(trie, word):
    current_dict = trie
    for letter in word:
        current_dict = current_dict.get(letter, None)
        if current_dict is None:
            # the trie doesn't contain this word.
            break
    else:
        del current_dict[_end]

dict = create_trie('foo', 'bar', 'baz', 'barz', 'bar')
print dict
print in_trie(dict, 'bar')
print in_trie(dict, 'bars')
insert_word(dict, 'bars')
print dict
print in_trie(dict, 'bars')
```

4.6　利用 matplotlib 进行可视化

如今，Python 有很多绘图包，而 matplotlib 是它们中极其热门的一个软件包。Python 社区正逐渐认识到 matplotlib 强大的作图能力。matplotlib 的创造者兼项目负责人 John

Hunter 总结到：matplotlib tries 让一些原本简单的操作进一步简化，而困难的操作更有可能得到解决。利用 matplotlib，你可以轻松做出高质量的、可用于出版的图像。本节，我们将举例说明 matplotlib 的强大。

4.6.1 词云

在词云中，单词在文本中出现的频率越高，它在图像中就越突出。词云也叫作标签云或是加权后的单词。你可以利用不同的字体、布局方式、配色方案来调整词云。一个词的重要程度，即它在文章中出现的次数将通过它在词云中的大小来展现。词云中最大的单词也就是在文本中出现次数最多的单词。

词云除直接展示文本中不同单词的出现次数，还可用于社交媒体和市场营销。它的一些应用如下：

- 企业可以通过词云了解客户是如何看待他们的产品的。一些机构用创造性方法让其追随者用几个词语来描述他们对公司品牌的看法，然后将这些词语放到词云中，以更好地理解他们的产品品牌在客户心中的共同印象。
- 通过考察竞争品牌的网络形象来了解竞争对手。利用网络上的信息制作词云，以便更好地理解什么样的词语和主题对产品的目标市场更具有吸引力。

为了制作词云，你可以自行编写一段 Python 代码，或者利用一些已经存在的词云制作程序。来自纽约大学数据科学中心的 Andreas Mueller 在 Python 中制作了一个非常简单且便于使用的词云程序。下一节给出了这个程序的安装过程。

4.6.2 安装词云

为了快速安装，你必须通过 sudo 权限使用 pip 命令进行安装，代码如下：

```
sudo pip install git+git://github.com/amueller/word_cloud.git
```

除了上面的方法，你也可以通过 wget（Linux）或 curl（Mac OS）进行安装，安装代码如下：

```
wget https://github.com/amueller/word_cloud/archive/master.zip
unzip master.zip
rm master.zip
cd word_cloud-master
sudo pip install -r requirements.txt
```

对于 Anaconda IDE，你需要利用 conda 进行安装，安装过程分为以下三步：

```
#step-1 command
conda install wordcloud

Fetching package metadata: ....
Error: No packages found in current osx-64 channels matching: wordcloud

You can search for this package on Binstar with
# This only means one has to search the source location
binstar search -t conda wordcloud

Run 'binstar show <USER/PACKAGE>' to get more details:
Packages:
                          Name | Access        | Package Types    |
     -------------------------- | ------------ | ---------------- |
              derickl/wordcloud | public        | conda            |
Found 1 packages

# step-2 command
binstar show derickl/wordcloud

Using binstar api site https://api.binstar.org
Name:    wordcloud
Summary:
Access:  public
Package Types:  conda
Versions:
   + 1.0

To install this package with conda run:
conda install --channel https://conda.binstar.org/derickl wordcloud

# step-3 command
conda install --channel https://conda.binstar.org/derickl wordcloud
Fetching package metadata: ......
Solving package specifications: .
Package plan for installation in environment /Users/MacBook/anaconda:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
```

```
    cython-0.22                |           py27_0         2.2 MB
    django-1.8                 |           py27_0         3.2 MB
    pillow-2.8.1               |           py27_1         454 KB
    image-1.3.4                |           py27_0          24 KB
    setuptools-15.1            |           py27_1         435 KB
    wordcloud-1.0              |       np19py27_1          58 KB
    conda-3.11.0               |           py27_0         167 KB
    ------------------------------------------------------------
                                           Total:         6.5 MB

The following NEW packages will be INSTALLED:
    django:     1.8-py27_0
    image:      1.3.4-py27_0
    pillow:     2.8.1-py27_1
    wordcloud:  1.0-np19py27_1

The following packages will be UPDATED:
    conda:      3.10.1-py27_0 --> 3.11.0-py27_0
    cython:     0.21-py27_0   --> 0.22-py27_0
    setuptools: 15.0-py27_0   --> 15.1-py27_1


The following packages will be DOWNGRADED:

    libtiff:    4.0.3-0       --> 4.0.2-1

Proceed ([y]/n)? y
```

4.6.3 词云的输入

本节，你可以从两个资源中提取单词，构建词云。第一个例子展示了如何从一些已知网站上提取文本数据，并从中提取单词。第二个例子展示了如何从 tweet 上利用搜索词提取文本。这些例子需要用到 feedparser 包和 tweepy 包，和前面安装其他包类似，你可以轻松安装它们。

我们的方法是从这些例子中收集单词，并将它们作为一个普通词云程序的输入。

1. 信息来源

大部分新闻和技术服务网站都存在着结构清晰的 RSS 或日志（atom）信息源。尽管我们的目标是将提取的文本内容严格限定在技术领域，但我们还是可以先确定少量的信息

源列表，如下面的代码所示。为了能够解析这些信息源，我们需要使用 feedparser 当中的 parser() 函数。词云有它自己的 stopwords 列表，但我们在收集数据的同时也可以自定义一个 stopwords 列表（本例中的 stopwords 列表不够全面，但是你可以从网上找到更多），代码如下：

```
import feedparser
from os import path
import re

d = path.dirname(__file__)
mystopwords = [  'test', 'quot', 'nbsp']

feedlist = ['http://www.techcrunch.com/rssfeeds/',
'http://www.computerweekly.com/rss',
'http://feeds.twit.tv/tnt.xml',
'https://www.apple.com/pr/feeds/pr.rss',
'https://news.google.com/?output=rss'
'http://www.forbes.com/technology/feed/'                    'http://rss.
nytimes.com/services/xml/rss/nyt/Technology.xml',           'http://www.
nytimes.com/roomfordebate/topics/technology.rss',
'http://feeds.webservice.techradar.com/us/rss/reviews'
'http://feeds.webservice.techradar.com/us/rss/news/software',
'http://feeds.webservice.techradar.com/us/rss',
'http://www.cnet.com/rss/',
'http://feeds.feedburner.com/ibm-big-data-hub?format=xml',
'http://feeds.feedburner.com/ResearchDiscussions-DataScien
ceCentral?format=xml',          'http://feeds.feedburner.com/
BdnDailyPressReleasesDiscussions-BigDataNews?format=xml',
'http://http://feeds.feedburner.com/ibm-big-data-hub-
galleries?format=xml',          'http://http://feeds.feedburner.com/
PlanetBigData?format=xml',
'http://rss.cnn.com/rss/cnn_tech.rss',
'http://news.yahoo.com/rss/tech',
'http://slashdot.org/slashdot.rdf',
'http://bbc.com/news/technology/']

def extractPlainText(ht):
    plaintxt=''
    s=0
    for char in ht:
        if char == '<': s = 1
        elif char == '>':
            s = 0
            plaintxt += ' '
        elif s == 0: plaintxt += char
    return plaintxt

def separatewords(text):
    splitter = re.compile('\\W*')
    return [s.lower() for s in splitter.split(text) if len(s) > 3]
```

```
def combineWordsFromFeed(filename):
    with open(filename, 'w') as wfile:
      for feed in feedlist:
        print "Parsing " + feed
        fp = feedparser.parse(feed)
        for e in fp.entries:
          txt = e.title.encode('utf8') +
               extractPlainText(e.description.encode('utf8'))
          words = separatewords(txt)

          for word in words:
            if word.isdigit() == False and word not in mystopwords:
              wfile.write(word)
              wfile.write(" ")
          wfile.write("\n")
    wfile.close()
    return

combineWordsFromFeed("wordcloudInput_FromFeeds.txt")
```

2. Twitter 文本

为了接入 Twitter 的 API，你需要访问令牌和用户授权，它们由 4 个参数组成：access_token、access_token_secret、consumer_key 和 consumer_secret。为了获取这些参数，你需要有一个 Twitter 账号。在 Twitter 的网页上，有获取这些参数方法的说明，获取说明的步骤如下：

1. 登陆 Twitter 账号。

2. 跳转到 developer.twitter.com，点击“Manage My Apps”并遵循上面的提示，最终获得需要的 4 个参数。

假设你已经获取了所有参数，利用 tweepy 包，你就可以通过 Python 接入到 tweet。下面的代码展示了一个简单的流监听器（stream listener）。这里，由于 tweet 的数据是流状的，一个监听器可以监听现有的情形并把它记录到文件中。这些记录之后可以用于制作词云。

流（stream）利用一个筛选器来挑选出那些内容关于 Python 编程、数据可视化、大数据、机器学习和统计的 Twitter 文本。tweepy 的流可以提供抽取后的 tweet。由于 Twitter 上有着近乎无限的数据，提取数据的程序可以永远运行下去。那么，我们应该如何让程序停止？由于程序接入 Twitter 的速度通常比你想象得要慢，所以你要考虑为了制作一个词云需要提取大约多少数量的 tweet 消息。因此，我们通过设置 MAX_TWEETS 参数来对提取 tweet 消息的数量进行限制，在下面的代码中，MAX_TWEETS 设置为 50：

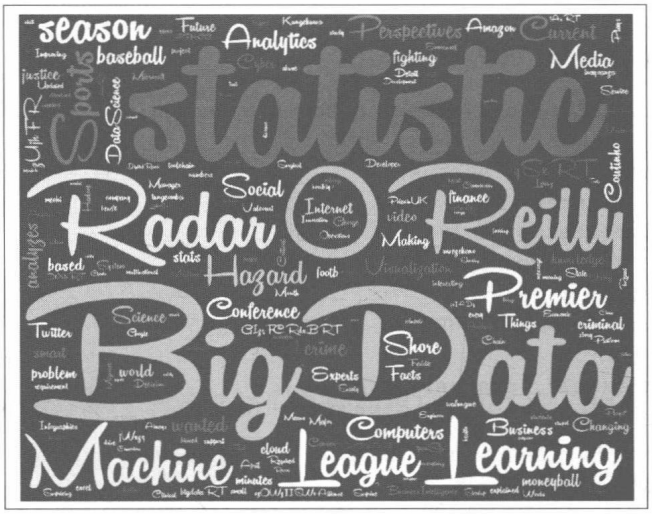
```
import tweepy
import json
import sys
```

```
import codecs

counter = 0
MAX_TWEETS = 500

#Variables that contains the user credentials to access Twitter API
access_token = "Access Token"
access_token_secret = "Access Secret"
consumer_key = "Consumer Key"
consumer_secret = "Consumer Secret"

fp = codecs.open("filtered_tweets.txt", "w", "utf-8")

class CustomStreamListener(tweepy.StreamListener):

     def on_status(self, status):
        global counter
        fp.write(status.text)
        print "Tweet-count:" +str(counter)
        counter += 1
        if counter >= MAX_TWEETS: sys.exit()

    def on_error(self, status):
        print status

if __name__ == '__main__':

    auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_token, access_token_secret)
    streaming_api = tweepy.streaming.Stream(auth,
                CustomStreamListener(), timeout=60)

    streaming_api.filter(track=['python program', 'statistics',
             'data visualization', 'big data', 'machine learning'])
```

利用包中的语句，你可以用少于20行的Python代码建立一个词云对象。然后利用matplotlib.pyplot中的imshow()函数将词云对象绘制成图像。下面的词云代码可以应用于任意由单词构成的输入文件：

```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
from os import path

d = path.dirname("__file__")
text = open(path.join(d, 'filtered_tweets.txt')).read()

wordcloud = WordCloud(
    font_path='/Users/MacBook/kirthi/RemachineScript.ttf',
    stopwords=STOPWORDS,
```

```
        background_color='#222222',
        width=1000,
        height=800).generate(text)

# Open a plot of the generated image.
plt.figure(figsize=(13,13))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```



代码中需要的字体文件可以从很多网站上下载（如 http://www.dafont.com/remachine-script.font）。对于本地的字体文件，你需要把文件的精确路径赋值到 font_path 上。对于使用网页信息源获取的数据，只需修改上述一行代码，具体的更改如下：

```
text = open(path.join(d, 'wordcloudInput_fromFeeds.txt')).read()
```

用类似的方法可以提取 tweet 文本绘制词云。你可以利用 iPhone、Samsung Galaxy、Amazon Fire、LG Optimus 和 Nokia Lumia 等关键字提取有关手机供应商的文字，进而分析用户对不同品牌手机的偏好。在这种情况下，你还需要一些额外的信息集，即与词语相关的表示积极与消极的情感数值。

这里有一些能够帮助你进行 tweet 内容情感分析的方法。首先，最直接的方法是对每个词语赋予一个权重，积极情绪的权重用 w_p 表示，消极情绪的权重用 w_n 表示，用 $p(+)$ 表示出现一个积极情绪的概率，用 $p(-)$ 表示出现一个消极情绪的概率：

$$p(+) = e^{\frac{1}{w_p - w_n + 1}}$$
$$p(-) = 1 - p(+)$$

第二种方法是利用自然语言处理工具，并应用经过训练的分类器获得更好的结果。如 TextBlob 是一个文本处理包，它也包含情感分析功能（http://textblob.readthedocs.org/en/dev）。

TextBlob 建立了一个文本分类系统，并建立一组 JSON 格式的训练集。之后利用训练集拟合朴素贝叶斯分类器进行情感分析。在后面的章节中，我们将尝试使用这种工具来展示我们的例子。

4.6.4　绘制股票价格图

美国两个最大的证券交易所是：成立于 1792 年的纽约证券交易所（NYSE）和成立于 1971 年的纳斯达克证券交易所（NASDAQ）。今天，大部分证券市场交易都采用电子化。甚至股票本身几乎全部记录在电子表格中，而不再是实物证书。除 NASDAQ 和 NYSE 之外，还有为数众多的网站提供实时的股价数据服务。

获取数据

Yahoo 作为获取数据的其中一个网站，通过 API 提供数据。例如，从网站 http://chartapi.finance.yahoo.com/instrument/1.0/amzn/chartdata;type=quote;range=3y/csv 上可以获取 Amazon 股票，包括涨、跌、开盘、收盘、交易量等股价数据。基于所选的绘图方法，你需要进行一些数据转换。例如从数据源获取的时间数据往往不具有可直接识别的时间数据结构，如下所示：

```
uri:/instrument/1.0/amzn/chartdata;type=quote;range=3y/csv
ticker:amzn
Company-Name:Amazon.com, Inc.
Exchange-Name:NMS
unit:DAY
timestamp:
first-trade:19970516
last-trade:20150430
currency:USD
```

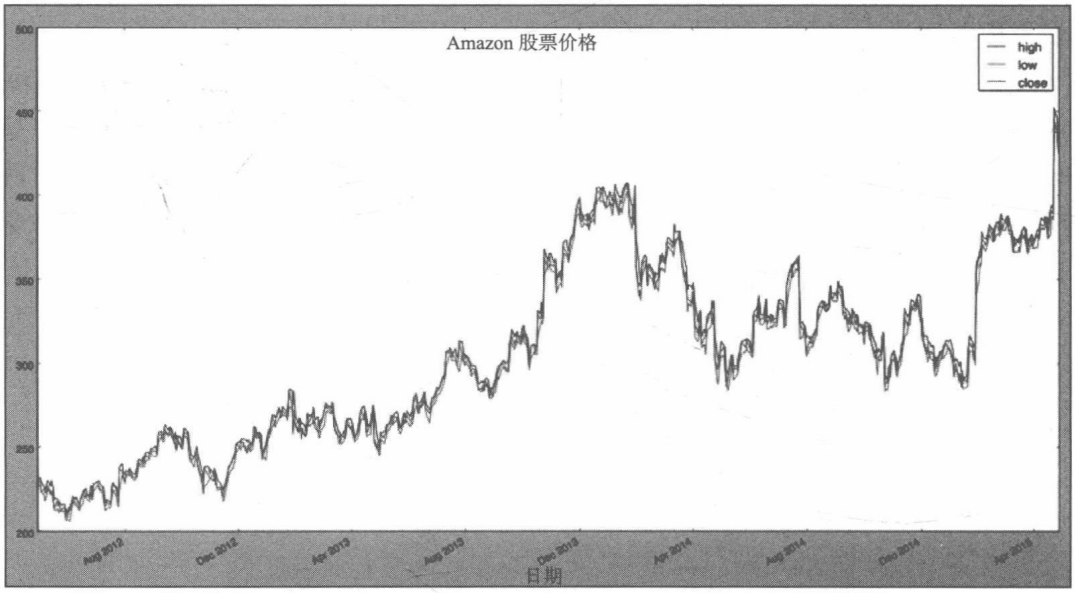
```
previous_close_price:231.9000
Date:20120501,20150430
labels:20120501,20120702,20121001,20130102,20130401,20130701,20131001,
20140102,20140401,20140701,20141001,20150102,20150401
values:Date,close,high,low,open,volume
close:208.2200,445.1000
high:211.2300,452.6500
low:206.3700,439.0000
open:207.4000,443.8600
volume:984400,23856100
20120501,230.0400,232.9700,228.4000,229.4000,6754900
20120502,230.2500,231.4400,227.4000,227.8200,4593400
20120503,229.4500,232.5300,228.0300,229.7400,4055500
...
...
20150429,429.3700,434.2400,426.0300,426.7500,3613300
20150430,421.7800,431.7500,419.2400,427.1100,3609700
```

我们将讨论三种绘图方法，每一种方法都有其优点与局限。

第一种方法，我们使用 matplotlib.cbook 包和 pylab 包进行绘图，代码如下：

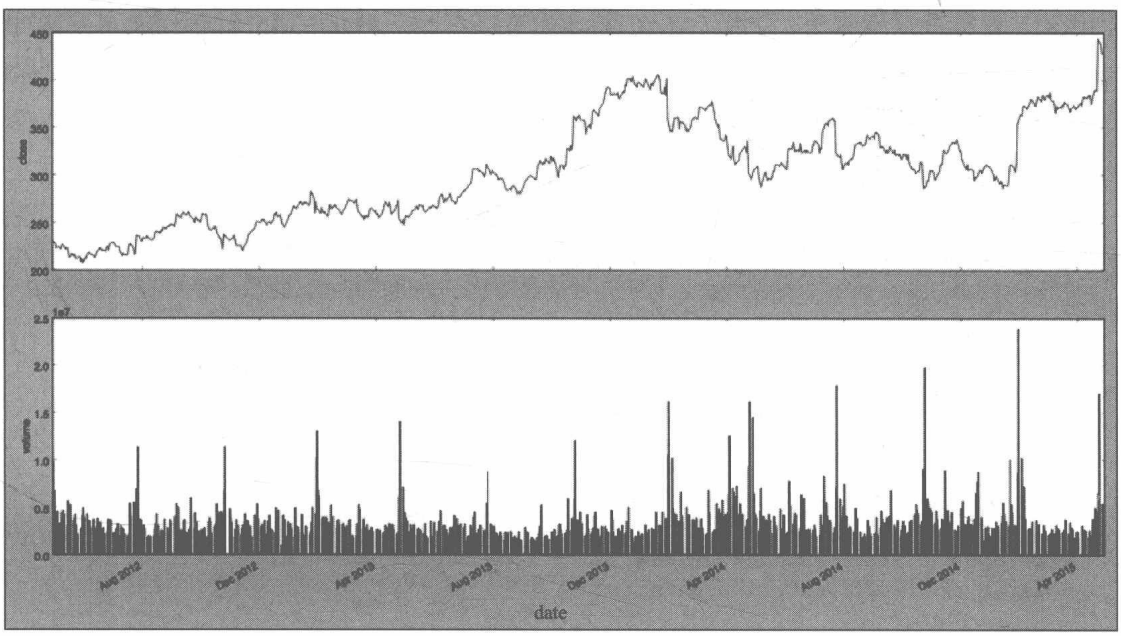
```
from pylab import plotfile show, gca
import matplotlib.cbook as cbook
fname = cbook.get_sample_data('/Users/MacBook/stocks/amzn.csv',
asfileobj=False)
plotfile(fname, ('date', 'high', 'low', 'close'), subplots=False)
show()
```

利用这种方法绘制的图像如下图所示：



采用这种方法进行绘图时，我们需要先把时间数值从形如 20150430 的表示方法转化为 %d-%b-%Y 的形式。我们可以把绘图分为两部分，一部分用来展示股票的价格，另一部分用来展示交易量，具体的代码如下：

```
from pylab import plotfile show, gca
import matplotlib.cbook as cbook
fname = cbook.get_sample_data('/Users/MacBook/stocks/amzn.csv',
asfileobj=False)
plotfile(fname, (0,1,5), plotfuncs={f:'bar'})
show()
```



第二种方法是采用 matplotlib.mlab 和 matplotlib.finance 的子包。这种方法能够很方便地从 http://ichart.finance.yahoo.com/table.csv?s=GOOG&a=04&b=12&c=2014&d=0 6&e=20&f=2015&g=d 上获取股票数据。为了展示一个例子，我们给出一个代码片段：

```
ticker='GOOG'

import matplotlib.finance as finance
import matplotlib.mlab as mlab
import datetime

startdate = datetime.date(2014,4,12)
today = enddate = datetime.date.today()

fh = finance.fetch_historical_yahoo(ticker, startdate, enddate)
r = mlab.csv2rec(fh); fh.close()
```

```
r.sort()
print r[:2]

[ (datetime.date(2014, 4, 14), 538.25, 544.09998, 529.56, 532.52002,
2568000, 532.52002)  (datetime.date(2014, 4, 15), 536.82001,
538.45001, 518.46002, 536.44, 3844500, 536.44)]
```

当你尝试利用图像对不同股票的价格进行比较时，会发现图像中无法显示交易量信息。因为每一种股票的交易量是不同的。另一方面，如果存在交易量信息，图像也会变得过于杂乱。

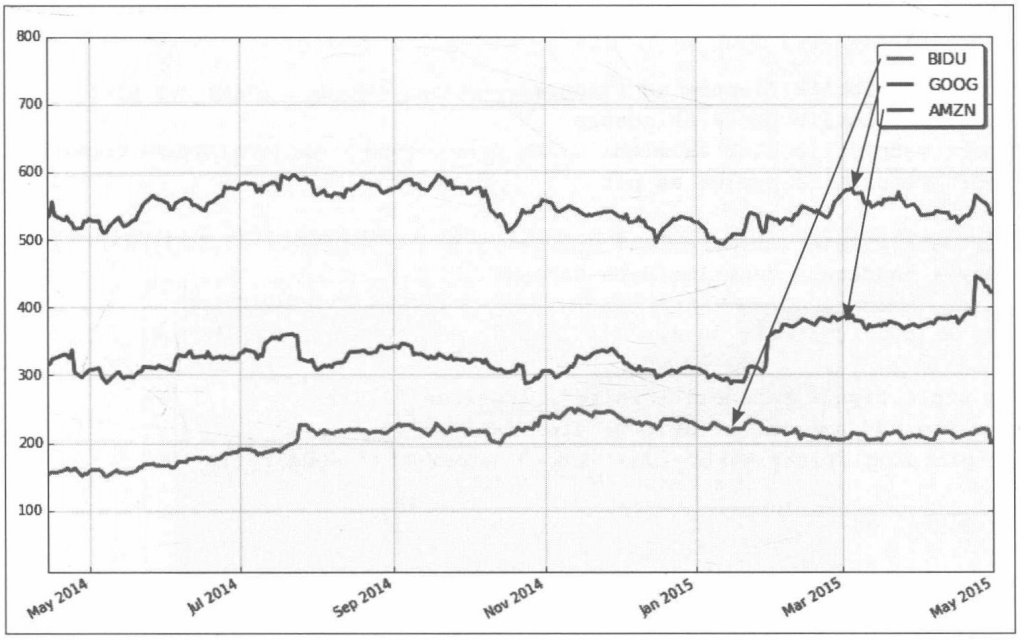
matplotlib 中具有一个绘制股票图像的实例，该实例讲解详细，并且包含相对强度指标（RSI）和移动平均收敛 / 发散（MACD）。你可以在 http://matplotlib.org/examples/pylab_examples/finance_work2.html 上找到该实例。想了解更对关于 RSI 和 MACD 的细节，你可以参考 http://easyforextrading.co/how-to-trade/indicators/（包含对它们的有趣的理解），或者在互联网上搜寻更多的资料。

对现有代码进行修改使得它能够用于绘制多幅图像，函数 plotTicker() 就产生了。该函数可将每只股票画在同一个坐标系下，其代码如下：

```
import datetime
import numpy as np

import matplotlib.finance as finance
import matplotlib.dates as mdates
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

startdate = datetime.date(2014,4,12)
today = enddate = datetime.date.today()

plt.rc('axes', grid=True)
plt.rc('grid', color='0.75', linestyle='-', linewidth=0.5)
rect = [0.4, 0.5, 0.8, 0.5]

fig = plt.figure(facecolor='white', figsize=(12,11))

axescolor = '#f6f6f6' # the axes background color

ax = fig.add_axes(rect, axisbg=axescolor)
ax.set_ylim(10,800)

def plotTicker(ticker, startdate, enddate, fillcolor):
  """
     matplotlib.finance has fetch_historical_yahoo() which fetches
     stock price data the url where it gets the data from is
     http://ichart.yahoo.com/table.csv stores in a numpy record
     array with fields:
      date, open, high, low, close, volume, adj_close
  """
```

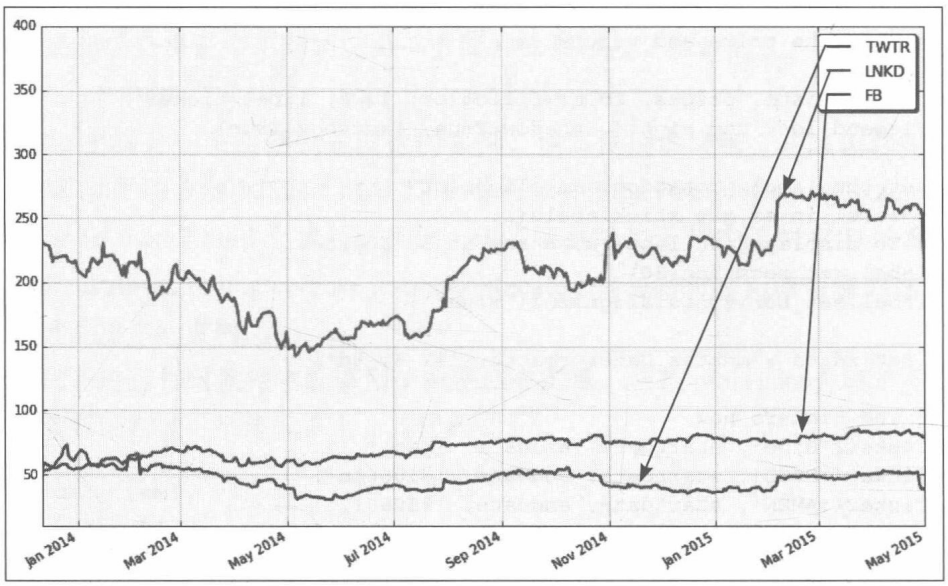
```
    fh = finance.fetch_historical_yahoo(ticker, startdate, enddate)
    r = mlab.csv2rec(fh);
    fh.close()
    r.sort()
    ### plot the relative strength indicator
    ### adjusted close removes the impacts of splits and dividends
    prices = r.adj_close

    ### plot the price and volume data

    ax.plot(r.date, prices, color=fillcolor, lw=2, label=ticker)
    ax.legend(loc='top right', shadow=True, fancybox=True)

    # set the labels rotation and alignment
    for label in ax.get_xticklabels():
      # To display date label slanting at 30 degrees
      label.set_rotation(30)
      label.set_horizontalalignment('right')

    ax.fmt_xdata = mdates.DateFormatter('%Y-%m-%d')

#plot the tickers now
plotTicker('BIDU', startdate, enddate, 'red')
plotTicker('GOOG', startdate, enddate, '#1066ee')
plotTicker('AMZN', startdate, enddate, '#506612')

plt.show()
```

用这种方法比较 Baidu、Google 和 Amazon 的股价时，所得图像看起来如下图所示：



可用如下代码绘制图像，比较 Twitter、Facebook 和 LinkedIn 的股价：

```
plotTicker('TWTR', startdate, enddate, '#c72020')
plotTicker('LNKD', startdate, enddate, '#103474')
plotTicker('FB', startdate, enddate, '#506612')
```

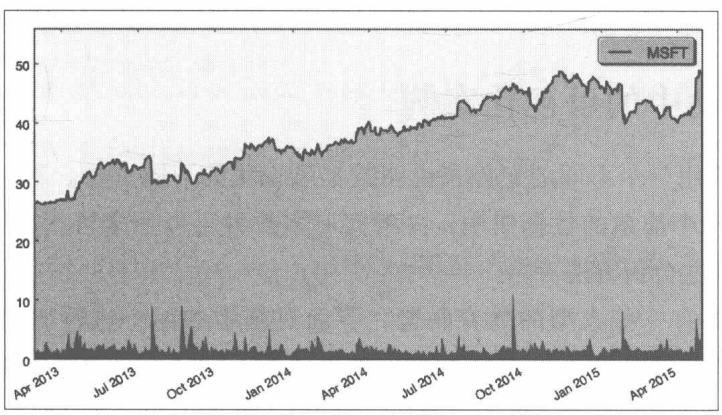


现在，你也可以向图像中加入交易量。利用下面的代码可以得到一个加入交易量的图表：

```
import datetime

import matplotlib.finance as finance
import matplotlib.dates as mdates
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

startdate = datetime.date(2013,3,1)
today = enddate = datetime.date.today()

rect = [0.1, 0.3, 0.8, 0.4]

fig = plt.figure(facecolor='white', figsize=(10,9))
ax = fig.add_axes(rect, axisbg='#f6f6f6')
def plotSingleTickerWithVolume(ticker, startdate, enddate):

    global ax

    fh = finance.fetch_historical_yahoo(ticker, startdate, enddate)

    # a numpy record array with fields:
```

```
    #     date, open, high, low, close, volume, adj_close
    r = mlab.csv2rec(fh);
    fh.close()
    r.sort()

    plt.rc('axes', grid=True)
    plt.rc('grid', color='0.78', linestyle='-', linewidth=0.5)

    axt = ax.twinx()
    prices = r.adj_close

    fcolor = 'darkgoldenrod'

    ax.plot(r.date, prices, color=r'#1066ee', lw=2, label=ticker)
    ax.fill_between(r.date, prices, 0, prices, facecolor='#BBD7E5')
    ax.set_ylim(0.5*prices.max())

    ax.legend(loc='upper right', shadow=True, fancybox=True)

    volume = (r.close*r.volume)/1e6  # dollar volume in millions
    vmax = volume.max()

    axt.fill_between(r.date, volume, 0, label='Volume',
                     facecolor=fcolor, edgecolor=fcolor)

    axt.set_ylim(0, 5*vmax)
    axt.set_yticks([])

    for axis in ax, axt:
        for label in axis.get_xticklabels():
            label.set_rotation(30)
            label.set_horizontalalignment('right')

        axis.fmt_xdata = mdates.DateFormatter('%Y-%m-%d')

plotSingleTickerWithVolume ('MSFT', startdate, enddate)
plt.show()
```

基于前面修改后的代码，可以绘制出一个带交易量的股票价格变化图，所得图像与下图类似：



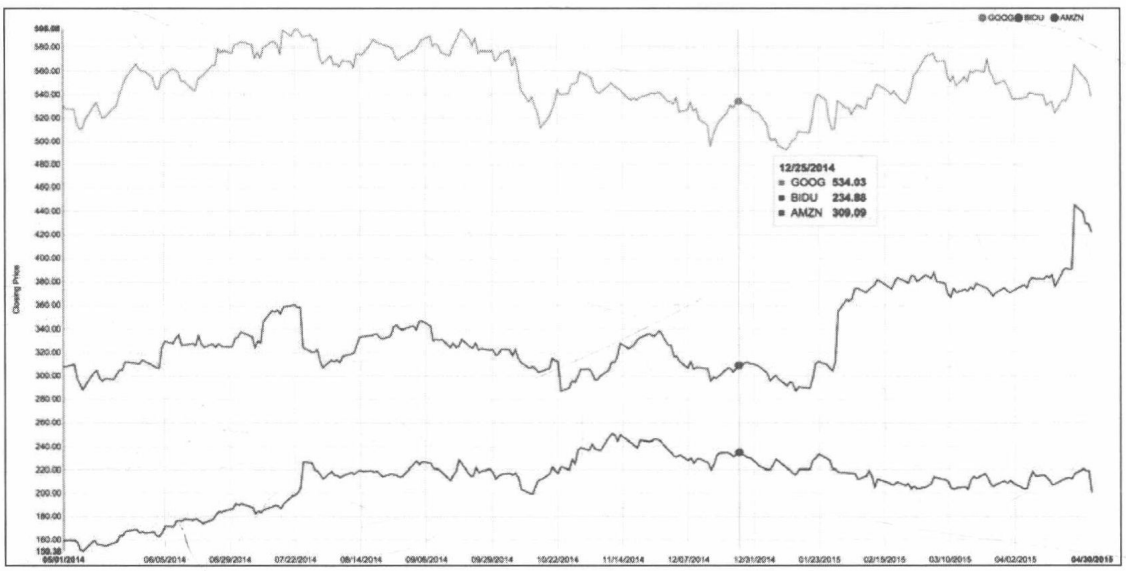
第三种绘制图像的方法是使用 blockspring 包。为了安装 blockspring 包，你需要用以下的 pip 命令：

```
pip install blockspring
```

Blockspring 方法是生成 HTML 代码，它能够自动生成 JavaScript 格式的用于绘图的数据。通过 D3.js 集成后，它可以绘制出非常好看的交互式图像。令人震惊的是，这种方法只需要两行代码：

```
import blockspring
import json

print blockspring.runParsed("stock-price-comparison",
   { "tickers": "FB, LNKD, TWTR",
   "start_date": "2014-01-01", "end_date": "2015-01-01" }).params
```

根据不同的操作系统，代码运行时，它会在系统默认的区域内生成 HTML 代码。



4.7 体育运动中的可视化案例

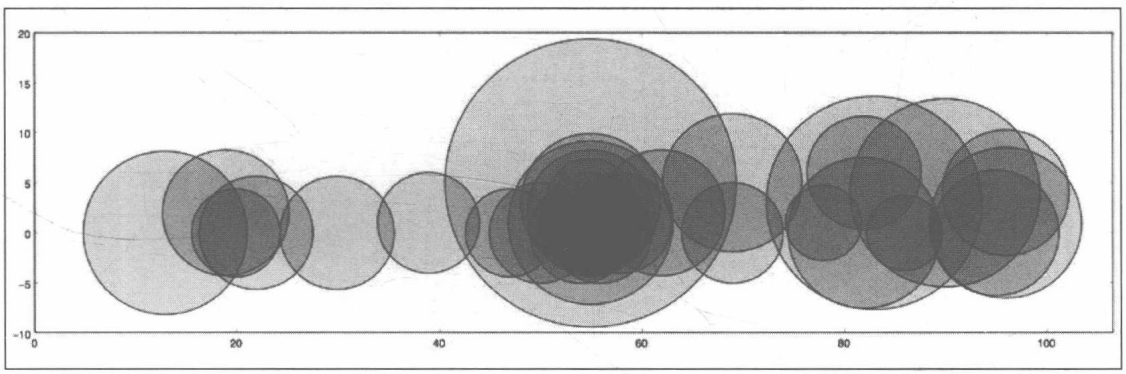
下面，让我们用一个不同的案例来说明数据可视化的不同方法。不考虑计算问题，我们将分析限定在一个简单的数据集中，并考察有多少种可以用来可视化分析数据的方法，以帮助我们明确这些分析方法。

在北美体育界有一些大型的体育联盟，下面对其中的四个进行比较：**全国足球联盟**（NFL）、**美国职棒大联盟**（MLB）、**美国职业篮球联赛**（NBA）和**国家曲棍球联盟**。NFL 的

球队总价值为 91.3 亿美元，年总收入为 95.8 亿美元。我们将选取 NFL 下各个球队的价值和冠军数量数据（这里只展示部分数据）：

	Team.name	Team.value	Years.Completed	Num.Championships	Championships.yr.average
1	Dallas Cowboys	3210	55	5	23.18
2	Washington Redskins	2400	83	3	12.22
3	New York Giants	2100	90	4	13.88
4	Houston Texans	1850	13	0	5.00
5	New York Jets	1810	55	1	8.63
6	Philadelphia Eagles	1750	82	0	5.00
7	Chicago Bears	1700	96	1	7.08
8	New England Patriots	1635	55	4	15.90
9	San Francisco 49ers	1600	69	5	19.49
10	Baltimore Ravens	1500	19	2	26.05

球队价值是衡量不同球队水平的重要指标，而冠军数量也是具有重要参考价值的指标。用 x 轴表示不同年份，y 轴表示冠军的数量，气泡大小代表每年平均获得的冠军数量，可绘制出图像如下：

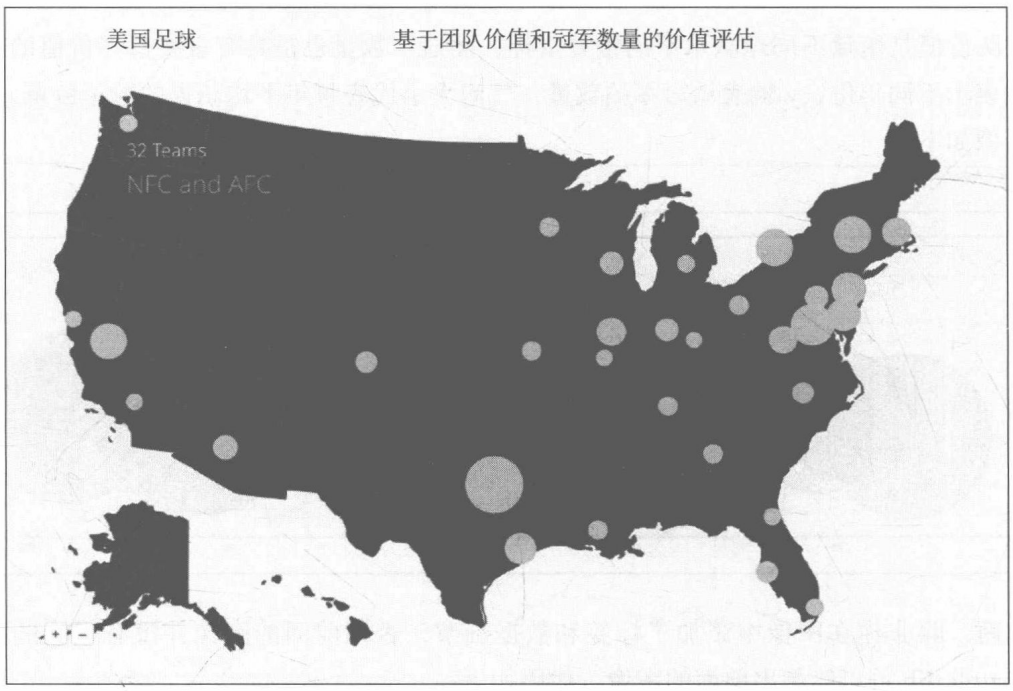


然而，除非你在图像中添加了标签和数据细节，否则前面的图像并没有很大的用处。利用 matplotlib 可以绘制出前面的图像，代码如下：

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(15,10), facecolor='w')

def plotCircle(x,y,radius,color, alphaval):
  circle = plt.Circle((x, y), radius=radius, fc=color,\
   alpha=alphaval)
  fig.gca().add_patch(circle)
  nofcircle = plt.Circle((x, y), radius=radius, ec=color, \
   fill=False)
  fig.gca().add_patch(nofcircle)

x = [55,83,90,13,55,82,96,55,69,19,55,95,62,96,82,30,22,39, \
```

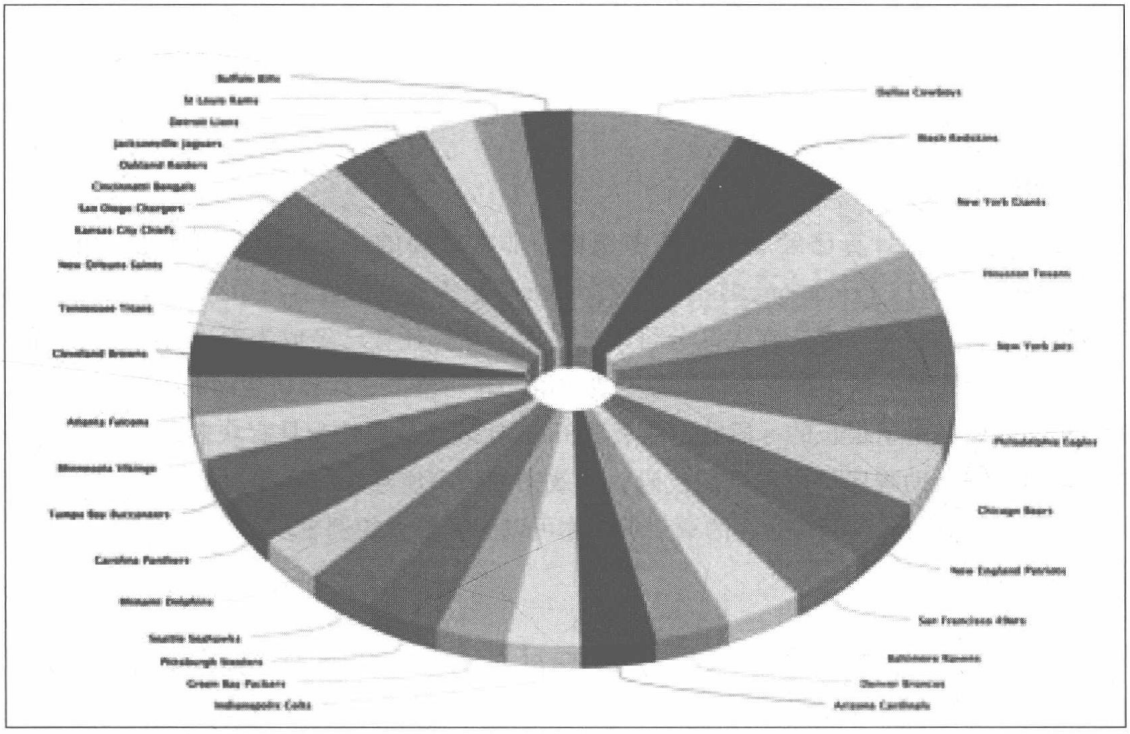
```
  54,50,69,56,58,55,55,47,55,20,86,78,56]
y = [5,3,4,0,1,0,1,3,5,2,2,0,2,4,6,0,0,1,0,0,0,0,1,1,0,0,3,0, \
  0,1,0]
r = [23,17,15,13,13,12,12,11,11,10,10,10,10,10,9,9,9,8,8,8,8, \
    8,8,8,7,7,7,7,6,6,6]
for i in range(0,len(x)):
  plotCircle(x[i],y[i],r[i],'b', 0.1)


