

O'REILLY®

2nd Edition

# 利用PYTHON 进行数据分析

DATA WRANGLING WITH PANDAS,  
NUMPY, AND IPYTHON



powered by



SeanCheney等译

Wes McKinney

# 有关此电子书试读版的说明

本人可以帮助你找到你要的高清PDF电子书，计算机类，文学，艺术，设计，医学，理学，经济，金融等等。

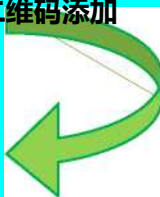
质量都很清晰，为方便读者阅读观看，每本100%。都带可跳转的书签索引和目录。

一般情况下，出版半年左右就会有PDF 极个别的书出PDF 时间要长一些

如看到试读版信息.说明已经有完整版，需求完整版即可联系我。

请添加 **QQ 312082710** 和**微信**

312082710或扫描微信二维码添加



PDF 代找说明：

本人已经帮助了上万人找到了他们需要的PDF ,其实网上有很多PDF,

大家如果在网上不到的话，可以联系我**QQ 312082710**或**微信312082710**

大部分我都可以找到，而且每本100%带书签索引目录。

因PDF 电子书都有版权，请不要随意传播，如果您有经济购买能力，请尽量购买正版。

提供各种书籍的高清PDF 电子版代找服务，如果你找不到自己想要的书的PDF 电子版，我们可以帮您找到，如有需要，请联系QQ 2028969416微信2028969416 备用 QQ 202896416  
若以上联系方式失效，您可通过以下电子邮件获取有效联系方式。邮箱：

[312082710@qq.com](mailto:312082710@qq.com)

若您没有QQ 通讯工具，请点击下面链接与客服取得联系。&

<https://item.taobao.com/item.htm?id=565681036651>

声明：本人只提供代找服务，每本100%索引|书签和目录，因寻找和后期制作pdf 电子书有一定难度

仅收取代找费用。如因PDF 产生的版权纠纷，与本人无关，我们仅仅只是帮助你寻找到你要的PDF 而已。

## 利用 **Python** 进行数据分析 中文第二版

---

译者: [SeanCheney](#)

来源: <https://www.jianshu.com/c/52882df3377a>

下载本

书: <http://www.jianshu.com/p/fad9e41c1a42> (更新为GitHub链接)

下载本书代码: <https://github.com/wesm/pydata-book> (建议把代码下载下来之后, 安装好Anaconda 3.6, 在目录文件夹中用Jupyter notebook打开)

---

本书是2017年10月20号正式出版的, 和第1版的不同之处有:

- 包括Python教程内的所有代码升级为Python 3.6 (第1版使用的是Python 2.7)
  - 更新了Anaconda和其它包的Python安装方法
  - 更新了Pandas为2017最新版
  - 新增了一章, 关于更高级的Pandas工具, 外加一些tips
  - 简要介绍了使用StatsModels和scikit-learn
- 

对有些内容进行了重新排版。

译者注1: 最大的改变是把第1版附录中的Python教程, 单列成了现在的第2章和第3章, 并且进行了扩充。可以说, 本书第2版对新手更为友好了!

译者注2：毫无疑问，本书是学习Python数据分析最好的参考书。本来想把书名直接译为《Python数据分析》，这样更简短。但是为了尊重第1版的翻译，考虑到继承性，还是用老书名。这样读过第一版的老读者可以方便的用之前的书名检索到第二版。作者在写第二版的时候，有些文字是照搬第一版的。所以第二版的翻译也借鉴copy了第一版翻译：即，如果第二版中有和第一版相同的文字，则copy第一版的中文译本，觉得不妥的地方会稍加修改，剩下的不同的内容就自己翻译。这样做也是为读过第一版的老读者考虑——相同的内容可以直接跳过。

## 第1章 准备工作

---

## 1.1 本书的内容

---

本书讲的是利用**Python**进行数据控制、处理、整理、分析等方面的具体细节和基本要点。我的目标是介绍**Python**编程和用于数据处理的库和工具环境，掌握这些，可以让你成为一个数据分析专家。虽然本书的标题是“数据分析”，重点确实**Python**编程、库，以及用于数据分析的工具。这就是数据分析要用到的**Python**编程。

## 什么样的数据？

当书中出现“数据”时，究竟指的是什么呢？主要指的是结构化数据（**structured data**），这个故意含糊其辞的术语代指了所有通用格式的数据，例如：

- 表格型数据，其中各列可能是不同的类型（字符串、数值、日期等）。比如保存在关系型数据库中或以制表符/逗号为分隔符的文本文件中的那些数据。
- 多维数组（矩阵）。
- 通过关键列（对于**SQL**用户而言，就是主键和外键）相互联系的多个表。
- 间隔平均或不平均的时间序列。

这绝不是一个完整的列表。大部分数据集都能被转化为更加适合分析和建模的结构化形式，虽然有时这并不是很明显。如果不行的话，也可以将数据集的特征提取为某种结构化形式。例如，一组新闻文章可以被处理为一张词频表，而这张词频表就可以用于情感分析。

大部分电子表格软件（比如**Microsoft Excel**，它可能是世界上使用最广泛的数据分析工具了）的用户不会对此类数据感到陌生。

## 1.2 为什么要使用Python进行数据分析

---

许许多多的人（包括我自己）都很容易爱上Python这门语言。自从1991年诞生以来，Python现在已经成为最受欢迎的动态编程语言之一，其他还有Perl、Ruby等。由于拥有大量的Web框架（比如Rails（Ruby）和Django（Python）），自从2005年，非常流行使用Python和Ruby进行网站建设工作。这些语言常被称作脚本（scripting）语言，因为它们可以用于编写简短而粗糙的小程序（也就是脚本）。我个人并不喜欢“脚本语言”这个术语，因为它好像在说这些语言无法用于构建严谨的软件。在众多解释型语言中，由于各种历史和文化的因素，Python发展出了一个巨大而活跃的科学计算（scientific computing）社区。在过去的10年，Python从一个边缘或“自担风险”的科学计算语言，成为了数据科学、机器学习、学界和工业界软件开发最重要的语言之一。

在数据分析、交互式计算以及数据可视化方面，Python将不可避免地与其他开源和商业的领域特定编程语言/工具进行对比，如R、MATLAB、SAS、Stata等。近年来，由于Python的库（例如pandas和scikit-learn）不断改良，使其成为数据分析任务的一个优选方案。结合其在通用编程方面的强大实力，我们完全可以只使用Python这一种语言构建以数据为中心的应用。