plt.axis('scaled')
plt.show()
```

你甚至可以将数值数据转化为另外一种 JavaScript 可以理解的 JSON 数据类型，从而让数据能够交互地展现在 SVG 地图中，在地图上展现这些数据，如下图所示：

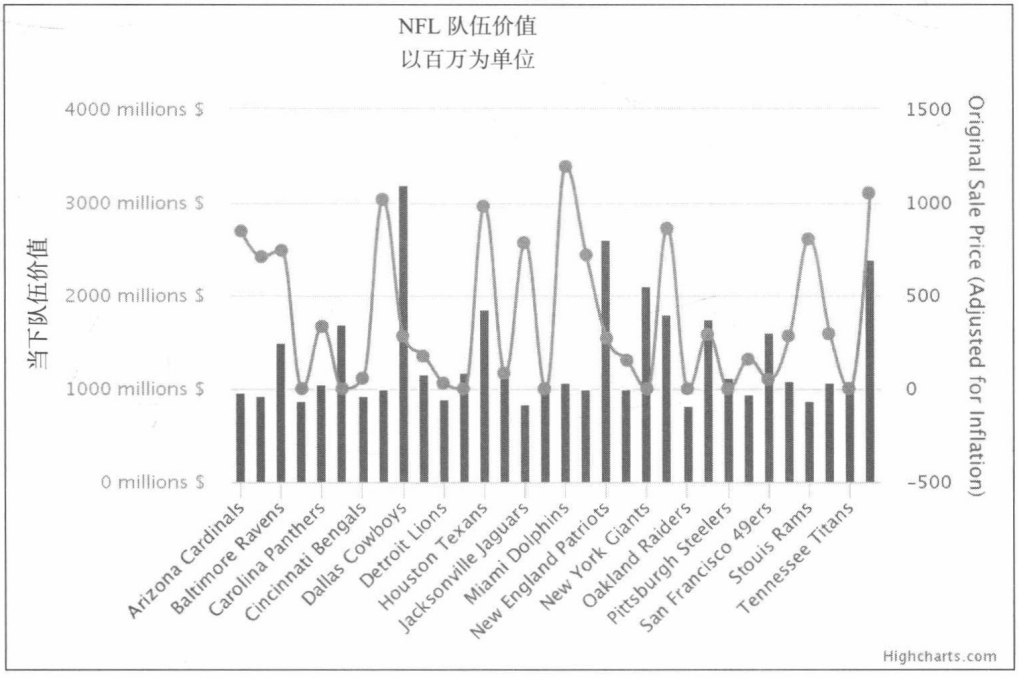


如果在前面带有气泡的地图上加入相应的标签，那么地图的可视化效果将进一步提升。然而，由于地图空间有限，一种更好的方法是在地图中加入交互式的工具，从而使得数据信息能够通过导航的形式进行展现，大幅增强可视化的效果。

原始数据可从 http://tinyurl.com/oyxk72r 或者 http://www.knapdata.com/python/nfl_franch.html 网站上获得。除平面气泡图和绘制在地图上的气泡图外，你还可以选择很多其他可视化方法。如果你想将全部 32 个队伍的数据用一个扇形图或条形图进行展现，那么图像会变得非常杂乱，同时，你也很难识别图像上的标签。这告诉我们，当我们尝试去展现这种类型的数据的时候，需要寻找一些其他更好的可视化方法，如下图所示：



如果你将队伍价值位于某一特定范围的队伍合并到一起，那么通过在图像中合并他们，你可以更加规整的展现这些队伍的价值，如下图所示：

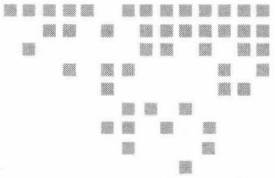


上图是展现队伍价值的一种方法，它通过将队伍划分到不同的组来展现不同队伍的价值。同时用“2300 百万美元”替代“$2300 000 000”，使得标签更易读。

4.8 总结

近几十年来，计算已经成为了很多领域中非常重要的组成部分。事实上，Stanford、UC-Berkeley、MIT、Princeton、Harvard、Caltech 等很多高校的计算科学课程内容已经调整，并适应跨学科的课程要求。在大部分科学学科中，计算任务已经成为理论与实践之外的重要补充。大量的理论和实践论文都包含一些数值计算、模拟、计算机建模的内容。

Python 已经走了很长的一段路，今天 Python 社区已经成长为具有众多资源和工具的平台，它们能够帮助程序员用最少的代码，高效完成计算所涉及的几乎所有问题。本章，我们只列举了几个例子，但在后面的章节中，你可以看到更多的例子。



第 5 章 Chapter 3

金融和统计模型

金融和经济模型充分运用概率统计学知识，主要有助于数据的简化和提取。了解数据往往非常重要，就像在数据分析前，我们应该先通过绘图了解数据特点一样。通过数据可视化，我们通常可以发现诸如不良数据、离群点和缺失数据，等等。不良数据应该尽可能修正或直接删掉。然而，在某些特殊情况下，例如股市，离群点反而需要保留。总的来说，发现和理解异常值和离群点同等重要，这是做出适当处理的前提。模型中的变量选择显得尤为重要。

因为模型的本质决定了我看到的事实，因此模型中的变量选择十分重要。例如，为了度量通货膨胀，通过建模你才能够理解价格的实际变化以及与通货膨胀直接联系的价格变化。

有很多有趣的模型及应用值得讨论，但考虑到本书的范畴，我们将有选择的介绍一些案例。在某些情况下，例如蒙特卡洛，我们也将选择一些体育领域的例子。后面几节讨论下述内容：

- 蒙特卡洛模拟——多领域应用举例
- 价格模型及应用举例
- 举例理解波动性度量
- 阈值模型——Shelling 的分离模型
- 有绘图选项的贝叶斯回归方法
- 几何布朗运动、基于扩散的模拟和投资组合估值
- 分析和创建实时交互图
- 统计与机器学习综述

计算金融学是计算机科学的一个领域，处理金融建模过程中涉及的数据和算法。有些读者能很好理解本章内容，但对于其他读者，接触这些概念将有助于学到一些有益于生活，或能运用于他们感兴趣领域的新见解。

在了解蒙特卡罗模拟方法的应用之前，我们先看一个一段时期内投资和收益率的简单例子。

5.1 确定性模型

投资的最终目的是为了盈利，并且投资收益或损失取决于价格变动和持有资产。通常，投资者对与初始资本规模高度相关的收益感兴趣。投资回报之所以能用于衡量资本，主要是因为它本身也是一种资产。例如，股票、债券，或股票和债券的一个投资组合用一种变化的形式来定义，而价格用初始价格的分数形式表达。下面不妨看看总收益的例子。

总收入

假设 P_t 是时刻 t 时的投资金额。一种简单的总收益可表示为：

$$\frac{P_{t+1}}{P_t}=1+R_{t+1}$$

其中，P_{t+1} 是返回的投资价值，R_{t+1} 是收益率。例如，当 P_t = 10，P_{t+1}= 10.6 时，R_{t+1} = 0.06 = 6%。收益率是没有尺度的，意味着它们不随着单位变化而变化。但是收益率跟时间 t 的单位有关（小时、天等）。换言之，如果 t 以年为单位，那么更精确地说，净收益率为每年 6%。

最近 k 年的总收入是 k 年间（从 t–k 到 t）每年总收入的乘积，如下所示：

$$\begin{aligned}1+R_t(k)&=\frac{P_t}{P_{t-k}}\\&=\left(\frac{P_t}{P_{t-1}}\right)\left(\frac{P_{t-1}}{P_{t-2}}\right)\cdots\left(\frac{P_{t-k+1}}{P_{t-k}}\right)\\&=(1+R_t)(1+R_{t-1})\cdots(1+R_{t-k+1})\end{aligned}$$

这是一个确定性模型的例子，但还有一点需要注意：必须在方程中纳入每年的通货膨胀率。不妨假设与收益率 R_t 对应的通货膨胀率为 F_t，由上述方程可得：

$$1+R_t(k)=\frac{(1+R_t)}{(1+F_t)}\frac{(1+R_{t-1})}{(1+F_{t-1})}\cdots\frac{(1+R_{t-k+1})}{(1+F_{t-k+1})}$$

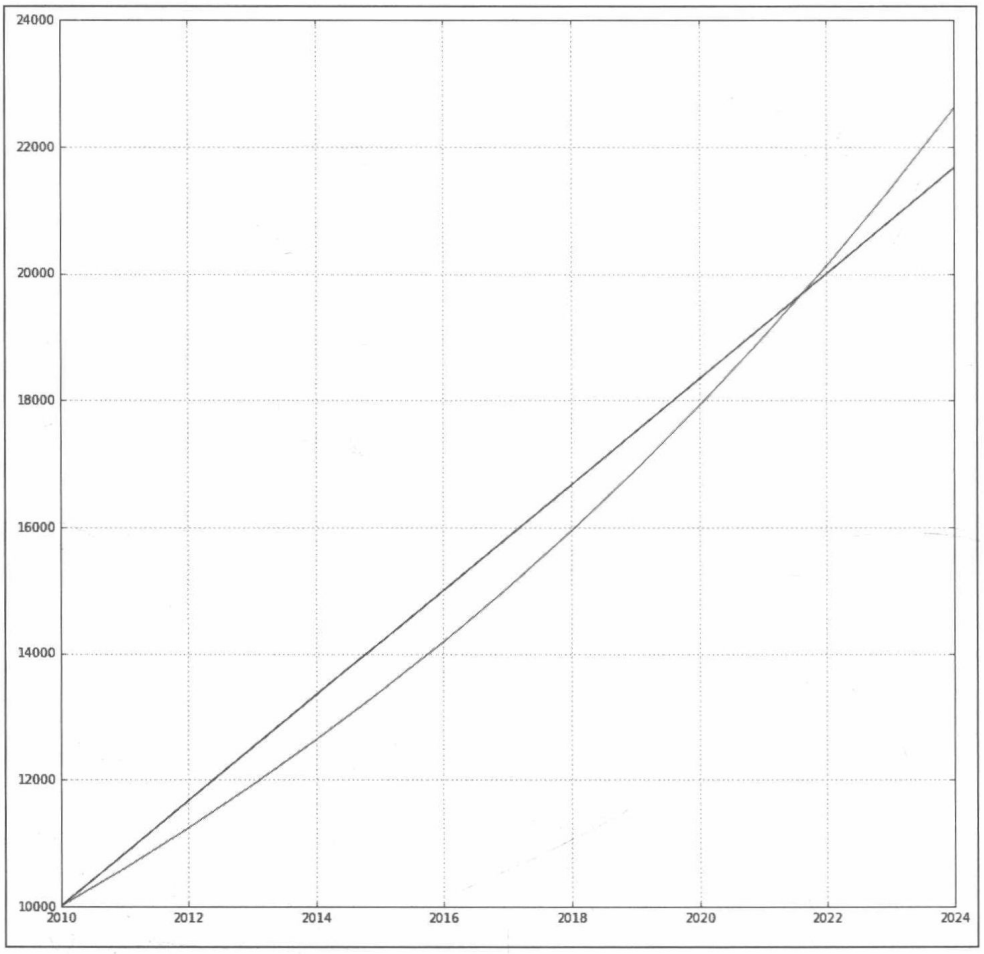
假设 F_t=0，则前面的公式仍然适用。在不考虑通货膨胀率的情况下，提出这样一个问题："如果 2010 年投资 $10 000，年收益率为 6%，那么经过多少年投资价值会翻倍？"

让我们试着用 Python 程序解决这个问题。在这个程序中，我们也要添加一条逼近 $y = 2x$ 的直线，看看它与以"年"为 x 轴、收益为 y 轴的曲线在何处相交。首先，我们不绘制这条直线，观察投资价值是否能在 12 年内几乎翻一番。然后，计算直线的斜率 m= 10 000/12 = 833.33。因此，我们在程序中加入直线斜率 833.33，展示收益和这条直线。下面代码比较了收入与这条直线间的关系。

```
import matplotlib.pyplot as plt

principle_value=10000  #invested amount
grossReturn = 1.06     # Rt

return_amt = []
x = []
y = [10000]
year=2010
return_amt.append(principle_value)
x.append(year)
for i in range(1,15):
  return_amt.append(return_amt[i-1] * grossReturn)
  print "Year-",i," Returned:",return_amt[i]

  year += 1
  x.append(year)
  y.append(833.33*(year-2010)+principle_value)

# set the grid to appear
plt.grid()

# plot the return values curve
plt.plot(x,return_amt, color='r')
plt.plot(x,y, color='b')
```

```
Year- 1 Returned: 10600.0
Year- 2 Returned: 11236.0
Year- 3 Returned: 11910.16
Year- 4 Returned: 12624.7696
Year- 5 Returned: 13382.255776
Year- 6 Returned: 14185.1911226
Year- 7 Returned: 15036.3025899
Year- 8 Returned: 15938.4807453
Year- 9 Returned: 16894.78959
Year- 10 Returned: 17908.4769654
Year- 11 Returned: 18982.9855834
Year- 12 Returned: 20121.9647184
Year- 13 Returned: 21329.2826015
Year- 14 Returned: 22609.0395575
```

看过这个图之后，你会想知道是否有一种方法能找到银行提供了按揭贷款的金额。这个问题我们将留给读者。



有趣的是，曲线与直线在2022年前相交。交点处，收益正好是$20 000。然而，在2022年，收益约为$20 121。我们已经了解过总收入，那么股票也类似吗？很多股票，尤其对于成熟的企业，需要按照上述方程来支付利息。

如果在时间 t 之前支付股息（或利息）D_t，则在时刻 t 时的总收益定义如下：

$$1+R_t=\frac{P_t+D_t}{P_{t-1}}$$

另一个例子是抵押贷款，其中一定量的贷款是按照利息率从某家金融机构借来的。为了了解业务属性，我们选择一笔30年期，利息率为5%的贷款$ 350 000。这是美国抵押贷款（贷款金额和利息率的变化取决于贷款者的信用记录和市场利率）的一个典型例子。

一个简单的利息计算方式是 $P\ (1 + rt)$，其中 P 为本金，r 是利率，t 是时间期限，因此30年后的总累积金额为：

$$350\ 000\times\left(1+\frac{5}{100}\times 30\right)=350\ 000\times\frac{5}{2}=875\ 000$$

事实证明，在第30年年末，你将支付两倍以上的贷款金额（计算结果没有考虑不动产税）：

```
from decimal import Decimal
import matplotlib.pyplot as plt

colors = [(31, 119, 180),(174, 199, 232),(255,128,0),(255, 15, 14),
      (44, 160, 44),(152, 223, 138),(214, 39, 40),(255,173, 61),
      (148, 103, 189),(197, 176, 213),(140, 86, 75),(196, 156, 148),
      (227, 119, 194),(247, 182, 210),(127, 127, 127),
      (199, 199, 199),(188, 189, 34), (219, 219, 141),
      (23, 190, 207), (158, 218, 229)]

# Scale the RGB values to the [0, 1] range, which is the format
matplotlib accepts.
for i in range(len(colors)):
    r, g, b = colors[i]
    colors[i] = (r / 255., g / 255., b / 255.)

def printHeaders(term, extra):
    # Print headers
    print "\nExtra-Payment: $"+str(extra)+" Term:"+str(term)+" years"
    print "------------------------------------------------------------"
    print 'Pmt no'.rjust(6), ' ', 'Beg. bal.'.ljust(13), ' ',
    print 'Payment'.ljust(9), ' ', 'Principal'.ljust(9), ' ',
    print 'Interest'.ljust(9), ' ', 'End. bal.'.ljust(13)
    print ''.rjust(6, '-'), ' ', ''.ljust(13, '-'), ' ',
    print ''.rjust(9, '-'), ' ', ''.ljust(9, '-'), ' ',
    print ''.rjust(9, '-'), ' ', ''.ljust(13, '-'), ' '

def amortization_table(principal, rate, term, extrapayment,
printData=False):
    xarr=[]
    begarr = []

    original_loan = principal
    money_saved=0
    total_payment=0
    payment = pmt(principal, rate, term)
    begBal = principal

    # Print data
    num=1
    endBal=1
    if printData == True: printHeaders(term, extrapayment)
```

```
    while  (num < term + 1) and (endBal >0):

        interest = round(begBal * (rate / (12 * 100.0)), 2)
        applied = extrapayment+round(payment - interest, 2)
        endBal = round(begBal - applied, 2)
        if (num-1)%12 == 0 or (endBal < applied+extrapayment):
          begarr.append(begBal)
          xarr.append(num/12)
          if printData == True:
              print '{0:3d}'.format(num).center(6), ' ',
              print '{0:,.2f}'.format(begBal).rjust(13), ' ',
              print '{0:,.2f}'.format(payment).rjust(9), ' ',
              print '{0:,.2f}'.format(applied).rjust(9), ' ',
              print '{0:,.2f}'.format(interest).rjust(9), ' ',
              print '{0:,.2f}'.format(endBal).rjust(13)
        total_payment += applied+extrapayment
        num +=1
        begBal = endBal
    if extrapayment > 0 :
      money_saved = abs(original_loan - total_payment)
      print '\nTotal Payment:','{0:,.2f}'.format(total_payment).
rjust(13)
      print '  Money Saved:','{0:,.2f}'.format(money_saved).rjust(13)
    return xarr, begarr, '{0:,.2f}'.format(money_saved)

def pmt(principal, rate, term):
    ratePerTwelve = rate / (12 * 100.0)

    result = principal * (ratePerTwelve / (1 - (1 + ratePerTwelve) **
(-term)))

    # Convert to decimal and round off to two decimal
    # places.
    result = Decimal(result)
    result = round(result, 2)
    return result

plt.figure(figsize=(18, 14))

#amortization_table(150000, 4, 180, 500)
i=0
markers = ['o','s','D','^','v','*','p','s','D','o','s','D','^','v','*
','p','s','D']
markersize=[8,8,8,12,8,8,8,12,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8]

for extra in range(100,1700,100):
  xv, bv, saved = amortization_table(450000, 5, 360, extra, False)
  if extra == 0:
    plt.plot(xv, bv, color=colors[i], lw=2.2, label='Principal only',
marker=markers[i], markersize=markersize[i])
  else:
    plt.plot(xv, bv, color=colors[i], lw=2.2, label="Principal
plus\$"+str(extra)+str("/month, Saved:\$")+saved, marker=markers[i],
markersize=markersize[i])
  i +=1
```

```
plt.grid(True)
plt.xlabel('Years', fontsize=18)
plt.ylabel('Mortgage Balance', fontsize=18)
plt.title("Mortgage Loan For $350,000 With Additional Payment Chart",
fontsize=20)
plt.legend()
plt.show()
```

程序运行完成后，你会得到以每 12 个月为单位的所有额外支付的摊销时间表，从 $100 到 $1600。下面是其中一个例子：

```
Extra-Payment: $800 Term: 30 years
-------------------------------------------------------------------Pmt
no   Beg. bal.    Payment    Principal    Interest     End. bal.     -----
-   ----------   --------   ----------   ----------   ------
  1      350,000.00  1,878.88* 1,220.55  1,458.33    348,779.45
 13      335,013.07  1,878.88  1,282.99  1,395.89    333,730.08
 25      319,259.40  1,878.88  1,348.63  1,330.25    317,910.77
 37      302,699.75  1,878.88  1,417.63  1,261.25    301,282.12
 49      285,292.85  1,878.88  1,490.16  1,188.72    283,802.69
 61      266,995.41  1,878.88  1,566.40  1,112.48    265,429.01
 73      247,761.81  1,878.88  1,646.54  1,032.34    246,115.27
 85      227,544.19  1,878.88  1,730.78    948.10    225,813.41
 97      206,292.20  1,878.88  1,819.33    859.55    204,472.87
109      183,952.92  1,878.88  1,912.41    766.47    182,040.51
121      160,470.74  1,878.88  2,010.25    668.63    158,460.49
133      135,787.15  1,878.88  2,113.10    565.78    133,674.05
145      109,840.70  1,878.88  2,221.21    457.67    107,619.49
157       82,566.78  1,878.88  2,334.85    344.03     80,231.93
169       53,897.49  1,878.88  2,454.31    224.57     51,443.18
181       23,761.41  1,878.88  2,579.87     99.01     21,181.54
188        5,474.98  1,878.88  2,656.07     22.81      2,818.91
189        2,818.91  1,878.88  2,667.13     11.75        151.78

* $1878.88 includes $1078.88 plus $800 extra payment towards principal
Total Payment: $504,526.47 Money Saved: $154,526.47
```

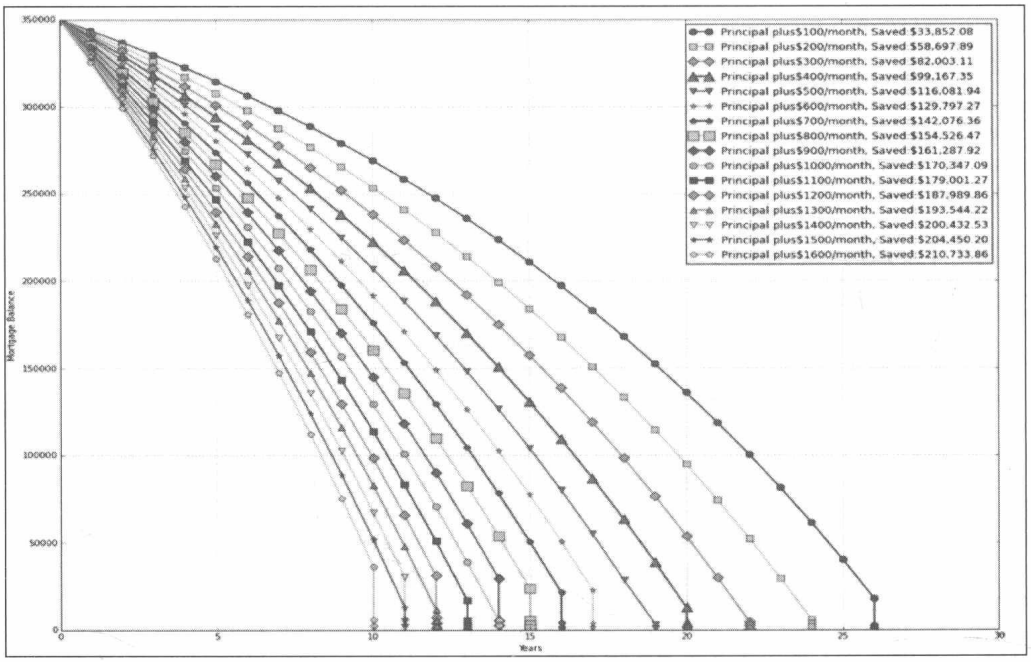
大约 15 年 10 个月之后，可以还清所有贷款，时间大概提前了一半。

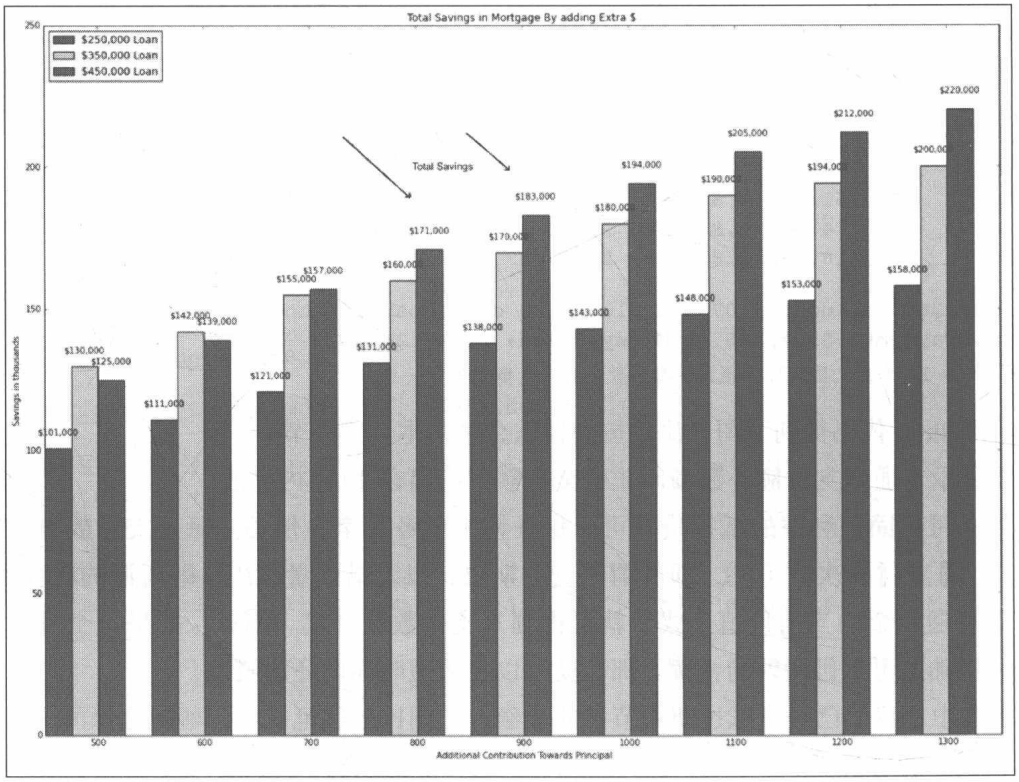
运行 Python 代码得到下图，比较按揭付款金额与本金额外节余。

上图显示了通过支付额外数额而非本金数额，按揭在 30 年内完成。

每月支付有固定利率的抵押贷款可以保证期末债务的全息偿还。每月支付数额取决于利率（r），月支付的次数（N），即所谓的贷款期限，以及借款量（P），即所谓的借款本金。改变一下普通年金的当前价值公式，就可得到月支付数额公式。不过，如果每个月在支付既定金额基础上再支付额外的金额，那么就可以在更短时间内还清贷款。

在下图中，我们已经尝试过使用节省的金额，绘图比较这笔款项与额外金额间（范围在 $500~$1300）存在的差异。仔细观察，我们发现，额外多支付 $800，几乎可以节省一半的贷款额度，并在半个期限内还清贷款。





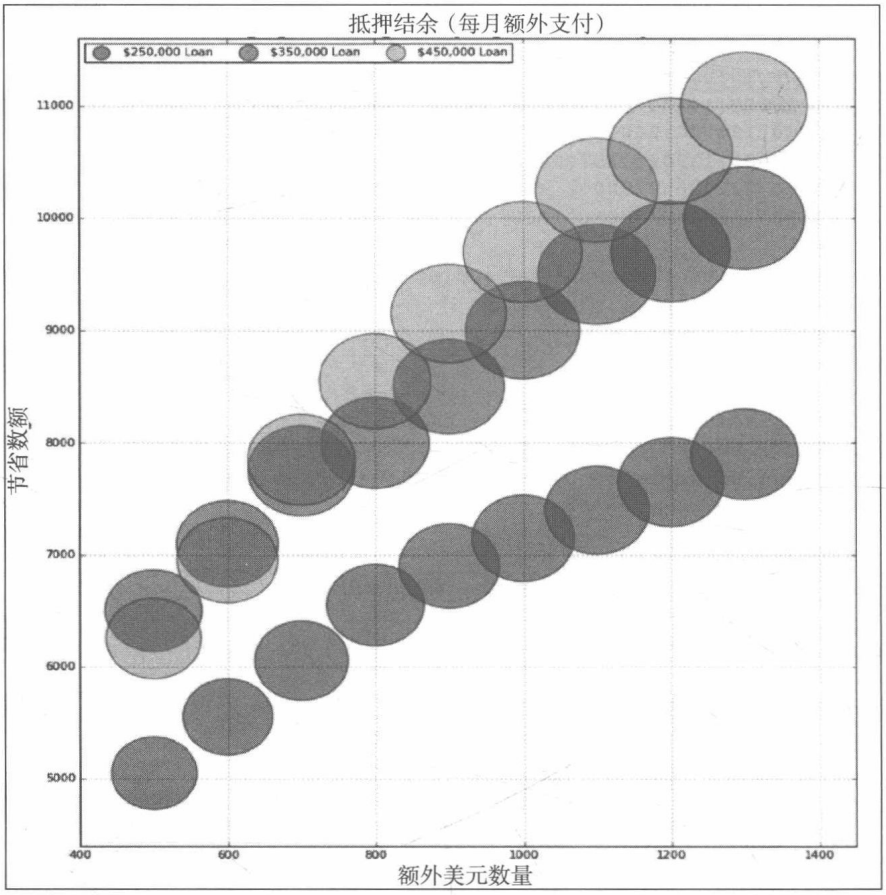
上图显示了三个不同的贷款数额的节省情况，其中，x 轴表示额外支付金额的贡献，y

轴表示以千为单位的节省数额。下面代码使用气泡图展示了按揭贷款中节省数额与额外金额及本金间的关系：

```
import matplotlib.pyplot as plt

# set the savings value from previous example
yvals1 = [101000,111000,121000,131000,138000,
143000,148000,153000,158000]
yvals2 = [130000,142000,155000,160000,170000,
180000,190000,194000,200000]
yvals3 = [125000,139000,157000,171000,183000,
194000,205000,212000,220000]
xvals = ['500','600','700', '800', '900','1000','1100','1200','1300']

#initialize bubbles that will be scaled
bubble1 = []
bubble2 = []
bubble3 = []

# scale it on something that can be displayed
# It should be scaled to 1000, but display will be too big
# so we choose to scale by 5% (divide these by 20 again to relate
# to real values)

for i in range(0,9):
  bubble1.append(yvals1[i]/20)
  bubble2.append(yvals2[i]/20)
  bubble3.append(yvals3[i]/20)

#plot yvalues with scaled by bubble sizes
#If bubbles are not scaled, they don't fit well
fig, ax = plt.subplots(figsize=(10,12))
plt1 = ax.scatter(xvals,yvals1, c='#d82730', s=bubble1, alpha=0.5)
plt2 = ax.scatter(xvals,yvals2, c='#2077b4', s=bubble2, alpha=0.5)
plt3 = ax.scatter(xvals,yvals3, c='#ff8010', s=bubble3, alpha=0.5)

#Set the labels and title
ax.set_xlabel('Extra Dollar Amount', fontsize=16)
ax.set_ylabel('Savings', fontsize=16)
ax.set_title('Mortgage Savings (Paying Extra Every Month)',
        fontsize=20)

#set x and y limits
ax.set_xlim(400,1450)
ax.set_ylim(90000,230000)

ax.grid(True)
ax.legend((plt1, plt2, plt3), ('$250,000 Loan', '$350,000 Loan',
     '$450,000 Loan'), scatterpoints=1, loc='upper left',
      markerscale=0.17, fontsize=10, ncol=1)

fig.tight_layout()
plt.show()
```

通过绘制散点图，很容易看出哪一个贷款范围能为贷款人提供节省空间。但为了保持图像简洁，我们只比较三个贷款金额：$ 250 000、$350 000 和 $450 000。

下面的散点图展示了通过每月额外支付可以节省的数额：



5.2 随机性模型

我们已经讨论了确定性模型，其中定量输入值的单个输出没有随机性。“随机性(stochastic)”一词源于希腊词汇“Stochastikos”。意思是擅长猜测或冒险。其反义词是“一定”、“确定”或“当然”。随机性模型根据事件发生的可能性或者概率加权可能性来预测事件发生的结果。例如，抛出的硬币最终一定落地，但究竟是正面朝上还是反面朝上是随机的。

5.2.1 蒙特卡洛模拟

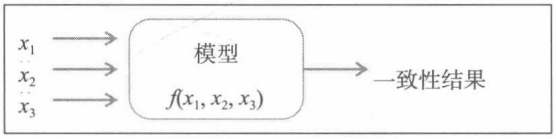
蒙特卡罗模拟作为一种概率模拟，用来研究风险和不确定性因素对预测模型的影响。

蒙特卡罗方法是1940年Stanislaw Ulam在Los Alamos国家实验室参与核武器项目时提出的。今天，随着计算机的发展，我们可以方便快速地生成随机数和运行模拟。但令人惊奇的是，他多年前发现这个方法的时候计算还相当困难。

在预测模型或决策未来的任何模型中，都需要先做出一些假定。这些假定可能关于投资组合的投资，或完成一定任务所需的时间。因为这些都是有关未来的课题，因此最好是完成预期值的估计。

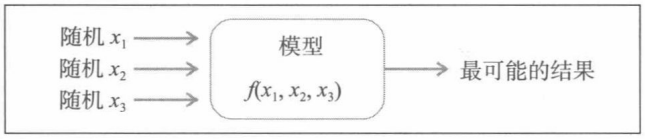
1. 究竟什么是蒙特卡洛模拟

蒙特卡罗模拟是一种输入一系列随机数反复评估确定性模型的方法。该方法适用于复杂模型、非线性模型或包含很多不确定参数的模型。模拟通常会涉及超过10万甚至百万次模型评估。让我们来看看确定性模型和随机性模型间的区别。确定性模型将输入一些实际值，保证结果一致，如下图所示：



让我们看看概率性模型与确定性模型之间的区别。

随机性模型的输入值是概率性的，而且来自概率密度函数，生成的结果也是概率性的。随机性模型如下图所示：



现在，我们如何用文字描述上图？

首先，创建一个模型。比如，已知三个随机输入：$x1$、$x2$和$x3$，一种方法：$f(x1, x2, x3)$，并产生一组10 000随机数的输入（在某些情况下，可以少一点或多一点）。然后评估模型，重复，结果记为yi，其中，i从1到10 000。分析并选择一个最有可能的结果。

比如，如果我们要回答下面的问题："洛杉矶快船队赢得第七场比赛的概率多大？"根据篮球的相关知识，我们输入一些随机数，就可以通过蒙特卡罗模拟找到答案：他们有45%的机会获胜。事实上，他们输掉了比赛。

蒙特卡罗模拟在很大程度上取决于随机数生成器。因此，需要弄清楚完成蒙特卡罗模拟最快而有效的方式是什么？ Hans Petter Langtangen曾在http://hplgit.github.io/teamods/MC_cython/sphinx/main_MC_ cython.html上展示了用Cython运行蒙特卡罗模拟比用C语

言更加有效。

让我们通过几个例子具体说明蒙特卡洛模拟。第一个例子是库存问题。后面我们将讨论蒙特卡洛模拟在体育方面的应用（蒙特卡罗模拟适用于体育分析）。

2. 蒙特卡洛模拟中的库存问题

水果零售员每天出售水果，每份订单有 Y 个单位。每卖出一个单位会有 60 美分的利润，在一天结束时未售出的部分以每件 40 美分的损失抛售。每天的需求 D 服从均匀分布 [80，140]。请问：为了使预期利润最大化，需要有多少单位的订单？

让我们用 P 表示利润。根据前面的描述，我们用 s 表示的销售数量，d 表示的需求量，则方程表示为：

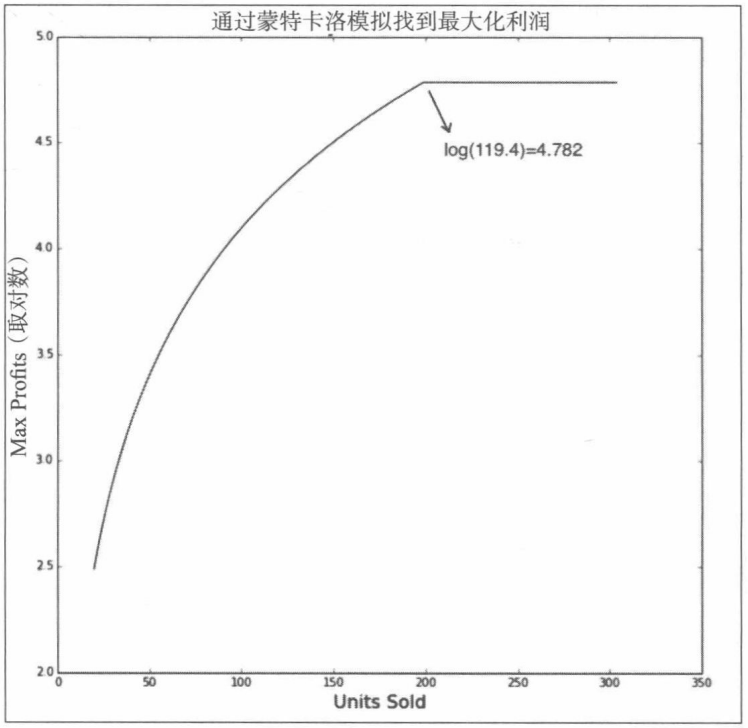
$$P=\begin{cases}0.6s & \text{若 } d \geqslant s \\ 0.6d-0.4(s-d) & \text{若 } s>d\end{cases}$$

用这种利润表示法，下面 Python 代码实现了最大化利润：

```
import numpy as np
from math import log

import matplotlib.pyplot as plt

x=[]
y=[]

#Equation that defines Profit
def generateProfit(d):

   global s

   if d >= s:
     return 0.6*s
   else:
     return 0.6*d - 0.4*(s-d)

# Although y comes from uniform distribution in [80,140]
# we are running simulation for d in [20,305]

maxprofit=0

for s in range (20, 305):

  # Run a simulation for n = 1000
  # Even if we run for n = 10,000 the result would
  # be almost the same
  for i in range(1,1000):
```

```
        # generate a random value of d
        d = np.random.randint(10,high=200)
        # for this random value of d, find profit and
        # update maxprofit
        profit = generateProfit(d)
        if profit > maxprofit:
            maxprofit = profit

    #store the value of s to be plotted along X axis
    x.append(s)

    #store the value of maxprofit plotted along Y axis
    y.append(log(maxprofit)) # plotted on log scale

plt.plot(x,y)
print "Max Profit:",maxprofit

# Will display this
Max Profit: 119.4
```

下图显示当销售数量增加时，利润也会增加。但当需求得到满足时，最大利润会保持不变：



上图对最大利润取对数，即意味着通过 1000 次模拟得到的利润为 119.4。现在，让我们看一种分析方法，比较模拟结果与分析方法结果的接近程度。

由于需求（D）服从 [80,140] 均匀分布，预期的利润可以通过下述积分求得：

$$
\begin{aligned}
\text{利润} &= \int_s^{140} \frac{0.6s}{60}\,dx + \int_{80}^{s} \frac{0.6x - 0.4(s-x)}{60}\,dx \\
&= \int_s^{140} \frac{0.6s}{60}\,dx + \int_{80}^{s} \frac{(x-0.4s)}{60}\,dx \\
&= \frac{s}{100}(140-s) + \frac{s^2}{600} + \frac{8}{15}s - \frac{160}{3} \\
&= \frac{7}{5}s - \frac{s^2}{100} + \frac{s^2}{600} + \frac{8}{15}s - \frac{160}{3} \\
&= -\frac{5}{600}s^2 + \frac{29}{15}s - \frac{160}{3} \\
&-\frac{s}{60} + \frac{29}{15} = 0 \Rightarrow s = \frac{29\times 60}{15} = 116
\end{aligned}
$$

使用分析方法得到的答案是 116，蒙特卡洛模拟也会在 119 周围产生波动，例如 118 或 116。这取决于试验的次数。

让我们来看看另一个简单的例子：“一个班有 30 名学生，至少有两位学生在同一天生日的概率是多少？”我们假定不是闰年，不用月和日来表示生日，只用日历年的天数，即 365 天。这个逻辑很简单。下面的代码展示了如何计算一个班 30 名学生中同一天生日超过一人的概率：

```
import numpy as np
numstudents = 30
numTrials = 10000
numWithSameBday = 0

for trial in range(numTrials):
    year = [0]*365

    for i in range(numstudents):
        newBDay = np.random.randint(365)
        year[newBDay] = year[newBDay] + 1

    haveSameBday = False
    for num in year:
        if num > 1:
            haveSameBday = True

    if haveSameBday == True:
        numWithSameBday = numWithSameBday + 1

prob = float(numWithSameBday) / float(numTrials)
print("The probability of a shared birthday in a class of ",
numstudents, " is ", prob)
```

```
('The probability of a shared birthday in a class of ', 30, ' is ',
0.7055)
```

换句话说，在一个由 30 名学生组成的班级中，两个人在同一天生日的概率为 70%。下面的例子说明现如今运用蒙特卡洛模拟计算比赛获胜的似然函数比过去更常见。

3. 蒙特卡罗模拟在篮球中的应用

让我们考虑用 JavaScript 处理一个由可汗学院举办的篮球比赛的例子。问题是："当落后 3 分且离比赛结束还有 30 秒时，应该尝试投难度较大的 3 分球还是更容易的 2 分球后再控制下一个球？"(Lebron James 问到)。

大多数读者可能对篮球比赛有所了解，这里我们只强调其中一些重要的规则。每支球队有 24 秒守住球的时间。在这个时间内，如果他们得分（甚至更短时间），对方球队会在接下来的 24 秒守住球。然而，当比赛剩余时间只有 30 秒，如果一个游戏者可以在大概不到 10 秒钟进行快速 3 分球，那么会留给对方球队大概 20 秒。与任何其他运动类似，篮球是一项竞争性很强的运动，而且由于球员的目标是减少比分的差距，那么他们对投三分球最感兴趣。让我们试着写 Python 程序解答这个问题。

在展现模拟程序之前，不妨先看一下这一问题中涉及的相关参数。下面给出的本队球员和对方球员的统计量对于确定投 3 分球是否更有利于增加获胜的概率非常重要。球员得 3 分的概率（threePtPercent）和得 2 分的概率（twoPtPercent）不仅决定了他的优势，还决定了对方的球队得 2 分的概率（oppTwoPtPercent）和对方球队在罚球命中率（oppFtPercent）情况下取得的优势。

还有很多其他组合，但是为了简单起见，我们就不详细讨论了。对方球队的罚球命中率越高，我们更倾向于得出球队投 3 分的结论。你可以降低 oppFtPercent 的值，不妨看看下面的讨论内容。在这个例子中，我们给出一些不同的代码片段，你可以用自己感觉舒服的方式把这些片段放在一起，并用它们来运行。首先，我们需要使用 NumPy 软件包中的随机数生成器。我们也需要 matplotlib 软件包进行绘图。代码如下：

```
import numpy as np
import matplotlib.pyplot as plt
```

在许多例子中，我们将使用 tableau 中的标准颜色，而且可能希望把它放在一个单独的文件中。下面是一组可在任何可视化绘图中使用的颜色代码：

```
colors = [(31, 119, 180), (174, 199, 232), (255, 127,  14),
    (255, 187, 120), (44, 160, 44), (214,  39,  40), (148,103,189),
    (152, 223, 138), (255,152,150), (197, 176, 213), (140, 86, 75),
    (196, 156, 148), (227,119,194), (247, 182, 210), (127,127,127),
    (199, 199, 199),(188,189, 34),(219, 219, 141), (23, 190,207),
```

```
    (158, 218, 229),(217,217,217)]

# Scale RGB values to the [0, 1] range, format matplotlib accepts.
for i in range(len(colors)):
  r, g, b = colors[i]
  colors[i] = (r / 255., g / 255., b / 255.)
```

让我们来尝试投 3 分球。如果 threePtPercent 比随机数大，比赛加时的可能性越大，那么赢得比赛就有保证。看看下面的代码：

```
def attemptThree():
  if np.random.randint(0, high=100) < threePtPercent:
    if np.random.randint(0, high=100) < overtimePercent:
      return True #We won!!
  return False #We either missed the 3 or lost in OT
```

选择投 2 分球与离比赛结束的时间，以及球在哪个球队手里有关。假设平均意义上，尝试投 2 分球平均需 5 秒钟，而且球员 twoPtPercent 的概率相当大，那么他们会得到 2 分球，并从 pointsDown 中扣除。下面函数描述了投 2 分球的尝试：

```
def attemptTwo():
  havePossession = True
  pointsDown = 3
timeLeft = 30
while (timeLeft > 0):
  #What to do if we have possession
  if (havePossession):
    #If we are down by 3 or more, we take the
    #2 quickly.  If we are down by 2 or less
    #We run down the clock first
    if (pointsDown >= 3):
      timeLeft -= timeToShoot2
    else:
      timeLeft = 0

    #Do we make the shot?
    if (np.random.randint(0, high=100) < twoPtPercent):
      pointsDown -= 2
      havePossession = False
  else:
    #Does the opponent team rebound?
    #If so, we lose possession.
    #This doesn't really matter when we run
    #the clock down
    if (np.random.randint(0, high=100) >= offenseReboundPercent):
      havePossession = False
    else:   #cases where we don't have possession
      if (pointsDown > 0):  #foul to get back possession

        #takes time to foul
```

```
        timeLeft -= timeToFoul

        #opponent takes 2 free throws
        if (np.random.randint(0, high=100) < oppFtPercent):
          pointsDown += 1

        if (np.random.randint(0, high=100) < oppFtPercent):
          pointsDown += 1
          havePossession = True
      else:
        if (np.random.randint(0, high=100) >= ftReboundPercent):
          #you were able to rebound the missed ft
          havePossession = True
        else:
          #tied or up so don't want to foul;
          #assume opponent to run out clock and take
          if (np.random.randint(0, high=100) < oppTwoPtPercent):
            pointsDown += 2 #They made the 2
          timeLeft = 0


if (pointsDown > 0):
  return False
else:
  if (pointsDown < 0):
    return True
  else:
    if (np.random.randint(0, high=100) < overtimePercent):
      return True
    else:
      return False
```

为了便于比较，我们会选择五名球员中擅长 3 分或 2 分球，或都擅长的球员，如下面的代码所示：

```
plt.figure(figsize=(14,14))
names=['Lebron James', 'Kyrie Irving', 'Steph Curry',
       'Kyle Krover', 'Dirk Nowitzki']
threePercents = [35.4,46.8,44.3,49.2, 38.0]
twoPercents = [53.6,49.1,52.8, 47.0,48.6]
colind=0

for i in range(5):  # can be run individually as well
  x=[]
  y1=[]
  y2=[]
  trials = 400 #Number of trials to run for simulation
  threePtPercent = threePercents[i] # % chance of making 3-pt shot
  twoPtPercent = twoPercents[i] # % chance of making a 2-pt shot
```

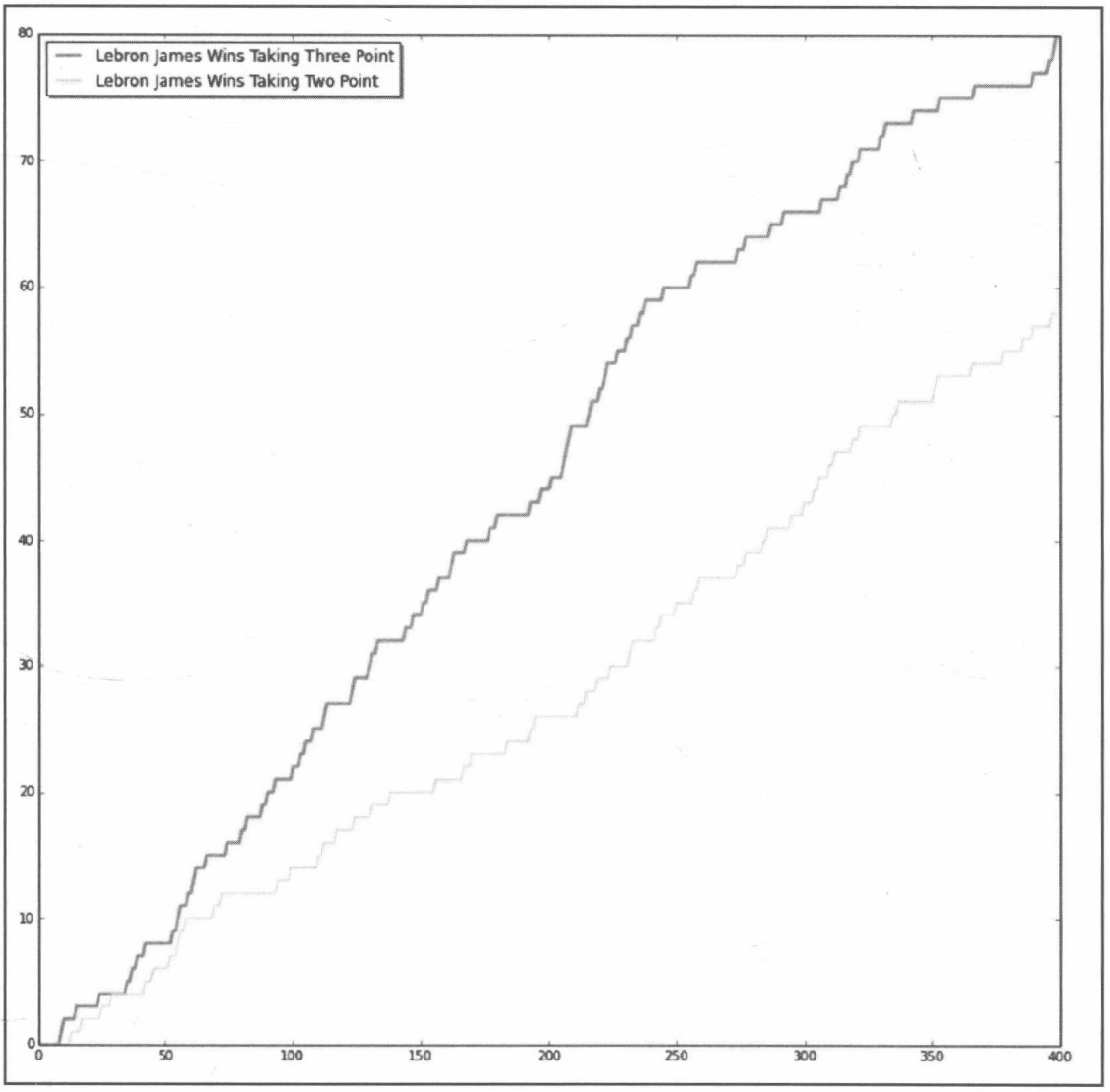
```
    oppTwoPtPercent = 40 #Opponent % chance making 2-pter
    oppFtPercent = 70 #Opponent's FT %
    timeToShoot2 = 5 #How many seconds elapse to shoot a 2
    timeToFoul = 5 #How many seconds elapse to foul opponent
    offenseReboundPercent = 25 #% of regular offense rebound
    ftReboundPercent = 15 #% of offense rebound after missed FT
    overtimePercent = 50 #% chance of winning in overtime

    winsTakingThree = 0
    lossTakingThree = 0
    winsTakingTwo = 0
    lossTakingTwo = 0
    curTrial = 1

    while curTrial < trials:
      #run a trial take the 3
      if (attemptThree()):
        winsTakingThree += 1
      else:
        lossTakingThree += 1
        #run a trial taking a 2
        if attemptTwo() == True :
          winsTakingTwo += 1
        else:
          lossTakingTwo += 1

        x.append(curTrial)
        y1.append(winsTakingThree)
        y2.append(winsTakingTwo)
        curTrial += 1


    plt.plot(x,y1, color=colors[colind], label=names[i]+" Wins Taking
Three Point", linewidth=2)
    plt.plot(x,y2, color=colors[20], label=names[i]+" Wins Taking Two
Point", linewidth=1.2)
    colind += 2

legend = plt.legend(loc='upper left', shadow=True,)
for legobj in legend.legendHandles:
    legobj.set_linewidth(2.6)
plt.show()
```

通过设定步长 1 以及仅包括球员的名字和统计量，我们对单个球员运行程序。在所有的情况下，对于所有球员来说，当对方球队得 2 分的百分比很高（70%）时，蒙特卡罗模拟结果是：建议投 3 分球赢得胜利。让我们来看看单个球员和全部球员放在一起的绘图结果。

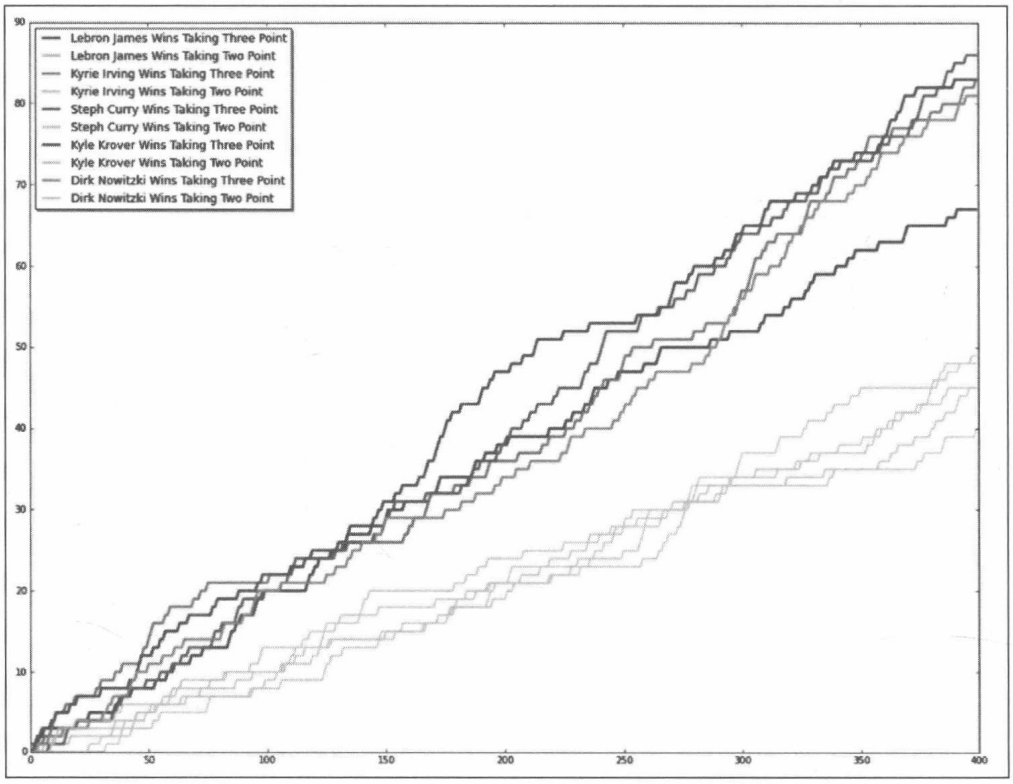
我们已经根据 http://www.basketball-reference.com/teams/ 网站上的最新统计数据，

选出 3 分球命中率相当不错的球员。这里的统计量截止时间为 2015 年 5 月 12 日。在所有的情况下，投 3 分球有更大的获胜机会。如果对方球队平均罚球命中率较低，那么结果将有所不同。

下面两幅图显示了从 NBA 联赛（2015）选出的一名球员和五名球员：



前面的图显示出 Lebron 的 3 分和两个 3 分球尝试结果。下图比较了其他四名球员尝试的结果：



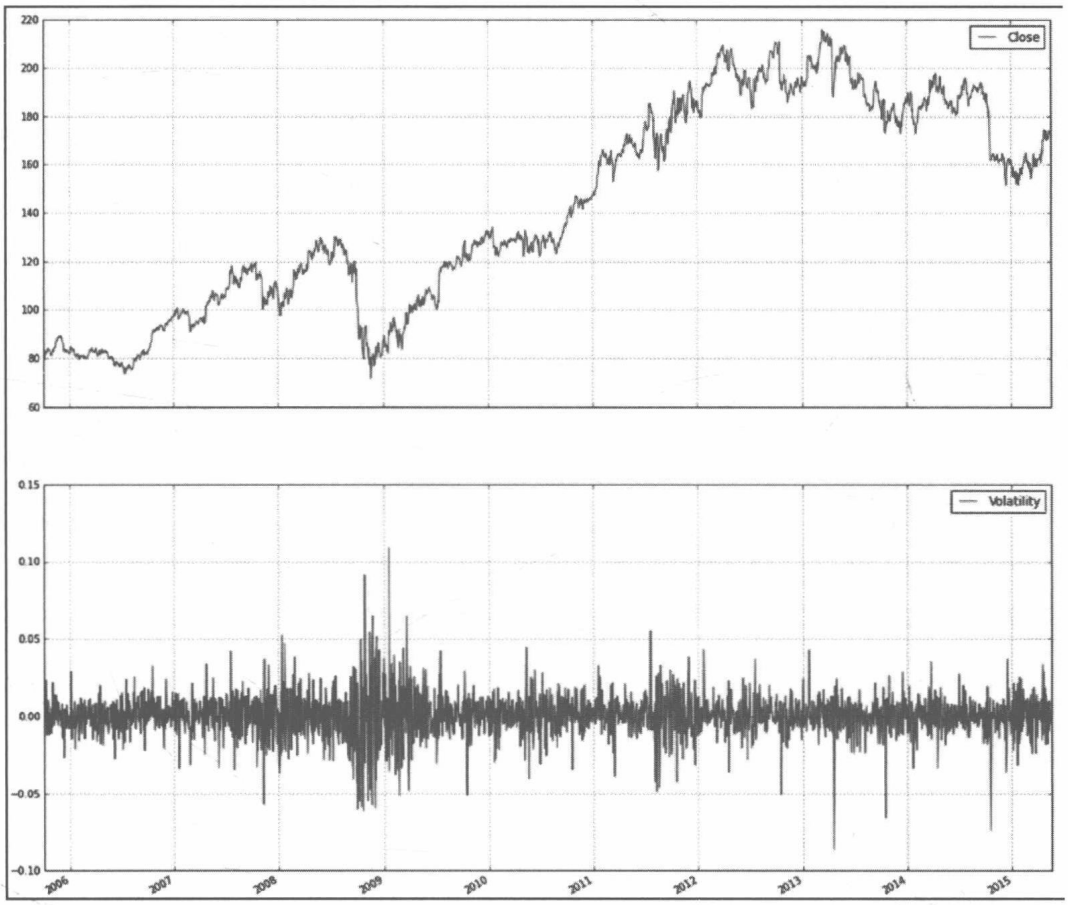
4. 波动图

到目前为止，我们已经看到了许多有用的 Python 软件包。我们已经知道如何使用 matplotlib。但这里，我们将展示如何运用 pandas，通过少数几行代码快速绘制金融图表。标准差是一个统计术语，用于度量平均变异程度，也称之为波动性测度。

根据定义，波动性是实际值与平均值之间的差异。为了用最接近的数值绘制波动性，下面的例子说明了如何从给定的起始日期读图，如何完成一支特定的股票（如 IBM），还有如何用下面的代码观察波动性：

```
import pandas.io.data as stockdata
import numpy as np
r,g,b=(31,  119, 180)
colornow=(r/255.,g/255.,b/255.)
ibmquotes = stockdata.DataReader(name='IBM', data_source='yahoo',
start='2005-10-1')
ibmquotes['Volatility'] = np.log(ibmquotes['Close']/
    ibmquotes['Close'].shift(1))
ibmquotes[['Close', 'Volatility']].plot(figsize=(12,10), \
    subplots=True, color=colornow)
```

下面的图展示了波动图：



现在，我们来看看波动性是如何度量的。波动性度量了价格的变化程度，可以具有多个峰值。此外，指数函数能够显示出随时间变化的增长率；对数函数（指数的逆函数）能够显示出随增长率变化对时间的度量。下面的代码显示出对数波动性的度量：

```
%time
ibmquotes['VolatilityTest'] = 0.0
for I in range(1, len(ibmquotes)):
  ibmquotes['VolatilityTest'] =
    np.log(ibmquotes['Close'][i]/ibmquotes['Close'][i-1])
```

如果我们关注时间，前面代码将会实现：

```
CPU times: user 1e+03 ns, sys: 0 ns, total: 1e+03 ns Wall time: 5.01
µs
```

为了分解并显示如何使用 %time，以及通过收盘价的变化值这一比例分配波动性：

```
%time
ibmquotes['Volatility'] = np.log(ibmquotes['Close']/
ibmquotes['Close'].shift(1))
```

如果我们关注时间，前面代码将会实现：

```
CPU times: user 2 µs, sys: 3 µs, total: 5 µs Wall time: 5.01 µs.
```

结果表明，方差越大，波动性越大。在我们试图绘制执行价格波动的隐含波动率之前，不妨先看看 VSTOXX 数据。这些数据可以从 http://www.stoxx.com 或 http://www.eurexchange.com/advanced-services/ 网站上下载。数据如下所示：

```
              V2TX   V6I1   V6I2   V6I3   V6I4   V6I5   V6I6   V6I7
V6I8
  Date
2015-05-18  21.01  21.01  21.04    NaN  21.12  21.16  21.34  21.75
21.84
2015-05-19  20.05  20.06  20.15  17.95  20.27  20.53  20.83  21.38
21.50
2015-05-20  19.57  19.57  19.82  20.05  20.22  20.40  20.63  21.25
21.44
2015-05-21  19.53  19.49  19.95  20.14  20.39  20.65  20.94  21.38
21.55
2015-05-22  19.63  19.55  20.07  20.31  20.59  20.83  21.09  21.59
21.73
```

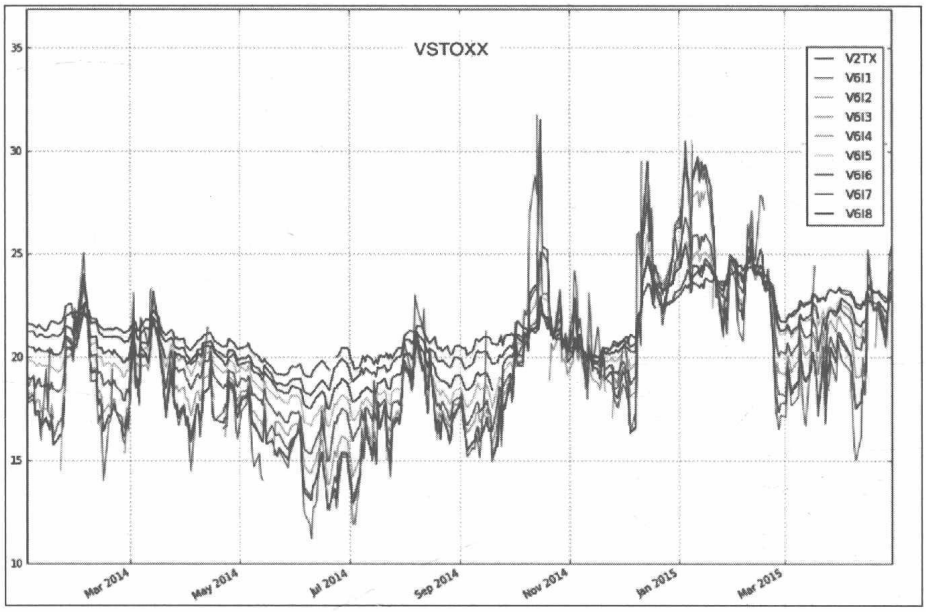
该数据文件包含 Euro Stoxx 波动指数，筛选出 2011 年 12 月 31 日到 2015 年 1 月 1 日的数据并绘图。下面的代码可以用来对 VSTOXX 数据绘图：

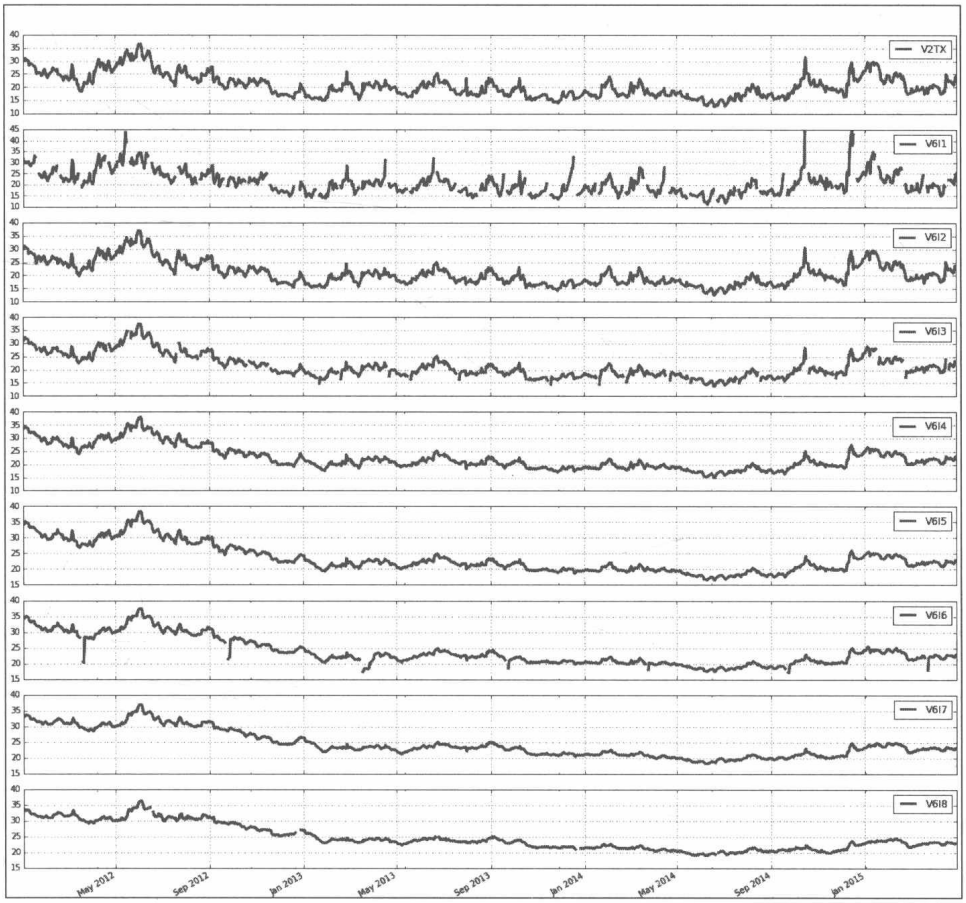
```
import pandas as pd

url = 'http://www.stoxx.com/download/historical_values/h_vstoxx.txt'
vstoxx_index = pd.read_csv(url, index_col=0, header=2,
                           parse_dates=True, dayfirst=True,
                           sep=',')
vstoxx_short = vstoxx_index[('2011/12/31' < vstoxx_index.index)
                            & (vstoxx_index.index < '2015/5/1')]
# to plot all together
vstoxx_short.plot(figsize=(15,14))
```

通过运行上面的代码，可以绘图比较 Euro Stoxx 波动指数。

上图显示出所有指数绘制在一起的情况。但如果要对它们分别绘图，那么可能需要将下面的子图设置为 True：

```
# to plot in subplots separately
vstoxx_short.plot(subplots=True, grid=True, color='r',
     figsize=(20,20), linewidth=2)
```


VSTOXX
V2TX
V6I1
V6I2
V6I3
V6I4
V6I5
V6I6
V6I7
V6I8
35
30
25
20
15
10
Mar 2014
May 2014
Jul 2014
Sep 2014
Nov 2014
Jan 2015
Mar 2015


V2TX
V6I1
V6I2
V6I3
V6I4
V6I5
V6I6
V6I7
V6I8
May 2012
Sep 2012
Jan 2013
May 2013
Sep 2013
Jan 2014
May 2014
Sep 2014
Jan 2015

5. 隐含波动率

布莱克－斯克尔斯－默顿（Black-Scholes-Merton）模型是一个金融市场的统计模型。根据这个模型，我们可以得到欧式期权价格的一种估计。这个公式得到广泛使用，而且许多实证结果表明：布莱克－斯科尔斯的价格“相当接近”观察到的实际价格。在1973年的论文中，费雪·布莱克和迈伦·斯克尔斯在“*The Pricing of Options and Corporate Liabilities*”这个期刊中首次发表该模型。模型背后的主要思想是通过恰当的方式购买和销售潜在资产，避免期权风险。

在这个模型中，在非支付股息的股票的欧式看涨期权的价格如下：

$$C_o = S_o N(d_1) - Xe^{-rT} N(d_2)$$

$$d_1 = \frac{ln\left(\frac{S_o}{X}\right) + \left(r + \frac{\sigma^2}{T}\right)T}{\sigma\sqrt{T}}$$

$$d_2 = \frac{ln\left(\frac{S_o}{X}\right) + \left(r - \frac{\sigma^2}{T}\right)T}{\sigma\sqrt{T}}$$

$N(d)$ 是标准正态分布

这里，

S_0 是股票价格　　T 是满期时间

X 是行使价或成交价　　r 是无风险利率

σ 是对数返回值的标准差（波幅）

对于一个给定的欧式看涨期权 Cg，隐含波动率可以通过上面的方程计算得到（对数收益率的标准差）。有关波动性的期权定价公式的偏导数称为 Vega。这是一个数字，表示波动性每变动正1%，期权价格将变化的方向和范围，如下面的方程所示：

$$Vega = \frac{\partial C_o}{\partial \sigma} = S_o N'(d_1)\sqrt{T}$$

波动率模型（例如 BSM 模型）用于预测波动率、金融领域模型的价值以及预测未来的收入。这样的预测主要用于风险管理和规避、择时交易、投资组合选择，以及其他金融活动。美式看涨期权或看跌期权给你提供任何时间行驶练习的权力，但是欧式看涨期权或看跌期权只能在到期日才可以行使权力。

Black–Scholes–Merton (BSM) 没有显示解，但运用牛顿迭代法（也称为牛顿－拉夫逊方法）可以通过迭代得到近似值。无论何时用到迭代法，都会有一定数量的阈值用以确定迭代

的最终条件。让我们来看看 Python 代码如何通过迭代（Newton 的方法）找到答案并绘图如下：

$$\frac{\partial C(\sigma_n)}{\partial \sigma_n} = -\left(\frac{C_{n+1} - C^*}{\sigma_{n+1} - \sigma_n}\right)$$

$$\Rightarrow \sigma_{n+1} - \sigma_n = -\left(\frac{C_{n+1} - C^*}{\frac{\partial C(\sigma_n)}{\partial \sigma_n}}\right)$$

$$\Rightarrow \sigma_{n+1} = \sigma_n - \left(\frac{C_{n+1} - C^*}{vega}\right)$$

```
from math import log, sqrt, exp
from scipy import stats
import pandas as pd
import matplotlib.pyplot as plt

colors = [(31, 119, 180), (174, 199, 232), (255,128,0),
 (255, 15, 14), (44, 160, 44), (152, 223, 138), (214, 39, 40),
 (255, 152, 150),(148, 103, 189), (197, 176, 213), (140, 86, 75),
(196, 156, 148),(227, 119, 194), (247, 182, 210), (127, 127, 127),
(199, 199, 199),(188, 189, 34), (219, 219, 141), (23, 190, 207),
(158, 218, 229)]

# Scale the RGB values to the [0, 1] range, which is the format
matplotlib accepts.
for i in range(len(colors)):
  r, g, b = colors[i]
  colors[i] = (r / 255., g / 255., b / 255.)

def black_scholes_merton(S, r, sigma, X, T):
  S = float(S) # convert to float
  logsoverx = log (S/X)
  halfsigmasquare = 0.5 * sigma ** 2
  sigmasqrtT = sigma * sqrt(T)

  d1 = logsoverx + ((r + halfsigmasquare) * T) / sigmasqrtT
  d2 = logsoverx + ((r - halfsigmasquare) * T) / sigmasqrtT

  # stats.norm.cdf -> cumulative distribution function
  value = (S * stats.norm.cdf(d1, 0.0, 1.0) -
     X * exp(-r * T) *   stats.norm.cdf(d2, 0.0, 1.0))

  return value

def vega(S, r, sigma, X, T):

  S = float(S)
```

```
    logsoverx = log (S/X)
    halfsigmasquare = 0.5 * sigma ** 2
    sigmasqrtT = sigma * sqrt(T)
    d1 = logsoverx + ((r + halfsigmasquare) * T) / sigmasqrtT
    vega = S * stats.norm.cdf(d1, 0.0, 1.0) * sqrt(T)

    return vega

def impliedVolatility(S, r, sigma_est, X, T, Cstar, it):

    for i in range(it):
      numer = (black_scholes_merton(S, r, sigma_est, X, T) - Cstar)
      denom = vega(S,r, sigma_est, X, T)
      sigma_est -= numer/denom

    return sigma_est
```

这些准备好使用的函数。可以用到单个文件中并完成导入，或者嵌入到代码中仅运行一次。输入文件从 stoxx.com 网站上的一个名为 vstoxx_data.h5 的文件获得，代码如下：

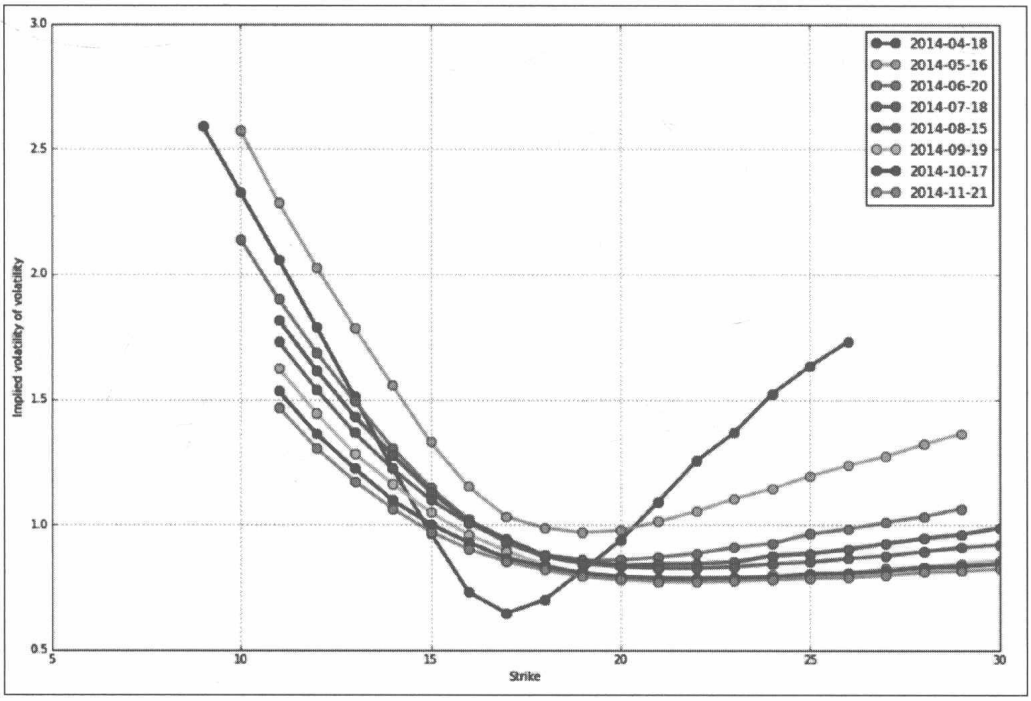
```
h5 = pd.HDFStore('myData/vstoxx_data_31032014.h5', 'r')

futures_data = h5['futures_data'] # VSTOXX futures data
options_data = h5['options_data'] # VSTOXX call option data

h5.close()

options_data['IMP_VOL'] = 0.0

V0 = 17.6639  # the closing value of the index
r=0.04        # risk free interest rate
sigma_est=2
tol = 0.5     # tolerance level for moneyness
```

现在，让我们用 options_data 和 futures_data 数值形式运行迭代运算：

```
for option in options_data.index:
  # iterating over all option quotes
  futureval = futures_data[futures_data['MATURITY'] ==
      options_data.loc[option]['MATURITY']]['PRICE'].values[0]

  # picking the right futures value
  if (futureval * (1 - tol) < options_data.loc[option]['STRIKE']
    < futureval * (1 + tol)):
    impliedVol = impliedVolatility(V0,r,sigma_est,
            options_data.loc[option]['STRIKE'],
            options_data.loc[option]['TTM'],
            options_data.loc[option]['PRICE'],  #Cn
            it=100)                             #iterations
    options_data['IMP_VOL'].loc[option] = impliedVol

plot_data = options_data[options_data['IMP_VOL'] > 0]
```

```
maturities = sorted(set(options_data['MATURITY']))

plt.figure(figsize=(15, 10))

i=0
for maturity in maturities:

  data = plot_data[options_data.MATURITY == maturity]

  # select data for this maturity
  plot_args = {'lw':3, 'markersize': 9}
  plt.plot(data['STRIKE'], data['IMP_VOL'], label=maturity.date(),
       marker='o', color=colors[i], **plot_args)
  i += 1

plt.grid(True)

plt.xlabel('Strike rate $X$', fontsize=18)
plt.ylabel(r'Implied volatility of $\sigma$', fontsize=18)
plt.title('Short Maturity Window (Volatility Smile)', fontsize=22)

plt.legend()
plt.show()
```

下图是上述程序的运行结果，展示出从 http://vstoxx.com 网站上下载数据的隐含波动率与执行利率的关系。我们也可以从 http://knapdata.com/python/vstoxx_data_31032014.h5 网站上下载数据。下图显示了 Euro Stoxx 中隐含波动率与执行利率的关系：



5.2.2 投资组合估值

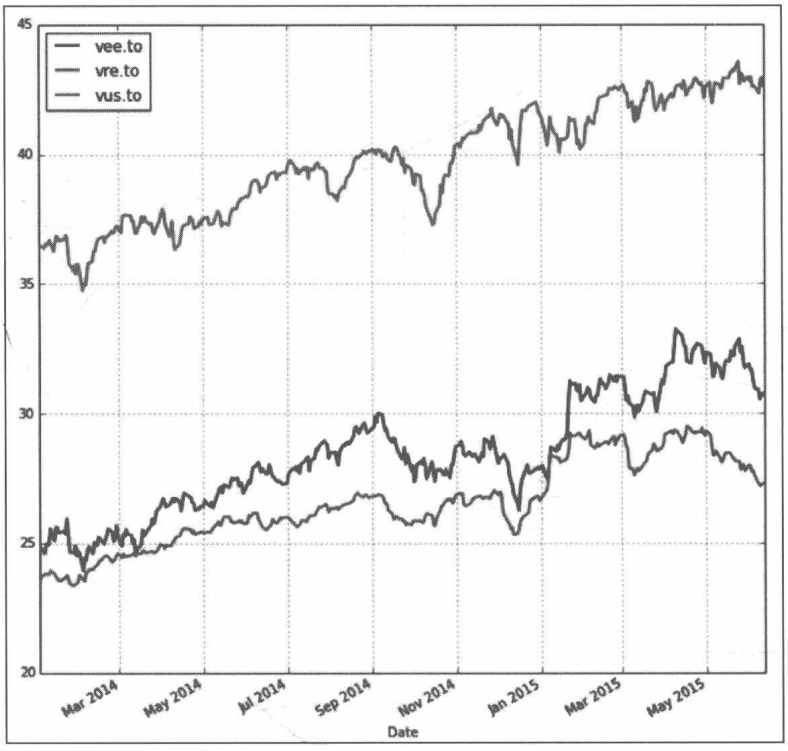
实体的投资组合估值一般是指估计其当前价值。估值通常在金融资产或金融负债价值评估中使用，其中包括股票、期权、经营企业或无形资产。为了理解估值的目的以及可视化方法，我们将选择共同基金，进行绘图、比较并找出他们的相关性。

假设我们以一个单位的货币来进行投资组合的估值。这样显著简化了投资组合价值的聚合。

我们从 Vanguard（基金公司）中选择三支基金，如 Vanguard US Total (vus.to)、Vanguard Canadian Capped (vre.to) 和 Vanguard Emerging Markets (vee.to)。下面的代码比较了这 3 个基金。

```
import pandas as pd   #gets numpy as pd.np
from pandas.io.data import get_data_yahoo
import matplotlib.pyplot as plt

# get data
data = get_data_yahoo(["vus.to","vre.to","vee.to"],
  start = '2014-01-01')['Adj Close']

data.plot(figsize=(10,10), lw=2)
plt.show()
```

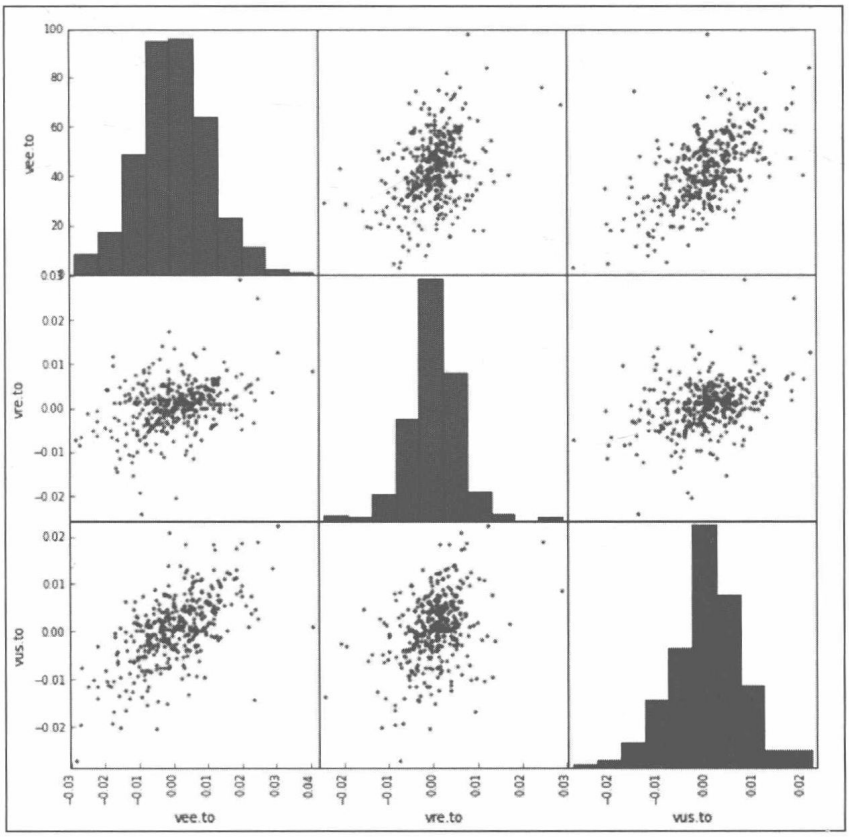
我们也可以使用 get_data_yahoo() 从 pandas.io.data 中获得数据，如下图所示：



除了绘图，为了尺度化价格，我们需要取价格的对数，从而得到对数价格的相关系数矩阵，代码如下所示：

```
#convert prices to log returns
retn=data.apply(pd.np.log).diff()

# make corr matrix
retn.corr()

#make scatterplot to show correlation
pd.tools.plotting.scatter_matrix(retn, figsize=(10,10))
plt.show()

# some more stats
retn.skew()
retn.kurt()
# Output
vee.to     0.533157
vre.to     3.717143
vus.to     0.906644
dtype: float64
```

相关系数图如下图所示。首先使用 skew() 和 kurt()，再使用 pandas 中的 scatter_matrix 功能：



5.2.3 模拟模型

模型呈现了构建过程和系统功能。与它所代表的系统相似，模型更容易理解。一个系统的模拟是一个系统的工作模型。该模型通常是可重构的，而且可以进行多次实验。该模型的运行对研究模型非常有用。在系统存在之前，模拟有助于减少失败的可能性以满足要求。

什么时候特定系统适合模拟模型？一般情况下，每当系统中有建模和分析随机性的需求时，模拟就是合适的工具。

5.2.4 几何布朗运动模拟

布朗运动是一种随机游动，广泛用于构建物理过程，如扩散和生物过程、社会和金融过程（如股票市场的波动）。

布朗运动是一种复杂的方法。这是基于一个植物的过程，由 R. 布朗在 1827 年发现，并且被广泛应用，如图像建模噪音、产生分形、晶体增长和股市模拟。考虑到内容的相关性，我们将选择股市模拟作为案例。

M.F.M 奥斯本研究了普通股的价格对数和金钱的价值，表明他们在统计均衡中的共同影响。以随机的方式进行统计和股票的选择，能够获得非常类似于布朗运动的分布函数。

几何布朗运动的定义：

随机过程（S_t）遵循几何布朗运动，并且满足下列随机微分方程：

$$dS_t = uS_t dt + \sigma S_t dW_t$$

$$\frac{dS_t}{S_t} = udt + \sigma dW_t$$

对等式两边进行积分，设定初始状态 $S_t = S_o$，可以求解得到：

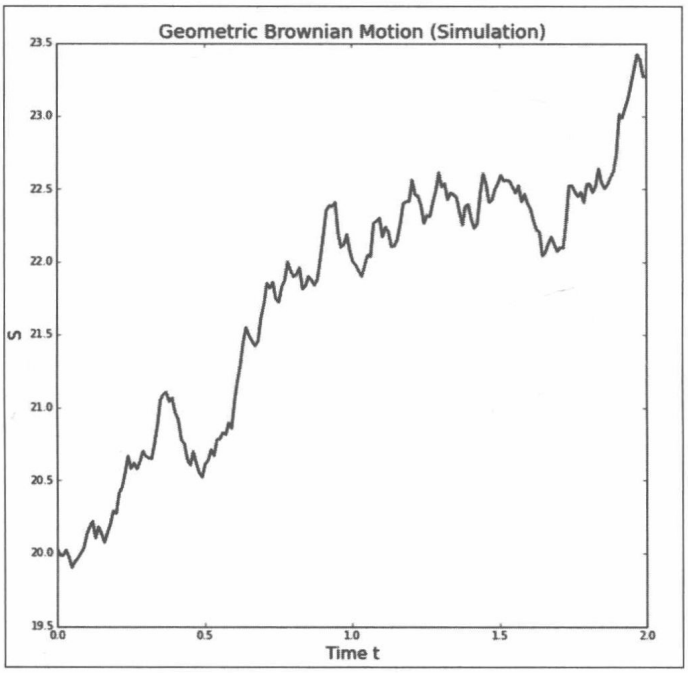
$$S_t = S_o exp^{\left(\left(u-\frac{\sigma^2}{2}\right)t+\sigma W_t\right)}$$

根据前面的推导，我们使用插值法得到布朗运动的解：

```
import matplotlib.pyplot as plt
import numpy as np

'''
Geometric Brownian Motion with drift!
u=drift factor
sigma: volatility
T: time span
dt: length of steps
S0: Stock Price in t=0
W: Brownian Motion with Drift N[0,1]
'''
```

```
rect = [0.1, 5.0, 0.1, 0.1]
fig = plt.figure(figsize=(10,10))

T = 2
mu = 0.1
sigma = 0.04
S0 = 20
dt = 0.01
N = round(T/dt)
t = np.linspace(0, T, N)

# Standare normal distrib
W = np.random.standard_normal(size = N)
W = np.cumsum(W)*np.sqrt(dt)

X = (mu-0.5*sigma**2)*t + sigma*W

#Brownian Motion
S = S0*np.exp(X)

plt.plot(t, S, lw=2)
plt.xlabel("Time t", fontsize=16)
plt.ylabel("S", fontsize=16)
plt.title("Geometric Brownian Motion (Simulation)",
  fontsize=18)
plt.show()
```

布朗运动模拟的结果如下图所示：



利用布朗运动的模拟股票价格如下面的代码所示：

```
import pylab, random

class Stock(object):
    def __init__(self, price, distribution):
        self.price = price
        self.history = [price]
        self.distribution = distribution
        self.lastChange = 0
    def setPrice(self, price):
        self.price = price
        self.history.append(price)

    def getPrice(self):
        return self.price

    def walkIt(self, marketBias, mo):
        oldPrice = self.price
        baseMove = self.distribution() + marketBias
        self.price = self.price * (1.0 + baseMove)
        if mo:
            self.price = self.price + random.gauss(.5, .5)*self.
lastChange
        if self.price < 0.01:
            self.price = 0.0
        self.history.append(self.price)
        self.lastChange = oldPrice - self.price

    def plotIt(self, figNum):
        pylab.figure(figNum)
        pylab.plot(self.history)
        pylab.title('Closing Price Simulation Run-' + str(figNum))
        pylab.xlabel('Day')
        pylab.ylabel('Price')


def testStockSimulation():
    def runSimulation(stocks, fig, mo):
        mean = 0.0
        for s in stocks:
            for d in range(numDays):
                s.walkIt(bias, mo)
            s.plotIt(fig)
            mean += s.getPrice()
        mean = mean/float(numStocks)
        pylab.axhline(mean)
    pylab.figure(figsize=(12,12))
    numStocks = 20
    numDays = 400
    stocks = []
    bias = 0.0
    mo = False
    startvalues = [100,500,200,300,100,100,100,200,200, 300,300,400,50
```

```
0,00,300,100,100,100,200,200,300]
    for i in range(numStocks):
        volatility = random.uniform(0,0.2)
        d1 = lambda: random.uniform(-volatility, volatility)
        stocks.append(Stock(startvalues[i], d1))
    runSimulation(stocks, 1, mo)

testStockSimulation()
pylab.show()
```

利用服从均匀分布的随机数模拟收盘价的结果如下图所示：

5.2.5 基于扩散模拟

随机模型提供了反应扩散过程的更具体的理解。这样一种描述通常对生物系统建模很有必要。现在已经有很多研究过的模拟模型，就本章而言，我们将考虑平方根扩散模型。

考克斯（Cox）、英格索尔（Ingersoll）和罗斯（Ross）（1985 年）将平方根扩散模型推广到金融领域，用于构建均值回归量（如利率和波动率）。这个过程的随机微分方程如下：

$$dx_t = \underbrace{k(\theta - x_t)dt}_{\text{漂移量}} + \underbrace{\sigma\sqrt{x_t}dW_t}_{\text{扩散}}$$

其中，x_t 为卡方分布，但在离散情况下，可用正态分布近似，即使用欧拉数值方法通过迭代法近似计算，如下述等式所示：

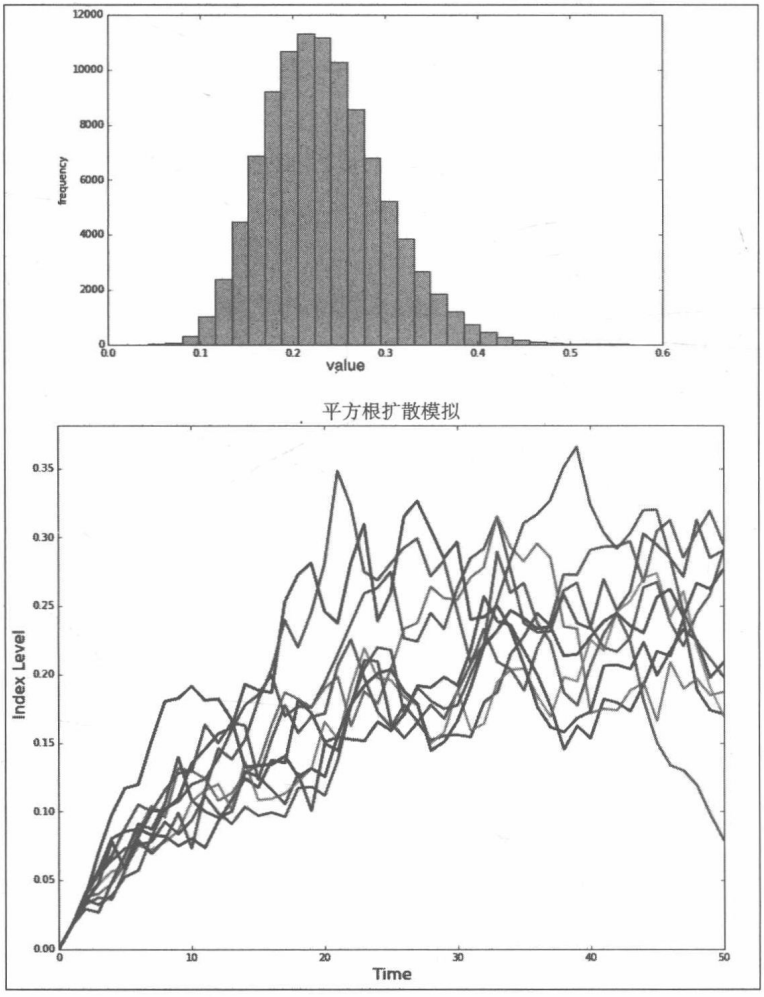
$$x_t^{新} = x_s^{新} + k\left(\theta - x_s^+\right)\Delta t + \sigma\sqrt{x_s^+ \Delta t}\, w_t$$

$$x_t = x_t^+$$

$$这里，x_s^+ = \max\left(x_s, 0\right) \quad , \quad x_t^+ = \max\left(x_t, 0\right)$$

```
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as npr

S0 = 100 # initial value
r = 0.05
sigma = 0.25
T = 2.0

x0=0
k=1.8
theta=0.24
i = 100000
M = 50
dt = T / M
def srd_euler():
  xh = np.zeros((M + 1, i))
  x1 = np.zeros_like(xh)
  xh[0] = x0
  x1[0] = x0
  for t in range(1, M + 1):
   xh[t] = (xh[t - 1]
    + k * (theta - np.maximum(xh[t - 1], 0)) * dt
    + sigma * np.sqrt(np.maximum(xh[t - 1], 0)) * np.sqrt(dt)
    * npr.standard_normal(i))
  x1 = np.maximum(xh, 0)
  return x1
  x1 = srd_euler()

  plt.figure(figsize=(10,6))
  plt.hist(x1[-1], bins=30, color='#98DE2f', alpha=0.85)
  plt.xlabel('value')
  plt.ylabel('frequency')
  plt.grid(False)

  plt.figure(figsize=(12,10))
  plt.plot(x1[:, :10], lw=2.2)
  plt.title("Square-Root Diffusion - Simulation")
  plt.xlabel('Time', fontsize=16)
```

```
plt.ylabel('Index Level', fontsize=16)
#plt.grid(True)
plt.show()
```



5.3　阈值模型

阈值模型通过设定阈值来区分不同取值范围，通过以某种重要的模型收敛方式完成预测。Schelling 尝试对一种动态分离现象进行建模，这主要源自由两种模拟模型构建的个体间相互作用。

Schelling 分离模型

Schelling 分离模型由 Thomas C. Schelling 建立。该模型是首批有自组织功能的系统构

建模型。

Schelling是一种在棋盘上放置硬币和分铸币，并根据各种规则进行移动的尝试。实验中，他将棋盘比作一个城市，棋盘方格比作住所，方格数比作邻居。硬币和分铸币（视觉效果不同）可以表示两组吸烟者、非吸烟者、男性、女性、高管、非高管、学生或老师。

模拟规则的终止条件是：没有人移出当前的位置，因为他们很幸福，这意味着只有不幸福的人才可能搬离。

Schelling模型用来模拟教室分座。模型展示了即便不想成为邻座的同学的分离模式。

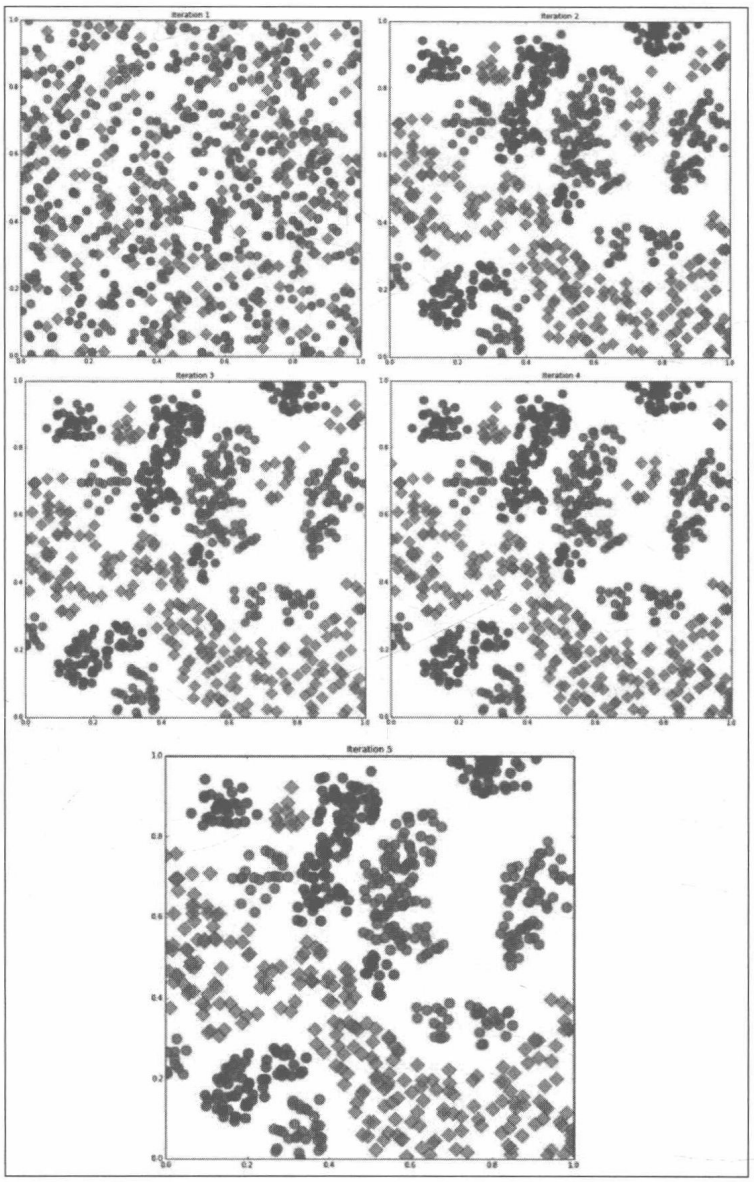
试想我们有三种类型的学生分类，对他们的三种优势——体育、优秀的学术水平和合格——分别赋值为0、1和2。

这里为了演示，我们将假定每类有250名高中生。每个代替物代表一个学生。这些代替物占据一个方格（可视化为高中建筑）。每个代替物的位置用(x, y)表示，这里$0 < x, y < 1$。如果一名学生周围的（欧式距离最近）12位同学中有超过一半跟她是同一类型的，那么这名学生是幸福的。每位同学的初始位置相互独立，服从二元均匀分布，相关代码如下所示：