## Python作为胶水语言

Python能变为成功的科学计算工具的部分原因是，它能够轻松地集成C、C++以及Fortran代码。大部分现代计算环境都利用了一些Fortran和C库来实现线性代数、优选、积分、快速傅里叶变换以及其他诸如此类的算法。许多企业和国家实验室也利用Python来“粘合”那些已经用了多年的遗留软件系统。

大多数软件都是由两部分代码组成的：少量需要占用大部分执行时间的代码，以及大量不经常执行的“胶水代码”。大部分情况下，胶水代码的执行时间是微不足道的。开发人员的精力几乎都是花在优化计算瓶颈上面，有时更是直接转用更低级的语言（比如C）。

## 解决“两种语言”问题

很多组织通常都会用一种类似于领域特定的计算语言（如**SAS**和**R**）对新的想法进行研究、原型构建和测试，然后再将这些想法移植到某个更大的生产系统中去（可能是用**Java**、**C#**或**C++**编写的）。人们逐渐意识到，**Python**不仅适用于研究和原型构建，同时也适用于构建生产系统。为什么一种语言就够了，却要使用两个语言的开发环境呢？我相信越来越多的企业也会这样看，因为研究人员和工程技术人员使用同一种编程工具将会给企业带来非常显著的组织效益。

## 为什么不选Python

虽然Python非常适合构建分析应用以及通用系统，但它对不少应用场景适用性较差。

由于Python是一种解释型编程语言，因此大部分Python代码都要比用编译型语言（比如Java和C++）编写的代码运行慢得多。由于程序员的时间通常都比CPU时间值钱，因此许多人也愿意在这里做一些权衡。但是，在那些要求延迟非常小或高资源利用率的应用中（例如高频交易系统），耗费时间使用诸如C++这样更低级、更低生产率的语言进行编程也是值得的。

对于高并发、多线程的应用程序而言（尤其是拥有许多计算密集型线程的应用程序），Python并不是一种理想的编程语言。这是因为Python有一个叫做全局解释器锁（Global Interpreter Lock, GIL）的组件，这是一种防止解释器同时执行多条Python字节码指令的机制。有关“为什么会存在GIL”的技术性原因超出了本书的范围。虽然很多大数据处理应用程序为了能在较短的时间内完成数据集的处理工作都需要运行在计算机集群上，但是仍然有一些情况需要用单进程多线程系统来解决。

这并不是说Python不能执行真正的多线程并行代码。例如，Python的C插件使用原生的C或C++的多线程，可以并行运行而不被GIL影响，只要它们不频繁地与Python对象交互。

## 1.3 重要的Python库

---

考虑到那些还不太了解Python科学计算生态系统和库的读者，下面我先对各个库做一个简单的介绍。

## NumPy

NumPy（Numerical Python的简称）是Python科学计算的基础包。本书大部分内容都基于NumPy以及构建于其上的库。它提供了以下功能（不限于此）：

- 快速高效的多维数组对象`ndarray`。
- 用于对数组执行元素级计算以及直接对数组执行数学运算的函数。
- 用于读写硬盘上基于数组的数据集的工具。
- 线性代数运算、傅里叶变换，以及随机数生成。
- 成熟的C API，用于Python插件和原生C、C++、Fortran代码访问NumPy的数据结构和计算工具。

除了为Python提供快速的数组处理能力，NumPy在数据分析方面还有另外一个主要作用，即作为在算法和库之间传递数据的容器。对于数值型数据，NumPy数组在存储和处理数据时要比内置的Python数据结构高效得多。此外，由低级语言（比如C和Fortran）编写的库可以直接操作NumPy数组中的数据，无需进行任何数据复制工作。因此，许多Python的数值计算工具要么使用NumPy数组作为主要的数据结构，要么可以与NumPy进行无缝交互操作。

## pandas

pandas提供了快速便捷处理结构化数据的大量数据结构和函数。自从2010年出现以来，它助使Python成为强大而高效的数据分析环境。本书用得最多的pandas对象是**DataFrame**，它是一个面向列（**column-oriented**）的二维表结构，另一个是**Series**，一个一维的标签化数组对象。

pandas兼具NumPy高性能的数组计算功能以及电子表格和关系型数据库（如**SQL**）灵活的数据处理功能。它提供了复杂精细的索引功能，以便更为便捷地完成重塑、切片和切块、聚合以及选取数据子集等操作。因为数据操作、准备、清洗是数据分析最重要的技能，pandas是本书的重点。

作为一点背景，我是在2008年初开始开发pandas的，那时我任职于**AQR Capital Management**，一家量化投资管理公司，我有许多工作需求都不能用任何单一的工具解决：

- 有标签轴的数据结构，支持自动或清晰的数据对齐。这可以防止由于数据不对齐，和处理来源不同的索引不同的数据，造成的错误。
- 集成时间序列功能。
- 相同的数据结构用于处理时间序列数据和非时间序列数据。
- 保存元数据的算术运算和压缩。
- 灵活处理缺失数据。

- 合并和其它流行数据库（例如基于**SQL**的数据库）的关系操作。

我想只用一种工具就实现所有功能，并使用通用软件开发语言。**Python**是一个不错的候选语言，但是此时没有集成的数据结构和工具来实现。我一开始就是想把**pandas**设计为一款适用于金融和商业分析的工具，**pandas**专注于深度时间序列功能和工具，适用于时间索引化的数据。

对于使用**R**语言进行统计计算的用户，肯定不会对**DataFrame**这个名字感到陌生，因为它源自于**R**的**data.frame**对象。但与**Python**不同，**data frames**是构建于**R**和它的标准库。因此，**pandas**的许多功能不属于**R**或它的扩展包。

**pandas**这个名字源于**panel data**（面板数据，这是多维结构化数据集在计量经济学中的术语）以及**Python data analysis**（**Python**数据分析）。

## **matplotlib**

**matplotlib**是最流行的用于绘制图表和其它二维数据可视化的**Python**库。它最初由**John D.Hunter**（JDH）创建，目前由一个庞大的开发人员团队维护。它非常适合创建出版物上用的图表。虽然还有其它的**Python**可视化库，**matplotlib**却是使用最广泛的，并且它和其它生态工具配合也非常完美。我认为，可以使用它作为默认的可视化工具。



## IPython和Jupyter

IPython项目起初是Fernando Pérez在2001年的一个用以加强和Python交互的子项目。在随后的16年中，它成为了Python数据栈最重要的工具之一。虽然IPython本身没有提供计算和数据分析的工具，它却可以大大提高交互式计算和软件开发的生产率。IPython鼓励“执行-探索”的工作流，区别于其它编程软件的“编辑-编译-运行”的工作流。它还可以方便地访问系统的shell和文件系统。因为大部分的数据分析代码包括探索、试错和重复，IPython可以使工作更快。

2014年，Fernando和IPython团队宣布了Jupyter项目，一个更宽泛的多语言交互计算工具的计划。

IPython web notebook变成了Jupyter notebook，现在支持40种编程语言。IPython现在可以作为Jupyter使用Python的内核（一种编程语言模式）。

IPython变成了Jupyter庞大开源项目（一个交互和探索式计算的高效环境）中的一个组件。它最老也是最简单的模式，现在是一个用于编写、测试、调试Python代码的强化shell。你还可以使用通过Jupyter Notebook，一个支持多种语言的交互式网络代码“笔记本”，来使用IPython。IPython shell 和Jupyter notebooks特别适合进行数据探索和可视化。

Jupyter notebooks还可以编写Markdown和HTML内容，提供了一种创建代码和文本的富文本方法。其它编程语言也在Jupyter中植入了内核，好让在Jupyter中可

以使用**Python**另外的语言。

对我个人而言，我的大部分**Python**都要用到**IPython**，包括运行、调试和测试代码。

在本书的**GitHub**页面，你可以找到包含各章节所有代码实例的**Jupyter notebooks**。

## SciPy

SciPy是一组专门解决科学计算中各种标准问题域的包的集合，主要包括下面这些包：

- **scipy.integrate**: 数值积分例程和微分方程求解器。
- **scipy.linalg**: 扩展了由**numpy.linalg**提供的线性代数例程和矩阵分解功能。
- **scipy.optimize**: 函数优化器（最小化器）以及根查找算法。
- **scipy.signal**: 信号处理工具。
- **scipy.sparse**: 稀疏矩阵和稀疏线性系统求解器。
- **scipy.special**: **SPECFUN**（这是一个实现了许多常用数学函数（如伽玛函数）的**Fortran**库）的包装器。
- **scipy.stats**: 标准连续和离散概率分布（如密度函数、采样器、连续分布函数等）、各种统计检验方法，以及更好的描述统计法。

**NumPy**和**SciPy**结合使用，便形成了一个相当完备和成熟的计算平台，可以处理多种传统的科学计算问题。

## scikit-learn

2010年诞生以来，**scikit-learn**成为了Python的通用机器学习工具包。仅仅七年，就汇聚了全世界超过1500名贡献者。它的子模块包括：

- 分类：**SVM**、近邻、随机森林、逻辑回归等等。
- 回归：**Lasso**、岭回归等等。
- 聚类：**k-均值**、谱聚类等等。
- 降维：**PCA**、特征选择、矩阵分解等等。
- 选型：网格搜索、交叉验证、度量。
- 预处理：特征提取、标准化。

与**pandas**、**statsmodels**和**IPython**一起，**scikit-learn**对于Python成为高效数据科学编程语言起到了关键作用。虽然本书不会详细讲解**scikit-learn**，我会简要介绍它的一些模型，以及用其它工具如何使用这些模型。

## statsmodels

**statsmodels**是一个统计分析包，起源于斯坦福大学统计学教授Jonathan Taylor，他设计了多种流行于R语言的回归分析模型。Skipper Seabold和Josef Perktold在2010年正式创建了**statsmodels**项目，随后汇聚了大量的使用者和贡献者。受到R的公式系统的启发，Nathaniel Smith发展出了Patsy项目，它提供了**statsmodels**的公式或模型的规范框架。

与**scikit-learn**比较，**statsmodels**包含经典统计学和经济计量学的算法。包括如下子模块：

- 回归模型：线性回归，广义线性模型，健壮线性模型，线性混合效应模型等等。
- 方差分析（ANOVA）。
- 时间序列分析：AR，ARMA，ARIMA，VAR和其它模型。
- 非参数方法：核密度估计，核回归。
- 统计模型结果可视化。

**statsmodels**更关注与统计推断，提供不确定估计和参数p-值。相反的，**scikit-learn**注重预测。

同**scikit-learn**一样，我也只是简要介绍**statsmodels**，以及如何用NumPy和pandas使用它。

## 1.4 安装和设置

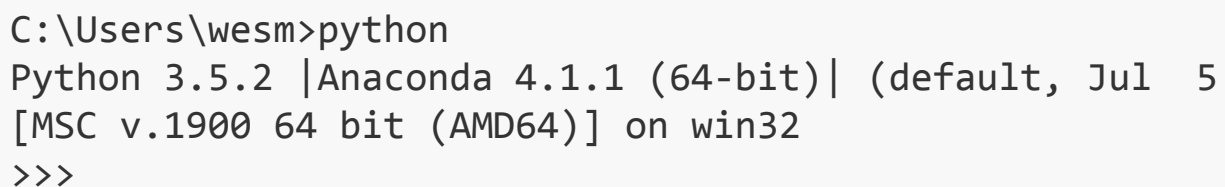
---

由于人们用Python所做的事情不同，所以没有一个普适的Python及其插件包的安装方案。由于许多读者的Python科学计算环境都不能完全满足本书的需要，所以接下来我将详细介绍各个操作系统上的安装方法。我推荐免费的Anaconda安装包。写作本书时，Anaconda提供Python 2.7和3.6两个版本，以后可能发生变化。本书使用的是Python 3.6，因此推荐选择Python 3.6或更高版本。

## Windows

要在Windows上运行，先下载[Anaconda安装包](#)。推荐跟随Anaconda下载页面的Windows安装指导，安装指导在写作本书和读者看到此文的的这段时间内可能发生变化。

现在，来确认设置是否正确。打开命令行窗口（`cmd.exe`），输入`python`以打开Python解释器。可以看到类似下面的Anaconda版本的输出：



```
C:\Users\wesm>python
Python 3.5.2 |Anaconda 4.1.1 (64-bit)| (default, Jul  5
[MSC v.1900 64 bit (AMD64)] on win32
>>>
```

要退出shell，按Ctrl-D（Linux或macOS上），Ctrl-Z（Windows上），或输入命令`exit()`，再按Enter。

## Apple (OS X, macOS)

下载OS X Anaconda安装包，它的名字类似Anaconda3-4.1.0-MacOSX-x86\_64.pkg。双击.pkg文件，运行安装包。安装包运行时，会自动将Anaconda执行路径添加到.bash\_profile文件，它位于/Users/\$USER/.bash\_profile。

为了确认成功，在系统shell打开IPython:

```
$ ipython
```

要退出shell，按Ctrl-D，或输入命令exit()，再按Enter。



## GNU/Linux

Linux版本很多，这里给出Debian、Ubuntu、CentOS和Fedora的安装方法。安装包是一个脚本文件，必须在shell中运行。取决于系统是32位还是64位，要么选择x86 (32位)或x86\_64 (64位)安装包。随后你会得到一个文件，名字类似