```
from random import uniform, seed
from math import sqrt
import matplotlib.pyplot as plt

num = 250             # These many agents of a particular type
numNeighbors = 12     # Number of agents regarded as neighbors
requireSameType = 8   # At least this many neighbors to be same type

seed(10)  # for reproducible random numbers

class StudentAgent:

    def __init__(self, type):
        #Students of different type will be shown in colors
        self.type = type
        self.show_position()

    def show_position(self):
        # position changed by using uniform(x,y)
        self.position = uniform(0, 1), uniform(0, 1)

    def get_distance(self, other):
        #returns euclidean distance between self and other agent.
        a = (self.position[0] - other.position[0])**2
        b = (self.position[1] - other.position[1])**2
        return sqrt(a + b)

    def happy(self, agents):
```

```
        "returns True if reqd number of neighbors are the same type."
        distances = []

        for agent in agents:
            if self != agent:
                distance = self.get_distance(agent)
                distances.append((distance, agent))
        distances.sort()
        neighbors = [agent for d, agent in distances[:numNeighbors]]
        numSameType = sum(self.type == agent.type
            for agent in neighbors)
        return numSameType >= requireSameType

    def update(self, agents):
        "If not happy, randomly choose new positions until happy."
        while not self.happy(agents):
            self.show_position()


def plot_distribution(agents, cycle_num):

    x1,y1 = [],[]
    x2,y2 = [],[]
    x3,y3 = [],[]

    for agent in agents:
        x, y = agent.position
        if agent.type == 0:
            x1.append(x); y1.append(y)
        elif agent.type == 1:
            x2.append(x); y2.append(y)
        else:
            x3.append(x); y3.append(y)

    fig, ax = plt.subplots(figsize=(10,10))
    plot_args = {'markersize' : 8, 'alpha' : 0.65, 'markersize': 14}
    ax.set_axis_bgcolor('#ffffff')
    ax.plot(x1, y1, 'o', markerfacecolor='#1b62a5', **plot_args)
    ax.plot(x2, y2, 'o', markerfacecolor='#279321', **plot_args)
    ax.plot(x3, y3, 'D', markerfacecolor='#fd6610', **plot_args)
    ax.set_title('Iteration {}'.format(cycle_num))
    plt.show()

agents = [StudentAgent(0) for i in range(num)]
agents.extend(StudentAgent(1) for i in range(num))
agents.extend(StudentAgent(2) for i in range(num))
count = 1
terminate=False
while terminate == False:
    plot_distribution(agents, count)
```

```
        count += 1
        no_one_moved = True
    for agent in agents:
        old_position = agent.position
        agent.update(agents)
        if agent.position != old_position:
            no_one_moved = False
    if no_one_moved:
        terminate=True
```



5.4 统计与机器学习综述

人工智能（AI）不是一个新领域。如果我们还记得 30 年前研究 AI 的时候，除了机器人外，我们对这一领域未来的发展趋势知之甚少。现在，特别是在过去 10 年，我们对研究人工智能和机器学习的热情和兴趣得到极大的提高。从广义上讲，这些领域的目的是“发现和了解一些有用的东西”。所收集的信息有利于发现新算法并提出新问题，如“如何处理高维数据以及解决不确定性”？

机器学习的目的是生成对于人类理解足够简单的分类表达形式。它们必须充分模仿人类推理能力，在决策过程中提供见解。与统计方法类似，在开发阶段可以使用背景知识。统计学习在许多科学领域起着重要作用，学习科学在统计学领域、数据挖掘和人工智能也扮演着重要角色，这与工程等其他学科领域也相互交叉。

统计和机器学习之间的区别在于：统计学强调推断，而机器学习强调预测。运用统计的一般方法是通过生成数据进行统计推断。机器学习是通过某些变量预测数据的某些特征。统计学习和机器学习有很多重合的部分，一些专家经常讨论哪种方法更好。在后面的章节中，我们会介绍有关机器学习的例子。下面列出了一些算法：

- 回归或预测
- 线性和二次判别分析
- 分类
- 最近邻
- 朴素贝叶斯
- 支持向量机
- 决策树
- 聚类

机器学习算法广义上主要分为监督学习、非监督学习、强化学习和深度学习。监督学习方法就是将数据进行分类并标记为测试集数据，就像一个老师，对课堂进行监管。无监督的学习没有任何标记的训练集数据，而监督学习包含完全标记的训练集数据。半监督学习处于监督和无监督学习之间。它使用未标记的数据作为训练集。

由于本书的内容是数据可视化，因此，我们只在后面的章节中讨论一些算法。

5.4.1 k- 最近邻算法

第一个机器学习算法是 k- 最近邻（k-NN）算法。k- 最近邻算法不用从训练集数据中建立模型。它逐一比较无标签数据和每一个有标签数据。然后，取最相似的数据部分（最近的邻居），并查看其标签。现在，从已知的数据集中取前 k 条最相似的数据（k 为整数，并且通

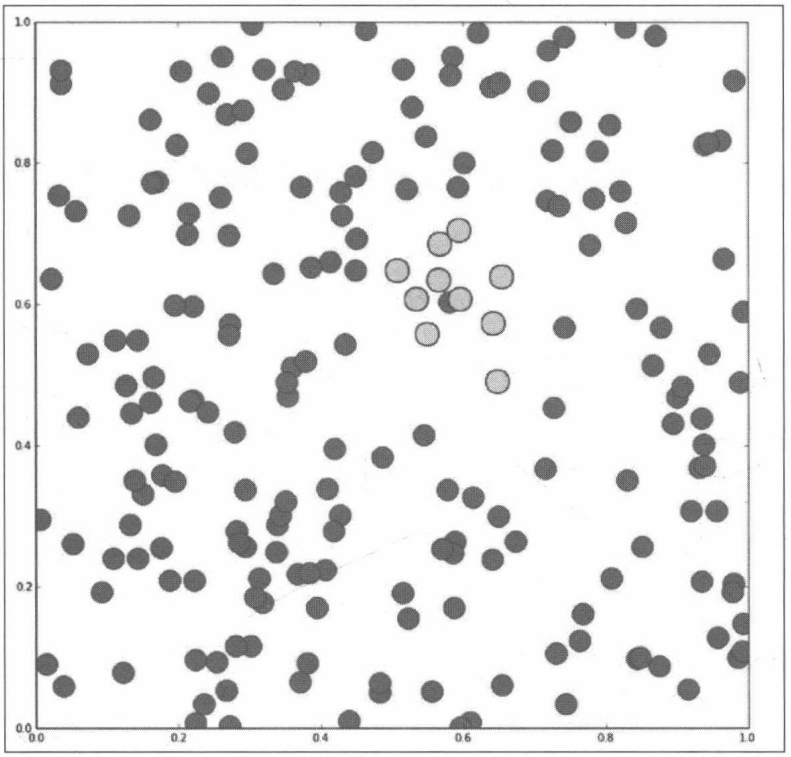
常小于 20）。下面的代码显示如何绘制 k- 最近邻图：

```
from numpy import random,argsort,sqrt
from pylab import plot,show
import matplotlib.pyplot as plt

def knn_search(x, data, K):

  """ k nearest neighbors """

  ndata = data.shape[1]
  K = K if K < ndata else ndata
  # euclidean distances from the other points
  sqd = sqrt(((data - x[:,:ndata])**2).sum(axis=0))
  idx = argsort(sqd) # sorting
  # return the indexes of K nearest neighbors
  return idx[:K]

data = random.rand(2,200) # random dataset
x = random.rand(2,1) # query point

neig_idx = knn_search(x,data,10)

plt.figure(figsize=(12,12))

# plotting the data and the input point
plot(data[0,:],data[1,:],'o',  x[0,0],x[1,0],'o', color='#9a88a1',
   markersize=20)

# highlighting the neighbors
plot(data[0,neig_idx],data[1,neig_idx],'o',
  markerfacecolor='#BBE4B4',markersize=22,markeredgewidth=1)

show()
```

使用 k- 最近邻步骤如下：

- ❑ 收集数据
- ❑ 准备计算距离的数值
- ❑ 用任何适当的方法进行分析
- ❑ 没有训练（不涉及训练）
- ❑ 测试并计算误差率
- ❑ 计算 k 最近邻搜索、确定前 *k* 个最近邻

为了测试分类结果，你可以从已知数据开始，这样可以隐藏分类结果，并测试自己的分类预测结果是否正确。



5.4.2　广义线性模型

回归是估计变量之间的关系的统计过程。更具体地说，回归有助于了解当自变量变化时因变量的变化。

线性回归是最传统的回归模型，可以用于插值，但它不适合预测分析。这种回归对异常值和相关性十分敏感。

贝叶斯回归是一种加罚估计量，比传统线性回归更加灵活和稳健。贝叶斯回归假设建模者具备回归系数的一些先验知识，并在贝叶斯推断下进行统计分析。

我们将讨论一些因变量（y）和自变量（x_1，x_2，…，x_n）关系的模型。符号如下：

$\hat{y}$的预测值如下：

$$\hat{y}(w,x) = w_o + w_1x_1 + w_2x_2 + \cdots + w_nx_n = w_o + \sum_{i=0}^{n} w_ix_i$$

现在，让我们来看看贝叶斯线性回归模型。有人会问：“为什么要使用贝叶斯？”答案如下：

- 贝叶斯模型更加灵活
- 贝叶斯模型在小样本的情况下更精确（可能依赖于先验）

❑ 贝叶斯模型能够结合先验信息

贝叶斯线性回归

首先，让我们来看看线性回归的图模型。在这个模型中，比如我们得到的数据为D=$((x_1, y_1),\cdots,(x_n, y_n))$。我们的目标是对这个数据构建一种模型，方程如下所示：

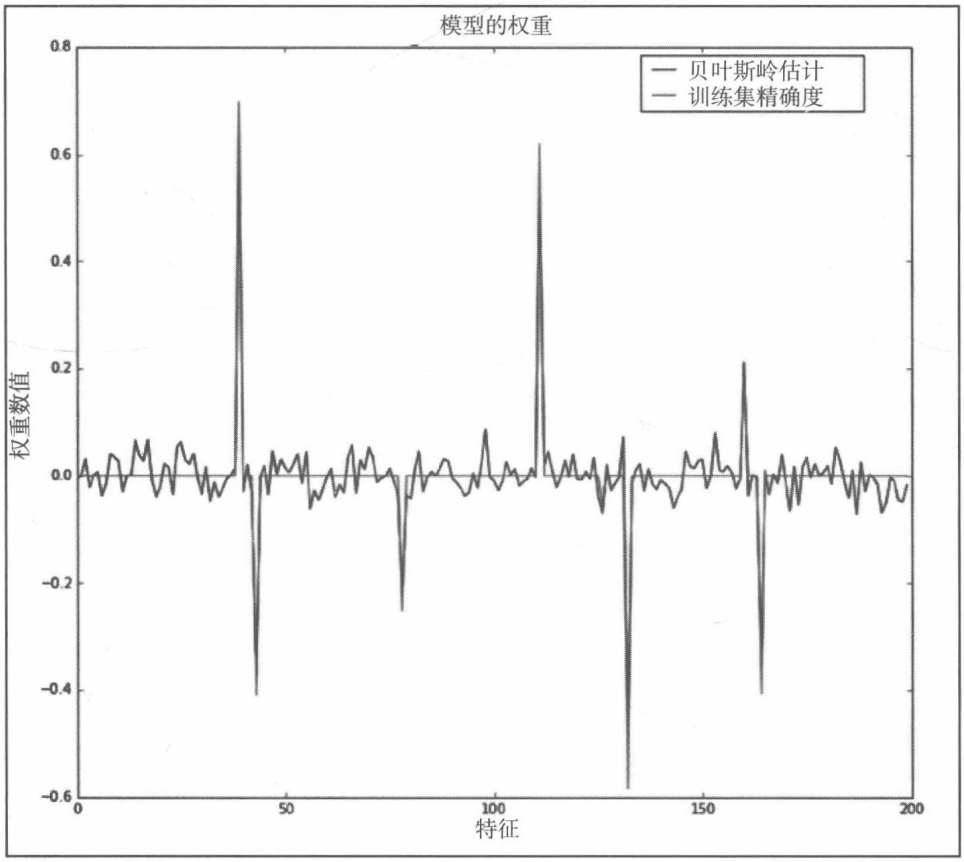
$$f(x)=w^{\mathrm{T}}\phi(x)$$
$$w\sim N(0,\sigma_0^2 I)$$
$$Y_i\sim N(w^{\mathrm{T}}\phi(x_i),\sigma^2)$$

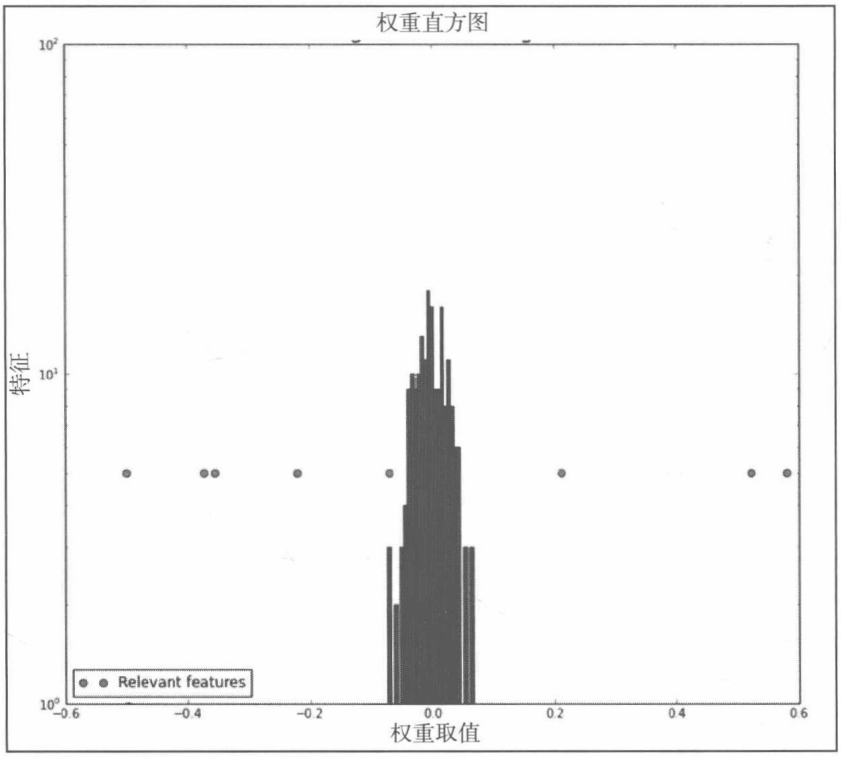
这里，w 为向量，表示权重。每个 Y_i 在新变量 x 已知的条件下服从正态分布。Y_i 是随机变量，由 $Y_i=y_i$ 我们可以用新变量 x 预测相应的 y，代码如下所示：

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

from sklearn.linear_model import BayesianRidge
from sklearn.linear_model import LinearRegression

np.random.seed(0)
n_samples, n_features = 200, 200

X = np.random.randn(n_samples, n_features)  # Gaussian data
# Create weights with a precision of 4.
theta = 4.
w = np.zeros(n_features)

# Only keep 8 weights of interest
relevant_features = np.random.randint(0, n_features, 8)
for i in relevant_features:
    w[i] = stats.norm.rvs(loc=0, scale=1. / np.sqrt(theta))

alpha_ = 50.
noise = stats.norm.rvs(loc=0, scale=1. / np.sqrt(alpha_), size=n_
samples)
y = np.dot(X, w) + noise

# Fit the Bayesian Ridge Regression
clf = BayesianRidge(compute_score=True)
clf.fit(X, y)

# Plot weights and estimated and histogram of weights
plt.figure(figsize=(11,10))
plt.title("Weights of the model", fontsize=18)
plt.plot(clf.coef_, 'b-', label="Bayesian Ridge estimate")
```

```
plt.plot(w, 'g-', label="Training Set Accuracy")
plt.xlabel("Features", fontsize=16)
plt.ylabel("Values of the weights", fontsize=16)
plt.legend(loc="best", prop=dict(size=12))
plt.figure(figsize=(11,10))
plt.title("Histogram of the weights", fontsize=18)
plt.hist(clf.coef_, bins=n_features, log=True)
plt.plot(clf.coef_[relevant_features], 5 * np.ones(len(relevant_
features)),
         'ro', label="Relevant features")
plt.ylabel("Features", fontsize=16)
plt.xlabel("Values of the weights", fontsize=16)
plt.legend(loc="lower left")
plt.show()
```

下面两个图是程序运行的结果：





5.5 创建动画和交互图

现在有一些可供选择的绘制交互式图形的工具，比如 Bokeh、Plotly 和 VisPy。

Bokeh 允许你通过 JavaScript 平台绘制 matplotlib 对象，这让交互部分的绘图更加容易。例如，Bokeh 可用于绘制交互型的地图。在 Bokeh 中，可以使用 JavaScript 绘制 D3.js 图形，同时通过 Web 浏览器实现可视化。Bokeh 处理大型数据时具有十分出色的表现。通过 conda 或 pip 安装 bokeh 十分方便，代码如下所示：

```
conda install bokeh
   OR
pip install bokeh
```

```
    import collections
from bokeh.sampledata import us_counties, unemployment
from bokeh.plotting import figure, show, output_file
from bokeh.models import HoverTool

county_coordinate_xs=[
us_counties.data[code]['lons'] for code in us_counties.data
if us_counties.data[code]['state'] == 'ca'
]
```

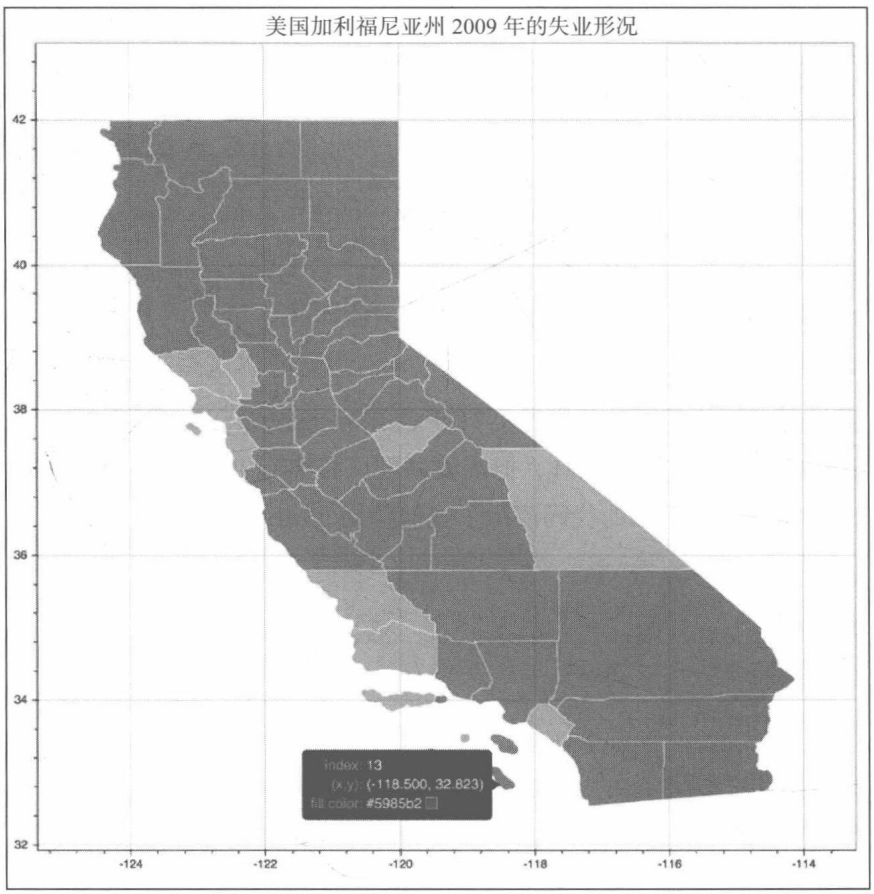
```
county_coordinate_ys=[
us_counties.data[code]['lats'] for code in us_counties.data
if us_counties.data[code]['state'] == 'ca'
]

colors = ["#e6f2ff", "#cce5ff", "#99cbff", "#b2d8ff", "#73abe5",
"#5985b2"]
county_colors = []
for county_id in us_counties.data:
  if us_counties.data[county_id]['state'] != 'ca':
    continue
  try:
    rate = unemployment.data[county_id]
    idx = min(int(rate/2), 5)
    county_colors.append(colors[idx])
  except KeyError:
    county_colors.append("black")

output_file("california.html", title="california.py example")

TOOLS="pan,wheel_zoom,box_zoom,reset,hover,save"
p = figure(title="California Unemployment 2009", width=1000,
height=1000, tools=TOOLS)

p.patches(county_coordinate_xs, county_coordinate_ys,
fill_color=county_colors, fill_alpha=0.7,
line_color="white", line_width=0.5)

mouse_hover = p.select(dict(type=HoverTool))
mouse_hover.point_policy = "follow_mouse"
mouse_hover.tooltips = collections.OrderedDict([
("index", "$index"), ("(x,y)", "($x, $y)"),
("fill color", "$color[hex, swatch]:fill_color"),
])
show(p)
```

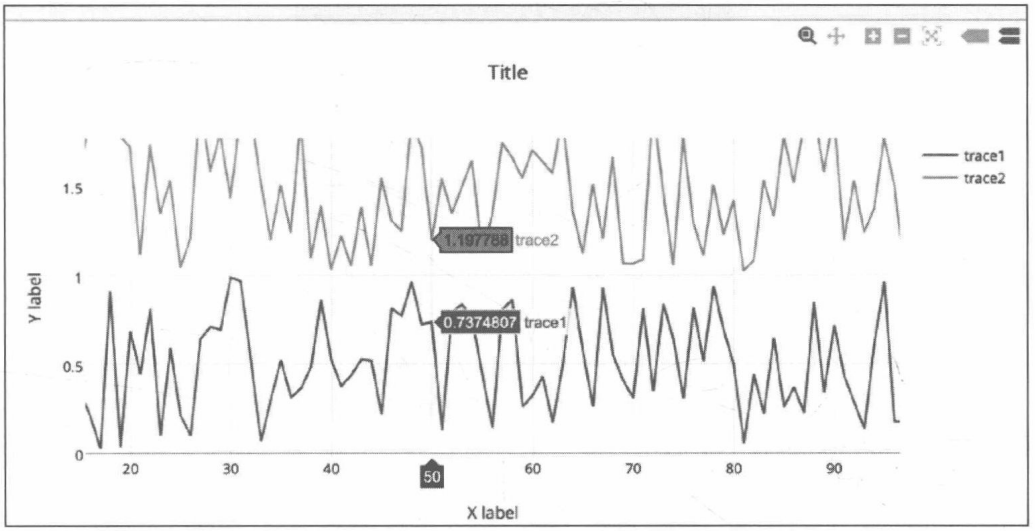
为了查看结果，可能需要使用浏览器打开 California.html：

Plotly 是另一个交互式绘图工具，但需要联网，并要有 Plotly 账户。使用 Plotly 画出来的图形看起来非常漂亮。下面的代码演示了如何使用 plotly 绘图：

```
from pylab import *
import plotly
#py = plotly.plotly('me', 'mykey')


def to_plotly(ax=None):
    if ax is None:
        ax = gca()
    lines = []
    for line in ax.get_lines():
        lines.append({'x': line.get_xdata(),
```

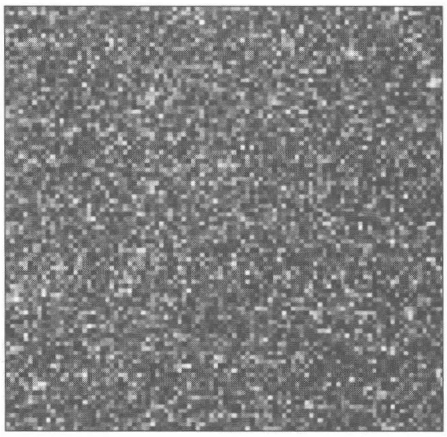
```
                    'y': line.get_ydata(),
                    'name': line.get_label(),
                    })

    layout = {'title':ax.get_title(),
              'xaxis':{'title':ax.get_xlabel()},
              'yaxis':{'title':ax.get_ylabel()}
              }
    filename = ax.get_title()  if ax.get_title() != '' else 'Untitled'
    print filename
    close('all')
    #return lines, layout
    return py.iplot(lines,layout=layout, filename = filename)

plot(rand(100), label = 'trace1')
plot(rand(100)+1, label = 'trace2')
title('Title')
xlabel('X label')
ylabel('Y label ')

response = to_plotly()
response
```



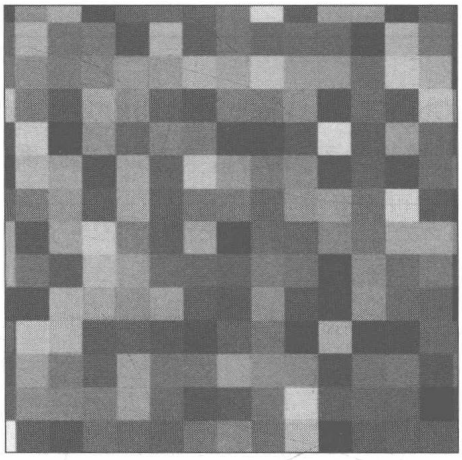


VisPy 是基于 Python 和 OpenGL 的具备更高性能的交互式绘图工具；因此，它提供了更强大的 CPU。VisPy 是新型成熟的画图工具，它为用户提供了很好的可视化库。下面的例子显示了如何使用 vispy 创建可交互缩放图像：

```
import sys

from vispy import scene
from vispy import app
import numpy as np

canvas = scene.SceneCanvas(keys='interactive')
canvas.size = 800, 800
canvas.show()

# Set up a viewbox to display the image with interactive pan/zoom
view = canvas.central_widget.add_view()

# Create the image
img_data = np.random.normal(size=(100, 100, 3), loc=128,
                            scale=40).astype(np.ubyte)
image = scene.visuals.Image(img_data, parent=view.scene)

# Set 2D camera (the camera will scale to the contents in the scene)
view.camera = scene.PanZoomCamera(aspect=1)

if __name__ == '__main__' and sys.flags.interactive == 0:
    app.run()
```



上图展示了初次出现的图形，我们可以通过移动鼠标并进行放大，如下图所示：



5.6 总结

本章讨论了典型的金融案例及机器学习的相关内容。我们还简要介绍了确定性模型，讨论了按揭支付中的总收入分析和节省数额。

本章使用真实数据进行分析，也讨论了 VSTOXX 波动率指数中欧式看涨期权的隐含波动率。我们也了解了蒙特卡罗模拟方法，并展示了如何将蒙特卡洛方法用于库存问题和篮球比赛分析。

此外，我们还以证券市场模型作为例子学习模拟模型（例如几何布朗运动和基于扩散模拟）。本章也讨论了用于计算漂移率和波动率的扩散模拟。

我们还学习了贝叶斯线性回归模型和交互式绘图方法。然后，讨论了 k 最近邻算法，基于实例的学习性能和机器学习算法。这个例子只为了激发你兴趣，并给出这些算法的原理。在后面的章节中，我们将看到更有趣的统计和机器学习算法。

第 6 章 Chapter 6

统计与机器学习

利用机器学习可以创建和使用计算机算法，学习这些算法，完成修正并用过去未知的任何新模式加以改进。你也可以从这些数据发现的新模式中提取观点。例如，你可能对计算机识别图像中的邮政编码感兴趣。另一个例子是，如果有一个特定的任务，比如确定垃圾信息，你可以用计算机寻找一些现成的方法，得到精确结果，而不必自己动手编写程序。

近年来，机器学习在人工智能中越来越重要。伴随着强大的计算能力，我们很有可能用机器学习方法构建智能系统。当前已具备的计算能力使得完成这些任务比 20 年前更简单。机器学习的主要目标是研发在真实世界中值得推崇的算法。除了时间和空间效率之外，这些学习算法所需的数据量也具有挑战性。作为数据驱动的机器学习算法，你可以看到为什么如今在该领域有如此多不同的算法。本章我们将举例讨论下述主题：

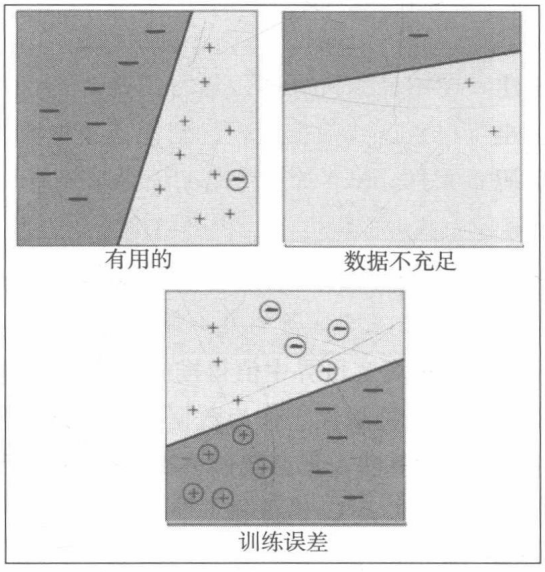
- 分类方法——决策树和线性以及 k- 最近邻
- 朴素贝叶斯、线性回归和 logistic 回归
- 支持向量机
- 基于树的回归和无监督学习
- 主成分分析
- 基于相似性的聚类
- 分类测度表现

6.1 分类方法

机器学习算法可以解决真实世界中的很多应用问题，例如，判断你对气候精确预测还是疾病诊断感兴趣。这项学习通常基于一些已知的行为或观测。这意味着机器学习是指基于对过去的经验或观测加以学习，并在某些方面得到改善。

机器学习算法广义上分为有监督学习、无监督学习、加强学习和深度学习。有监督学习的分类学习方法（这里的测试数据加了标签）类似于一位监督不同班级的老师。当我们指定一个目标变量时，有监督学习借助算法来学习数据。构建一个精确的分类器需要下面的特征：

- 一套好的训练案例
- 训练集上有相当不错的表现
- 与先验预期相关的分类器方法



二项分类器需要数据项，并将它们分别放在两个类别中（对更高维类别而言，数据项被放到 k 个类别中）。二项分类器的实例确定了能否用某些疾病阳性或阴性的概率来诊断一个人的病情。分类器算法是概率性的。在误差范围内，一个人能够被诊断为阳性或阴性。这些算法都可以通过一种通用的方法来完成，顺序如下：

- 从一个可靠的来源收集数据
- 准备或重新识别有特定结构的数据。对于一个二项分类器，距离计算是必需的
- 用任意合适的方法分析数据
- 训练（二项分类器不适用）

❑ 测试（计算误差比率）

本章中，我们的讨论将关注于哪些工具可以用于可视化输入和结果，但不太关注机器学习概念。要更深入研究该领域，你可以参考相关材料。让我们看一个例子，逐渐研究各种可选择的方案。

6.1.1 理解线性回归

一种简单的场景是在 GPA 得分和 SAT 得分数据的基础上，预测一个学生是否有可能被大学本科培养计划（如普林斯顿大学）接受，样本数据如下所示：

	SAT.Score	GPA	Accepted
1	2400	4.4	Y
2	2350	4.5	Y
3	2400	4.2	Y
4	2290	4.3	N
5	2100	4.0	N
6	2380	4.1	Y
7	2300	3.9	N
8	2280	4.0	N
9	2210	4.3	Y
10	2390	4.5	Y

为了能够考虑录取情况与 SAT 与 GPA 得分相结合的分数间的关系，这里仅为了说明一个案例（需要注意的是，这不像真实的录取过程），我们将试图弄清楚分割线。由于 SAT 得分沿 x 轴从 2100 到 2390，我们能够从 y=2490 – 2 × i × 2000 尝试 5 个值。下一个例子中，我们用 2150 而不是 2000。GPA 沿 y 轴有极值 3.3 和 5.0；因此，我们从 3.3 开始用增量值，用 3.3+0.2i 从一个极值递增，用 5.0–0.2i 从另一个极值递减（步长为 0.2）。

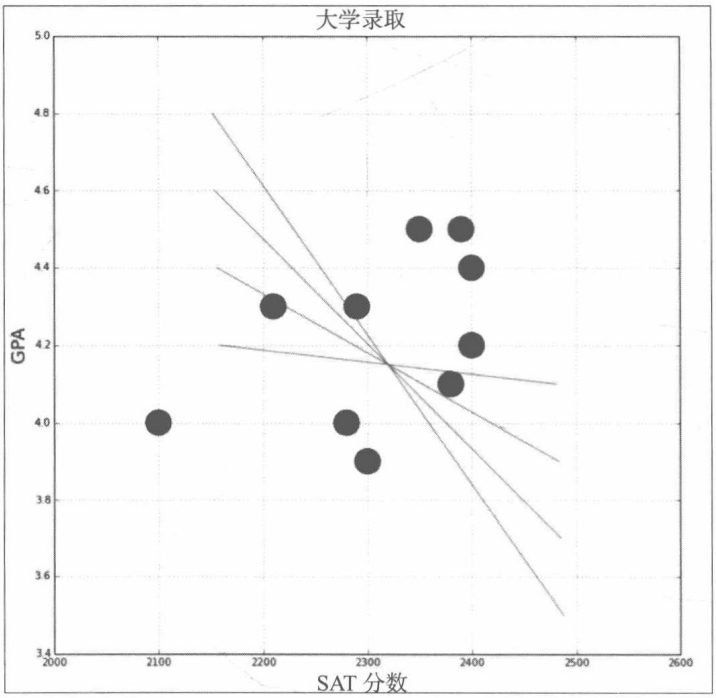
这是如何进行数据可视化的第一次尝试，我们将试图用 matplotlib 和 numpy 进行探索。用 x 和 y 轴的 SAT 和 GPA 得分绘制散点图，我们将试图发现下面的例子中的分割线：

```
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np

mpl.rcParams['axes.facecolor']= '#f8f8f8'
mpl.rcParams['grid.color'] = '#303030'
mpl.rcParams['grid.color']= '#303030'
mpl.rcParams['lines.linestyle'] = '--'
#SAT Score
x=[2400,2350,2400,2290,2100,2380,2300,2280,2210,2390]

#High school GPA
y=[4.4,4.5,4.2,4.3,4.0,4.1,3.9,4.0,4.3,4.5]

a = '#6D0000'
```

```
r = '#00006F'
#Acceptance or rejections core
z=[a,a,a,r,r,a,r,r,a,a]

plt.figure(figsize=(11,11))
plt.scatter(x,y,c=z,s=600)

# To see where the separation lies
for i in range(1,5):
  X_plot = np.linspace(2490-i*2,2150+i*2,20)
  Y_plot = np.linspace(3.3+i*0.2,5-0.2*i,20)
  plt.plot(X_plot,Y_plot, c='gray')

plt.grid(True)

plt.xlabel('SAT Score', fontsize=18)
plt.ylabel('GPA', fontsize=18)
plt.title("Acceptance in College", fontsize=20)
plt.legend()

plt.show()
```

在上面的代码中，我们无法完成任何回归或分类。这仅仅是一个理解数据视觉样貌的尝试。你也可以绘制一些分割线，直观理解线性回归的工作原理。

可以看到，一种精确的预测方法所需的测试数据不够充足。然而，如果我们试图得到更多数据，并用一些著名的软件包实现机器学习算法，我们就能够更好地理解结果。例如，增加课外活动（比如体育和音乐）。



6.1.2 线性回归

使用线性回归的主要目标是预测一个数值目标值。完成这项任务的一种方法是写出目标取值与输入的方程。例如，假定我们正在尽力预测一名在体育和音乐方面全面发展、但属于低收入家庭的学生的录取率。

一个可能的方程是：录取 = 0.0015 × 收入 + 0.49 × 得分；这是一个回归方程。利用一个简单线性回归用单个特征变量预测定量响应变量。形式如下：

$$y=\beta_0+\beta_1 x$$

这里，y 是响应变量，β_0 是截距，β_1 是 x 的系数。

β_0 和 β_1 为模型参数。为了创建我们的模型，你必须学习这些参数的实际含义。这样才能够合理地用模型预测录取比率。

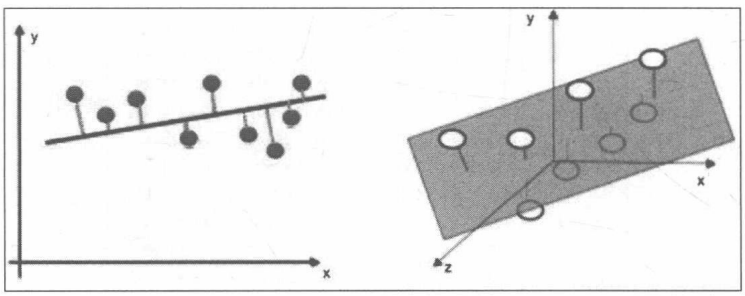
这些参数通过最小二乘准则进行估计，这意味着我们将从数学层面找到分割线，并且最小化平方残差和。下面的案例用到一部分数据：

	X	academic	sports	music	acceptance
1	1	230.1	37.8	62.9090909	81.851852
2	2	44.5	39.3	41.0000000	38.518519
3	3	17.2	45.9	63.0000000	34.444444
4	4	151.5	41.3	68.5185185	68.518519
5	5	180.8	10.8	53.0909091	47.777778
6	6	8.7	48.9	68.1818182	26.666667
7	7	57.5	32.8	21.3636364	43.703704
8	8	120.2	19.6	10.5454545	48.888889
9	9	8.6	2.1	0.9090909	17.777778
10	10	199.8	2.6	19.2727273	39.259259
11	11	66.1	5.8	22.0000000	31.851852
12	12	214.7	24.0	64.4444444	64.444444
13	13	23.8	35.1	59.9090909	34.074074
14	14	97.5	7.6	6.5454545	35.925926
15	15	204.1	32.9	70.3703704	70.370370
16	16	195.4	47.7	82.9629630	82.962963
17	17	67.8	36.6	103.6363636	46.296296
18	18	281.4	39.6	90.3703704	90.370370
19	19	69.2	20.5	16.6363636	41.851852
20	20	147.3	23.9	17.3636364	54.074074

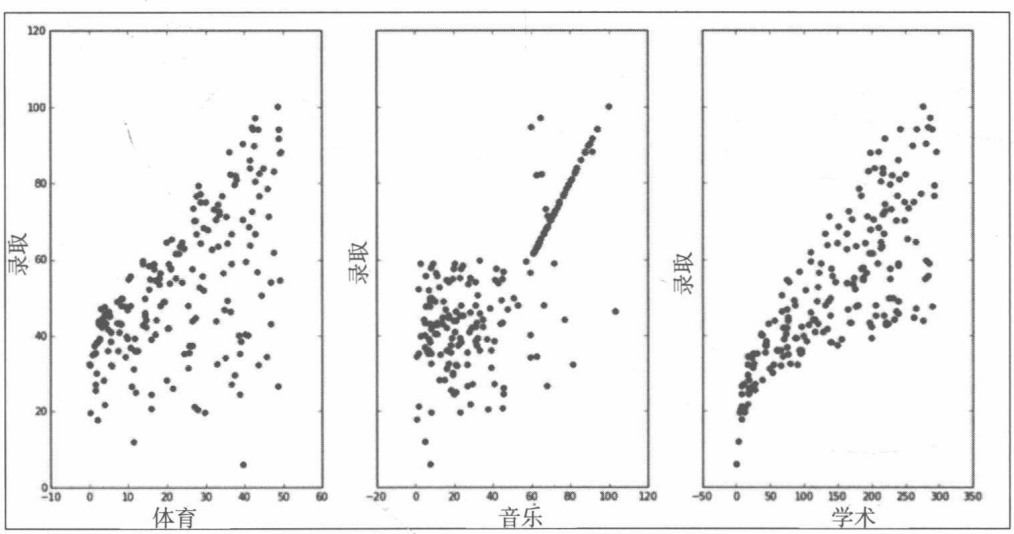
下面的 Python 代码展示了如何尝试用散点图确定变量间的相关性：

```
from matplotlib import pyplot as pplt

import pandas as pds

import statsmodels.formula.api as sfapi

df = pds.read_csv('/Users/myhomedir/sports.csv', index_col=0)
fig, axs = plt.subplots(1, 3, sharey=True)
```

```
df.plot(kind='scatter', x='sports', y='acceptance', ax=axs[0],
figsize=(16, 8))
df.plot(kind='scatter', x='music', y='acceptance', ax=axs[1])
df.plot(kind='scatter', x='academic', y='acceptance', ax=axs[2])

# create a fitted model in one line
lmodel = sfapi.ols(formula='acceptance ~ music', data=df).fit()

X_new = pd.DataFrame({'music': [df.music.min(), df.music.max()]})
predictions = lmodel.predict(X_new)

df.plot(kind='scatter', x='music', y='acceptance', figsize=(12,12),
s=50)

plt.title("Linear Regression - Fitting Music vs Acceptance Rate",
fontsize=20)
plt.xlabel("Music", fontsize=16)
plt.ylabel("Acceptance", fontsize=16)

# then, plot the least squares line
```



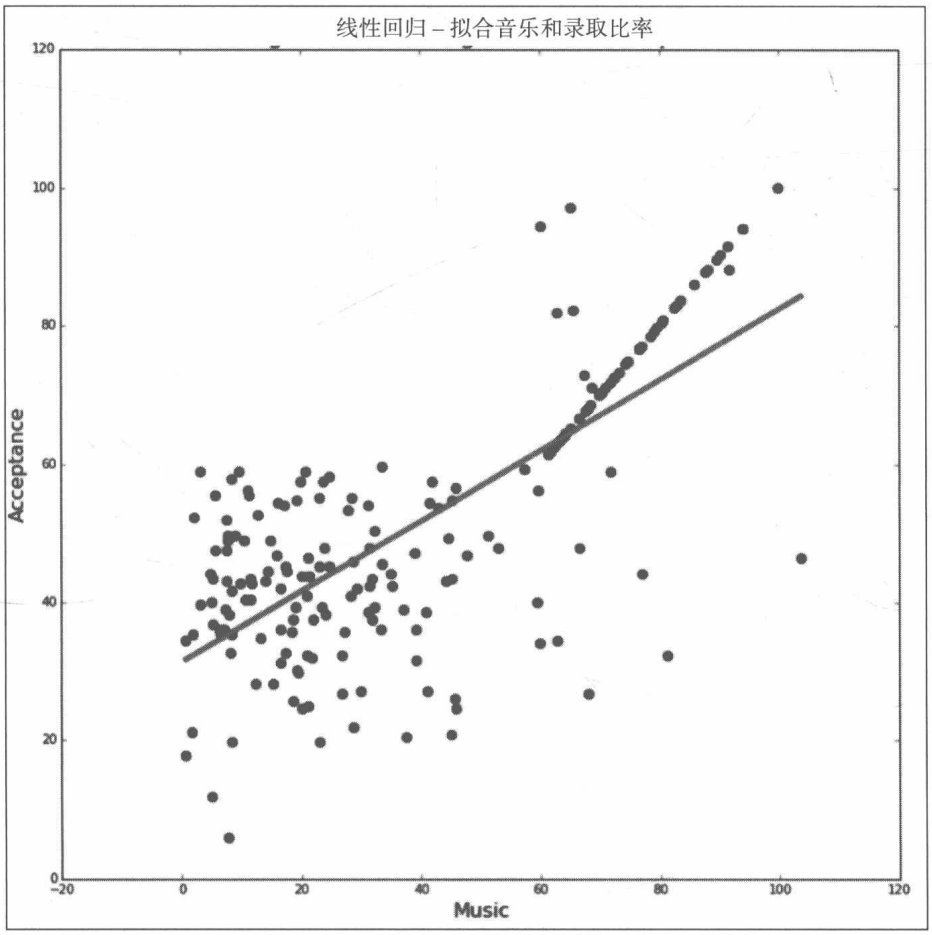
如上图所示，点是观测值 (x, y)，对角线是基于 (x, y) 数值的最小平方拟合。细线是残差，表示观察值和最小平方线间的距离。



用 statsmodels、pandas 和 matplotlib（如上图所示），我们可以假定有某种得分，它基于一所大学如何评估学生在学术、体育和音乐方面的贡献。

为了测试一个分类器，我们可以从一些已知数据和未知答案开始，用分类器寻找答案，得到最好的结果。此外，我们可以增加分类器错误的次数，并通过错误比率的测试总数进行划分。

下面是从上面的 Python 代码得出的线性回归分析图。



还有很多其他用于线性回归分析的 Python 库，如 scikit-learn、seaborn、statsmodels 和 mlpy，这些是比较著名和常用的库。网上已有很多用这些软件包进行线性回归分析的案例。想了解 scikit-learn 软件包的更多细节，请参见：http://scikit-learn.org/stable/modules/generated/sklearn.linear_ model.LinearRegression.html。