于Anaconda3-4.1.0-Linux-x86\_64.sh。用bash进行安装：

```
$ bash Anaconda3-4.1.0-Linux-x86_64.sh
```

笔记：某些Linux版本在包管理器中有满足需求的Python包，只需用类似apt的工具安装就行。这里讲的用Anaconda安装，适用于不同的Linux安装包，也很容易将包升级到最新版本。

接受许可之后，会向你询问在哪里放置Anaconda的文件。我推荐将文件安装到默认的home目录，例如/home/\$USER/anaconda。

Anaconda安装包可能会询问你是否将bin/目录添加到\$PATH变量。如果在安装之后有任何问题，你可以修改文件.bashrc（或.zshrc，如果使用的是zsh shell）为类似以下内容：

```
export PATH=/home/$USER/anaconda/bin:$PATH
```

做完之后，你可以开启一个新窗口，或再次用 ~/.bashrc 执行.bashrc。

## 安装或升级Python包

在你阅读本书的时候，你可能想安装另外的不在Anaconda中的Python包。通常，可以用以下命令安装：

```
conda install package_name
```

如果这个命令不行，也可以用pip包管理工具：

```
pip install package_name
```

你可以用conda update命令升级包：

```
conda update package_name
```

pip可以用--upgrade升级：

```
pip install --upgrade package_name
```

本书中，你有许多机会尝试这些命令。

注意：当你使用conda和pip二者安装包时，千万不要用pip升级conda的包，这样会导致环境发生问题。当使用Anaconda或Miniconda时，最好首先使用conda进行升级。

## Python 2 和 Python 3

第一版的Python 3.x出现于2008年。它有一系列的变化，与之前的Python 2.x代码有不兼容的地方。因为从1991年Python出现算起，已经过了17年，Python 3 的

出现被视为吸取一些列教训的更优结果。

2012年，因为许多包还没有完全支持Python 3，许多科学和数据分析社区还是在使用Python 2.x。因此，本书第一版使用的是Python 2.7。现在，用户可以在Python 2.x和Python 3.x间自由选择，二者都有良好的支持。

但是，Python 2.x在2020年就会到期（包括重要的安全补丁），因此再用Python 2.7就不是好的选择了。因此，本书使用了Python 3.6，这一广泛使用、支持良好的稳定版本。我们已经称Python 2.x为“遗留版本”，简称Python 3.x为“Python”。我建议你也是如此。

本书基于Python 3.6。你的Python版本也许高于3.6，但是示例代码应该是向前兼容的。一些示例代码可能在Python 2.7上有所不同，或完全不兼容。

## 集成开发环境（**IDEs**）和文本编辑器

当被问到我的标准开发环境，我几乎总是回答“IPython加文本编辑器”。我通常在编程时，反复在IPython或Jupyter notebooks中测试和调试每条代码。也可以交互式操作数据，和可视化验证数据操作中某一特殊集合。在shell中使用pandas和NumPy也很容易。

但是，当创建软件时，一些用户可能更想使用特点更为丰富的IDE，而不仅仅是原始的蕾西Emacs或Vim的文本编辑器。以下是一些IDE：

- PyDev（免费），基于Eclipse平台的IDE；
- JetBrains的PyCharm（商业用户需要订阅，开源开发者免费）；
- Visual Studio（Windows用户）的Python Tools；
- Spyder（免费），Anaconda附带的IDE；
- Komodo IDE（商业）。

因为Python的流行，大多数文本编辑器，比如Atom和Sublime Text 3，对Python的支持也非常好。

## 1.5 社区和会议

---

除了在网上搜索，各式各样的科学和数据相关的Python邮件列表是非常有帮助的，很容易获得回答。包括：

- **pydata**: 一个Google群组列表，用以回答Python数据分析和pandas的问题；
- **pystatsmodels**: statsmodels或pandas相关的问题；
- **scikit-learn**和Python机器学习邮件列表， [scikit-learn@python.org](mailto:scikit-learn@python.org)；
- **numpy-discussion**: 和NumPy相关的问题；
- **scipy-user**: SciPy和科学计算的问题；

因为这些邮件列表的URLs可以很容易搜索到，但因为可能发生变化，所以没有给出。

每年，世界各地会举办许多Python开发者大会。如果你想结识其他有相同兴趣的人，如果可能的话，我建议你参加一个。许多会议会对无力支付入场费和差旅费的人提供财力帮助。下面是一些会议：

- **PyCon**和**EuroPython**: 北美和欧洲的两大Python会议；
- **SciPy**和**EuroSciPy**: 北美和欧洲两大面向科学计算的会议；
- **PyData**: 世界范围内，一些列的地区性会议，专注数据科学和数据分析；

- 国际和地区的PyCon会议（<http://pycon.org>有完整列表）。

## 1.6 本书导航

---

如果之前从未使用过Python，那你可能需要先看看本书的第2章和第3章，我简要介绍了Python的特点，IPython和Jupyter notebooks。这些知识是为本书后面的内容做铺垫。如果你已经掌握Python，可以选择跳过。

接下来，简单地介绍了NumPy的关键特性，附录A中是更高级的NumPy功能。然后，我介绍了pandas，本书剩余的内容全部是使用pandas、NumPy和matplotlib处理数据分析的问题。我已经尽量让全书的结构循序渐进，但偶尔会有章节之间的交叉，有时用到的概念还没有介绍过。

尽管读者各自的工作任务不同，大体可以分为几类：

- 与外部世界交互  
阅读编写多种文件格式和数据商店；
- 数据准备  
清洗、修改、结合、标准化、重塑、切片、切割、转换数据，以进行分析；
- 转换数据  
对旧的数据集进行数学和统计操作，生成新的数据集（例如，通过各组变量聚类成大的表）；
- 建模和计算  
将数据绑定统计模型、机器学习算法、或其他计算工具；
- 展示

创建交互式 and 静态的图表可视化和文本总结。



## 代码示例

本书大部分代码示例的输入形式和输出结果都会按照其在IPython shell或Jupyter notebooks中执行时的样子进行排版：

```
In [5]: CODE EXAMPLE
```

```
Out[5]: OUTPUT
```

但你看到类似的示例代码，就是让你在in的部分输入代码，按Enter键执行（Jupyter中是按Shift-Enter）。然后就可以在out看到输出。

## 示例数据

各章的示例数据都存放在GitHub

上：<http://github.com/pydata/pydata-book>。下载这些数据的方法有二：使用git版本控制命令行程序；直接从网站上下载该GitHub库的zip文件。如果遇到了问题，可以到我的个人主页，<http://wesmckinney.com/>，获取最新的指导。

为了让所有示例都能重现，我已经尽我所能使其包含所有必需的东西，但仍然可能会有一些错误或遗漏。如果出现这种情况的话，请给我发邮

件：[wesmckinn@gmail.com](mailto:wesmckinn@gmail.com)。报告本书错误的最好方法是O'Reilly的errata页面，[http://www.bit.ly/pyDataAnalysis\\_errata](http://www.bit.ly/pyDataAnalysis_errata)。

## 引入惯例

Python社区已经广泛采取了一些常用模块的命名惯例：

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import statsmodels as sm
```

也就是说，当你看到`np.arange`时，就应该想到它引用的是NumPy中的`arange`函数。这样做的原因是：在Python软件开发过程中，不建议直接引入类似NumPy这种大型库的全部内容（`from numpy import *`）。

## 行话

由于你可能不太熟悉书中使用的一些有关编程和数据科学方面的常用术语，所以我在这里先给出其简单定义：

### 数据规整（Munge/Munging/Wrangling）

指的是将非结构化和（或）散乱数据处理为结构化或整洁形式的整个过程。这几个词已经悄悄成为当今数据黑客们的行话了。**Munge**这个词跟**Lunge**押韵。

### 伪码（Pseudocode）

算法或过程的“代码式”描述，而这些代码本身并不是实际有效的源代码。

### 语法糖（Syntactic sugar）

这是一种编程语法，它并不会带来新的特性，但却能使代码更易读、更易写。

---

## 第2章 Python语法基础，IPython和Jupyter Notebooks

---

当我在2011年和2012年写作本书的第一版时，可用的学习Python数据分析的资源很少。这部分上是一个鸡和蛋的问题：我们现在使用的库，比如pandas、scikit-learn和statsmodels，那时相对来说并不成熟。2017年，数据科学、数据分析和机器学习的资源已经很多，原来通用的科学计算拓展到了计算机科学家、物理学家和其它研究领域的工作人员。学习Python和成为软件工程师的优秀书籍也有了。

因为这本书是专注于Python数据处理的，对于一些Python的数据结构和库的特性难免不足。因此，本章和第3章的内容只够你能学习本书后面的内容。

在我来看，没有必要为了数据分析而去精通Python。我鼓励你使用IPython shell和Jupyter试验示例代码，并学习不同类型、函数和方法的文档。虽然我已尽力让本书内容循序渐进，但读者偶尔仍会碰到没有之前介绍过的内容。

本书大部分内容关注的是基于表格的分析和处理大规模数据集的数据准备工具。为了使用这些工具，必须首先将混乱的数据规整为整洁的表格（或结构化）形式。幸好，Python是一个理想的语言，可以快速整理数据。使用Python越熟练，越容易准备新的数据集以进行分析。

本书中使用的工具最好在IPython和Jupyter中亲自尝

试。当你学会了如何启用Ipython和Jupyter，我建议你跟随示例代码进行练习。与任何键盘驱动的操作环境一样，记住常见的命令也是学习曲线的一部分。

笔记：本章没有介绍Python的某些概念，如类和面向对象编程，你可能会发现它们在Python数据分析中很有用。

为了加强Python知识，我建议你学习官方Python教程，<https://docs.python.org/3/>，或是通用的Python教程书籍，比如：

- Python Cookbook，第3版，David Beazley和Brian K. Jones著（O'Reilly）
- 流畅的Python，Luciano Ramalho著 (O'Reilly)
- 高效的Python，Brett Slatkin著 (Pearson)

## 2.1 Python解释器

Python是解释性语言。Python解释器同一时间只能运行一个程序的一条语句。标准的交互Python解释器可以在命令行中通过键入python命令打开：

```
$ python
Python 3.6.0 | packaged by conda-forge | (default, Jan 1
[GCC 4.8.2 20140120 (Red Hat 4.8.2-15)] on linux
Type "help", "copyright", "credits" or "license" for mor
>>> a = 5
>>> print(a)
5
```

>>>提示输入代码。要退出Python解释器返回终端，可以输入exit()或按Ctrl-D。

运行Python程序只需调用Python的同时，使用一个.py文件作为它的第一个参数。假设创建了一个hello\_world.py文件，它的内容是：

```
print('Hello world')
```

你可以用下面的命令运行它（hello\_world.py文件必须位于终端的工作目录）：

```
$ python hello_world.py
Hello world
```

一些Python程序员总是这样执行Python代码的，从事数据分析和科学计算的人却会使用IPython，一个强化

的Python解释器，或Jupyter notebooks，一个网页代码笔记本，它原先是IPython的一个子项目。在本章中，我介绍了如何使用IPython和Jupyter，在附录A中有更深入的介绍。当你使用`%run`命令，IPython会同样执行指定文件中的代码，结束之后，还可以与结果交互：

```
$ ipython
Python 3.6.0 | packaged by conda-forge | (default, Jan 1
Type "copyright", "credits" or "license" for more inform

IPython 5.1.0 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's feat
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for

In [1]: %run hello_world.py
Hello world

In [2]:
```

IPython默认采用序号的格式`In [2]:`，与标准的`>>>`提示符不同。



## 2.2 IPython基础

---

在本节中，我们会教你打开运行IPython shell和jupyter notebook，并介绍一些基本概念。

## 运行IPython Shell

你可以用`ipython`在命令行打开IPython Shell，就像打开普通的Python解释器：

```
$ ipython
Python 3.6.0 | packaged by conda-forge | (default, Jan 1
Type "copyright", "credits" or "license" for more inform

IPython 5.1.0 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's feat
%quickref        -> Quick reference.
help            -> Python's own help system.
object?         -> Details about 'object', use 'object??' for

In [1]: a = 5
In [2]: a
Out[2]: 5
```

你可以通过输入代码并按**Return**（或**Enter**），运行任意Python语句。当你只输入一个变量，它会显示代表的对象：

```
In [5]: import numpy as np

In [6]: data = {i : np.random.randn() for i in range(7)}

In [7]: data
Out[7]:
{0: -0.20470765948471295,
 1: 0.47894333805754824,
 2: -0.5194387150567381,
 3: -0.55573030434749,
```

```
4: 1.9657805725027142,  
5: 1.3934058329729904,  
6: 0.09290787674371767}
```

前两行是Python代码语句；第二条语句创建一个名为`data`的变量，它引用一个新创建的Python字典。最后一行打印`data`的值。

许多Python对象被格式化为更易读的形式，或称作`pretty-printed`，它与普通的`print`不同。如果在标准Python解释器中打印上述`data`变量，则可读性要降低：