还有一种有趣的机器学习模型叫作**决策树学习**，有时可以指**分类树**。另一个类似的模型是**回归树**。这里，我们将讨论它们间的不同以及哪种方法比其他方法更有意义。

6.1.3 决策树

分类树用于将数据分为响应变量的几个类别。响应变量通常有两类：是或否（1或0），晴天或下雨。如果目标变量有不止两个类别，可以使用C4.5。C4.5改进了ID3算法，以适用于连续属性、离散属性和后续的构建过程。

和大多数学习算法类似，分类树算法分析了训练集，然后在训练集上构建分类器，以在将来。它可以像分类训练集一样正确地分类新数据。测试案例为输入对象，算法必须能预测出输出数值。分类树适用于响应或目标变量本质上是分类型变量的情况。

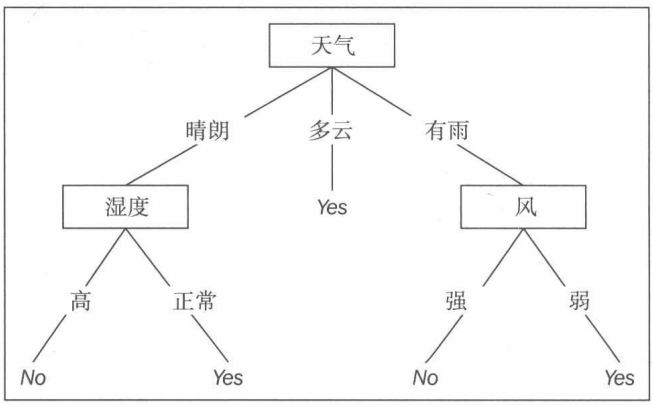
相反，当响应变量是连续型和非离散型时，需要用回归树。例如，产品价格的预测。回归树的建立是通过二项划分完成的。这是一个迭代过程，将数据分为分隔或分支，然后继续将每个分隔划分为更小的群，直到该方法经过每一个分隔或分支。换句话说，回归树适用于预测问题而不是分类问题。想要了解更多的细节，我们建议你参考分类和回归树方面的书籍。

当预测变量与响应变量间存在线性关系时，一个标准的回归树更为适合。当预测变量与响应变量间存在非线性关系时，应该使用C4.5。故当响应变量只有两类时，应该用分类树算法。

一个例子

对于确定一个玩网球还是高尔夫的决策树算法，我们可以简单地通过提问对决策过程进行分类，也就是，是下雨还是晴天？而且基于答案在每个问题处进行分支来绘制决策图。体育运动的本质绝大多数相同（网球与高尔夫）而且在任何体育赛事中，如果刮风下雨，将不适合进行体育运动。

如果天气晴朗但湿度很高，就推荐打网球。类似地，如果下雨刮风，则网球比赛将变得非常糟糕。因此，在这些情况下，打网球就变得没有乐趣。下图展示了所有可能的情况：



我们还可以添加离散属性（如温度），在什么温度范围内打网球没有意义？可能，如果温度大于 70 华氏度[⊖]，即温度太高的时候。我们可以结合这些条件编写如下规则：

```
If (Outlook = Sunny) and (Humidity = High) then play=No
If (Outlook = Rain) and (Wind = Strong) then play=No
If (Outlook = Sunny) and (Humidity = Normal) or
  (Outlook = Overcast) or (Outlook=Rain and Wind=Weak) then play=Yes
```

用下面的训练集，我们可以运用算法选择下一个最佳分类器：

天气	温度	湿度	风	是否打网球
晴朗	炎热	高	弱	否
晴朗	炎热	高	强	否
多云	炎热	高	弱	是
多云	凉爽	正常	强	是
晴朗	温和	高	弱	否
晴朗	凉爽	正常	弱	是
有雨	温和	高	弱	是
有雨	凉爽	正常	弱	是
有雨	凉爽	正常	强	否
有雨	温和	正常	弱	是
晴朗	温和	正常	强	是
多云	温和	高	强	是
多云	炎热	正常	弱	是
有雨	温和	高	强	否

自上而下的**决策树归纳**（induction of decision tree，ID3）是遵守下面规则的方法：

❑ 在叶节点上迭代直到满足停止条件：

1. 在遍历下一个节点时，确定最佳决策属性。
2. 分配第 1 步中的最佳节点为决策属性。
3. 对于那些最佳节点的每个数值，创建那些节点的新后代。
4. 将训练数据分类为叶节点。
5. 停止迭代条件：如果训练数据被分到阈值范围。

线性回归和决策树算法间的一个明显区别在于：决策树的边界与数轴平行。比如，如果我们有两个特征（x_1 和 x_2），就可以创建规则，如 $x_1 \geqslant 5.2$，$x_2 \geqslant 7.2$。决策树算法的优势是误差的稳健性，这意味着训练集可以有误差。此外，它对算法的影响不大。

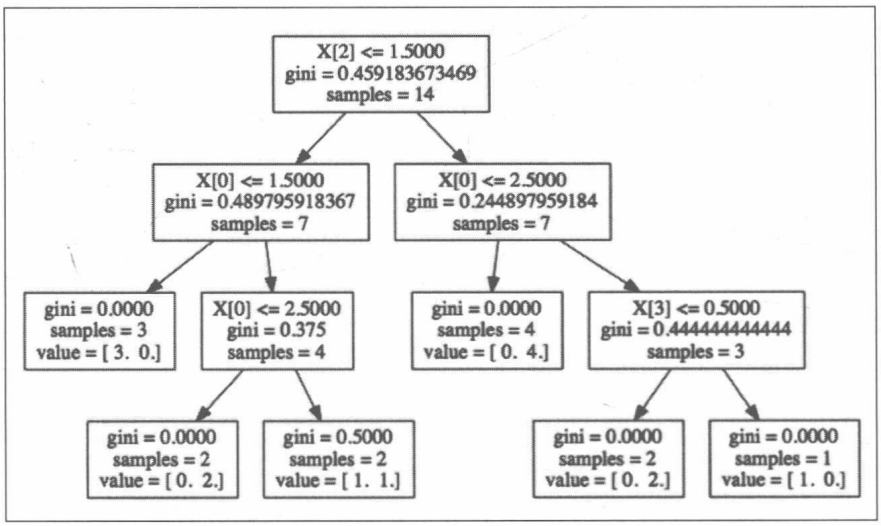
使用 scikit-learn（scikit-learn.org）的 sklearn 软件包和下面的代码，我们可以绘制决策

⊖ 约为 21 摄氏度。

树分类器：

```
from sklearn.externals.six import StringIO
from sklearn import tree
import pydot

# Four columns from the table above with values
# 1st col - 1 for Sunny, 2 for Overcast, and 3 for Rainy
# 2nd col - 1 for Hot, 2 for Mild, 3 for Cool
# 3rd col - 1 for High and 2 for Normal
# 4th col - 0 for Weak and 1 for Strong

X=[[1,1,1,0],[1,1,1,1],[2,1,1,0],[2,3,2,1],[1,2,1,0],[1,3,2,0],\
[3,2,1,0],[3,3,2,0],[3,3,2,1],[3,2,2,0],[1,2,2,1],[2,2,1,1],\
[2,1,2,0],[3,2,1,0]]
# 1 for Play and 0 for Don't Play
Y=[0,0,1,1,0,1,1,1,0,1,1,1,1,0]

clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, Y)

dot_data = StringIO()
tree.export_graphviz(clf, out_file=dot_data)

graph = pydot.graph_from_dot_data(dot_data.getvalue())
graph.write_pdf("game.pdf")
```

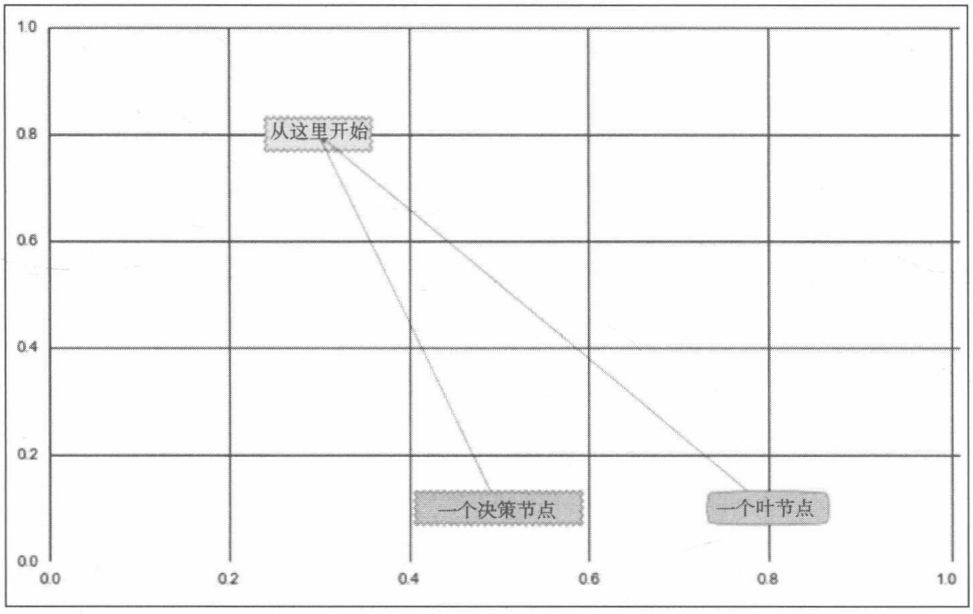
用 sklearn 的输出功能可以将树形图转换为一种看起来与下图类似的图形形式：



为了创建树形结构，可以用 matplotlib 绘图。matplotlib 可以通过标签创建树形图的注解展示树形图，代码如下所示：

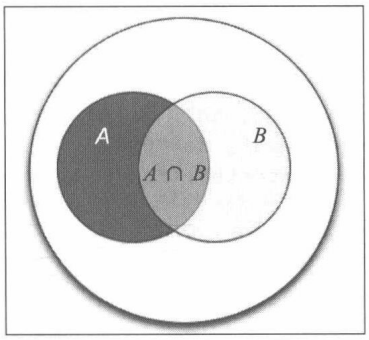
```
import matplotlib.pyplot as plt

#create nodes here
branchNode = dict(boxstyle="sawtooth", fc="0.8")
leafNode = dict(boxstyle="round4", fc="0.8")
startNode = dict(boxstyle="sawtooth", fc="0.9")
def createPlot():
    fig = plt.figure(1, facecolor='white')
    fig.clf()
    createPlot.ax1 = plt.subplot(111, frameon=False) #ticks for demo
purposes
    plotNode('from here', (0.3,0.8), (0.3, 0.8), startNode)
    plotNode('a decision node', (0.5, 0.1), (0.3, 0.8), branchNode)
    plotNode('a leaf node', (0.8, 0.1), (0.3, 0.8), leafNode)
    plt.show()
...
```

这通常是用 matplotlib 从零开始创建树形结构的想法。基本上，上面的例子展示了如何创建 3 个节点以形成一棵小树。代码的运行结果如下所示：



6.1.4 贝叶斯理论

首先，为了理解贝叶斯理论，在试图研究朴素贝叶斯分类方法之前，不妨先考虑下面的例子。假定在 u 宇宙的所有人中，患乳腺癌的人为集合 A，集合 B 是经过筛选试验却不幸诊断为乳腺癌阳性的集合。A ∩ B 表示重叠的区域，如下图所示：



有两个需要关注的异常区域：$B - A \cap B$ 或者那些没有乳腺癌却测试为阳性的人，以及事件 $A - A \cap B$ 或者那些有乳腺癌却测试为阴性的人。现在，我们尽量回答我们是否知道随机被抽中的人的测试结果是阳性。进而，判断一个人患乳腺癌的概率有多大？这在视觉上转换为我们是否知道一个人属于 B，那么这个人出现在 $A \cap B$ 的概率？数学上，这种转换就是概率（给定 B 的情况 A）。条件概率方程如下：

$$P(A|B)=\frac{|A\cap B|}{|B|}=\frac{\frac{|A\cap B|}{|U|}}{\frac{|B|}{|U|}}$$

$$P(A|B)=\frac{P(A\cap B)}{P(B)}$$

类似地，如果知道一个随机抽中的人患有癌症，那么诊断测试结果为阳性的概率有多大？这种转换概率（给定 A 的情况 B）如下所示：

$$P(B|A)=\frac{|A\cap B|}{|B|}=\frac{\frac{|A\cap B|}{|U|}}{\frac{|A|}{|U|}}$$

$$P(B|A)=\frac{P(A\cap B)}{P(A)}$$

$$\Rightarrow P(A\cap B)=P(B|A)P(A)=P(A|B)P(B)$$

$$P(A|B)=\frac{P(B|A)P(A)}{P(B)}$$

因此，我们推导出了贝叶斯理论，其中 A 和 B 是事件，且 $P(B)$ 非零。

6.1.5 朴素贝叶斯分类器

朴素贝叶斯分类器技术以贝叶斯理论为基础，而且适用于输入变量维度较高的情况。

尽管这看起来很简单，但是它在技术上比其他分类方法完成得要好。

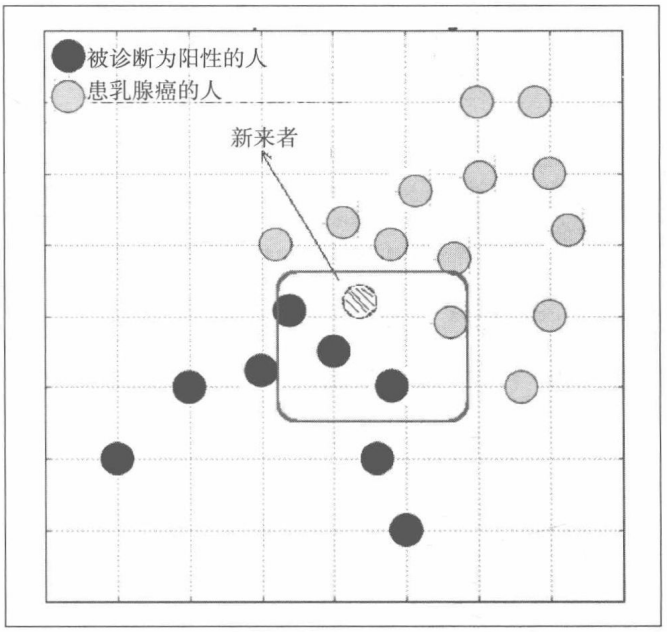
（更多信息请见：http://scikit-learn.org/stable/modules/naive_bayes.html 和 http://sebastianraschka.com/Articles/2014_naive_ bayes_1.html。）

让我们看看下面的例子，其中的研究对象展示为浅色和深色。如图所示，浅色的对象表示乳腺癌患者，深色的对象表示乳腺癌诊断为阳性的患者。我们的任务是能够对任意新数据加标签。在这种情况下，新来者的出现基于对象的既定结构或类别，并且确定新数据或新人属于哪个群或类。

在贝叶斯中，先验概率更倾向于接近对象当下被特征描述的模式或行为。这主要取决于一个事实：这里先验一词与先前的经验同义；因此，如果浅色对象所占的百分比比深色的大，那么我们预期预测结果应该更多是浅色比较有优势。

这里的方法是朴素贝叶斯和 k 最近邻算法的结合。对于一个纯粹的朴素贝叶斯分类方法，我们将用 TextBlob 讨论另外的案例（http://textblob.readthedocs.org/en/dev/）。

下图展示了还未分类的一个人：



用浅色和深色的先验概率，你可以计算 x 是浅色还是深色的后验概率，如下面的代码所示：

浅色的先验概率 =13/21

深色的先验概率 =8/21

给定浅色时 x 的可能性 = 区域内的红点数 / 红点总数 =1/13

$$给定深色时x的可能性=区域内的蓝点数/蓝点总数=3/8$$

$$x为浅色的后验概率=(1/13)\times(13/21)=1/21$$

$$x为深色的后验概率=(3/8)\times(8/21)=1/7$$

新来者最有可能被分类为乳腺癌诊断为阳性的患者。

6.1.6 用 TextBlob 构建朴素贝叶斯分类器

TextBlob 是一个有趣的库，是具备文本处理功能的工具集合。它配备有 API，能完成**自然语言处理**（natural language processing，NLP）任务，比如分类、名词短语提取、部分词性标注和情感分析。

有一些步骤来确保 TextBlob 可以使用。任何与 NLP 一起工作的库都需要 corpora，因此，下面的安装序列和配置需要在该库使用前完成：

- ❑ 安装 TextBlob（通过 conda 或者 pip）
- ❑ 下载 corpora

1. 安装 TextBlob

使用 binstar search -t conda textblob，anaconda 用户可以发现从哪进行安装。更多的细节请见附录。

2. 下载 corpora

下面是下载 corpora 的命令：

```
$ python -m textblob.download_corpora

[nltk_data] Downloading package brown to
[nltk_data]     /Users/administrator/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/administrator/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/administrator/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
[nltk_data] Downloading package conll2000 to
[nltk_data]     /Users/administrator/nltk_data...
[nltk_data]   Unzipping corpora/conll2000.zip.
[nltk_data] Downloading package maxent_treebank_pos_tagger to
[nltk_data]     /Users/administrator/nltk_data...
[nltk_data]   Unzipping taggers/maxent_treebank_pos_tagger.zip.
[nltk_data] Downloading package movie_reviews to
[nltk_data]     /Users/administrator/nltk_data...
[nltk_data]   Unzipping corpora/movie_reviews.zip.
Finished.
```

3. 使用 TextBlob 的朴素贝叶斯分类器

TextBlob 使创建自定义文本分类器变得容易。为了更好理解，你可能需要用其训练和测试数据进行一些实验。在 TextBlob 0.6.0 版本中，可用的分类器如下：

- BaseClassifier
- DecisionTreeClassifier
- MaxEntClassifier
- NLTKClassifier *
- NaiveBayesClassifier
- PositiveNaiveBayesClassifier

用 * 标记的分类器是包在 nltk.classify 模块周围的一个抽象类。

对于情感分析，可以用朴素贝叶斯分类器，并用该分类器和 textblob.en.sentiments.PatternAnalyzer 训练系统。一个简单的例子如下所示：

```
from textblob.classifiers import NaiveBayesClassifier
from textblob.blob import TextBlob

from textblob.classifiers import NaiveBayesClassifier
from textblob.blob import TextBlob

train = [('I like this new tv show.', 'pos'),
  # similar train sentences with sentiments goes here]
test = [ ('I do not enjoy my job', 'neg'),
  # similar test sentences with sentiments goes here]
]

cl = NaiveBayesClassifier(train)
cl.classify("The new movie was amazing.") # shows if pos or neg


cl.update(test)

# Classify a TextBlob
blob = TextBlob("The food was good. But the service was horrible. "
                "My father was not pleased.", classifier=cl)
print(blob)
print(blob.classify())

for sentence in blob.sentences:
    print(sentence)
    print(sentence.classify())
```

运行上面的代码将得到以下结果：

```
pos
neg
```

```
The food was good.
pos
But the service was horrible.
neg
My father was not pleased.
pos
```

无论是文本格式还是JSON格式的文件，我们都能从中读取训练数据。JSON格式的样本数据如下所示：

```
[
  {"text": "mission impossible three is awesome btw","label": "pos"},
  {"text": "brokeback mountain was beautiful","label":"pos"},
  {"text": " da vinci code is awesome so far","label":"pos"},
  {"text": "10 things i hate about you + a knight's tale * brokeback
mountain","label":"neg"},
  {"text": "mission impossible 3 is amazing","label":"pos"},

    {"text": "harry potter = gorgeous","label":"pos"},
    {"text": "i love brokeback mountain too: ]","label":"pos"},
]

from textblob.classifiers import NaiveBayesClassifier
from textblob.blob import TextBlob
from nltk.corpus import stopwords

stop = stopwords.words('english')

pos_dict={}
neg_dict={}
with open('/Users/administrator/json_train.json', 'r') as fp:
     cl = NaiveBayesClassifier(fp, format="json")
print "Done Training"

rp = open('/Users/administrator/test_data.txt','r')
res_writer = open('/Users/administrator/results.txt','w')
for line in rp:
    linelen = len(line)
    line = line[0:linelen-1]
    sentvalue = cl.classify(line)
    blob = TextBlob(line)
    sentence = blob.sentences[0]
    for word, pos in sentence.tags:
       if (word not in stop) and (len(word)>3 \
            and sentvalue == 'pos'):
         if pos == 'NN' or pos == 'V':
           pos_dict[word.lower()] = word.lower()
       if (word not in stop) and (len(word)>3 \
            and sentvalue == 'neg'):
         if pos == 'NN' or pos == 'V':
           neg_dict[word.lower()] = word.lower()
```

```
    res_writer.write(line+" => sentiment "+sentvalue+"\n")

    #print(cl.classify(line))
print "Lengths of positive and negative sentiments",len(pos_dict),
len(neg_dict)

Lengths of positive and negative sentiments 203 128
```

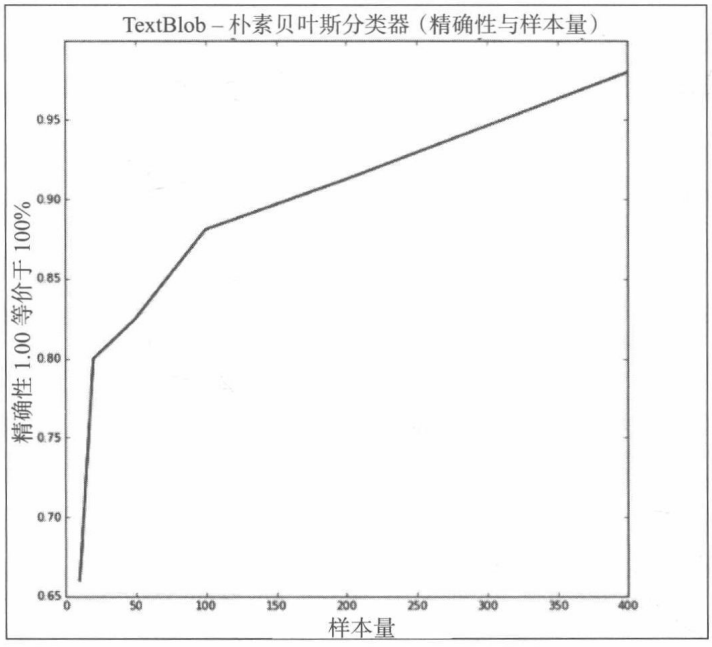
我们可以从 corpus 添加更多的训练数据，并用下面的代码评估分类器的精确性：

```
test=[
("mission impossible three is awesome btw",'pos'),
("brokeback mountain was beautiful",'pos'),
("that and the da vinci code is awesome so far",'pos'),
("10 things i hate about you =",'neg'),
("brokeback mountain is a spectacularly beautiful movie",'pos'),
("mission impossible 3 is amazing",'pos'),
("the actor who plays harry potter sucks",'neg'),
("harry potter = gorgeous",'pos'),
('The beer was good.', 'pos'),
('I do not enjoy my job', 'neg'),
("I ain't feeling very good today.", 'pos'),
("I feel amazing!", 'pos'),
('Gary is a friend of mine.', 'pos'),
("I can't believe I'm doing this.", 'pos'),
("i went to see brokeback mountain, which is beautiful(",'pos'),
("and i love brokeback mountain too: ]",'pos')
]

print("Accuracy: {0}".format(cl.accuracy(test)))
from nltk.corpus import movie_reviews

reviews = [(list(movie_reviews.words(fileid)), category)
for category in movie_reviews.categories()
for fileid in movie_reviews.fileids(category)]
new_train, new_test = reviews[0:100], reviews[101:200]

cl.update(new_train)
accuracy = cl.accuracy(test + new_test)
print("Accuracy: {0}".format(accuracy))

# Show 5 most informative features
cl.show_informative_features(4)
```

结果如下所示：

```
Accuracy: 0.973913043478
Most Informative Features
contains(awesome) = True          pos : neg    =     51.9 : 1.0
contains(with) = True             neg : pos    =     49.1 : 1.0
contains(for) = True              neg : pos    =     48.6 : 1.0
contains(on) = True               neg : pos    =     45.2 : 1.0
```

首先，训练集有 250 个样本，准确率是 0.813，随后从电影评论中添加了另外 100 个样本。准确率提高到 0.974。因此，我们试图使用不同的测试样本，绘制样本量与精确性，如下图所示：



6.1.7 用词云观察积极情绪

在任何给定的文本中，词云赋予出现频率更高的词以更大的显著性。它们也叫作标签云或加权云。就一个词出现的次数而言，其强度的显著性在视觉方面映射为它呈现的大小。换句话说，视觉上看起来最大的词是文本中出现次数最多的词。

除了通过形状和颜色展示出现率之外，词云还有一些社交媒体和市场营销方面的应用，如下所示：

- 企业可以了解他们的顾客以及他们对产品的评价。一些组织已经用一种创造性的方式，要求他们的粉丝或追随者用词汇描述他们对品牌的看法，用所有这些词汇创建一个词云，理解他们对产品品牌最普遍的印象。
- 通过确定那些网络受欢迎的品牌，寻找了解竞争者的方式。从内容创建一个词云，更好地理解与产品目标市场挂钩的词汇和主题。

为了创建一个词云，你可以使用 Python 代码或使用已经存在的工具。来自 NYU 数据科学中心的 Andreas Mueller 用 Python 创建了一个词云。这用起来非常简单网站上。RemachineScript.ttf 字体文件已从 http://www.fonts101.com/fonts/view/Script/63827/Remachine_Script 网站上下载。

STOPWORDS 包括极其普遍的词汇，比如 a、an、the、is、was、at、in 等。为了忽略它们，下面的代码用一列 STOPWORDS 创建了一个词云：

```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
from os import path

d = path.dirname("__file__")
text = open(path.join(d, '/Users/MacBook/kirthi/results.txt')).read()

wordcloud = WordCloud(
    font_path='/Users/MacBook/kirthi/RemachineScript.ttf',
    stopwords=STOPWORDS,
    background_color='#222222',
    width=1000,
    height=800).generate(text)
```

为了绘图，首先设定图的大小，并用 imshow() 展示词云：

```
# Open a plot of the generated image.
plt.figure(figsize=(13,13))

plt.imshow(wordcloud)
plt.axis("off")

plt.show()
```

总而言之，我们首先从 TextBlob 案例中提取情感，并假定被提取的结果保存为 results.txt。然后，我们用 matplotlib 软件包将这些词汇数据可视化为一个词云。

词云的结果如下图所示：

6.2 k- 最近邻

k- 最近邻（k-nearest neighbor，k-NN）分类器是最容易理解的分类方法之一（特别在少许或没有数据分布的先验知识的情况下）。k- 最近邻分类有一种方法来存储所有已知的情况，以及基于相似性测度对新情况分类（如欧氏距离函数）。k-NN 算法由于其简洁性而在统计估计和模式识别方面熟知。

对于 **1- 最近邻**（1-nearest neighbor，1-NN），一个特别点的标签被设定为最近训练点。当你推广到一个更高值 k 时，一个测试点的标签由 k 个最近训练点测量得到。k-NN 算法被认为是一种慵懒的学习算法，因为是局部最优化，而且计算被延后直到分类。

该方法有优点也有不足。优点是高准确性，对异常值不敏感以及对数据没有假设条件。k-NN 的不足是计算昂贵且需要大量内存。

可使用以下距离度量：

$$\text{欧氏距离：}\ \sqrt{\sum_{i=1}^{k}(x_i-y_i)^2}$$

$$\text{曼哈顿距离：}\ \sum_{i=1}^{k}|x_i-y_i|$$

$$\text{明氏距离：}\ \left(\sum_{i=1}^{k}(|x_i-y_i|)^q\right)^{\frac{1}{q}}$$

让我们考虑一个例子，给出了一大篮水果，只有苹果、香蕉和梨。假定只有红苹果，没有青苹果。用颜色这一特征区分这些水果。苹果是红色的，香蕉是黄色的，梨是绿色的。这些水果也可以用重量作为特征。为了说明这个例子，需要做如下假设：

形状特征可进行如下分类：

- 对苹果而言，形状数值在 1 和 3 之间，而重量在 6 和 7 盎司[⊖]之间
- 对梨而言，形状数值在 2 和 4 之间，而重量在 5 和 6 盎司之间
- 对香蕉而言，形状数值在 3 和 5 之间，而重量在 7 和 9 盎司之间

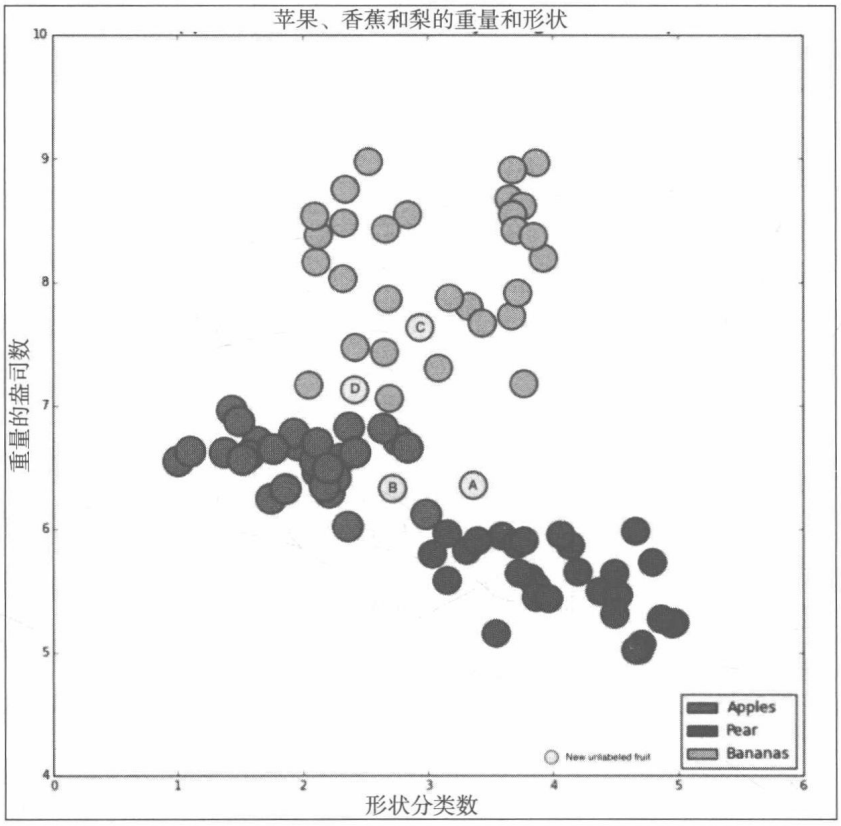
我们在篮子中有水果数据如下表所示。

如果我们有一个重量已知和颜色分类的无标签水果，运用 k- 最近邻方法（用任意距离方程）最有可能发现最近的 k 邻近点（如果他们是绿色、红色或黄色，无标签水果分别最有可能是梨、苹果或香蕉）。下面的代码用水果的形状和重量说明 k- 最近邻算法：

⊖ 1 盎司 =28.3495 克。

	Shape	Weight	Fruit
1	1.747993	6.244728	Apple
2	2.160436	6.548997	Apple
3	2.308360	6.568994	Apple
4	2.989498	6.116004	Apple
5	2.217408	6.298844	Apple
6	3.550124	5.148646	Banana
7	4.795393	5.729825	Banana
8	4.380994	5.491813	Banana
9	4.975395	5.243866	Banana
10	4.714245	5.061763	Banana
11	1.644232	6.710433	Apple
12	2.101244	8.531404	Pear
13	2.847359	8.541824	Pear
14	3.759746	8.609348	Pear
15	3.436196	7.667397	Pear
16	2.420651	7.471596	Pear
17	1.960733	6.678455	Apple
18	1.861635	6.320602	Apple

```
import csv
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt

count=0
x=[]
y=[]
z=[]

with open('/Users/myhome/fruits_data.csv', 'r') as csvf:
  reader = csv.reader(csvf, delimiter=',')
  for row in reader:
    if count > 0:
      x.append(row[0])
      y.append(row[1])
      if ( row[2] == 'Apple' ): z.append('r')
      elif ( row[2] == 'Pear' ): z.append('g')
      else: z.append('y')
    count += 1

plt.figure(figsize=(11,11))

recs=[]
classes=['Apples', 'Pear', 'Bananas']
class_colours = ['r','g','y']
plt.title("Apples, Bananas and Pear by Weight and Shape", fontsize=18)

plt.xlabel("Shape category number", fontsize=14)
plt.ylabel("Weight in ounces", fontsize=14)

plt.scatter(x,y,s=600,c=z)
```



让我们挑选 4 个无标签水果，它们的 x 和 y 取值为 A(3.5,6.2)、B(2.75,6.2)、C(2.9, 7.6)和 D(2.4, 7.2)，代码如下：

```
from math import pow, sqrt
dist=[]
def determineFruit(xv, yv, threshold_radius):
  for i in range(1,len(x)):
    xdif=pow(float(x[i])-xv, 2)
    ydif=pow(float(y[i])-yv, 2)
    sqrtdist = sqrt(xdif+ydif))
    if ( xdif < threshold_radius and
         ydif < thresholdradius and sqrtdist < threshold_radius):
      dist.append(sqrtdist)
    else:
      dist.append(99)
  pear_count=0
  apple_count=0
  banana_count=0
  for i in range(1,len(dist)):
      if dist[i] < threshold_radius:
        if z[i] == 'g': pear_count += 1
```

```
        if z[i] == 'r': apple_count += 1
        if z[i] == 'y': banana_count += 1
    if ( apple_count >= pear_count and apple_count >= banana_count ):
      return "apple"
    elif ( pear_count >= apple_count and pear_count >= banana_count):
      return "pear"
    elif ( banana_count >= apple_count and banana_count >= pear_count):
      return "banana"

dist=[]
determine = determineFruit(3.5,6.2, 1)
print determine

'pear'
```

6.3 逻辑斯谛回归

正如我们前面看到的，线性回归的一个问题是它倾向于数据欠拟合。它给出无偏估计量的最小均方误差。用欠拟合模型，我们无法得到最佳预测。有一些方法通过增加估计量偏差来减少均方误差。

逻辑斯谛回归是二分类（正确或错误）响应变量的模型拟合方法。线性回归不能直接预测所有概率，但逻辑斯谛回归可以。此外，与朴素贝叶斯的结果相比，预测概率能够校准得更好。

为了便于讨论，通过持续关注二项响应变量，我们可以设定 1 表示正确，0 表示错误。逻辑斯谛回归模型假定输入变量可以进行逆对数化处理，因此，另一种研究方法是用 y 的对数表示 x 的 n 个输入变量的线性组合，如下面的方程所示：

$$\log\frac{P(x)}{1-P(x)}=\sum_{j=0}^{n}b_j x_j=z$$

$$\frac{P(x)}{1-P(x)}=e^z$$

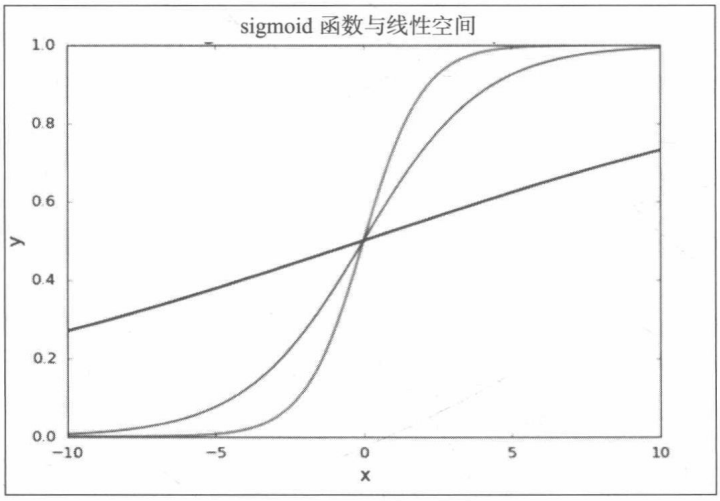
$$\Rightarrow P(x)=\frac{e^z}{1+e^z}=\frac{1}{1+e^{-z}}$$

作为对数函数的逆——指数函数，右侧的表达式看起来是变量 x 的线性组合的一个 sigmoid 的版本。这意味着分母永远不会是 1（除非 z 是 0）。因此，$P(x)$ 的数值严格大于 0，小于 1，代码如下所示：

```
import matplotlib.pyplot as plt
import matplotlib
import random, math
import numpy as np
```

```
import scipy, scipy.stats
import pandas as pd

x = np.linspace(-10,10,100)
y1 = 1.0 / (1.0+np.exp(-x))
y2 = 1.0 / (1.0+np.exp(-x/2))
y3 = 1.0 / (1.0+np.exp(-x/10))

plt.title("Sigmoid Functions vs LineSpace")
plt.plot(x,y1,'r-',lw=2)
plt.plot(x,y2,'g-',lw=2)
plt.plot(x,y3,'b-',lw=2)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

下图展示了一个标准的 sigmoid 函数：



下面是展示开心与悲伤概率的例子。

$$P(happy)=\frac{e^z}{1+e^z}$$

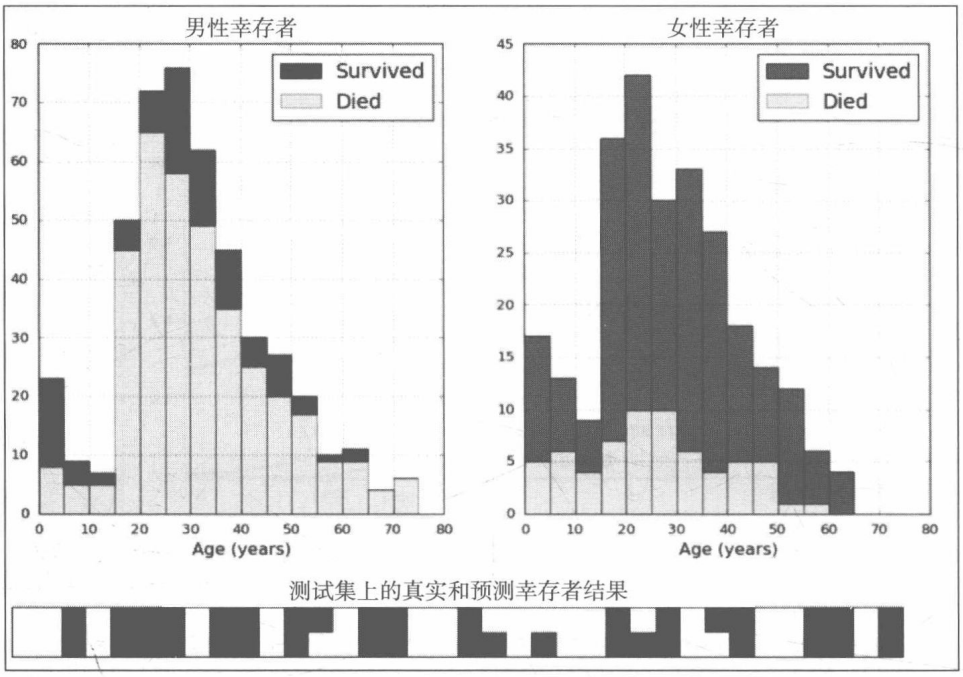
$$P(sad)=1-P(happy)=\frac{1}{1+e^z}$$

Kaggle 主办了所有机器学习比赛。它通常提供训练和测试数据。前一段时间，基于真实数据预测 Titanic 的幸存者在 Kaggle 上比赛。titanic_train.csv 和 titanic_test.csv 文件分别用于训练和测试。用 scikit-learn 中的 linear_model 软件包（包括逻辑斯谛回归），我们能看到下面是赢得比赛的那位作者代码的修订版：

```
Import numpy as np
import pandas as pd
import sklearn.linear_model as lm
import sklearn.cross_validation as cv
import matplotlib.pyplot as plt
train = pd.read_csv('/Users/myhome/titanic_train.csv')
test = pd.read_csv('/Users/myhome/titanic_test.csv')
train[train.columns[[2,4,5,1]]].head()

data = train[['Sex', 'Age', 'Pclass', 'Survived']].copy()
data['Sex'] = data['Sex'] == 'female'
data = data.dropna()

data_np = data.astype(np.int32).values
X = data_np[:,:-1]
y = data_np[:,-1]

female = X[:,0] == 1
survived = y == 1

# This vector contains the age of the passengers.
age = X[:,1]
# We compute a few histograms.
bins_ = np.arange(0, 121, 5)
S = {'male': np.histogram(age[survived & ~female],
                          bins=bins_)[0],
     'female': np.histogram(age[survived & female],
                            bins=bins_)[0]}
D = {'male': np.histogram(age[~survived & ~female],
                          bins=bins_)[0],
     'female': np.histogram(age[~survived & female],
                            bins=bins_)[0]}
bins = bins_[:-1]
plt.figure(figsize=(15,8))
for i, sex, color in zip((0, 1),('male', 'female'), ('#3345d0',
'#cc3dc0')):
    plt.subplot(121 + i)
    plt.bar(bins, S[sex], bottom=D[sex], color=color,
            width=5, label='Survived')
    plt.bar(bins, D[sex], color='#aaaaff', width=5, label='Died',
alpha=0.4)
    plt.xlim(0, 80)
    plt.grid(None)

    plt.title(sex + " Survived")
    plt.xlabel("Age (years)")
    plt.legend()

(X_train, X_test, y_train, y_test) = cv.train_test_split(X, y, test_
```

```
size=.05)
print X_train, y_train

# Logistic Regression from linear_model
logreg = lm.LogisticRegression();
logreg.fit(X_train, y_train)
y_predicted = logreg.predict(X_test)

plt.figure(figsize=(15,8));
plt.imshow(np.vstack((y_test, y_predicted)),
           interpolation='none', cmap='bone');
plt.xticks([]); plt.yticks([]);
plt.title(("Actual and predicted survival outcomes on the test set"))
```

下面是一个展示了 Titanic 男性和女性生存者的线性回归：



我们已经看到，scikit-learn 有一个很好的机器学习函数集合。他们也自带一些标准数据集，比如，用于分类的 iris 数据集和 digits 数据集，以及用于回归的 Boston 房价数据集。机器学习是一个学习数据性质和将这些性质用到新数据集的过程。

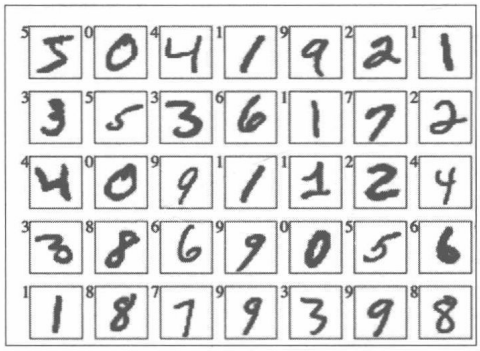
6.4 支持向量机

支持向量机（support vector machine，SVM）是可以用于回归或分类的有监督学习方法。这些学习方法是非线性模型的拓展，不仅从经验上提供了良好的性能，而且可以应用到很

多领域，比如生物信息学、文本、图像识别等。这些方法计算便宜且容易实现，但倾向于欠拟合和低准确率。

如何理解 SVM 的目标？其目标是映射或发现 x 和 y 间的一种模式，我们想要完成 $X \rightarrow Y$ ($x \in X$ 和 $y \in Y$) 的映射。这里，x 可以是一个对象，而 y 可以是一个标签。另一个简单的例子是，X 是一个 n- 维真实数值空间，而 y 是由 –1 和 1 组成的集合。

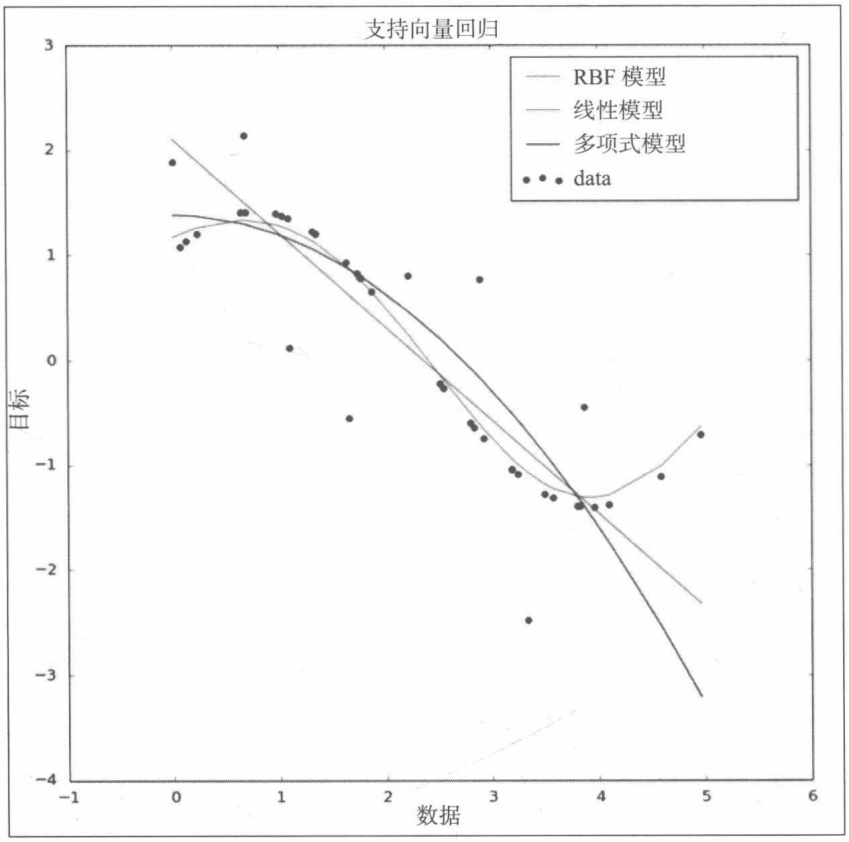
SVM 的一个经典的例子是：给出老虎和人的两张图，X 成为该组像素图像，而 Y 成为回答这个问题的标签，即给出一张未知图像："这是老虎还是人？"这里是字符识别问题的另一个例子：



网上有很多 SVM 的例子，但这里，我们将展示如何用 scikit-learn (sklearn) 将可视化方法运用到包括 SVM 的各种机器学习算法上。在 sklearn 中，sklearn.svm 软件包包括下面的 SVR 模型：

```
import numpy as np
from sklearn.svm import SVR
import matplotlib.pyplot as plt

X = np.sort(5 * np.random.rand(40, 1), axis=0)
y = (np.cos(X)+np.sin(X)).ravel()
y[::5] += 3 * (0.5 - np.random.rand(8))

svr_rbfmodel = SVR(kernel='rbf', C=1e3, gamma=0.1)
svr_linear = SVR(kernel='linear', C=1e3)
svr_polynom = SVR(kernel='poly', C=1e3, degree=2)
y_rbfmodel = svr_rbfmodel.fit(X, y).predict(X)
y_linear = svr_linear.fit(X, y).predict(X)
y_polynom = svr_polynom.fit(X, y).predict(X)

plt.figure(figsize=(11,11))
plt.scatter(X, y, c='k', label='data')
plt.hold('on')
plt.plot(X, y_rbfmodel, c='g', label='RBF model')
plt.plot(X, y_linear, c='r', label='Linear model')
```

```
plt.plot(X, y_polynom, c='b', label='Polynomial model')
plt.xlabel('data')
plt.ylabel('target')
plt.title('Support Vector Regression')
plt.legend()
plt.show()
```



6.5 主成分分析

主成分分析（principal component analysis，PCA）通过简单重组和转换来变换无标签数据的属性。对于没有任何意义的数据，我们可以寻找一些降维的方法加以处理。例如，当一个特定的数据集从某个特定角度沿轴线呈椭圆形分布时，如果变换一种方式沿 x 轴移动，而沿 y 轴没有变化迹象，那么就可能要忽略这些点。

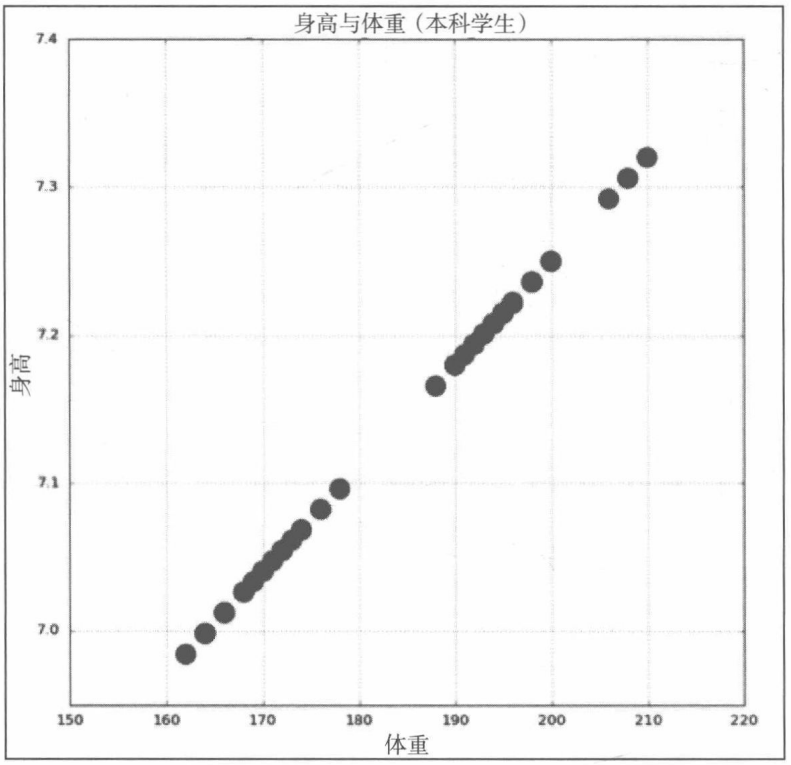
k- 均值聚类适用于聚类无标签数据。有时，可以用 PCA 将数据映射到更低维度，然后运用其他方法，比如 k- 均值，得到更小的维度更低的数据空间。

然而，因为任何降维都可能导致信息损失，所以认真完成降维很重要。而且算法在消除噪音时保留有用的数据部分是至关重要的。这里，我们将从两个角度研究 PCA，解释为

什么保留方差最大时才有意义：

- 相关性和冗余度
- 可视化

试想我们确实收集到了一个学校的学生数据，包括学生的性别、身高、体重、看电视时间、运动时间、学习时间、GPA 等。用这些维度调查这些学生时，我们发现身高和体重的相关性产生一个有趣的结论（通常，由于骨骼重量，学生越高，体重也越大，反之亦然）。这可能不是绝大多数情况（体重大不一定意味着长得高）。相关性也可以通过下面的代码进行可视化：




```
import matplotlib.pyplot as plt
import csv

gender=[]
x=[]
y=[]
with open('/Users/kvenkatr/height_weight.csv', 'r') as csvf:
  reader = csv.reader(csvf, delimiter=',')
  count=0
  for row in reader:
    if count > 0:
```

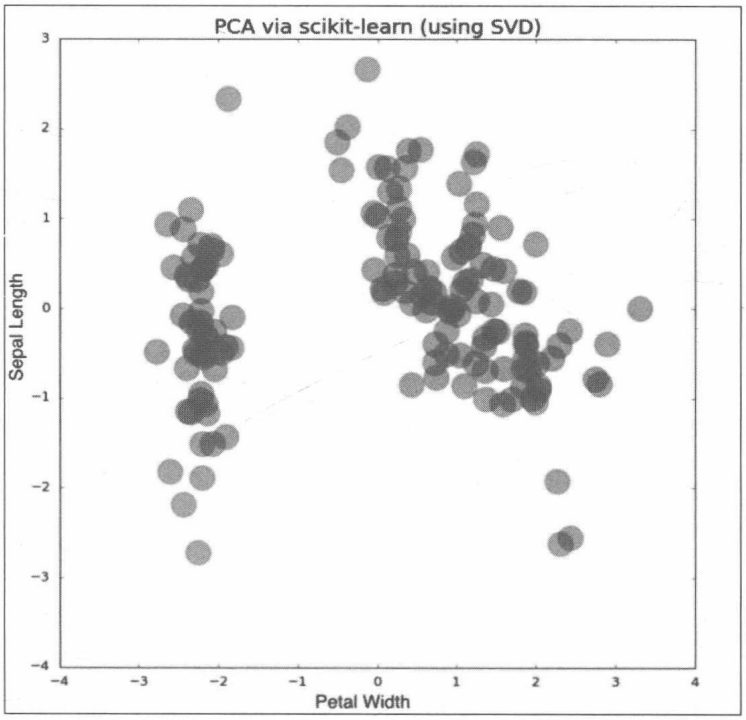
```
        if row[0] == "f": gender.append(0)
        else:  gender.append(1)
        height = float(row[1])
        weight = float(row[2])
        x.append(height)
        y.append(weight)
    count += 1

plt.figure(figsize=(11,11))
plt.scatter(y,x,c=gender,s=300)
plt.grid(True)
plt.xlabel('Weight', fontsize=18)
plt.ylabel('Height', fontsize=18)
plt.title("Height vs Weight (College Students)", fontsize=20)
plt.legend()

plt.show()
```

与 preprocessiny、datasets 和 decomposition 软件包一起，再次使用 sklearn，你可以编写一个简单的可视化代码，具体如下：

```
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

data = load_iris()
X = data.data

# convert features in column 1 from cm to inches
X[:,0] /= 2.54
# convert features in column 2 from cm to meters
X[:,1] /= 100
from sklearn.decomposition import PCA

def scikit_pca(X):

    # Standardize
    X_std = StandardScaler().fit_transform(X)

    # PCA
    sklearn_pca = PCA(n_components=2)
    X_transf = sklearn_pca.fit_transform(X_std)

    # Plot the data
    plt.figure(figsize=(11,11))
    plt.scatter(X_transf[:,0], X_transf[:,1], s=600, color='#8383c4',
alpha=0.56)
    plt.title('PCA via scikit-learn (using SVD)', fontsize=20)
    plt.xlabel('Petal Width', fontsize=15)
    plt.ylabel('Sepal Length', fontsize=15)
```

```
    plt.show()

scikit_pca(X)
```

以下图用 scikit-learn 软件包展示了 PCA：



安装 scikit-learn

下面是 scikit-learn 软件包的安装命令：

```
$ conda install scikit-learn
Fetching package metadata: ....
Solving package specifications: .
Package plan for installation in environment /Users/myhomedir/anaconda:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    nose-1.3.7                 |           py27_0         194 KB
    setuptools-18.0.1          |           py27_0         341 KB
```

```
    pip-7.1.0                 |           py27_0         1.4 MB
    scikit-learn-0.16.1       |       np19py27_0         3.3 MB
    ------------------------------------------------------------
                                          Total:         5.2 MB

The following packages will be UPDATED:

    nose:          1.3.4-py27_1       --> 1.3.7-py27_0
    pip:           7.0.3-py27_0       --> 7.1.0-py27_0
    scikit-learn: 0.15.2-np19py27_0 --> 0.16.1-np19py27_0
    setuptools:    17.1.1-py27_0      --> 18.0.1-py27_0

Proceed ([y]/n)? y
Fetching packages ...
```

对于 anaconda，正如 CLI 所有组件都通过 conda，它可以用 conda 安装。其他方法，我们总会尝试用默认的 pip 安装，然而，在任何情况下，我们应该检查安装文档。正如所有 scikit-learn 软件包已盛行了一段时间，其中并没有太大变化。在后面的章节中，我们将探索 k- 均值聚类来总结本章。

6.6 k- 均值聚类

k- 均值聚类源于信号处理，是数据挖掘中一种受欢迎的方法。k- 均值聚类的主要目的是发现一个数据集中的 m 个点，它们最能够代表数据集中 m- 群的中心。

k- 均值聚类也被认为是划分聚类。这意味着在聚类过程开始前需要指定类的个数。你可以定义一个目标函数：数据点和它最近类质心的欧几里得距离。可以按照一个系统过程来最小化目标函数，不断迭代找到能够降低目标函数的新类中心。

k- 均值聚类是聚类分析中一种受欢迎的方法。它不需要任何假设。这意味着当给定一个数据集、类的个数被预先设定为 k、而且运用 k- 均值算法，它能最小化距离的误差平方和。

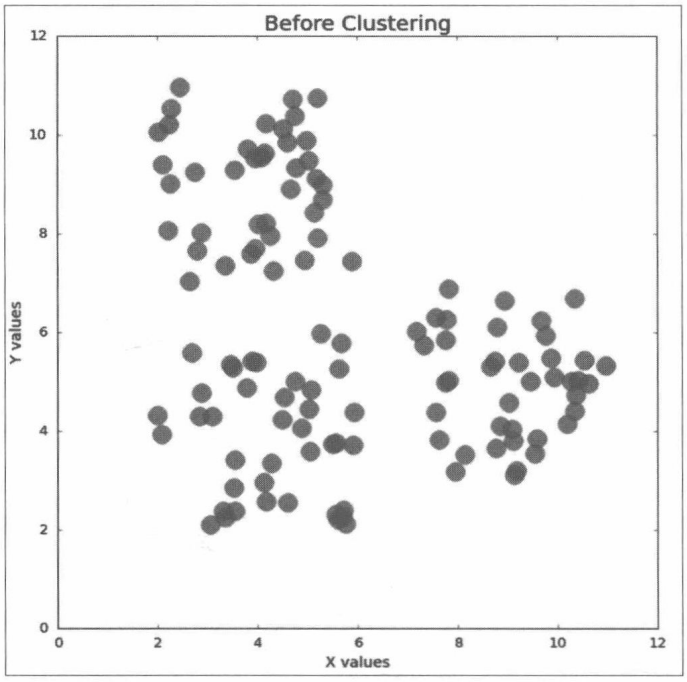
算法很容易理解，如下所示：

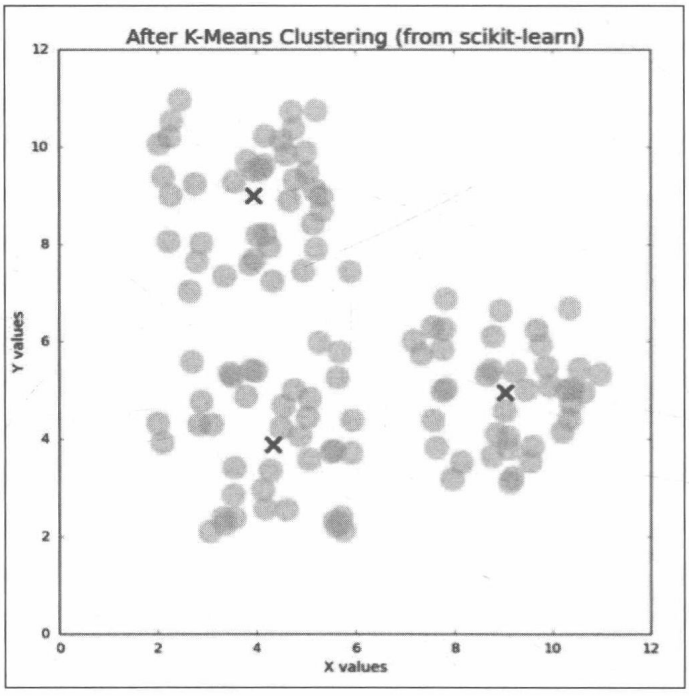
- 给定 n 个点和 k 个质心
- 对于每个 (x, y)，找到该点距离最近的质心（这确定了 (x, y) 所属的类）
- 在每一类中找到中位数，并将其设置为该类的质心，重复这个过程

使用 sklearn.cluster 软件包的 k- 均值方法，来看一个简单的例子（这可以用到大量点）。这个例子表明，用 scikit-learn 库编写最少的代码，可以实际 k- 均值聚类：

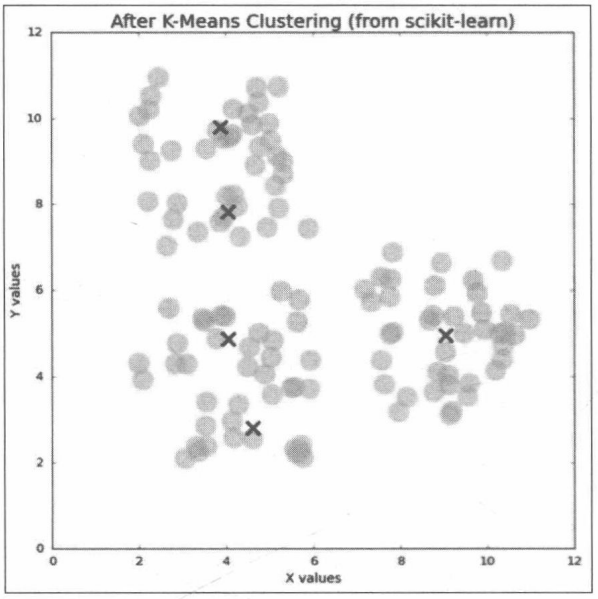
```
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

import csv

x=[]
y=[]

with open('/Users/myhomedir/cluster_input.csv', 'r') as csvf:
  reader = csv.reader(csvf, delimiter=',')
    for row in reader:
      x.append(float(row[0]))
      y.append(float(row[1]))

data=[]
for i in range(0,120):
  data.append([x[i],y[i]])

plt.figure(figsize=(10,10))

plt.xlim(0,12)
plt.ylim(0,12)

plt.xlabel("X values",fontsize=14)
plt.ylabel("Y values", fontsize=14)

plt.title("Before Clustering ", fontsize=20)

plt.plot(x, y, 'k.', color='#0080ff', markersize=35, alpha=0.6)

kmeans = KMeans(init='k-means++', n_clusters=3, n_init=10)
kmeans.fit(data)
plt.figure(figsize=(10,10))

plt.xlabel("X values",fontsize=14)
plt.ylabel("Y values", fontsize=14)

plt.title("After K-Means Clustering (from scikit-learn)", fontsize=20)

plt.plot(x, y, 'k.', color='#ffaaaa', markersize=45, alpha=0.6)

# Plot the centroids as a blue X
centroids = kmeans.cluster_centers_

plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=200,
  linewidths=3, color='b', zorder=10)

plt.show()
```

在聚类前绘制数据，如下图所示：





在这个例子中，如果我们用 k=5 表示 5 类，并且保持不变，那么其他两类一分为二以形成 5 类，如下图所示：



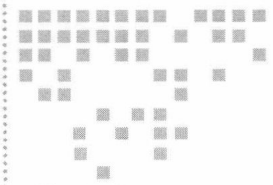
6.7 总结

本章举例说明了最受欢迎的机器学习算法。我们简单介绍了线性回归，并讨论了逻辑斯蒂回归。用大学录取标准的例子研究线性回归，用 Titanic 幸存者的例子研究逻辑斯蒂回归。本章也讲述了如何使用 statsmodels.formula.api、pandas 和 sklearn.linear_model 软件包来处理这些回归方法。在这些例子中，matplotlib 已用于可视化。

本章我们学习了决策树。以体育（高尔夫和网球）为案例，我们用 sklearn 和 pydot 软件包研究决策树，进而讨论了贝叶斯定理和朴素贝叶斯分类器。用 TextBlob 软件包和 nltk 库的电影评论数据，我们研究了用 wordcloud 软件包分析词云案例。

我们还学习了 k- 最近邻算法。这里，我们看到了基于权重和形状的水果分类案例，视觉上通过颜色加以区分。

我们还研究了形式最简单的 SVM，举出如何从 sklearn.svm 软件包生成数据、并用 matplotlib 库绘制结果的例子。我们学习了 PCA 以及确定冗余度消除变量的方法。我们用 sklearn.preprocesing 库分析 iris 数据来研究如何将结构可视化。最后，以用 sklearn.cluster 处理随机点为例，研究了 k- 均值聚类。这是最简单的聚类方法（最少代码）。下一章，我们将讨论各种生物信息学、遗传学和网络的案例。



Chapter 7 第 7 章

生物信息学、遗传学和网络模型

科学应用有多个黑匣子，这些匣子里面很复杂，通常认为是不可思议的。但是，它们都遵循一套系统的方法。这些方法在科研领域是众所周知的。例如，网络模型广泛用于呈现复杂结构数据，比如蛋白质网络、分子遗传学和化学结构。另一个有趣的科研领域是生物统计学。这是一个正在发展壮大的领域，近来在研究方面产生了重大的突破。

在生物界，有很多不同的复杂结构，比如 DNA 序列、蛋白质结构等。为了比较，让我们看一下这些结构中的一些未知元素。建立模型有助于视觉化展示它们。类似地，在图论或网络的应用中，能够将复杂图结构可视化本质上是有利的。

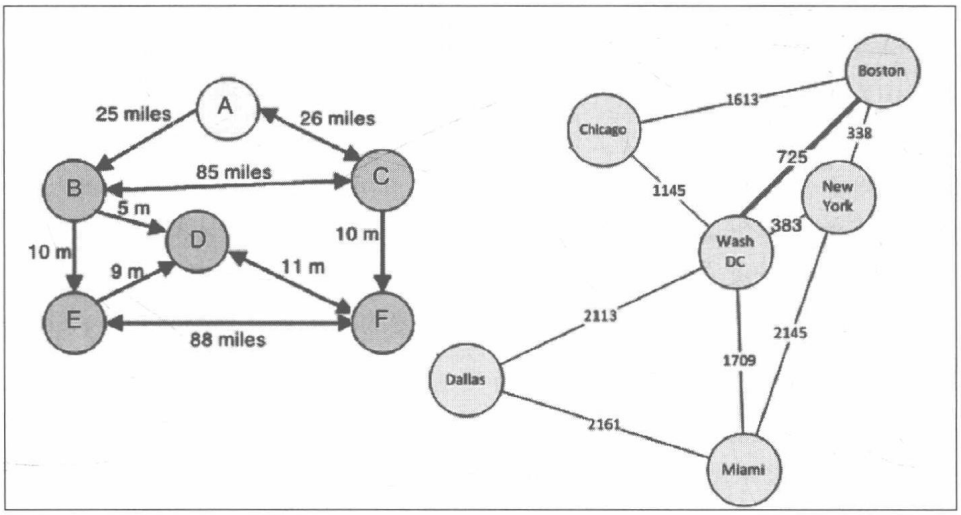
本章还将讨论一些有趣的例子，比如社交网络、现实生活中的有向图案例、适合于这些问题的数据结构和网络分析。为了演示这些例子，我们将用到特定的库，比如 metaseq、NetworkX、matplotlib、Biopython 和 ETE toolkit，包括以下主题：

- ❑ 有向图和多重图
- ❑ 图的聚集系数
- ❑ 社交网络分析
- ❑ 平面图测试和有向无环图测试
- ❑ 最大流量和最小切割
- ❑ 遗传编程案例
- ❑ 随机块模型和随机图

7.1　有向图和多重图

首先，我们将回顾有向图和多重图。随后，弄清楚在 Python 中如何生成这些图。我们也会看到一个可能需要有向图的例子。在从概念上描述图和有向图之前，让我们用不同方式理解何时用图和有向图。

同一所大学校园里相互连接的计算机可视为一个连接图。在这个图中，每台电脑视为一个节点或顶点。连线是一条边，在某些情况下，如果只有一个单向连接，那么它是一个有向图。例如，非常受限制的联邦网络不允许外界连接进去，但可能不会限制周围其他方式。下面是简单的图，展示不同地点间的距离：



在前面的案例中，图中城市标签 A 到 F 是一个有向图，右侧的另一个图是无向图。在有向图中，如果箭头指向两端，那么它是双向的，而在无向图中，则假定两种方式。如果我们用一些数据结构表示这些图，那会是什么？另外，如果要绘制这些图，我们会用到哪些库？如何完成？

7.1.1　存储图表数据

图表数据通常表示为一个邻接矩阵，除非它是稀疏的。一个邻接矩阵是一个有 V_2 行的矩阵，假定图有一个 V 顶点或节点。例如，对于前面图中所示的两个图表，邻接矩阵看起来与下表类似：

	A	B	C	D	E	F
A	0	25	26			
B		0	85	5	10	
C	26	85	0			10
D				0		11
E				9	0	88
F				11	88	0

	Chicago	Boston	New York	Wash DC	Miami	Dallas
Chicago	0	1613		1145		
Boston	1613	0	338	725		
New York		338	0	383	2145	
Wash DC	1145	725	383	0	1709	2113
Miami			2145	1709	0	2161
Dallas				2113	2161	0

无向图的对称性减少了一半的存储空间（不需要从 A 到 B 和从 B 到 A 存储所有信息）。空白条目展示了不够充足的距离数据。如果矩阵是稀疏的，即大多数条目没有填充，那么就可以存储为列表。幸运的是，scipy 中有很多便捷的方法来处理稀疏矩阵。下面的代码只是上图中的第一个图：

```
import scipy.sparse as sparse

matrixA = sparse.lil_matrix((6,6))

matrixA = sparse.lil_matrix( [[0,25,26,0,0,0], [0,0,85,5,10,0],
   [26,85,0,0,0,10], [0,0,0,0,0,11],[0,0,0,9,0,88],[0,0,0,11,88,0]])
print matrixA
(0, 1)  25
(0, 2)  26
(1, 2)  85
(1, 3)  5
(1, 4)  10
(2, 0)  26
(2, 1)  85
(2, 5)  10
(3, 5)  11
(4, 3)  9
(4, 5)  88
(5, 3)  11
(5, 4)  88
```

7.1.2　图表展示

上面的例子仅展示如何用 scipy 库（特别是 scipy.sparse 软件包）表示图表。然而，在后面的章节中，我们将看到如何展示这些图。尽管有众多可供选择的 Python 软件包来展示图，但最受欢迎的还是：NetworkX、igraph（从 igraph.org）和 graph-tool。让我们看看用这三个软件包展示图表的案例。

1. igraph

最初，igraph 是为 R 用户设计的，但后来，增加了 Python 版本。对于小一点的图，你可以容易地用增加顶点和边的方式来展示它们，但在大多数情况下，图都不小；因此，igraph 提供了从文件便捷地读取并展示图表数据的功能。

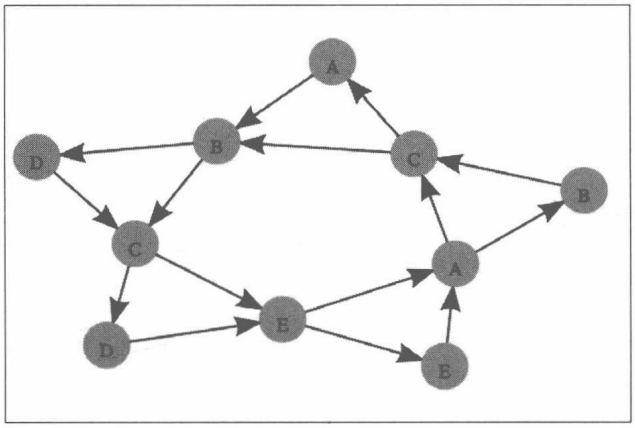
当前，igraph 提供一些格式，比如 dimacs、dl、edgelist、graml、graphdb、gml、lgl、ncol 和 pajek。GraphML 是一种基于 XML 的文件格式，能用于大型图。NCOL 图形格式适用于加权边列表的大型图。LGL 图形格式也能够用于大型加权图。大多数其他格式只用一种简单的文本格式。igraph 完全支持 DL 文件格式，但它只支持其他部分文件格式。

与很多其他 Python 软件包类似，igraph 的优点是它提供了非常便捷的方法来配置和展示图表，而且储存为 SVG 格式，因此，它们能被嵌入到一个 HTML 文件。

让我们看看一个涉及 pajek 格式的案例（要了解 pajek 的更多细节，你可以参考 http://vlado.fmf.uni-lj.si/pub/networks/pajek/）。这里有很多其他参数。其中有些是顶点形状的 labelcolor、vertexsize 和 radius。这里，我们看一下两个例子。第一个例子是一个有标签和边的小图，而第二个例子是从文件中读取一个图的数据，并展示出来。下面的例子展示了用 igraph 软件包的加标签的图：

```
from igraph import *

vertices = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J"]

edges = [(0,1),(1,2),(2,3),(3,4),(4,5),(5,6),(6,7),(7,1),
         (1,8),  (8,2),(2,4),(4,9),(9,5),(5,7),(7,0)]

graphStyle = { 'vertex_size': 20}
g = Graph(vertex_attrs={"label": vertices}, edges=edges,
directed=True)
g.write_svg("simple_star.svg", width=500, height=300, **graphStyle)
```

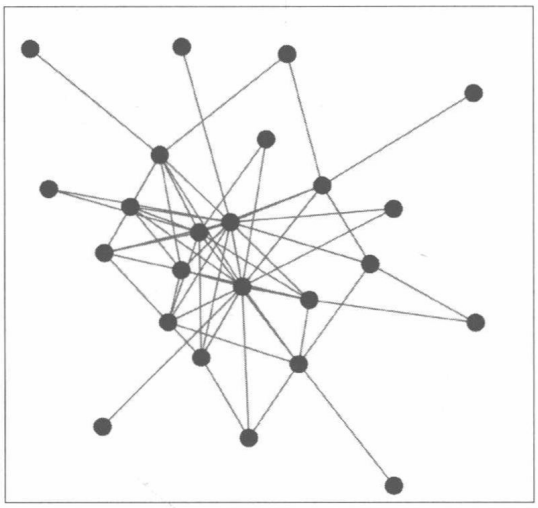
星形图中有 10 个顶点形成 5 个三角形和 1 个五边形。此外，也有 15 条边，因为 5 个三角组成边集。这是一个非常小的图，每条边通过从 0 开始相关联的顶点编号来定义。下面加标签的图是前面 Python 案例的结果：



第二个案例不仅说明了如何从文件读取图形数据，而且指出了怎样将图保存为 SVG 格式，这样你可以在 HTML 中嵌入 SVG 数据：

```
from igraph import read

g=read("ragusa.net",format="pajek")

g.vs["color"]="#3d679d"
g.es["color"]="red"

graphStyle={ 'vertex_size': 12, 'margin': 6}
#graphStyle["layout"]=g.layout("fr")  # optional

g.write_svg("ragusa_graph.svg", width=600, height=600,**graphStyle)
```

pajek 格式文件通过 igraph 的阅读功能进行读取。在设置边和顶点颜色时，你可以生成该图的 SVG 格式。你可以体验一些 igraph 提供的不同画图方法。下面展示了使用 igraph 软件包从文件中读取图形数据后绘制的图：



pajek 格式的图形数据可以从 pajek 网站上的 Rgausa16.net 文件获取 (http://vlado.fmf.uni-lj.si/pub/networks/pajek/data/gphs.htm)。一旦从这里下载数据文件，你可以以一种类似的方法使用，并展示为上图。如果我们用 tinamatr.net 数据，设置圆形布局，然后该图将以圆形布局显示，如下面的代码所示：

```
graphStyle["layout"]=g.layout("circle")
```

2. NetworkX

这个 Python 软件包称为 NetworkX 的原因是：它是一个网络和图形分析库。从寻找一个源节点或顶点到目的节点或顶点的最短路径，发现度的分布以区分与结节点类似的节点，发现一个图的聚集系数开始，总有一些方法来完成图形分析。

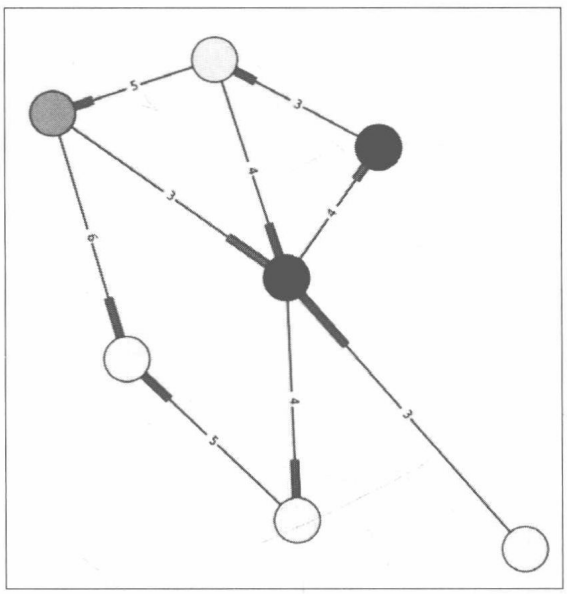
如今，图形研究已有一段时间，适用于神经生物学、化学、社交网络分析、网页排名，以及很多其他有趣的领域。在联合相似附属成员方面，社交网络是真正可选的，而生物网络却相反。换句话说，Facebook 用户或学会会员（合作者）间的友谊可以通过图形可视化。Python 软件包向用户提供了很多选项。通常，用户选择部分软件包来结合它们最好的单个功能。

NetworkX 提供图形构建和分析功能。你可以用标准和非标准的数据格式读取和编写网络数据、生成图形网络、分析它们的结构和构建一些模型。下面 Python 代码展示了如何只用 matplotlib 创建有向图：

```
import matplotlib.pyplot as plt
import pylab
from pylab import rcParams

import networkx as nx
import numpy as np

# set the graph display size as 10 by 10 inches
rcParams['figure.figsize'] = 10, 10

G = nx.DiGraph()

# Add the edges and weights
G.add_edges_from([('K', 'I'),('R','T'),('V','T')], weight=3)
G.add_edges_from([('T','K'),('T','H'),('I','T'),('T','H')], weight=4)
G.add_edges_from([('I','R'),('H','N')], weight=5)
G.add_edges_from([('R','N')], weight=6)
# these values to determine node colors
val_map = {'K': 1.5, 'I': 0.9, 'R': 0.6, 'T': 0.2}
values = [val_map.get(node, 1.0) for node in G.nodes()]

edge_labels=dict([((u,v,),d['weight'])
                  for u,v,d in G.edges(data=True)])

#set edge colors
red_edges = [('R','T'),('T','K')]
edge_colors = ['green' if not edge in red_edges else 'red' for edge in
```

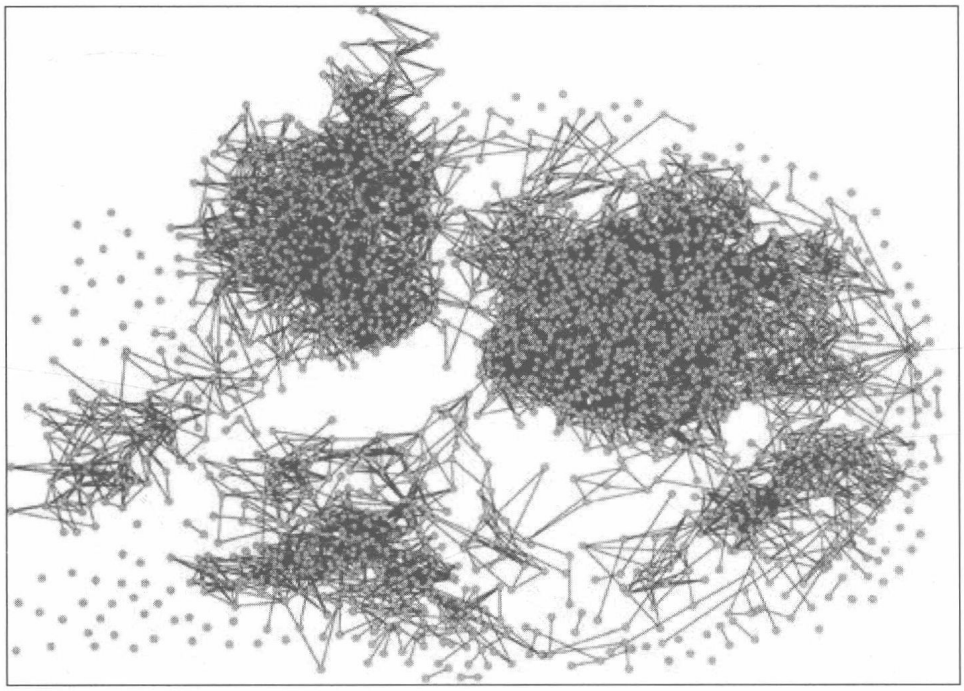
```
G.edges()]

pos=nx.spring_layout(G)

nx.draw_networkx_edges(G,pos,width=2.0,alpha=0.65)
nx.draw_networkx_edge_labels(G,pos,edge_labels=edge_labels)

nx.draw(G,pos, node_color = values, node_size=1500,
 edge_color=edge_colors, edge_cmap=plt.cm.Reds)

pylab.show()
```

下图说明如何用 NetworkX 形成边权重，实现符合视觉美学的图。在一些展示有向图的方法中，NetworkX 采取一种不同的方法：最后展示一个厚的条形，而不是确定图方向的箭头符号。



当我们对一组元素（事物或人）进行科学研究时，用图表示他们间的关联效果会更好，其中，这些元素是顶点或节点。大多数情况，中心性视觉上识别了显著重要的节点。Python 软件包（比如 NetworkX）有很多有用的图形分析（包括发现图中派系）函数。对于小一点的图，更容易视觉检查复杂细节，但对于大一点的图，你希望识别出一种行为模式，比如孤立簇群。

通常情况下，节点和边的标签与你试图展示为图的元素有关。比如，蛋白质交互作用可以展示为一个图。一个更复杂的案例是序列空间图，其中图的节点表示蛋白质序列，而边表示单个 DNA 突变。对于科学家来说，它变得更容易放大这些图像来观察模式，如下图所示。该案例不用 Python 而用交互式编程来放大和查看复杂的细节。



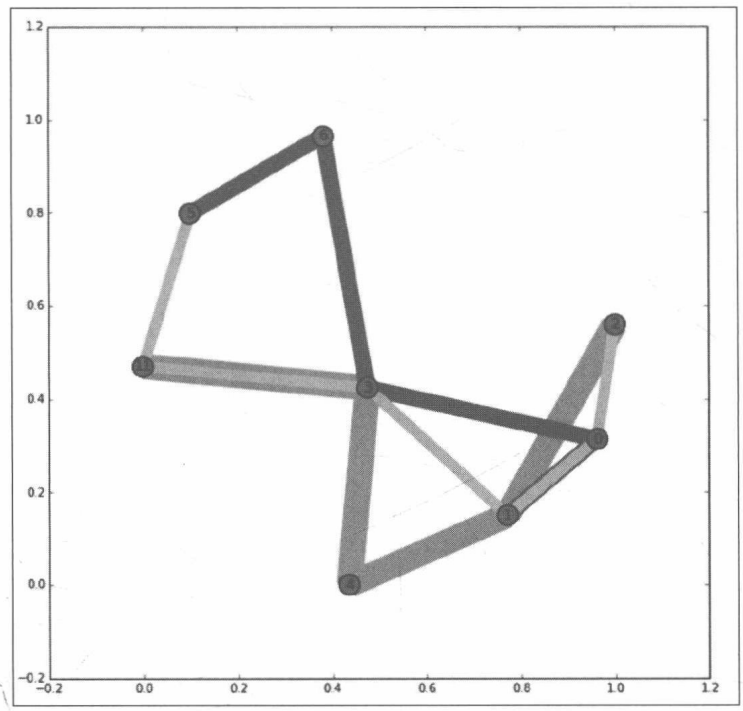
（上图可从 http://publications.csail.mit.edu/ 下载。）

有时，你会想要在一个图上突出不同路线。例如，如果要展示一个路线图，而且必须在地图上展示今年 Olympic 自行车队路线，你可以类似编写出如下代码：

```
import networkx as nx
from pylab import rcParams

# set the graph display size as 10 by 10 inches
rcParams['figure.figsize'] = 10, 10

def genRouteEdges(r):
    return [(r[n],r[n+1]) for n in range(len(r)-1)]

G=nx.Graph(name="python")
graph_routes = [[11,3,4,1,2], [5,6,3,0,1], [2,0,1,3,11,5]]
edges = []
for r in graph_routes:
    route_edges = genRouteEdges(r)
    G.add_nodes_from(r)
    G.add_edges_from(route_edges)
    edges.append(route_edges)

print("Graph has %d nodes with %d edges" %(G.number_of_nodes(),
G.number_of_edges()))
```

```
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G,pos=pos)
nx.draw_networkx_labels(G,pos=pos)

colors = ['#00bb00', '#4e86cc', 'y']
linewidths = [22,14,10]

for ctr, edgelist in enumerate(edges):
    nx.draw_networkx_edges(G,pos=pos,edgelist=edgelist,
      edge_color = colors[ctr], width=linewidths[ctr])
```

用 NetworkX 的一些便捷方法得到一个特定路线，你可以很容易地用不同的颜色和线的宽度来突出这些路径，如下图所示：



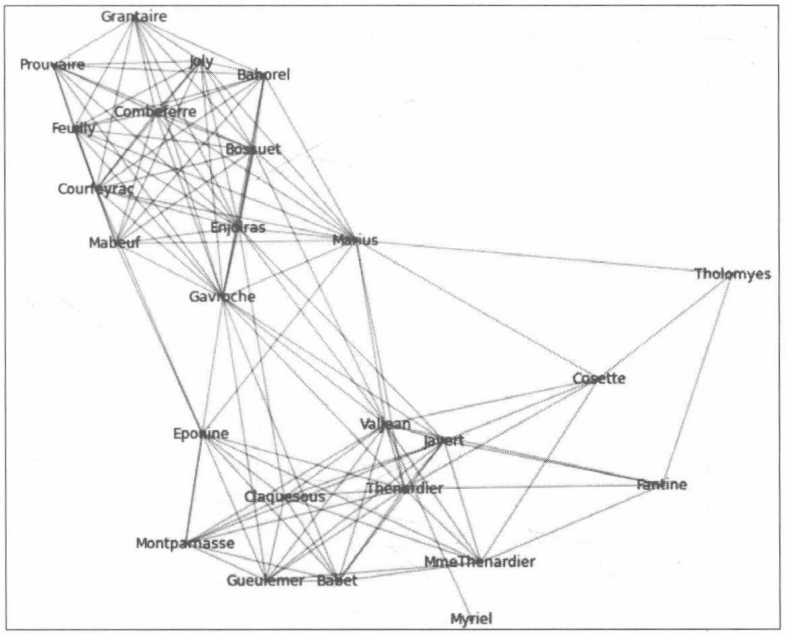
如上图所示，通过控制路线的亮点，你可以在同一个地图上识别不同的路线。

此外，NetworkX 提供了各种方法来完成图形分析，如从度的分布的最短路线到聚集系数。一种发现最短距离的简单方法可通过下面的代码实现：

```
import networkx as nx

g = nx.Graph()
g.add_edge('m','i',weight=0.1)
g.add_edge('i','a',weight=1.5)
g.add_edge('m','a',weight=1.0)
```

```
g.add_edge('a','e',weight=0.75)
g.add_edge('e','h',weight=1.5)
g.add_edge('a','h',weight=2.2)

print nx.shortest_path(g,'i','h')
nx.draw(g)

#printed shortest path as result
['i', 'a', 'h']
```

使用 NetworkX 的另一个案例（特别是读取 GML 格式的数据）是“小说《悲惨世界》中主人公的同时出现”，我们从 gephi.org 上下载数据集，网址是 https://gephi.org/datasets/lesmiserables.gml.zip。



上图是程序执行后的结果，它读取《悲惨世界》中相关联的主人公，创建了一个网络图，如下面的代码所示：

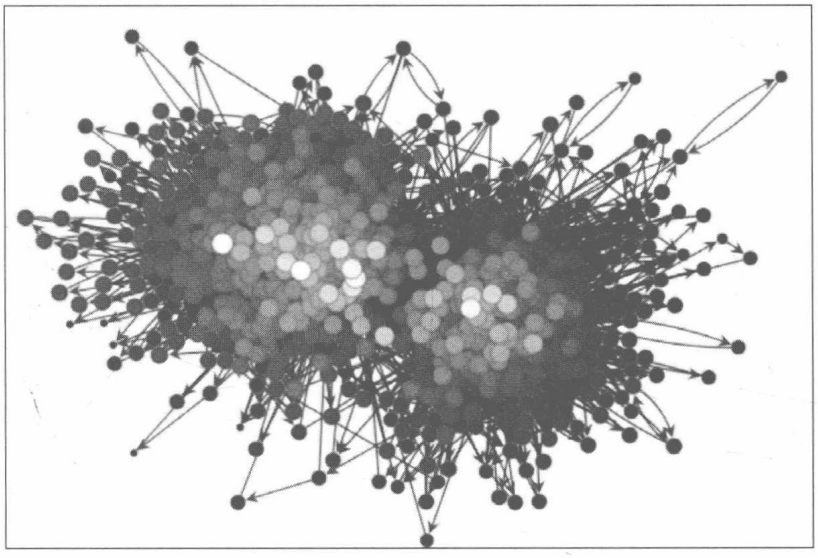
```
import networkx as nx
from pylab import rcParams
rcParams['figure.figsize'] = 12, 12

G = nx.read_gml('/Users/kvenkatr/Downloads/lesmiserables.gml',
relabel=True)
G8= G.copy()
dn = nx.degree(G8)
for n in G8.nodes():
  if dn[n] <= 8:
    G8.remove_node(n)
pos= nx.spring_layout(G8)
```

```
nx.draw(G8, node_size=10, edge_color='b', alpha=0.45, font_size=9,
pos=pos)
labels = nx.draw_networkx_labels(G8, pos=pos)
```

3. Graph-tool

在 igraph、networkx 和 graph-tool 这 3 个软件包中，graph-tool 软件包是最难安装的，尤其是在 Mac OS 系统上。Graph-tool 有很多便捷功能，用在中心性相关算法上也非常高效。这包括 k- 核、PageRank、最小生成树和单源最短路径。对比表可见 https://graph-tool.skewed.de/performance。前面提到的包括中心性相关算法的模块是 graph_tool.centrality。

```
import graph_tool.all as gtool

gr = gtool.collection.data["polblogs"]
gr = gtool.GraphView(gr, vfilt=gtool.label_largest_component(gr))

cness = gtool.closeness(gr)

gtool.graph_draw(gr, pos=gr.vp["pos"], vertex_fill_color=cness,
                 vertex_size=gtool.prop_to_size(cness, mi=5, ma=15),
                 vorder=cness, vcmap=matplotlib.cm.gist_heat,
                 output="political_closeness.pdf")
```



“centrality”（中心性）一词的前缀 centra（中心）真正意味着一些实体（在这种情况下，是一个节点或顶点）位于中心位置。此外，很多其他实体与中心实体相连。因此，我们可以提出一个合理的问题：哪些特征使一个顶点变得重要？在 graph_tool 模块中，提供了 9 种中心性相关算法，除了封闭性，PageRank 是其中一个算法。

PageRank

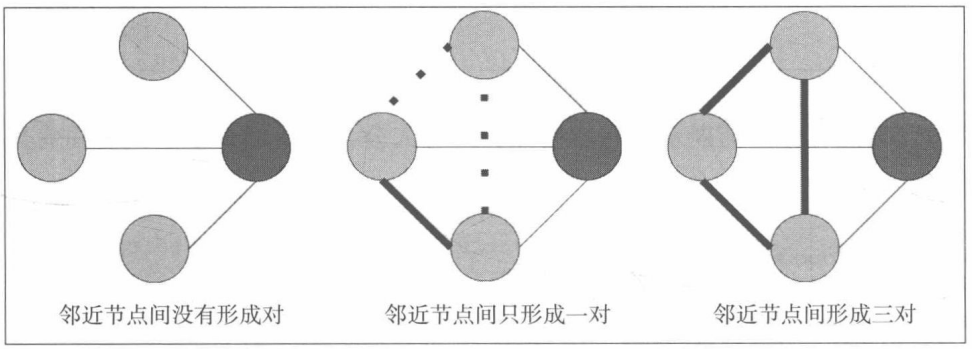
graph_tool.centrality .pagerank() 函数生成了 v 个顶点的 PageRank。大多数了解谷歌 PageRank 的人都理解这种测度如何起作用。简而言之，这是一种测度网页 A 有多重要的一种方法 (就依赖于网页 A 的外部 B 网站的数量，还有 A 依赖于多少网页——在图论当中，称为入度和出度)。此外，谷歌将很多外部因素应用到网页排序。在前面的例子中，如果我们用 PageRank 代替那些发现封闭性的代码，如下所示：