```
>>> from numpy.random import randn  
>>> data = {i : randn() for i in range(7)}  
>>> print(data)  
{0: -1.5948255432744511, 1: 0.10569006472787983, 2: 1.97  
3: 0.15455217573074576, 4: -0.24058577449429575, 5: -1.2  
6: 0.3308507317325902}
```

IPython还支持执行任意代码块（通过一个华丽的复制-粘贴方法）和整段Python脚本的功能。你也可以使用Jupyter notebook运行大代码块，接下来就会看到。

## 运行Jupyter Notebook

Jupyter项目的重要组件之一就是notebook，它是一个代码、文本（有标记或无标记）、数据可视化或其它输出的交互式文档。Jupyter Notebook需要与内核互动，内核是Jupyter与其它编程语言的交互编程协议。

Python的Jupyter内核是使用IPython。要启动Jupyter，在命令行中输入jupyter notebook：

```
$ jupyter notebook
[I 15:20:52.739 NotebookApp] Serving notebooks from local
/home/wesm/code/pydata-book
[I 15:20:52.739 NotebookApp] 0 active kernels
[I 15:20:52.739 NotebookApp] The Jupyter Notebook is run
http://localhost:8888/
[I 15:20:52.740 NotebookApp] Use Control-C to stop this
all kernels (twice to skip confirmation).
Created new window in existing browser session.
```

在多数平台上，Jupyter会自动打开默认的浏览器（除非指定了`--no-browser`）。或者，可以在启动notebook之后，手动打开网页<http://localhost:8888/>。图2-1展示了Google Chrome中的notebook。

笔记：许多人使用Jupyter作为本地的计算环境，但它也可以部署到服务器上远程访问。这里不做介绍，如果需要的话，鼓励读者自行到网上学习。



图2-1 Jupyter notebook启动页面

要新建一个notebook，点击按钮**New**，选择“**Python3**”或“**conda[默认项]**”。如果是第一次，点击空格，输入一行**Python**代码。然后按**Shift-Enter**执行。

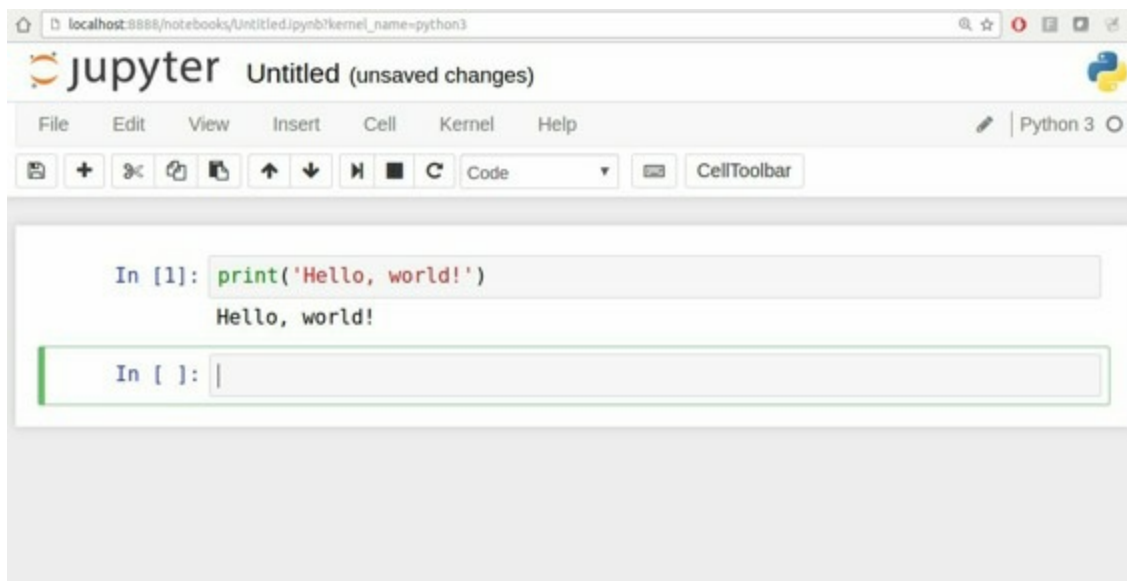


图2-2 Jupyter新notebook页面

当保存notebook时（**File**目录下的**Save and Checkpoint**），会创建一个后缀名为**.ipynb**的文件。这

是一个自包含文件格式，包含当前笔记本中的所有内容（包括所有已评估的代码输出）。可以被其它Jupyter用户加载和编辑。要加载存在的notebook，把它放到启动notebook进程的相同目录内。你可以用本书的示例代码练习，见图2-3。

虽然Jupyter notebook和IPython shell使用起来不同，本章中几乎所有的命令和工具都可以通用。

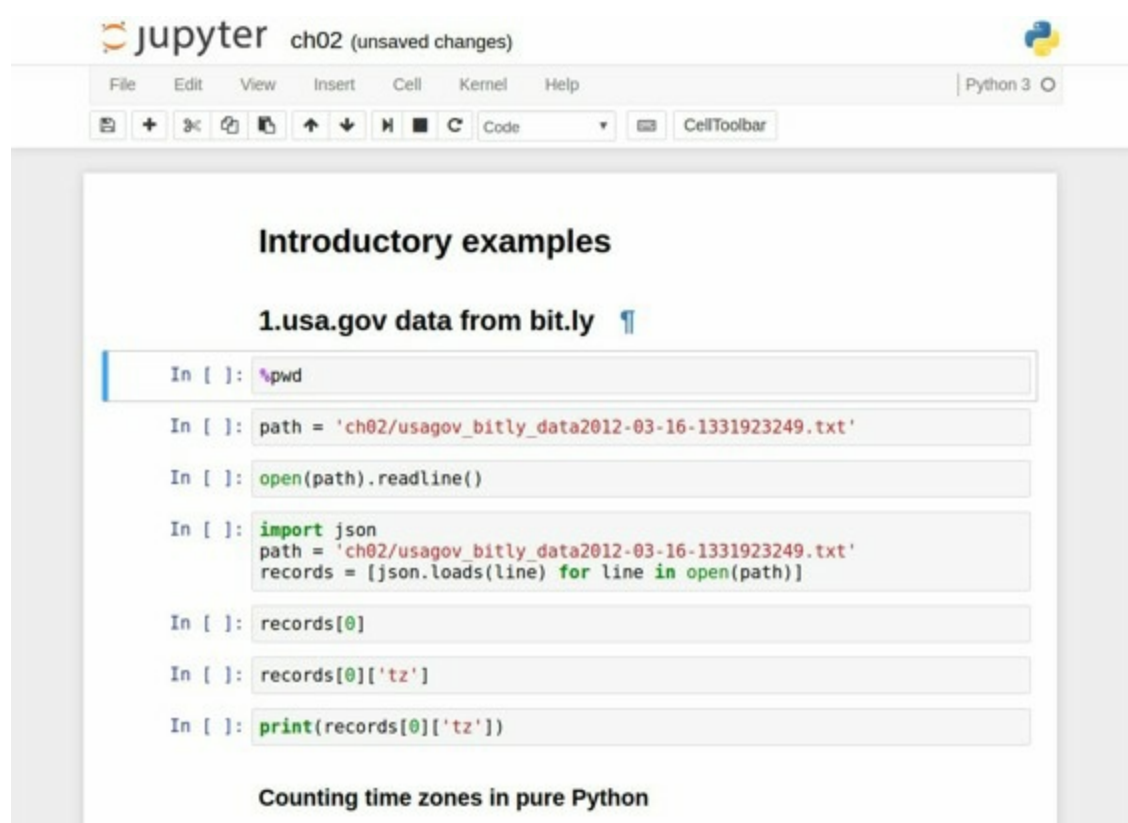


图2-3 Jupyter查看一个存在的notebook的页面

## Tab补全

从外观上，IPython shell和标准的Python解释器只是看起来不同。IPython shell的进步之一是其它IDE和交互计算分析环境都有的tab补全功能。在shell中输入表达式，按下Tab，会搜索已输入变量（对象、函数等等）的命名空间：

```
In [1]: an_apple = 27

In [2]: an_example = 42

In [3]: an<Tab>
an_apple    and                an_example  any
```

在这个例子中，IPython呈现除了之前两个定义的变量和Python的关键字和内建的函数any。当然，你也可以补全任何对象的方法和属性：

```
In [3]: b = [1, 2, 3]

In [4]: b.<Tab>
b.append    b.count      b.insert    b.reverse
b.clear     b.extend    b.pop       b.sort
b.copy      b.index     b.remove
```

同样也适用于模块：

```
In [1]: import datetime

In [2]: datetime.<Tab>
datetime.date          datetime.MAXYEAR      datetime.t
datetime.datetime     datetime.MINYEAR      datetime.t
```



在Jupyter notebook和新版的IPython（5.0及以上），自动补全功能是下拉框的形式。

笔记：注意，默认情况下，IPython会隐藏下划线开头的方法和属性，比如魔术方法和内部的“私有”方法和属性，以避免混乱的显示（和让新手迷惑！）这些也可以tab补全，但是你必须首先键入一个下划线才能看到它们。如果你喜欢总是在tab补全中看到这样的方法，你可以IPython配置中进行设置。可以在IPython文档中查找方法。

除了补全命名、对象和模块属性，Tab还可以补全其它的。当输入看似文件路径时（即使是Python字符串），按下Tab也可以补全电脑上对应的文件信息：



结合%run，tab补全可以节省许多键盘操作。

另外，tab补全可以补全函数的关键词参数（包括等于号=）。见图2-4。



```
In [12]: def func_with_keywords(abra=1, abbra=2, abbbra=3):  
         return abra, abbra, abbbra  
  
In [ ]: func_with_keywords(ab|  
         abbbra=  
         abbra=  
         abra=  
         abs
```

图2-4 Jupyter notebook中自动补全函数关键词  
我们来仔细看看函数。

## 自省

在变量前后使用问号？，可以显示对象的信息：

```
In [8]: b = [1, 2, 3]

In [9]: b?
Type:      list
String Form:[1, 2, 3]
Length:    3
Docstring:
list() -> new empty list
list(iterable) -> new list initialized from iterable's i

In [10]: print?
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, fl

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline
flush: whether to forcibly flush the stream.
Type:  builtin_function_or_method
```

这可以作为对象的自省。如果对象是一个函数或实例方法，定义过的文档字符串，也会显示出信息。假设我们写了一个如下的函数：

```
def add_numbers(a, b):
    """
    Add two numbers together
```

```
Returns
-----
the_sum : type of arguments
"""
return a + b
```

然后使用?符号，就可以显示如下的文档字符串：

```
In [11]: add_numbers?
Signature: add_numbers(a, b)
Docstring:
Add two numbers together

Returns
-----
the_sum : type of arguments
File:      <ipython-input-9-6a548a216e27>
Type:      function
```

使用??会显示函数的源码：

```
In [12]: add_numbers??
Signature: add_numbers(a, b)
Source:
def add_numbers(a, b):
    """
    Add two numbers together

    Returns
    -----
    the_sum : type of arguments
    """
    return a + b
File:      <ipython-input-9-6a548a216e27>
```

Type:           function

?还有一个用途，就是像Unix或Windows命令行一样搜索IPython的命名空间。字符与通配符结合可以匹配所有的名字。例如，我们可以获得所有包含load的顶级NumPy命名空间：

```
In [13]: np.*load*?
np.__loader__
np.load
np.loads
np.loadtxt
np.pkgload
```

## %run命令

你可以用`%run`命令运行所有的Python程序。假设有一个文件`ipython_script_test.py`:

```
def f(x, y, z):  
    return (x + y) / z  
  
a = 5  
b = 6  
c = 7.5  
  
result = f(a, b, c)
```

可以如下运行:

```
In [14]: %run ipython_script_test.py
```

这段脚本运行在空的命名空间（没有`import`和其它定义的变量），因此结果和普通的运行方式`python script.py`相同。文件中所有定义的变量（`import`、函数和全局变量，除非抛出异常），都可以在IPython shell中随后访问:

```
In [15]: c  
Out [15]: 7.5  
  
In [16]: result  
Out[16]: 1.4666666666666666
```

如果一个Python脚本需要命令行参数（在`sys.argv`中查找），可以在文件路径之后传递，就像在命令行上运行

一样。

笔记：如果想让一个脚本访问IPython已经定义过的变量，可以使用`%run -i`。

在Jupyter notebook中，你也可以使用`%load`，它将脚本导入到一个代码格中：

```
>>> %load ipython_script_test.py
```

```
def f(x, y, z):  
    return (x + y) / z  
a = 5  
b = 6  
c = 7.5  
  
result = f(a, b, c)
```

## 中断运行的代码

代码运行时按**Ctrl-C**，无论是`%run`或长时间运行命令，都会导致`KeyboardInterrupt`。这会导致几乎左右Python程序立即停止，除非一些特殊情况。

警告：当Python代码调用了一些编译的扩展模块，按**Ctrl-C**不一定将执行的程序立即停止。在这种情况下，你必须等待，直到控制返回Python解释器，或者在更糟糕的情况下强制终止Python进程。

## 从剪贴板执行程序

如果使用Jupyter notebook，你可以将代码复制粘贴到任意代码格执行。在IPython shell中也可以从剪贴板执行。假设在其它应用中复制了如下代码：