```
pagerank = gtool.pagerank(gr)
```

这应该生成一个强调 PageRank 的图。除了中心性测度之外，也有另一个因素——称为图的聚集系数。

7.2　图的聚集系数

图中一个节点或顶点的聚集系数依赖于邻近节点的接近程度。当接近程度达到一定水平时，他们就会形成一个派系（或一个小型完备图），如下图所示：



聚集系数有一个著名的公式，看起来由很多数学符号组成。然而，为了简化，请看下面的公式：

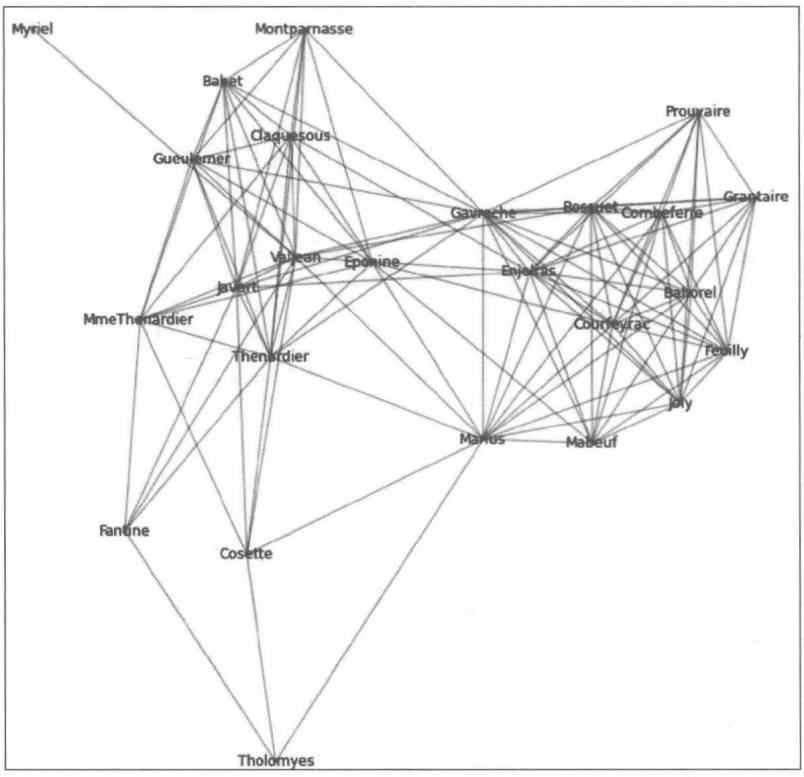
$$C_i = \frac{2\times(\text{节点 } i \text{ 的链接})}{n_b(n_b-1))}$$

n_b 是与节点 i 邻接的节点数

这包括记录每个顶点的链接，计算每个顶点的聚集指数，其中一个节点最明显的邻接节点只有一个链接远离该节点。聚集指数计算为：

$$C_i = \frac{2\times(\text{节点 } i \text{ 的链接})}{n_b(n_b-1))}$$

n_b 是与节点 i 邻接的节点数



下面的代码说明你如何展示《悲惨世界》的主人公，以及每一个主人公如何与其他主人公相连：

```
import networkx as nx
from pylab import rcParams
rcParams['figure.figsize'] = 12, 12

G = nx.read_gml('/Users/kvenkatr/Downloads/lesmiserables.gml',
relabel=True)
G8= G.copy()

dn = nx.degree(G8)

for n in G8.nodes():
  if dn[n] <= 8:
    G8.remove_node(n)

pos= nx.spring_layout(G8)
nx.draw(G8, node_size=10, edge_color='b', alpha=0.45, font_size=9,
pos=pos)
labels = nx.draw_networkx_labels(G8, pos=pos)

def valuegetter(*values):
```

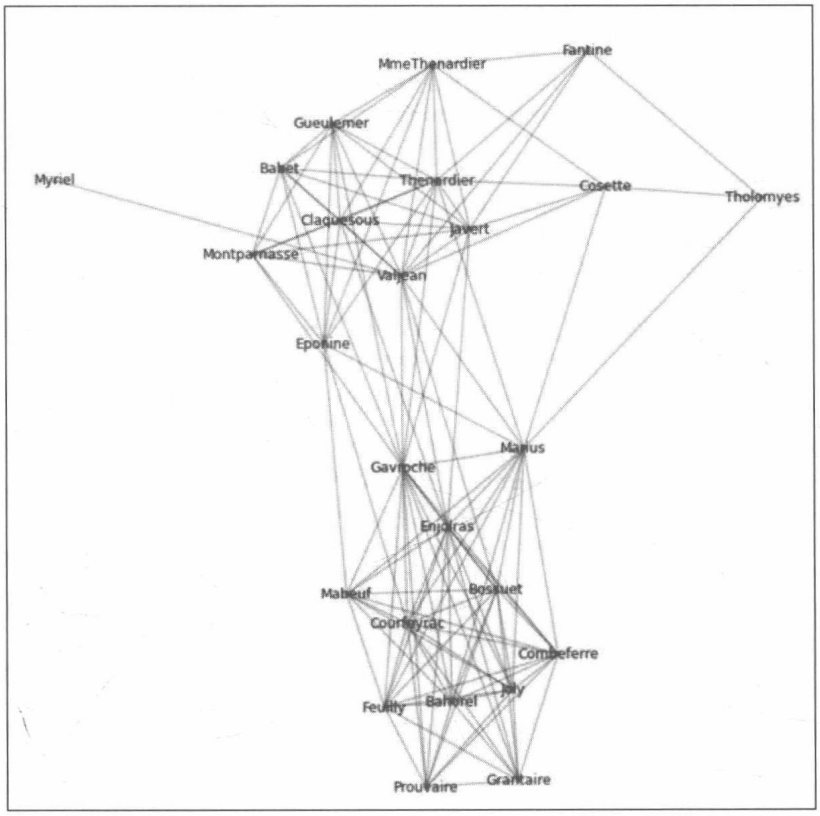
```
    if len(values) == 1:
        item = values[0]
        def g(obj):
            return obj[item]
    else:
        def g(obj):
            return tuple(obj[item] for item in values)
    return g

def clustering_coefficient(G,vertex):
    neighbors = G[vertex].keys()
    if len(neighbors) == 1: return -1.0
    links = 0
    for node in neighbors:
        for u in neighbors:
            if u in G[node]: links += 1
    ccoeff=2.0*links/(len(neighbors)*(len(neighbors)-1))
    return links, len(neighbors),ccoeff

def calculate_centrality(G):
    degc = nx.degree_centrality(G)
    nx.set_node_attributes(G,'degree_cent', degc)
    degc_sorted = sorted(degc.items(), key=valuegetter(1),
reverse=True)
    for key, value in degc_sorted[0:10]:
        print "Degree Centrality:", key, value
    return G, degc

print "Valjean", clustering_coefficient(G8,"Valjean")
print "Marius", clustering_coefficient(G8,"Marius")
print "Gavroche", clustering_coefficient(G8,"Gavroche")
print "Babet", clustering_coefficient(G8,"Babet")
print "Eponine", clustering_coefficient(G8,"Eponine")
print "Courfeyrac", clustering_coefficient(G8,"Courfeyrac")
print "Comeferre", clustering_coefficient(G8,"Combeferre")
calculate_centrality(G8)
```

上面的代码有两种结果：第一部分是输出文本，第二部分是绘制网络图，如下面的代码和图所示：

```
#Text Results printed
Valjean (82, 14, 0.9010989010989011)
Marius (94, 14, 1.032967032967033)
Gavroche (142, 17, 1.0441176470588236)
Babet (60, 9, 1.6666666666666667)
Eponine (36, 9, 1.0)
Courfeyrac (106, 12, 1.606060606060606)
Comeferre (102, 11, 1.8545454545454545)
```

```
Degree Centrality: Gavroche 0.708333333333
Degree Centrality: Valjean 0.583333333333
Degree Centrality: Enjolras 0.583333333333
Degree Centrality: Marius 0.583333333333
Degree Centrality: Courfeyrac 0.5
Degree Centrality: Bossuet 0.5
Degree Centrality: Thenardier 0.5
Degree Centrality: Joly 0.458333333333
Degree Centrality: Javert 0.458333333333
Degree Centrality: Feuilly 0.458333333333
```

图形结果如下图所示：



显然，到目前为止，我们已经发现 Comeferre 会有较大的聚集系数 (0.927)。通常，在我们绘制一个大的二维图时，不容易看到聚集系数。

7.3 社交网络分析

几年前，从社交网络（比如 LinkedIn、Facebook 或 Twitter）获得数据比现在更简单、

更容易。现在，大多数 API 都有限制。此外，数据获取的方法更多了。首先，必须获得认证（更早时使用过），然后借助一些方法访问其他朋友或链接。这里，我们只选择 Twitter 进行社交网络数据的分析，但你也能够用一种类似的方法发现其他社交媒体数据。

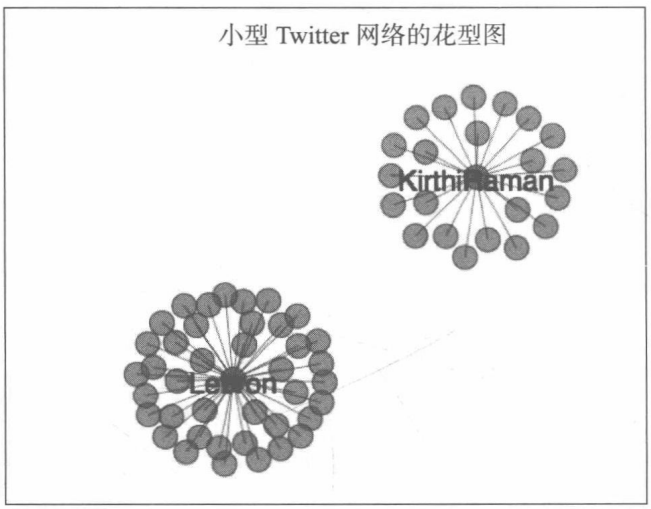
为了得到 Twitter 数据，正如我们从前面的章节中注意到的（当我们讨论词云时），你不得不获得认证密钥来得到他们的 API。有四种密钥：CONSUMER_KEY、CONSUMER_SECRET、ACCESS_TOKEN_KEYS 和 ACCESS_TOKEN_SECRET。一旦这些凭据成功地通过 Python 验证，你可以调用 GetFriends() 和 GetFollowers() 来获得朋友和追随者的名单。在 Python 中，有很多软件包可以用来获得 Twitter 数据。因此，选择使用哪些软件包会令人困惑。在过去的案例中，我们已经用过 tweepy。这里我们将用 Python-Twitter，因为它有便捷的模块来得到数据、进行总结，并储存在 cPickle 中，然后进行可视化，如以下代码：

```
import cPickle
import os
import twitter  # https://github.com/ianozsvald/python-twitter

# Usage:
# $ # setup CONSUMER_KEY, CONSUMER_SECRET, ACCESS_TOKEN_KEY, ACCESS_
TOKEN_SECRET
# as environment variables
# $ python get_data.py  # downloads friend and follower data to ./data

# Errors seen at runtime:
# raise URLError(err)
# urllib2.URLError: <urlopen error [Errno 104] Connection reset by
peer>

DATA_DIR = "data"  # storage directory for friend/follower data

# list of screen names that we'll want to analyze
screen_names = [ 'KirthiRaman', 'Lebron' ]

def get_filenames(screen_name):
    """Build the friends and followers filenames"""
    return os.path.join(DATA_DIR, "%s.friends.pickle" % (screen_
name)), os.path.join(DATA_DIR, "%s.followers.pickle" % (screen_name))

if __name__ == "__main__":

    # deliberately stripped my keys
    t = twitter.Api(consumer_key='k7atkBNgoGrioMS...',
                    consumer_secret='eBOx1ikHMkFc...',
                    access_token_key='8959...',
                    access_token_secret='O7it0...');

    print t.VerifyCredentials()

    for screen_name in screen_names:
```

```
fr_filename, fo_filename = get_filenames(screen_name)
print "Checking for:", fr_filename, fo_filename
if not os.path.exists(fr_filename):
    print "Getting friends for", screen_name
    fr = t.GetFriends(screen_name=screen_name)
    cPickle.dump(fr, open(fr_filename, "w"), protocol=2)
if not os.path.exists(fo_filename):
    print "Getting followers for", screen_name
    fo = t.GetFollowers(screen_name=screen_name)
    cPickle.dump(fo, open(fo_filename, "w"), protocol=2)
```

朋友和追随者信息被无序的堆叠到 cPickle 中。如 https://github.com/ianozsvald/python-twitter 中的解释，你可以运行下面的代码：

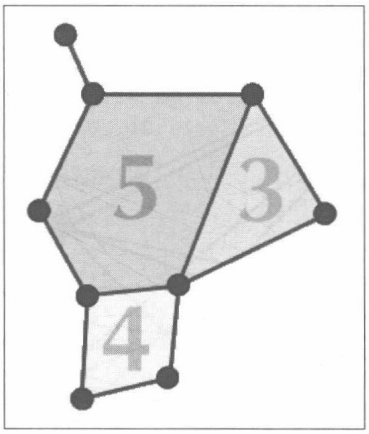
```
python get_data.py
python summarise_data.py
python draw_network.py
```



7.4 平面图测试

平面图可以画在一个没有任何交叉边的平面上。你要从一个顶点开始绘图，从边画到边，而且随绘图过程的继续，保持面轨迹。根据 Kuratowski，一个图如果不包含子图（这里子图是指 5 个顶点组成的完备图的一部分），那么它就是平面图。

下面是一个平面图的简单案例：



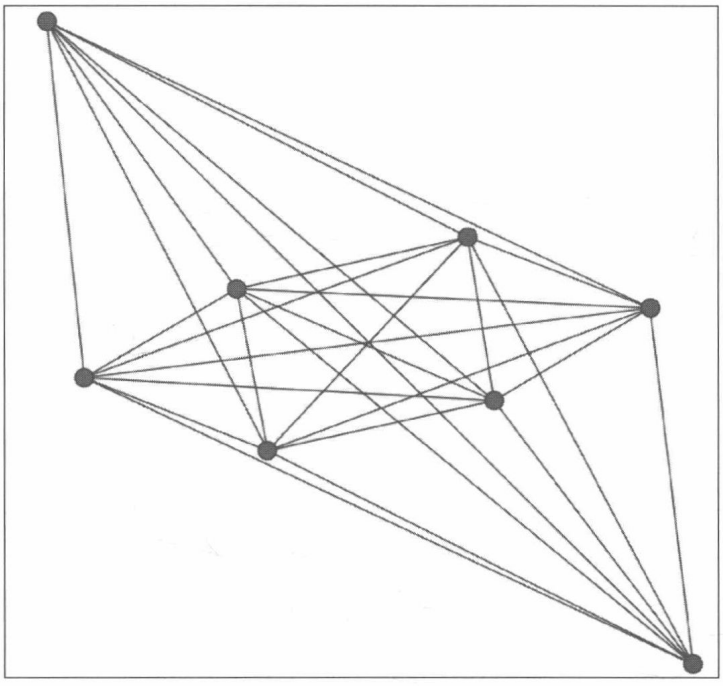
欧拉公式连接了一些顶点、边和面。根据欧拉公式，如果有限个、连接的平面图画在没有交叉边的平面上，而且 v 表示顶点的数量，e 表示边的数量，f 表示面的数量，那么就有：$v-e+f=2$。

除了 Mayavi、NetworkX 和 planarity 外，你还可以用 gamera 软件包创建和展示图形。然而，gamera 仅在 Windows 系统上可用。我们给出一个用 planarity 和 NetworkX 的简单的例子：

```
import planarity
import networkx as nx

# complete graph of 8 nodes, K8
G8=nx.complete_graph(8)

# K8 is not planar
print(planarity.is_planar(G8))

# Will display false because G8 is not planar subgraph
K=planarity.kuratowski_subgraph(G8)

# Will display the edges
print(K.edges())

#Will display the graph
nx.draw(G8)
```

```
False
[(0, 4), (0, 5), (0, 7), (2, 4), (2, 5), (2, 7), (3, 5), (3, 6), (3,
7), (4, 6)]
```

这个例子说明下面 8 个节点的完整图不是平面图：



上图表示一个 8 个节点的平面图，这可能看起来很凌乱，因此，图的节点越多，看起来就越复杂。

7.5 有向无环图测试

首先，让我们看看什么是有向无环图（directed acyclic graph, DAG）。一个有向无环图是有向图，这意味着给出顶点 A 到 B 的边将指向特定方向（A->B 或 B->A），而且是无环的。无环图是那些没有循环的图，这也意味着没有圆环（它们没有沿圆环进行连接）。

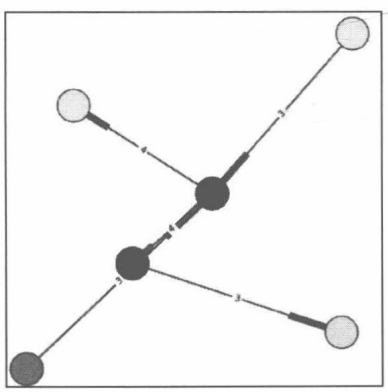
有什么好的 DAG 例子？一棵树，甚至一个线索。我们都知道它们是什么，因为这在本书前面章节中讨论了。用线索的一个案例是存储字典的词汇，而且有一个拼写检查算法。我们不必讨论细节，但在可视化背景下检查一个图是否无环，我们能确定提供无环图测试方法的 Python 软件包。

NetworkX 有一个便捷的函数，称为 is_directed_acyclic_graph (Graph)。这里给出一个无环图的案例，用这个函数，我们将测试它是否返回正确：

```
import matplotlib.pyplot as plt
import pylab
from pylab import rcParams
```

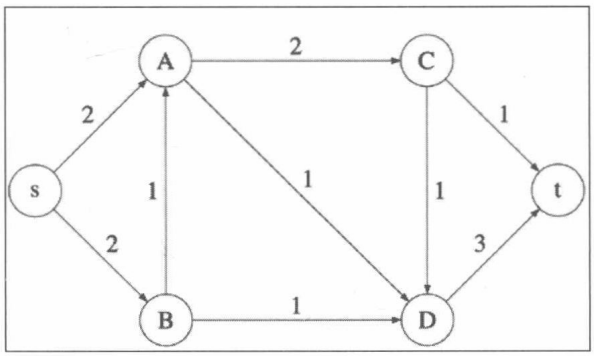
```
import networkx as nx
import numpy as np

# set the graph display size as 10 by 10 inches
rcParams['figure.figsize'] = 10, 10

G = nx.DiGraph()

# Add the edges and weights
G.add_edges_from([('K', 'I'),('R','T'),('V','T')], weight=3)
G.add_edges_from([('T','K'),('T','H'),('T','H')], weight=4)
# these values to determine node colors
val_map = {'K': 1.5, 'I': 0.9, 'R': 0.6, 'T': 0.2}
values = [val_map.get(node, 1.0) for node in G.nodes()]

edge_labels=dict([((u,v,),d['weight'])
                  for u,v,d in G.edges(data=True)])

#set edge colors
red_edges = [('R','T'),('T','K')]
edge_colors = ['green' if not edge in red_edges else 'red' for edge in
G.edges()]

pos=nx.spring_layout(G)

nx.draw_networkx_edges(G,pos,width=2.0,alpha=0.65)
nx.draw_networkx_edge_labels(G,pos,edge_labels=edge_labels)

nx.draw(G,pos, node_color = values, node_size=1500,
 edge_color=edge_colors, edge_cmap=plt.cm.Reds)

pylab.show()
nx.is_directed_acyclic_graph(G)

True
```

该例中的无环图如下图所示：



7.6 最大流量和最小切割

一个流量网络是一个从源头到终点的、沿每条边分配能力的有向图。正如我们可以将一个街道地图建立为一个有向图。为了发现从一个地点到另一个地点的最短路径，我们也可以将一个有向图解释为一个“流量网络”。流量网络的例子有：液体通过管道流动，电流通过电网传递和数据通过通讯网络传送。下面是一个流量图的例子：

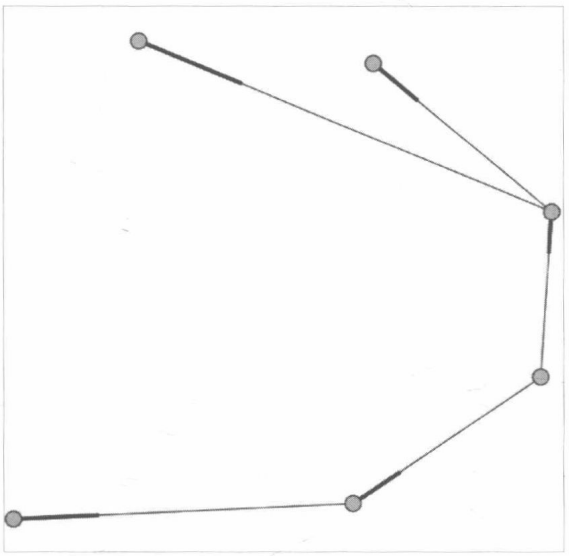


G 图的边被赋予不同的流量的承载能力。如果这种能力不被呈现，则假定能力是有限的。这里流量网络 G 的最大流量是 4。

在 NetworkX 软件包中，maximum_flow_value(Graph, from, to) 函数评价了图的最大流量，如下面的代码所示：

```
import networkx as nx
G = nx.DiGraph()
G.add_edge('p','y', capacity=5.0)
G.add_edge('p','s', capacity=4.0)
G.add_edge('y','t', capacity=3.0)
G.add_edge('s','h', capacity=5.0)
G.add_edge('s','o', capacity=4.0)

flow_value = nx.maximum_flow_value(G, 'p', 'o')

print "Flow value", flow_value
nx.draw(G, node_color='#a0cbe2')

Flow value 4.0
```

上面代码的图正在为 maximum_flow_value 测试，而且该图展示如下：



7.7　遗传编程示例

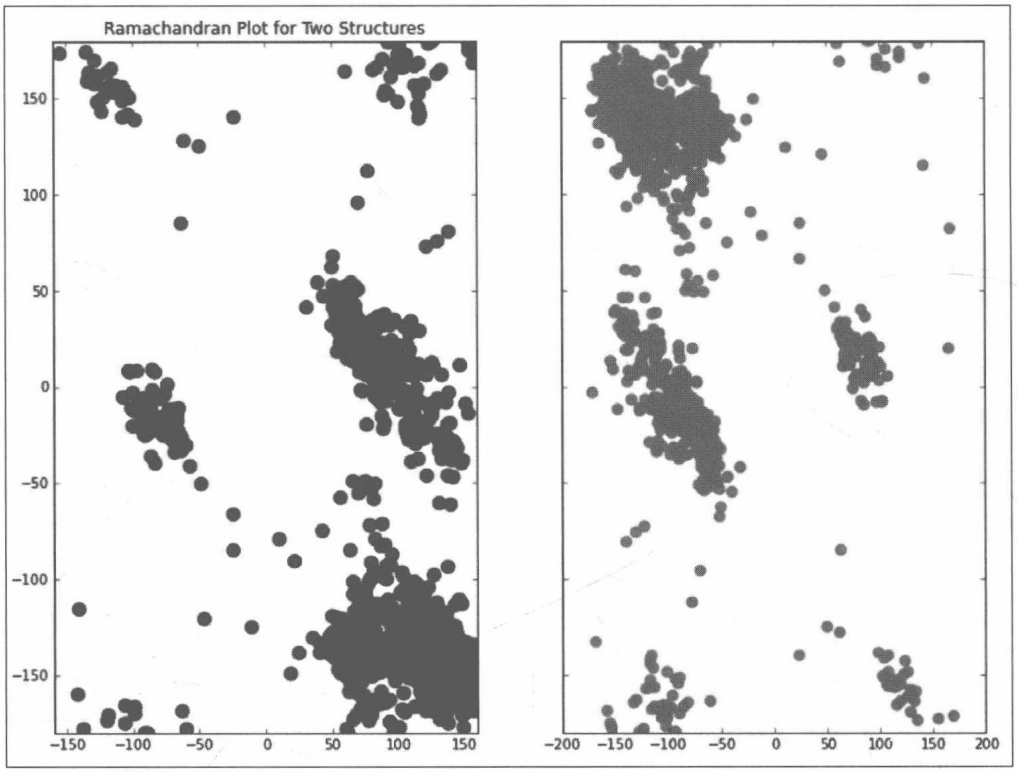
CnvKit 也是可用的，但它是一个命令行工具，而且不容易使用。此外，由全国健康研究所（National Institutes of Health，NIH）的 NCBI 的研究员研发的 PyCogent 也是一种有用的工具。然而，它们也不容易使用。我们将用一个名为 Bio 的软件包（https://github.com/biopython/biopython/tree/master/Bio）和来自 Python programming for biology 的库。

一般情况下，每一次实验、研究项目或研究将序列作为生物信息学中所使用的关键对象。作为一名数学家，我认为序列的可视化思维与某些模式的字符串（比如 ATAGCATATGCT）有关。这里列举一个简单的例子展示一个序列、GC 比率和密码子：

```
from Bio.Seq import Seq
from Bio.Alphabet import IUPAC
from Bio.SeqUtils import GC

def DNACodons(seq):
    end = len(seq) - (len(seq) % 3) - 1
    codons = [seq[i:i+3] for i in range(0, end, 3)]
    return codons DNACodons(my_seq)
my_seq = Seq('GGTCGATGGGCCTAGCAGCATATCTGAGC', IUPAC.unambiguous_dna)
print "GC Result==>", GC(my_seq)

DNACodons(my_seq)
[Seq('GGT', IUPACUnambiguousDNA()),
 Seq('CGA', IUPACUnambiguousDNA()),
 Seq('TGG', IUPACUnambiguousDNA()),
 Seq('GCC', IUPACUnambiguousDNA()),
 Seq('TAG', IUPACUnambiguousDNA()),
```

```
 Seq('CAG', IUPACUnambiguousDNA()),
 Seq('CAT', IUPACUnambiguousDNA()),
 Seq('ATC', IUPACUnambiguousDNA()),
 Seq('TGA', IUPACUnambiguousDNA())]

GC Result==> 58.6206896552
```

让我们考虑两种分子结构，收集一定的原子，并且尽量用它们的 Phi 和 Psi 角绘制它们的位置。这些分子结构是 DNA、RNA 和蛋白质。用 PythonForBiology 库中的 Modeling 和 Maths 模块，我们试图绘制这些结构，并挨着排列：



这两个图用到的数据来自于两个文件：testTransform.pdb 和 1A12.pub。它包含人类的 regulator of chromosome condensation (RCC1)，代码如下所示：

```
# bio_1.py
#
import matplotlib.pyplot as plt
from phipsi import getPhiPsi
from Modelling import getStructuresFromFile

def genPhiPsi(fileName):
  struc = getStructuresFromFile(fileName)[0]
```

```
    phiList = []
    psiList = []
    for chain in struc.chains:
      for residue in chain.residues[1:-1]:
        phi, psi = getPhiPsi(residue)
        phiList.append(phi)
        psiList.append(psi)

    return phiList, psiList

if __name__ == '__main__':

    phiList = []
    psiList = []
    phiList, psiList = genPhiPsi('examples/testTransform.pdb')

    phiList2 = []
    psiList2 = []
    phiList2, psiList2 = genPhiPsi('examples/1A12.pdb')

    plt.figure(figsize=(12,9))
    f, (ax1, ax2) = plt.subplots(1, 2, sharey=True, figsize=(12,9))

    ax1.scatter(phiList, psiList, s=90, alpha=0.65)
    ax1.axis([-160,160,-180,180])
    ax1.set_title('Ramachandran Plot for Two Structures')
    ax2.scatter(phiList2, psiList2, s=60, alpha=0.65, color='r')
    plt.show()
```

这个例子用到的库将可用于文件 PythonForBiology.zip 中的代码示例。假设已经安装了 numpy 和 matplotlib，你就可以通过一行命令压缩并运行该代码。

7.8　随机区组模型

在前面的章节中，我们已经用蒙特卡洛模拟讨论了随机模型。现在，我们讨论的是图和网络。因此，单从内容上来说，一个社群结构也可视为一个图。在这样的图中，节点经常聚在一起，形成密集连接的子图。在一般情况下，两个这样节点间边的概率是节点所属类别的函数。

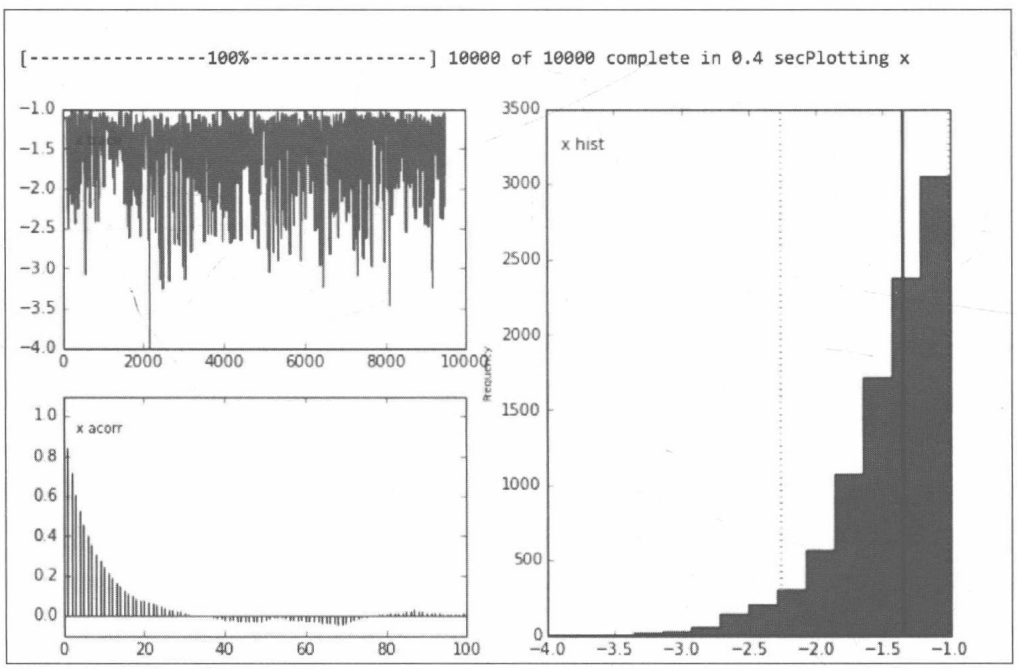
这种网络划分的一个受欢迎选择是随机区组模型。随机区组模型的一个简单定义是通过标量 n 描述图的特点。它表示群组的数量或类别的数量，以及一个展示节点及他们连接的矩阵。读者可以参考统计学的书籍获得更严格的数学定义。

在一些支持随机模型的 Python 软件包中，PyMC 提供了 Markov Chain Monte Carlo（MCMC）和概率模型的 3 个基石，比如随机的、确定的和可能的。除 PyMC 外，还有另

一个构建有趣的随机建模的软件包 StochPy。SSA 模块特别提供了一些便捷的方法（http://stochpy.sourceforge.net/examples.html）。第一个例子用 pymc 的正态分布展示一张复杂的图，另一个例子用了 MCMC 模型，如下面的代码所示：

```
import pymc as mc

from pylab import rcParams

# set the graph display size as 10 by 10 inches
rcParams['figure.figsize'] = 12, 12
z = -1.

#instead of 0 and 1, some unknown mu and std goes here:
X = mc.Normal( "x", 0, 1, value = -3. )

#Here below, one can place unknowns here in place of 1, 0.4
@mc.potential
def Y(x=X, z=z):
  return mc.lognormal_like( z-x, 1, 0.4,  )

mcmc = mc.MCMC([X])
mcmc.sample(10000,500)
mc.Matplot.plot(mcmc)
```

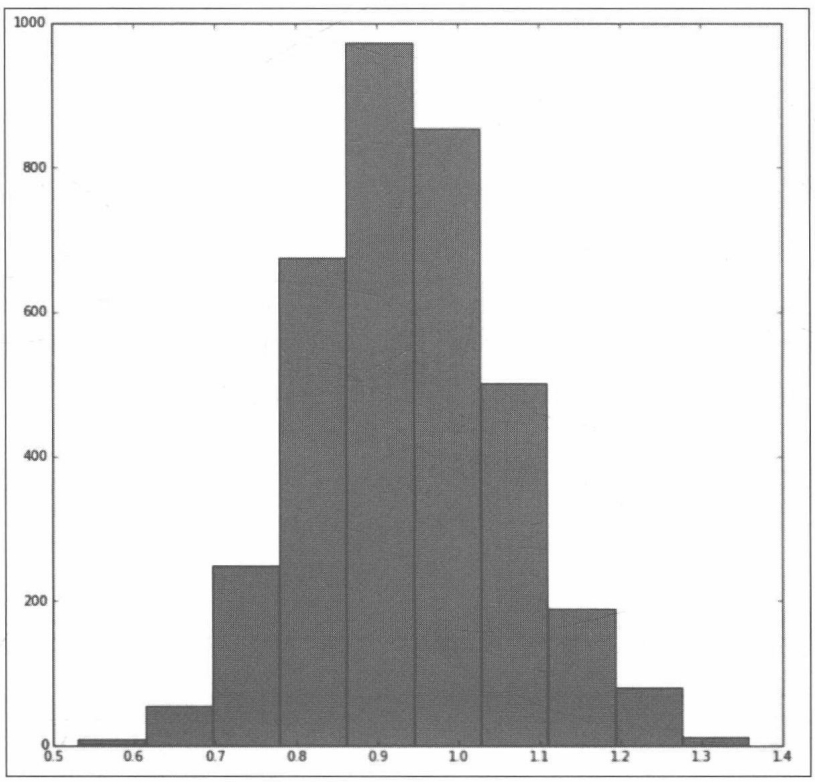
这里给出的例子说明你如何用非常少的代码展示一个复杂模型：



在 PyMC 中有 disaster-model 的案例，通过 MCMC 和 50 000 次简单迭代，模型展示如下：

```
from pymc.examples import disaster_model
from pymc import MCMC

from pylab import hist, show, rcParams

rcParams['figure.figsize'] = 10, 10

M = MCMC(disaster_model)
M.sample(iter=65536, burn=8000, thin=16)

hist(M.trace('late_mean')[:], color='#b02a2a')

show()
```

如果我们要展示模型均值的直方图，那么这就是用 PyMC 的一种选项：

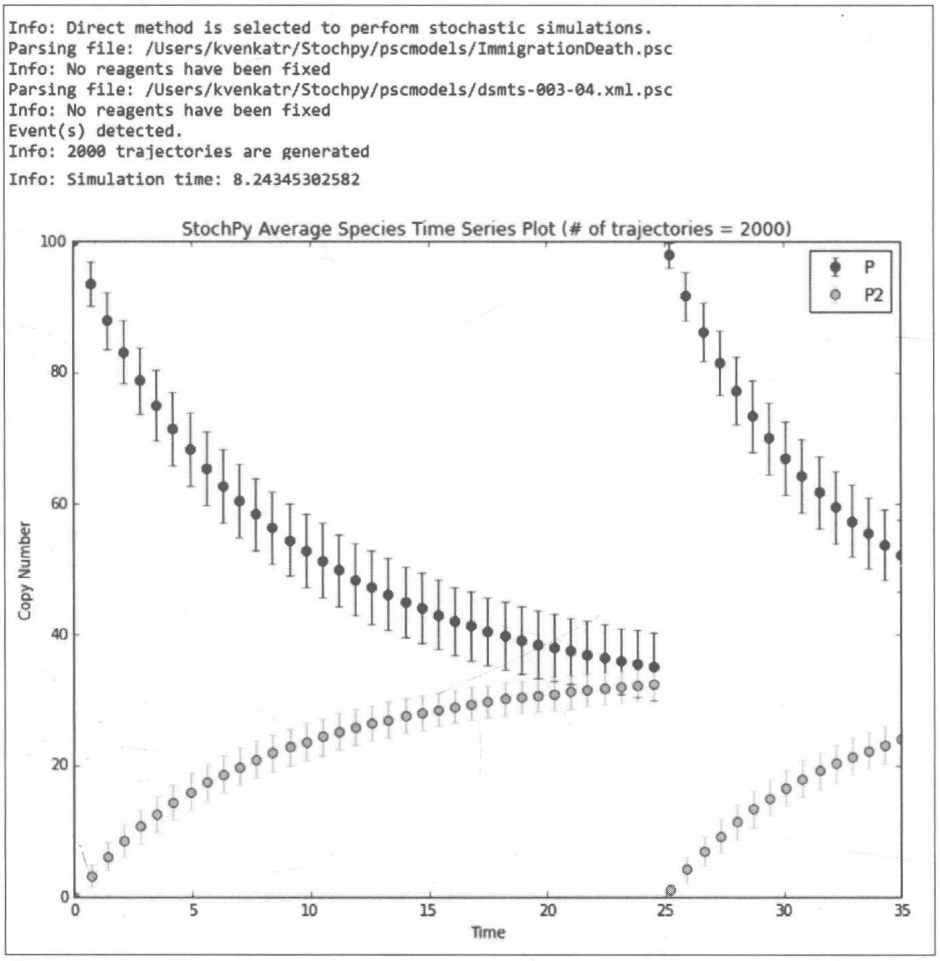


下面的代码用 stochpy 时间序列轨迹数据来完成模拟：

```
import stochpy as stp
smod = stp.SSA()

from pylab import rcParams
```

```
# set the graph display size as 10 by 10 inches
rcParams['figure.figsize'] = 12, 12

smod.Model('dsmts-003-04.xml.psc')
smod.DoStochSim(end=35,mode='time',trajectories=2000)
smod.GetRegularGrid()
smod.PlotAverageSpeciesTimeSeries()
```

StochPy 有一些便捷方法模拟随机模型，并展示出结果，如下图所示：



7.9 总结

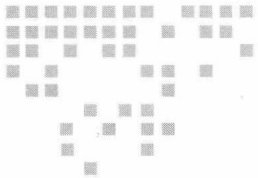
本章列举了网络和生物信息学的例子，而且选择 Python 软件包能够绘制这些结果。我们看到图和多重图的简要介绍，用稀疏矩阵和距离图来说明你如何用一些不同的软件包（比如，NetworkX、igraph（从 igraph.org）和 graph-tool）存储和展示图表。

聚集系数和图的中心性说明应该怎样计算聚集系数，以便了解图中的节点或顶点的显

著性大小。我们也可以用 Python-Twitter 软件包和 NetworkX 库，实现 Twitter 朋友和追随者的可视化研究社交网络数据分析。

通过说明如何用 bio 软件包发现 DNA 序列中的密码和计算 GC 比率，你也可以了解遗传编程样本。此外，我们示范了如何展示 DNA、RNA 或蛋白质的结构。

我们讨论了用 NetworkX 软件包进行的平面图测试，无环图测试和最大流量，以及一些如何测试的少数代码。此外，你可以选择如 PyMC 或 StochPy 等绘制随机区组模型，在下一章，我们将总结可供选择的高级可视化方法。



Chapter 8 第 8 章

高级可视化

几十年前，可视化方法已将传统的条形图和扇形图转换为极具创造力的表现形式。可视化设计已经不仅仅是从众多工具中选择一个特别的工具了。正确的可视化需要传递正确的信息，而错误的可视化会扭曲数据的含义，迷惑读者，甚至传递错误的信息。

计算机和存储设备在可视化设计中非常重要，它们能够利用数据结构实现体量巨大的数据块的存储，并且能够通过算法实现强大的计算能力。D3.js 的作者、顶级可视化专家 Michael Bostock 认为：我们应该将算法也可视化，而不只对放入其中的数据进行可视化。算法是任何一个程序或计算模型背后的核心部分，因此，算法成为可视化领域的一个重大应用。

可视化算法在近几年才开始被人们关注，visualgo.net 是了解算法可视化的一个有趣的网站，该网站中提供了一些讲授数据结构和算法的先进算法。Visualgo 包含的一些算法能在 Dr. Steven Halim 的著作 *Competitive Programming* 中找到。来自旧金山大学的 David Galles 教授也提供了不少类似的有趣的可视化方法（https://www.cs.usfca.edu/~galles/visualization/）。其中还有一些其他讲授算法和数据的内容。

我们已经讨论了很多不同的领域，包括数值计算、金融建模、统计和机器学习，以及网络模型。在本章，我们要讨论一些有关可视化和模拟以及信号处理案例的全新的，富有创意的想法。而且，我们还会讨论下述内容：

- 计算机模拟、信号处理和动画实例
- 基于 HTML5 的一些有趣的可视化方法
- Julia 与 Python 有哪些不同？——Julia 的优点与缺点
- 与 Python 的众多可视化方法相比较，为什么 D3.js 是更受欢迎的可视化工具？

- 建立仪表盘的工具

8.1 计算机模拟

计算机模拟是一门得到数十年公认的学科。它是一个试图模拟抽象模型的计算机程序。作为一种理解和评估隐含或未知情形的方式，计算机模拟模型有助于创建复杂系统。著名的计算机模拟建模的例子有：天气预测和用于训练飞行员的飞机模拟器。

计算机模拟已经成为众多领域中数学建模体系极具成效的部分，比如：物理、化学、生物、经济学、工程学、心理学和社会科学。

下面是模拟模型的优点：

- 加深对正在学习的算法和进程的理解
- 识别出算法和进程中存在的问题
- 测试出任何与算法模型有关的变化给模型带来的影响

模拟模型有以下几种类型：

- **离散模型**：系统仅在特定的时间点发生改变
- **连续模型**：一段时间内系统的状态会随着时间连续变化
- **混合模型**：同时具有离散和连续的成分

为了模拟，通常会利用随机概率输入。这是因为在进行模拟实验之前往往很难得到真实的数据。因此，常见的模拟实验中会有一些随机数来确定是否完成一个确定性模型。

让我们先来考虑 Python 中生成随机数的几种方法，并举几个模拟中的例子。

8.1.1 Python 的 random 包

Python 提供了一个名为 random 的包，这个软件包包含一些便捷的可以实现下述操作的函数：

- 生成从 0.0 到 1.0 之间，或指定区间的随机实数
- 生成指定范围内的随机整数
- 从一列数字或字母中生成一列随机数

```
import random

print random.random() # between 0.0 and 1.0
print random.uniform(2.54, 12.2) # between 2.54 and 12.2
print random.randint(5,10)  # random integer between 5 and 10

print random.randrange(25)  # random number between 0 and 25
#  random numbers from the range of 5 to 500 with step 5
```

```
print random.randrange(5,500,5)

# three random number from the list
print random.sample([13,15,29,31,43,46,66,89,90,94], 3)
# Random choice from a list
random.choice([1, 2, 3, 5, 9])
```

8.1.2 SciPy 的 random 函数

NumPy 和 SciPy 都是 Python 中具有数学和数值方法的模块。Numeric Python（NumPy）软件包提供了处理大型数值数组或矩阵的基本方法，scipy 软件包在 NumPy 基础上补充了很多算法和数学工具。

NumPy 具有内建的伪随机数生成器。生成器产生的这些数字是伪随机的，因为它们是通过一个种子数来确定的。使用同样的种子数，你就可以生成完全相同的随机数集，代码如下：

```
Import numpy as np
np.random.seed(65536)
```

在不提供种子数的情况下，该函数每次产生一个不同的随机数列。运行下面代码后，NumPy 每次会自动选择一个随机的种子（这种选择是基于时间的）：

```
np.random.seed()
```

从区间 [0.0, 1.0] 中生成一个具有 5 个随机数的数组，其代码如下：

```
import numpy as np
np.random.rand(5)
#generates the following
array([ 0.2611664,  0.7176011,  0.1489994,  0.3872102,  0.4273531])
```

rand 函数也可以用于生成二维随机数组，其代码如下：

```
np.random.rand(2,4)
array([
[0.83239852, 0.51848638, 0.01260612, 0.71026089],
[0.20578852, 0.02212809, 0.68800472, 0.57239013]])
```

为了产生随机整数，你可以使用 randint (min, max) 函数，代码如下：

```
np.random.randint(4,18)
```

利用下面的代码可以生成一个来自于 λ = 8.0 的离散泊松分布的随机数：

```
np.random.poisson(8.0)
```

利用下面的代码可以生成一个来自于均值 $\mu = 1.25$，标准差 $\sigma = 3.0$ 的连续高斯正态分布的随机数：

```
np.random.normal(2.5, 3.0)

#for mean 0 and variance 1
np.random.mormal()
```

8.1.3 模拟示例

在第一个例子中，我们将选择几何布朗运动（也被称为指数布朗运动），并用**随机微分方程**（SDE）对股票价格变动模式进行建模：

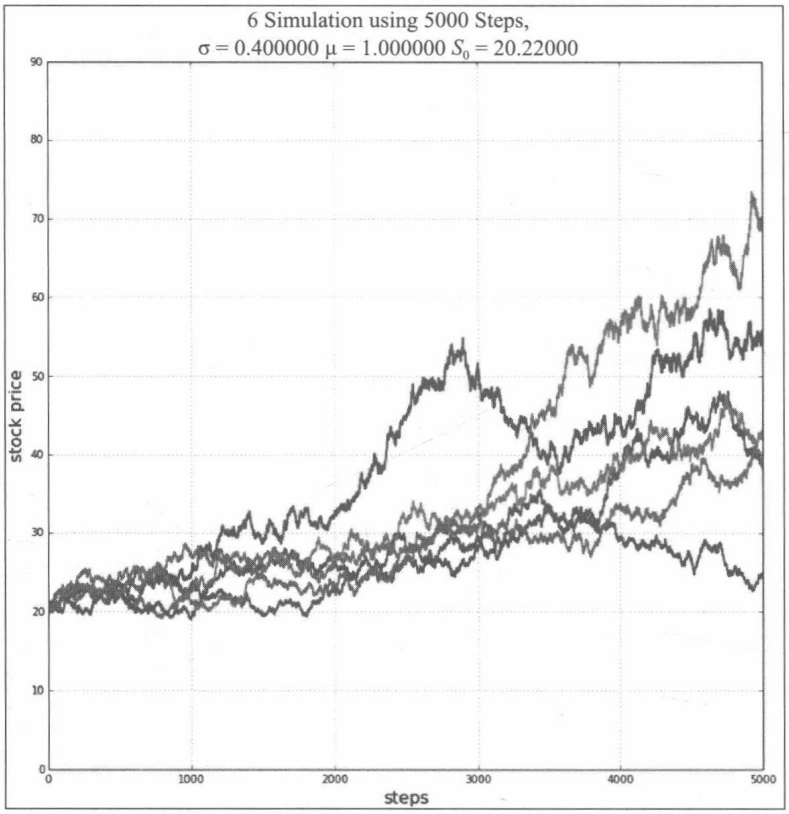
$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

在前面的方程中，W_t 是布朗运动，μ 是漂移百分比，σ 是波动百分比。通过下面代码对布朗运动绘制图像：

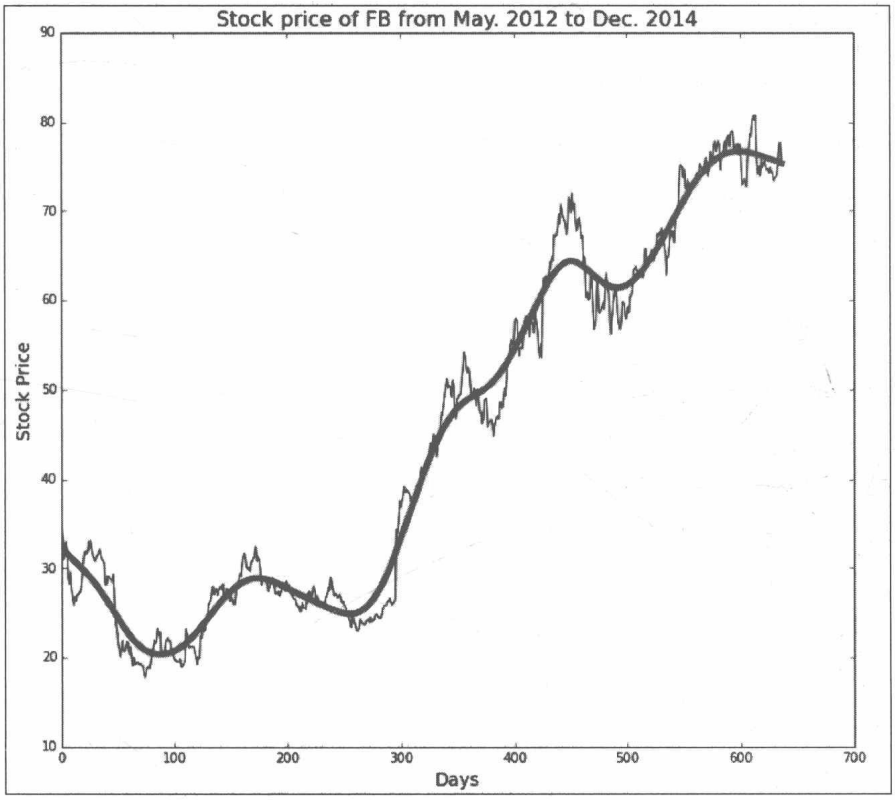
```
from numpy.random import standard_normal
from numpy import zeros, sqrt
import matplotlib.pyplot as plt

S_init = 20.222
T =1
tstep =0.0002
sigma = 0.4
mu = 1
NumSimulation=6

colors = [ (214,27,31), (148,103,189), (229,109,0), (41,127,214),
(227,119,194),(44,160,44),(227,119,194), (72,17,121), (196,156,148)]

# Scale the RGB values to the [0, 1] range.

for i in range(len(colors)):
    r, g, b = colors[i]
    colors[i] = (r / 255., g / 255., b / 255.)

plt.figure(figsize=(12,12))

Steps=round(T/tstep); #Steps in years
S = zeros([NumSimulation, Steps], dtype=float)
x = range(0, int(Steps), 1)

for j in range(0, NumSimulation, 1):

    S[j,0]= S_init
    for i in x[:-1]:
```

```
            S[j,i+1]=S[j,i]+S[j,i]*(mu-0.5*pow(sigma,2))*tstep+ \
                sigma*S[j,i]*sqrt(tstep)*standard_normal()
        plt.plot(x, S[j], linewidth=2., color=colors[j])

plt.title('%d Simulation using %d Steps, \n$\sigma$=%.6f $\mu$=%.6f
$S_0$=%.6f ' % (int(NumSimulation), int(Steps), sigma, mu, S_init),
          fontsize=18)
plt.xlabel('steps', fontsize=16)
plt.grid(True)
plt.ylabel('stock price', fontsize=16)
plt.ylim(0,90)

plt.show()
```

下图展示了利用布朗运动进行的 6 次模拟结果：



另一个模拟的例子展现了：在时间序列数据背景下，如何用 Hodrick– Prescott 筛选来获得一条表示股票价格数据的平滑曲线：



这里，我们将利用 matplotlib 的子软件包 finance 产生一组从 2012 年 5 月到 2014 年 12 月的股票价格数据。利用 matplotlib 包中的 hold 方法，我们可以用一条平滑的曲线展现股票的价格，具体代码如下：

```
from matplotlib import finance
import matplotlib.pyplot as plt

import statsmodels.api as sm

titleStr='Stock price of FB from May. 2012 to Dec. 2014'
plt.figure(figsize=(11,10))

dt1 = datetime.datetime(2012, 05, 01)
dt2 = datetime.datetime(2014, 12, 01)
sp=finance.quotes_historical_yahoo('FB',dt1,dt2,asobject=None)

plt.title(titleStr, fontsize=16)
plt.xlabel("Days", fontsize=14)
plt.ylabel("Stock Price", fontsize=14)

xfilter = sm.tsa.filters.hpfilter(sp[:,2], lamb=100000)[1]
```

```
plt.plot(sp[:,2])
plt.hold(True)
plt.plot(xfilter,linewidth=5.)
```

除了这些例子，你可以模拟一个排队系统，或者基于事件的任何过程。例如，你可以模拟一个神经网络，http://briansimulator.org 网站上有一个软件包能够帮助你快速构建一个神经网络模型。你可以从他们的样本程序中了解更多细节。

8.1.4 信号处理

信号处理有很多可供思考的例子，这里我们只选择一个卷积的例子。两个信号的卷积是指将它们合并到一起生成第三种滤波信号。在现实生活中，信号卷积常用于平滑图像。除此之外，信号卷积还可用来计算信号干扰。你可以从关于微波测量的书籍中获得更多的细节信息，这里我们仅列举几个简单的例子。

让我们看一个简单的例子。第一个例子展现了一个数字信号的卷积信号以及利用 hamming 函数模拟出的一个信号，具体代码如下：

```
import matplotlib.pyplot as plt
from numpy import concatenate, zeros, ones, hamming, convolve

digital = concatenate ( (zeros(20), ones(25), zeros(20)))
norm_hamming = hamming(80)/sum(hamming(80))
res = convolve(digital, norm_hamming)
plt.figure(figsize=(10,10))
plt.ylim(0, 0.6)
plt.plot(res, color='r', linewidth=2)
plt.hold(True)
plt.plot(data, color='b', linewidth=3)
plt.hold(True)
plt.plot(norm_hamming, color='g', linewidth=4)
plt.show()
```

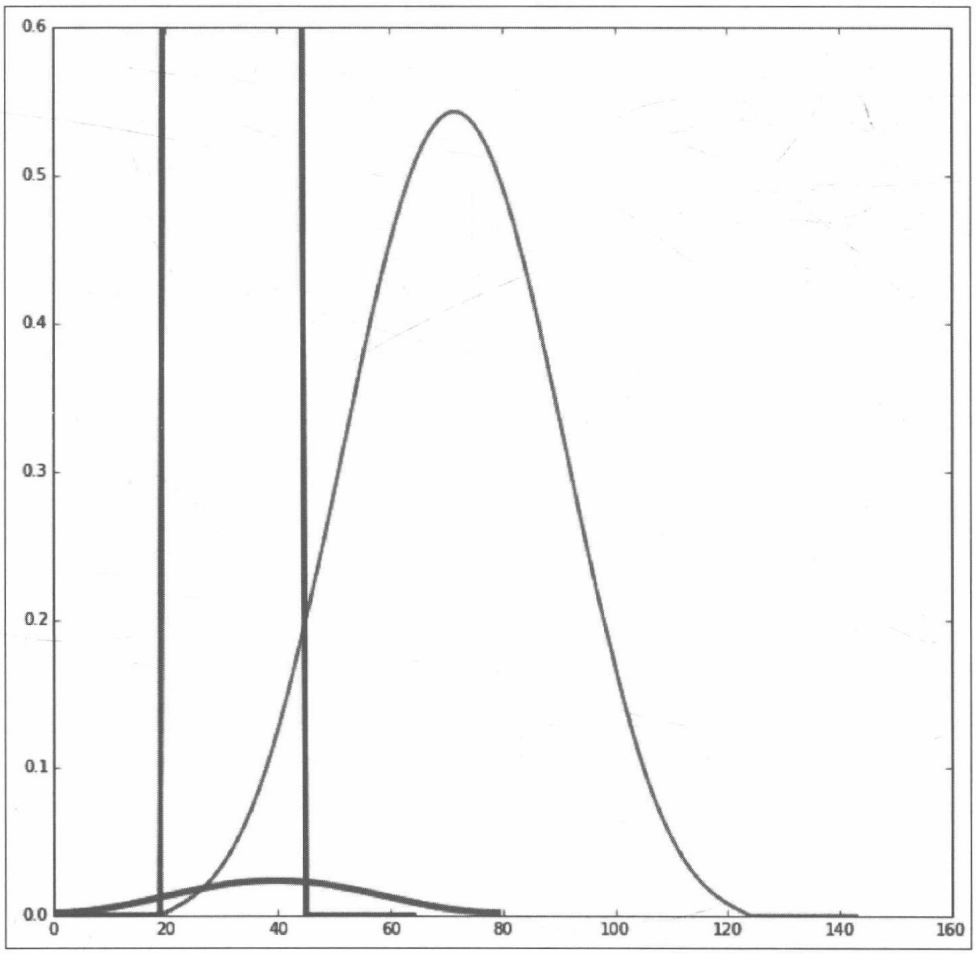
在这个例子中，我们利用 numpy 中的 concatenate、zeros 以及 ones 函数来制造数字信号，利用 hamming 函数产生相似的模拟信号，再利用 convolve 函数来实施卷积过程。

如果绘制出全部三个信号的图像，即数字信号、模拟信号以及卷积信号（res），我们可以看到卷积信号同预期的一样发生变化，如下图所示。

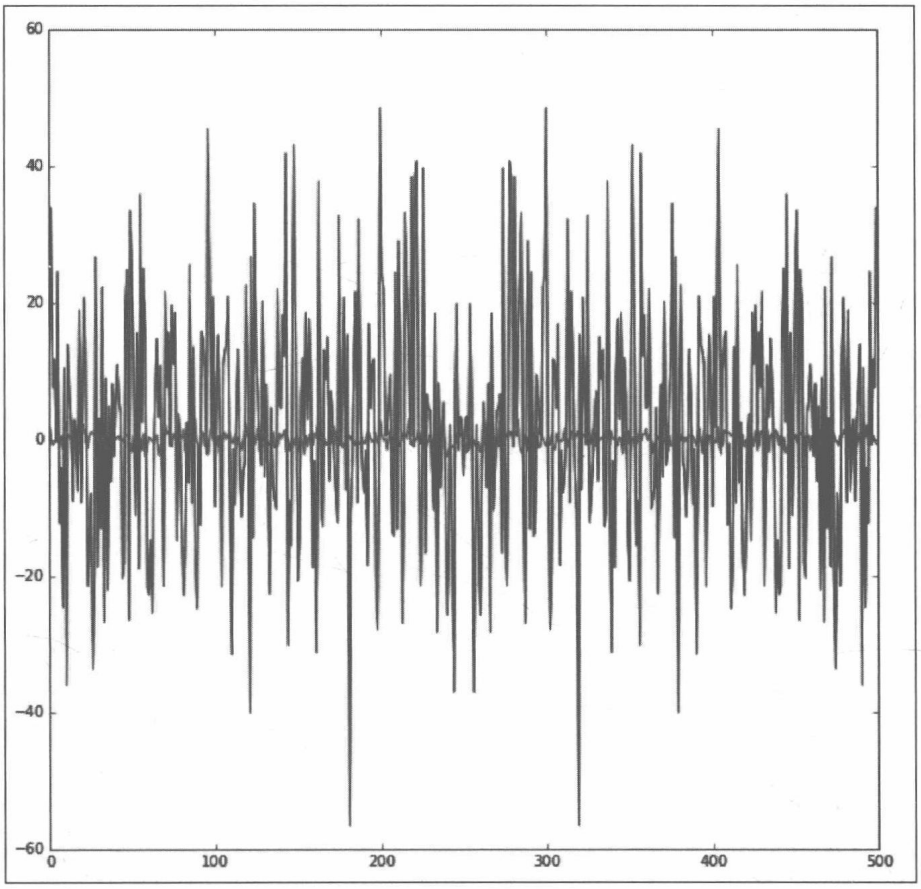
在下面这个例子中，我们将使用随机信号数据 random_data，并应用**快速傅里叶变换**（Fast Fourier Transform，FFT），代码如下所示：

```
import matplotlib.pyplot as plt
from scipy import randn
from numpy import fft
```

```
plt.figure(figsize=(10,10))
random_data = randn(500)
res = fft.fft(random_data)
plt.plot(res, color='b')
plt.hold(True)
plt.plot(random_data, color='r')
plt.show()
```



scipy 中的 randn 函数可以产生随机信号数据，而 numpy 中的 fft 函数能够实现快速傅里叶变换。利用 matplotlib 对数据信号进行绘图，得到下图结果，其中细线代表经过傅里叶变换后的信号，粗线代表原始的随机信号。



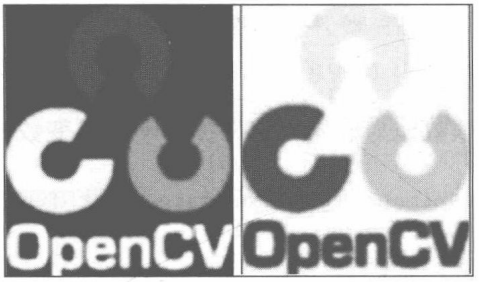
在第三个例子中，将简单说明如何利用 scipy 软件包创建一个反色图片。在编写实际的 Python 代码并得到结果之前，让我们尝试分析一下反色图片如何有助于数据可视化。

在某些情形中，反色图片的颜色能够减少视觉压力，让图片看起来更加舒服。如果将原始图片和反色图片放在一起对比，你会惊奇地发现反色图片能够帮助你看到一些在原始图片中难以观察到的区域。虽然这种现象并不在所有的图片中都存在，但至少在一些特定的图片上是真实存在的。下面的代码告诉你如何利用 scipy.misc.pilutil.Image() 函数将一幅图片转化为它的反色图片。

```
import scipy.misc as scm
from scipy.misc.pilutil import Image

# open original image
orig_image = Image.open('/Users/kvenkatr/Desktop/filter.jpg')

# extract image data into array
image1 = scm.fromimage(orig_image)
# invert array values
```

```
inv_image = 255 - image1

# using inverted array values, convert image
inverted_image = scm.toimage(inv_image)

#save inverted image
inverted_image.save('/Users/kvenkatr/Desktop/filter_invert.jpg').
```

反色图片与原始图片如下图所示：



类似地，使用如下函数可以对任何图像施加各种滤镜。

```
convolve()          Multidimensional convolution.
correlate()         Multi-dimensional correlation.
gaussian_filter()   Multidimensional Gaussian filter
```

在网站 http://tinyurl.com/3xubv9p 上可以找到图像滤镜函数的完整列表。

8.1.5 动画制作

在 Python 中利用 matplotlib 包可以实现动画制作，它会将制作结果保存为 MP4 格式的文件，供以后重复播放。用 Python 制作动画的基本步骤如下：

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation

# Set up the figure, axis, and the plot element to be animated
fig = plt.figure()
ax = plt.axes(xlim=(0, 3.2), ylim=(-2.14, 2.14))
line, = ax.plot([], [], lw=2)
```

首先要从 matplotlib 中加载 animation 包，设置坐标轴，并准备好必要的绘图变量（只是一条空白线）。如下所示：

```
# initialization function: plot the background of each frame
def init():
```

```
    line.set_data([], [])
    return line,
```

在开始制作动画之前，需要先进行绘图的初始化，它能够给动画的每一帧创建一个基础的框架，代码如下：

```
# animation function.  This is called sequentially
def animate(i):
    x = np.linspace(0, 2, 1000)
    xval = 2 * np.pi * (x - 0.01 * i)
    y = np.cos(xval) # Here we are trying to animate cos function
    line.set_data(x, y)
    return line,
```

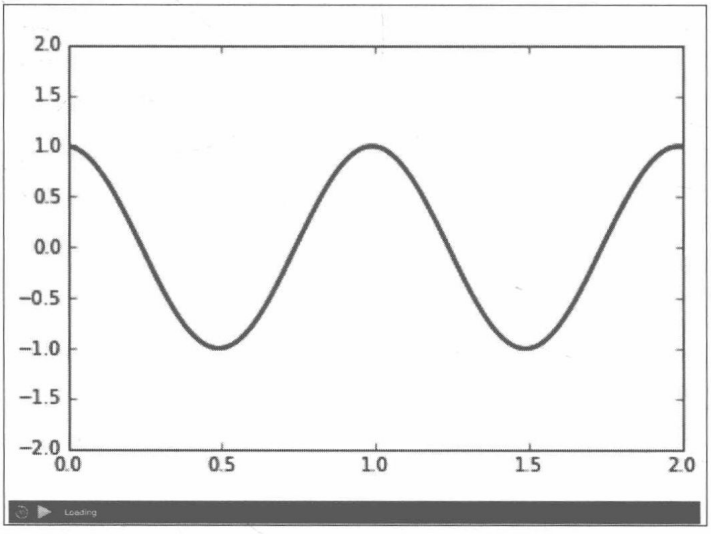
在 animation 函数中，帧数作为输入，定义 x 和 y 的可变数值，并设置好绘图变量：

```
anim = animation.FuncAnimation(fig, animate, init_func=init,\
            frames=200, interval=20, blit=True)
anim.save('basic_animation.mp4', fps=30)
plt.show()
```

实际的动画对象通过 FuncAnimation 产生，在 FuncAnimation 函数中放入初始化函数 init() 和动画函数 animate()，并设定动画的总帧数，每秒的帧数以及时间区间等参数。参数 blit=True 表示只有画面需要改变的地方才重画（blit=False 表示你会看到画面一直在闪烁）。

在你尝试制作一个动画之前，需确保已经安装了 mencoder 或 ffmpeg。否则，在运行程序时会出现以下错误：ValueError: Cannot save animation: no writers are available. Please install mencoder or ffmpeg to save animations.

下面的图片展示了三角函数曲线（如 sin 或 cos）的动画：



你可以把制作好的 MP4 文件镶嵌到 HTML 上进行展示，并通过点击左下角的播放按钮来观看动画。

在 https://jakevdp.github.io/blog/2012/08/18/matplotlib-animation-tutorial/ 上有由 Jake Vanderplas 制作的有趣的双摆动画，在 http://matplotlib.org/examples/animation/dynamic_image2.html 上也可以找到一个动态的图像动画。

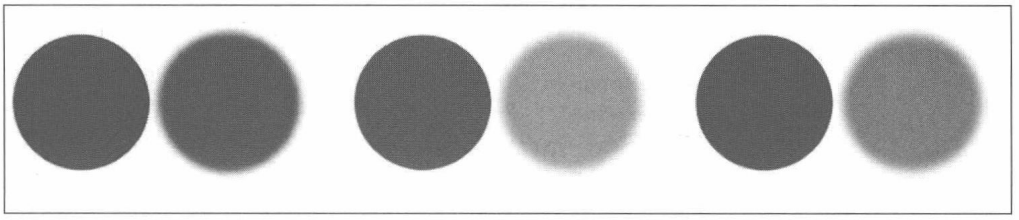
到目前为止，本书已经讨论了一些可视化方法，包括如何在 Python 上进行绘图，或者生成其他外部格式的可视化文件（如 MP4）。另外一种非常流行的可视化方法是基于 JavaScript 的可视化方法。这种可视化方法能够轻松地在 Web 页面上进行展现，并且能够关联一些事件驱动的动画。另一方面，向量图像（SVG）也是非常流行的可视化方法，它因具有经过任意的大小尺度变化下不损失任何细节的优点而获得了很多人的追捧。

8.1.6 利用 HTML5 进行可视化

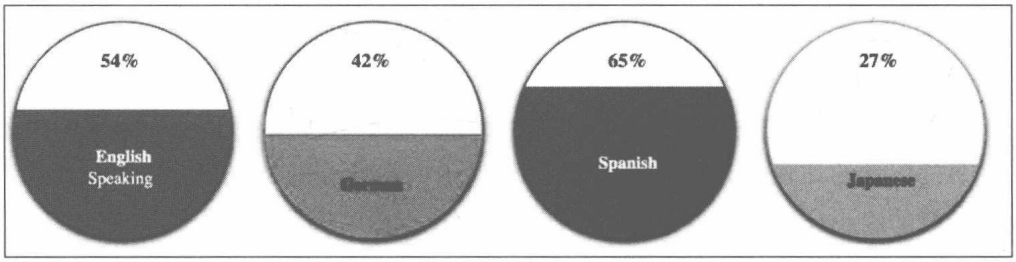
一个 SVG 的简单说明是利用 feGaussianBlur 绘制圆圈，代码如下：

```
  <svg width="230" height="120" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">
    <filter id="blurMe">
       <feGaussianBlur in="SourceGraphic" stdDeviation="5" />
    </filter>

    <circle cx="60"  cy="80" r="60" fill="#E90000" />
    <circle cx="190" cy="80" r="60" fill="#E90000"
      filter="url(#blurMe)" />
    <circle cx="360"  cy="80" r="60" fill="#4E9B01" />
    <circle cx="490" cy="80" r="60" fill="#4E9B01"
      filter="url(#blurMe)" />
    <circle cx="660"  cy="80" r="60" fill="#0080FF" />
    <circle cx="790" cy="80" r="60" fill="#0080FF"
      filter="url(#blurMe)" />
  </svg>
```

前两个圆圈半径均为 60 且颜色相同，但是第二个圆圈使用了模糊滤波器。类似地，相邻的绿色和蓝色圆圈也具有相同的特征（对于色彩的效应，详见 http://knapdata.com/dash/html/svg_circle.html）。这些圆圈的图像如下图所示：



当我们需要展示总体数据中的一部分，而不是把数据组合为一个整体时，我们要如何使用模糊的概念？模糊是什么意思？让我们来看两个例子，第一个例子考察一个班级的学生选修外语课的情况（在某些情况下，可能不止一种外语）。如果我们想用下图展示分布，我们应该如何做？



你可以利用 Python 程序生成 SVG 格式的图像，具体代码如下：

```
import os
display_prog = 'more' # Command to execute to display images.
svcount=1

class Scene:
    def __init__(self,name="svg",height=400,width=1200):
        self.name = name
        self.items = []
        self.height = height
        self.width = width
        return

    def add(self,item): self.items.append(item)

    def strarray(self):
        var = [ "<html>\n<body>\n<svg height=\"%d\" width=\"%d\" >\n"
% (self.height,self.width),
                "  <g id=\"setttings\">\n",
                "    <filter id=\"dropshadow\" height=\"160%\">\n",
                "     <feGaussianBlur in=\"SourceAlpha\"
stdDeviation=\"5\"></feGaussianBlur>\n",
                "       <feOffset dx=\"0\" dy=\"3\"
result=\"offsetblur\"></feOffset>\n",
                "       <feMerge>\n",
                "          <feMergeNode></feMergeNode>\n",
                "          <feMergeNode in=\"SourceGraphic\"></
feMergeNode>\n",
                "       </feMerg>\n",
                "    </filter>\n"]
        for item in self.items: var += item.strarray()
        var += [" </g>\n</svg>\n</body>\n</html>"]
        return var
```

```
    def write_svg(self,filename=None):
        if filename:
            self.svgname = filename
        else:
            self.svgname = self.name + ".html"
        file = open(self.svgname,'w')
        file.writelines(self.strarray())
        file.close()
        return

    def display(self,prog=display_prog):
        os.system("%s %s" % (prog,self.svgname))
        return

def colorstr(rgb): return "#%x%x%x" % (rgb[0]/16,rgb[1]/16,rgb[2]/16)

class Text:
    def __init__(self, x,y,txt, color, isItbig, isBold):
        self.x = x
        self.y = y
        self.txt = txt
        self.color = color
        self.isItbig = isItbig
        self.isBold = isBold
    def strarray(self):
        if ( self.isItbig == True ):
          if ( self.isBold == True ):
            retval = [" <text y=\"%d\" x=\"%d\" style=\"font-
size:18px;font-weight:bold;fill:%s\">%s</text>\n" %(self.y, self.x,
self.color,self.txt) ]
          else:
            retval = [" <text y=\"%d\" x=\"%d\" style=\"font-
size:18px;fill:%s\">%s</text>\n" %(self.y, self.x, self.color,self.
txt) ]
        else:
          if ( self.isBold == True ):
            retval = [" <text y=\"%d\" x=\"%d\" style=\"fill:%s;font-
weight:bold;\">%s</text>\n" %(self.y, self.x, self.color,self.txt) ]
          else:
            retval = [" <text y=\"%d\" x=\"%d\" style=\"fill:%s\">%s</
text>\n" %(self.y, self.x, self.color,self.txt) ]
        return retval

class Circle:
    def __init__(self,center,radius,color, perc):
        self.center = center #xy tuple
        self.radius = radius #xy tuple
        self.color = color   #rgb tuple in range(0,256)
        self.perc = perc
        return
```

```
    def strarray(self):
        global svcount
        diam = self.radius+self.radius
        fillamt = self.center[1]-self.radius - 6 + (100.0 - self.
perc)*1.9
        xpos = self.center[0] - self.radius
        retval = ["  <circle cx=\"%d\" cy=\"%d\" r=\"%d\"\n" %\
                (self.center[0],self.center[1],self.radius),
                "    style=\"stroke: %s;stroke-width:2;fill:white;filt
er:url(#dropshadow)\"  />\n" % colorstr(self.color),
               "  <circle clip-path=\"url(#dataseg-%d)\" fill=\"%s\"
cx=\"%d\" cy=\"%d\" r=\"%d\"\n" %\
                (svcount, colorstr(self.color),self.center[0],self.
center[1],self.radius),
                "    style=\"stroke:rgb(0,0,0);stroke-width:0;z-
index:10000;\"  />\n",
               "<clipPath id=\"dataseg-%d\"> <rect height=\"%d\"
width=\"%d\" y=\"%d\" x=\"%d\"></rect>" %(svcount,diam,
diam,fillamt,xpos),
               "</clipPath>\n"
                ]
        svcount += 1
        return retval

def languageDistribution():
    scene = Scene('test')
    scene.add(Circle((140,146),100,(0,128,0),54))
    scene.add(Circle((370,146),100,(232,33,50),42))
    scene.add(Circle((600,146),100,(32,119,180),65))
    scene.add(Circle((830,146),100,(255,128,0),27))
    scene.add(Text(120,176,"English", "white", False, True))
    scene.add(Text(120,196,"Speaking", "#e2e2e2", False, False))
    scene.add(Text(340,202,"German", "black", False, True))
    scene.add(Text(576,182,"Spanish", "white", False, True))
    scene.add(Text(804,198,"Japanese", "black", False, True))

    scene.add(Text(120,88,"54%", "black", True, True))
    scene.add(Text(350,88,"42%", "black", True, True))
    scene.add(Text(585,88,"65%", "black", True, True))
    scene.add(Text(815,88,"27%", "black", True, True))

    scene.write_svg()
    scene.display()
    return

if __name__ == '__main__': languageDistribution()
```

前面的例子为我们展现了一种创建传统的可视化 SVG 图片的思路。现在 Python 中有很多其他 SVG 生成器，但没有一种方法能够超越我们在这展现出的结果。其他编程语言，如 Julia，也有很多创建传统可视化方法的方式。Julia 已有 3 年历史，且非常适合进行数值计

算和科学计算。

8.1.7 Julia 和 Python 有什么区别

Julia 是一门动态的编程语言。然而，Julia 的效率却可以与 C 相提并论，因为 Julia 是一个低层次的、虚拟的、基于机器的实时编译器（JIT 编译器）。众所周知，在 Python 中，为了能够将 C 和 Python 结合利用，你可以去使用 Cython。

Julia 的一些显著优点如下：

- 运算效率和 C 语言一样快
- 具有内置包管理器
- 具有 lisp 宏
- 可以利用 PyCall 包来调用 Python 中的函数
- 可以直接调用 C 中的函数
- 基于分布式计算设计
- 用户自定义的类型和内建类型执行速度一样快

虽然 Julia 与 C 和 Python 有很多相似之处，但是用 Julia 仍存在一点不足——需要学习一门新的编程语言。

D3.js（其中 D3 是 DDD 的简写，即 document-driven data）是对 Python 极具竞争力的可视化框架。

8.1.8 用 D3.js 进行可视化

D3.js 是一个用于展现互联网上数据的 JavaScript 模块，它能够利用 HTML、SVG 和 CSS 来展示数据。

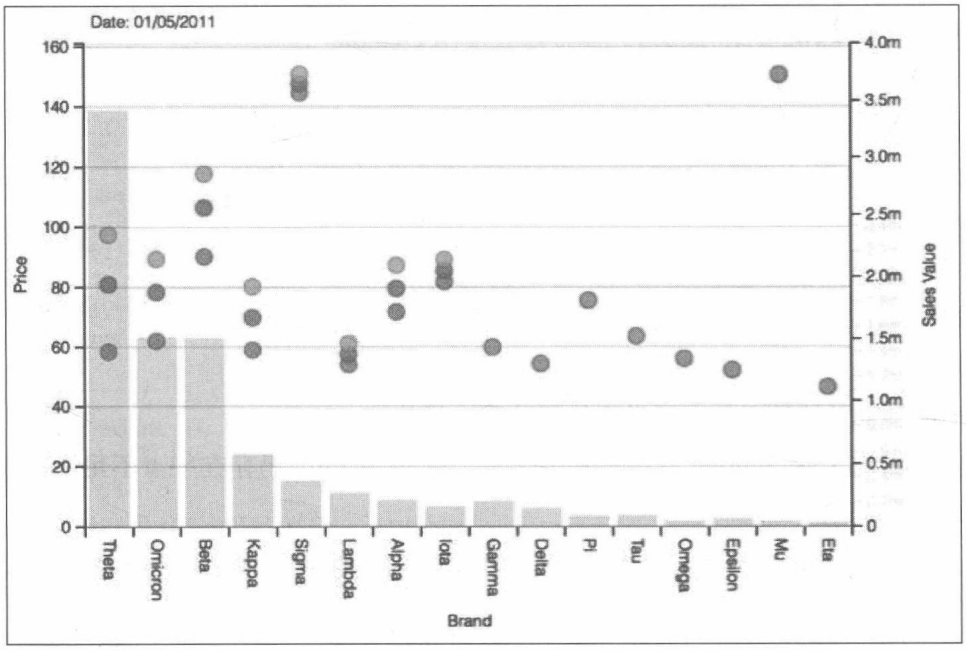
D3.js 将数据附到**文件对象模型**（DOM）中，因此可以利用 CSS3、HTML 和 SVG 来展示这些数据。更进一步，JavaScript 具有事件监听器，从而使数据变得具有交互能力。

Mike Bostock 在斯坦福可视化小组读博的时候创造了 D3.js。一开始，Mike 和斯坦福可视化小组的成员共同制作了 Protivis，最后 Protivis 演变为了 D3。Mike Bostock、Vadim Ogievetsky 和 Jeffrey Heer 发表了一篇题为“*Data-Driven Documents*”的文章。你可以从 http://vis.stanford.edu/papers/d3 网站上找到这篇文章。

在实践过程中，D3.js 的基本原理是利用 CSS 的选择器从 DOM 节点当中进行选择，再利用 jQuery 来处理他们，例如：

```
d3.selectAll("p")            // select all <p> elements
  .style("color", "#FF8000") // set style "color" to value "#FF8000"
  .attr("class", "tin")      // set attribute "class" to value "tin"
  .attr("x", 20);            // set attribute "x" to 20px
```

D3 的一个优点是：通过利用 DOM 的机制，你可以将数据用令人震惊的形式展现出来。另外一个优点是：结合先进时代强大的计算能力，你可以利用 JavaScript 的力量在可视化中非常方便的添加导航行为。在 http://bost.ocks.org/mike/ 网站上有大量这种可视化方法的例子。下图就是一个 D3 可视化的例子：



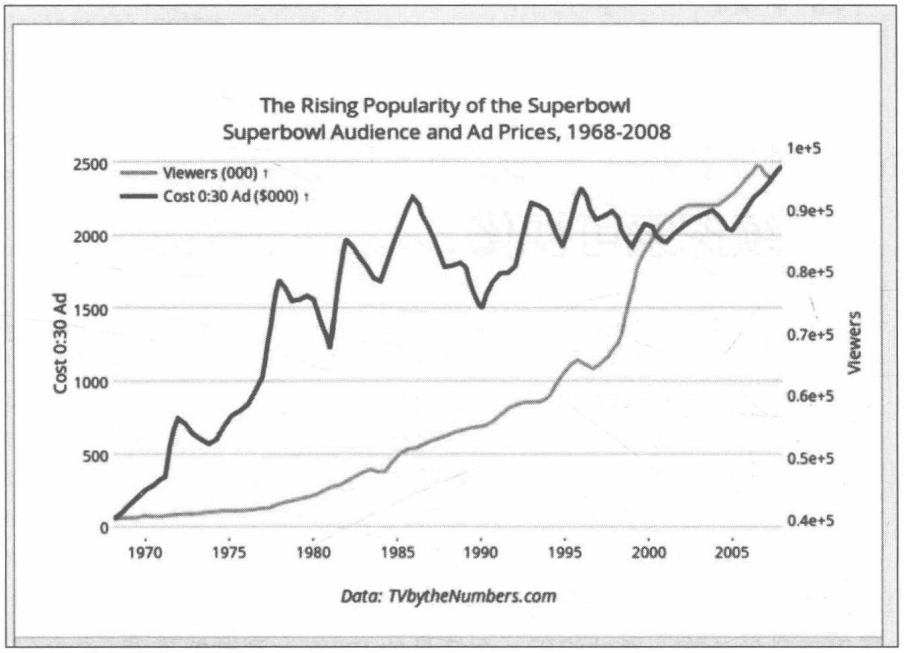
你自己也可以编写出很多可视化的例子，但是在 http://christopheviau.com/d3list/gallery.html#visualizationType=lollipop 提供的例子当中，我最喜欢的是那个利用多个系列与多个坐标系展现出数据中不同聚集体的例子。你可以从 http://tinyurl.com/p988v2u 上获取（本章前面的图片中也有该例子）。

8.1.9 仪表盘

与 D3 相比，Python 也有很多优点。当结合两者一起使用时，你会得到最好的可视化效果。例如，Python 提供很多具有优秀的数值计算和科学计算能力的包，因此它在学术界非常流行。

近几年来出现的不错的数据可视化协作工具并不多，Plotly（https://plot.ly）是其中一个。Python 的仪表盘内容可以从 https://plot.ly/python/dashboard/ 中获得。由于仪表盘最近才被研发出来，所以我们还没能深入发掘它的功能和作用。Splunk 提供了 SDK 包来创建基于 Python 的仪表盘（http://dev.splunk.com/view/SP-CAAADSR）。Pyxley 是软件包的集合，它具有 Python 和 JavaScript 的优势，能够创建基于页面的仪表盘。一个 Splunk Dashboard 的

例子如下：



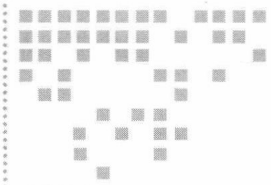
上图展示了一个 Plotly 的例子。它说明如何生成一个视觉效果不错、容易理解的图像，可见 http://tinyurl.com/pwmg5zr。

8.2 总结

本章讲述了前几章未涉及的话题，比如用 Python 进行信号处理和动画。此外，我们比较了 Python 与 D3.js 和 Julia 的差异，确定了他们的长处。此外，还讨论了一些信号处理的例子，介绍了用 numpy 和 matplotlib 计算模拟和数字信号频谱的卷积。

我们也讨论了一个动画的例子，演示了如何通过 Python 生成 MP4 格式的动画。我们也比较了 Julia 与 Python，并列出 Julia 的一些优势，比较了它们之间的差异。

随后，我们展示了 D3.js 的优势，强调基于 JavaScript 可视化工具和 Python 之间的差异。最后，我们讨论了仪表盘上可用的选项，并列出一些创建基于 Python 仪表盘的选项。



appendix 附录

继续探索可视化

自 1991 年以来，Python 受到众多科学家和工程师的欢迎。numpy、scipy 和 matplotlib 这些库已广泛用于科学计算。通过使用一个简易的 IPython 浏览器界面，Sage 可以解决代数、组合数学、计算数学、数论和微积分领域的问题。另一个受欢迎的软件包是 pandas，它可以存储和处理复杂的数据集。

有多种工具可以运行和编辑 Python 程序，Continuum 中的 Anaconda 就是其中一个。Anaconda 的优势之在于它没有任何消耗，只内置必要的软件包。管理环境和 Python 软件包的隐含命令行工具是 conda，Spyder 是编辑器。

过去，安装 Spyder 非常复杂，因为它涉及下载和安装的多个步骤。在最新版本中，安装已经非常简单，一步就可以自动下载和安装所有组件。

Conda 综述

Conda 是一个命令行工具，负责管理环境和 Python 软件包，而不是用 pip。现在有几种方法查询和搜索软件包，必要时创建新环境，以及在现存 conda 环境中安装和更新 Python 软件包。这个命令行工具也会跟踪软件包与平台细节间的依赖关系，帮助你从不同的软件包组合中创建工作环境。为了检验正在运行的 conda 是哪个版本，你可以在 Python 中输入 conda --version。比如，它将显示 conda 3.18.2 版。

Conda 环境是文件系统目录，包括特定集合的 conda 软件包。为了开始使用一个工作环境，简单的设置 PATH 变量，使它指向其 bin 目录。

下面是通过命令行用 conda 安装软件包的一个例子：

```
$ conda install scipy

Fetching package metadata: ....
Solving package specifications: .
Package plan for installation in environment /Users/MacBook/anaconda:

The following packages will be downloaded:

    package                    |        build
    ---------------------------|-----------------
    flask-0.10.1               |       py27_1         129 KB
    itsdangerous-0.23          |       py27_0          16 KB
    jinja2-2.7.1               |       py27_0         307 KB
    markupsafe-0.18            |       py27_0          19 KB
    werkzeug-0.9.3             |       py27_0         385 KB

The following packages will be linked:

    package                    |            build
    ---------------------------|-----------------
    flask-0.10.1               |       py27_1
    itsdangerous-0.23          |       py27_0
    jinja2-2.7.1               |       py27_0
    markupsafe-0.18            |       py27_0
    python-2.7.5               |       2
    readline-6.2               |       1
    sqlite-3.7.13              |       1
    tk-8.5.13                  |       1
    werkzeug-0.9.3             |       py27_0
    zlib-1.2.7                 |       1

Proceed ([y]/n)?
```

我们安装的软件包依赖包含自动识别、下载和连接。

下面给出通过命令行用 conda 更新软件包的一个例子：

```
$ conda update matplotlib
Fetching package metadata: ....
Solving package specifications: .
```

```
Package plan for installation in environment /Users/MacBook/anaconda:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    freetype-2.5.2             |                0         691 KB
    conda-env-2.1.4            |            py27_0          15 KB
    numpy-1.9.2                |            py27_0         2.9 MB
    pyparsing-2.0.3            |            py27_0          63 KB
    pytz-2015.2                |            py27_0         175 KB
    setuptools-15.0            |            py27_0         436 KB
    conda-3.10.1               |            py27_0         164 KB
    python-dateutil-2.4.2      |            py27_0         219 KB
    matplotlib-1.4.3           |       np19py27_1        40.9 MB
    ------------------------------------------------------------
                                           Total:        45.5 MB

The following NEW packages will be INSTALLED:

    python-dateutil: 2.4.2-py27_0

The following packages will be UPDATED:

    conda:              3.10.0-py27_0     --> 3.10.1-py27_0
    conda-env:        2.1.3-py27_0      --> 2.1.4-py27_0
    freetype:   2.4.10-1          --> 2.5.2-0
    matplotlib:     1.4.2-np19py27_0 --> 1.4.3-np19py27_1
    numpy:          1.9.1-py27_0      --> 1.9.2-py27_0
    pyparsing:      2.0.1-py27_0      --> 2.0.3-py27_0
    pytz:           2014.9-py27_0     --> 2015.2-py27_0
    setuptools:     14.3-py27_0       --> 15.0-py27_0

Proceed ([y]/n)?
```

有时候，通过 conda 安装软件包涉及更多步骤。例如，为了安装 wordcloud，你必须执行以下代码中给出的步骤：

```
#step-1 command
conda install wordcloud
```

```
Fetching package metadata: ....
Error: No packages found in current osx-64 channels matching: wordcloud

You can search for this package on Binstar with
# This only means one has to search the source location
binstar search -t conda wordcloud

Run 'binstar show <USER/PACKAGE>' to get more details:
Packages:
                          Name | Access        | Package Types    |
     ------------------------- | ------------- | ---------------- |
             derickl/wordcloud | public        | conda            |
Found 1 packages

# step-2 command
binstar show derickl/wordcloud

Using binstar api site https://api.binstar.org
Name:    wordcloud
Summary:
Access:  public
Package Types:  conda
Versions:
   + 1.0

To install this package with conda run:
conda install --channel https://conda.binstar.org/derickl wordcloud

# step-3 command
conda install --channel https://conda.binstar.org/derickl wordcloud

Fetching package metadata: ......
Solving package specifications: .
Package plan for installation in environment /Users/MacBook/anaconda:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    cython-0.22                |           py27_0         2.2 MB
    django-1.8                 |           py27_0         3.2 MB
```

```
    pillow-2.8.1                |           py27_1         454 KB
    image-1.3.4                 |           py27_0          24 KB
    setuptools-15.1             |           py27_1         435 KB
    wordcloud-1.0               |       np19py27_1          58 KB
    conda-3.11.0                |           py27_0         167 KB
    ------------------------------------------------------------
                                           Total:         6.5 MB

The following NEW packages will be INSTALLED:
    django:     1.8-py27_0
    image:      1.3.4-py27_0
    pillow:     2.8.1-py27_1
    wordcloud:  1.0-np19py27_1

The following packages will be UPDATED:
    conda:      3.10.1-py27_0 --> 3.11.0-py27_0
    cython:     0.21-py27_0   --> 0.22-py27_0
    setuptools: 15.0-py27_0   --> 15.1-py27_1


Finally, the following packages will be downgraded:

    libtiff:    4.0.3-0       --> 4.0.2-1

Proceed ([y]/n)? y
```

Anaconda是一个适用于科学计算的免费Python发行版。该发行版配备Python 2.x或Python 3.x，以及100多个跨平台测试和优化的Python软件包。Anaconda也可以创建混合和匹配不同Python版本的自定义环境。

用Anaconda安装的软件包

下面的命令展示了Anaconda环境中的所有软件包：

```
conda list
```

Anaconda中典型的软件包是Astropy、Cython、h5py、IPython、LLVM、LLVMpy、matplotlib、Mayavi、NetworkX、NLTK、Numexpr、Numba、numpy、pandas、Pytables、scikit-image、scikit-learn、scipy、Spyder、Qt/PySide和VTK。

为了检查 Anaconda 安装的软件包，导向命令行输入 conda list 命令，以快速展示默认环境中安装的所有软件包列表。另外，你也可以在当前和最新发布的软件包列表中检查 Continuum Analytics 的详细信息。

此外，你总可以用通常的方法安装一个软件包。例如，用 pip install 命令或使用 setup.py 的源文件。尽管人们更倾向于用 conda 安装软件包，但 Anaconda 也没有特别妨碍标准 Python 包工具的使用。

IPython 不是必须的，但我强烈推荐使用。IPython 应该安装在 Python、GNU Readline 和 PyReadline 之后。Anaconda 和 Canopy 默认完成这些工作。本书中有一些所有案例都使用的 Python 软件包。下一节，我们已经更新了这个列表。

软件包网站

这里有本书提到的几个 Python 软件包以及相应的网站，通过这些网站可以获得大多信息信息：

- IPython：交互式计算的丰富架构（http://ipython.org）。
- NumPy：适用于多维数组的高性能和向量计算（http://www.numpy.org）。
- SciPy：适用于高级数值算法（http://www.scipy.org）。
- Matplotlib：适用于在交互式可视化中画图和执行（http://matplotlib.org）。
- matplotlib-basemap：matplotlib 的一个映射工具箱（http://matplotlib.org/basemap/）。
- Seaborn：适用于 matplotlib 的统计数据可视化（http://stanford.edu/~mwaskom/software/seaborn）。
- Scikit：适用于 Python 中的机器学习（http://scikit-learn.org/stable）。
- NetworkX：适用于处理图表（http://networkx.lanl.gov）。
- Pandas：适用于处理任何一种表格数据（http://pandas.pydata.org）。
- Python Imaging Library (PIL)：适用于图像处理算法（http://www.pythonware.com/products/pil）。
- PySide：作为图形用户界面（graphical user interface，GUI）的 Qt 外包（http://qt-project.org/wiki/PySide）。
- PyQt：与 PySide 类似，但许可证不同（http://www.riverbankcomputing.co.uk/software/pyqt/intro）。
- Cython：适用于在 Python 中使用 C 代码（http://cython.org）。

关于 matplotlib

matplotlib 软件包配备很多便捷地创建可视化图表的方法。其中只有极少数在本书中得以探索。如果想要进一步研究 matplotlib，可参考下面的资源：

- http://www.labri.fr/perso/nrougier/teaching/matplotlib/
- http://matplotlib.org/Matplotlib.pdf

你也可以参考前面列出的其他软件包，这些库使绘图变得更具吸引力。

Python数据可视化

Mastering Python Data Visualization

随着海量信息的增长，需要处理的数据越来越多。这些数据包含着许多掌控当代成功命脉的看法与见解。能够发现数据、清洗数据，并使用正确的工具实现可视化至关重要。本书讲解了用Python软件包实现数据可视化的一些不同方法，并给出很多不同领域的案例，比如，数值计算、财务模型、统计机器学习，以及遗传学与网络等。

通过阅读本书，你会学到：

- 收集、清理、获取数据，并将数据映射到可视化框架
- 识别可用的可视化方法，学习数据可视化的最佳案例
- 熟悉以读者驱动为导向的故事和以作者驱动为导向的故事以及感知原则
- 理解为何Python能够成为像MATLAB一样有效的数值计算工具，探索一些相应的有趣的数据结构
- 探索Python在金融学和统计学领域相关计算的多种可视化选择
- 了解为什么Python成为继Java后的第二种选择，以及为何Python在机器学习领域使用的如此频繁
- 比较Python和其他可视化方法，包括Julia及一种基于JavaScript的框架（如D3.js）
- 发现Python如何与NoSQL（如Hive）结合使用，在分布式环境中高效生成结果

[PACKT] PUBLISHING

投稿热线：(010) 88379604
客服热线：(010) 88379426 88361066
购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn

上架指导：计算机/数据分析

ISBN 978-7-111-56090-6

9 787111 560906 >

定价：69.00元