```
x = 5
y = 7
if x > 5:
    x += 1

    y = 8
```

最简单的方法是使用`%paste`和`%cpaste`函数。`%paste`可以直接运行剪贴板中的代码：

```
In [17]: %paste
x = 5
y = 7
if x > 5:
    x += 1

    y = 8
## -- End pasted text --
```

`%cpaste`功能类似，但会给出一条提示：

```
In [18]: %cpaste
Pasting code; enter '--' alone on the line to stop or us
:x = 5
:y = 7
:if x > 5:
:    x += 1
:
:
```



```
:    y = 8
:--
```

使用`%cpaste`，你可以粘贴任意多的代码再运行。你可能想在运行前，先看看代码。如果粘贴了错误的代码，可以用**Ctrl-C**中断。

键盘快捷键

IPython有许多键盘快捷键进行导航提示（类似Emacs文本编辑器或UNIX bash Shell）和交互shell的历史命令。表2-1总结了常见的快捷键。图2-5展示了一部分，如移动光标。

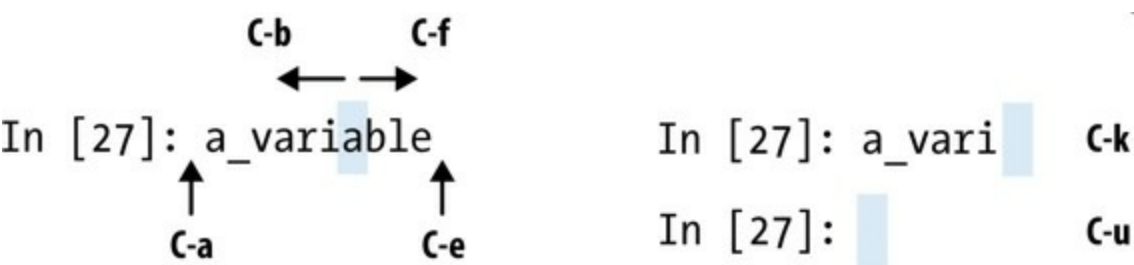


图2-5 IPython shell中一些快捷键的说明

快捷键	说明
Ctrl-P 或 ↑箭头	用当前输入的文本搜索之前的命令
Ctrl-N 或 ↓箭头	用当前输入的文本搜索之后的命令
Ctrl-R	Readline 方式翻转历史搜索（部分匹配）
Ctrl-Shift-V	从剪贴板粘贴文本
Ctrl-C	中断运行的代码
Ctrl-A	将光标移动到一行的开头
Ctrl-E	将光标移动到一行的末尾
Ctrl-K	删除光标到行尾的文本
Ctrl-U	删除当前行的所有文本
Ctrl-F	光标向后移动一个字符
Ctrl-B	光标向前移动一个字符
Ctrl-L	清空屏幕

表2-1 IPython的标准快捷键

Jupyter notebooks有另外一套庞大的快捷键。因为它的快捷键比IPython的变化快，建议你参阅Jupyter notebook的帮助文档。

## 魔术命令

IPython中特殊的命令（Python中没有）被称作“魔术”命令。这些命令可以使普通任务更便捷，更容易控制IPython系统。魔术命令是在指令前添加百分号%前缀。例如，可以用`%timeit`（这个命令后面会详谈）测量任何Python语句，例如矩阵乘法，的执行时间：

```
In [20]: a = np.random.randn(100, 100)

In [20]: %timeit np.dot(a, a)
10000 loops, best of 3: 20.9 µs per loop
```

魔术命令可以被看做IPython中运行的命令行。许多魔术命令有“命令行”选项，可以通过？查看：

```
In [21]: %debug?
Docstring:
::

    %debug [--breakpoint FILE:LINE] [statement [statement ...]]

Activate the interactive debugger.
```

This magic command support two ways of activating debugger. One is to activate debugger before executing code. This can set a break point, to step through the code from the You can use this mode by giving statements to execute and a breakpoint.

The other one is to activate debugger in post-mortem mode. activate this mode simply running %debug without any arguments. If an exception has just occurred, this lets you inspect

frames interactively. Note that this will always work on the last traceback that occurred, so you must call this quickly after an exception that you wish to inspect has fired, because if another occurs, it clobbers the previous one.

If you want IPython to automatically do this on every exception, use the `%pdb` magic for more details.

positional arguments:

statement	Code to run in debugger. You can use <code>%run</code> magic mode.
-----------	--

optional arguments:

<code>--breakpoint &lt;FILE:LINE&gt;, -b &lt;FILE:LINE&gt;</code>	Set break point at LINE in FILE.
---	----------------------------------

魔术函数默认可以不用百分号，只要没有变量和函数名相同。这个特点被称为“自动魔术”，可以用`%automagic`打开或关闭。

一些魔术函数与Python函数很像，它的结果可以赋值给一个变量：

```
In [22]: %pwd
Out[22]: '/home/wesm/code/pydata-book'

In [23]: foo = %pwd

In [24]: foo
Out[24]: '/home/wesm/code/pydata-book'
```

IPython的文档可以在shell中打开，我建议你用`%quickref`或`%magic`学习下所有特殊命令。表2-2列出了

一些可以提高生产率的交互计算和Python开发的IPython指令。

命令	说明
%quickref	显示 IPython 的快速参考。
%magic	显示所有魔术命令的详细文档。
%debug	在出现异常的语句进入调试模式。
%hist	打印命令的输入（可以选择输出）历史。
%pdb	出现异常时自动进入调试。
%paste	执行剪贴板中的代码。
%cpaste	开启特别提示，手动粘贴待执行代码。
%reset	删除所有命名空间中的变量和名字。
%page OBJECT	美化打印对象，分页显示。
%run script.py	运行代码。
%prun statement	用 CProfile 运行代码，并报告分析器输出。
%time statement	报告单条语句的执行时间。
%timeit statement	多次运行一条语句，计算平均执行时间。适合执行时间短的代码。
%who, %who_ls, %whos	显示命名空间中的变量，三者显示的信息级别不同。
%xdel variable	删除一个变量，并清空任何对它的引用。

表2-2 一些常用的IPython魔术命令

## 集成Matplotlib

IPython在分析计算领域能够流行的原因之一是它非常好的集成了数据可视化和其它用户界面库，比如matplotlib。不用担心以前没用过matplotlib，本书后面会详细介绍。`%matplotlib`魔术函数配置了IPython shell和Jupyter notebook中的matplotlib。这点很重要，其它创建的图不会出现（notebook）或获取session的控制，直到结束（shell）。

在IPython shell中，运行`%matplotlib`可以进行设置，可以创建多个绘图窗口，而不会干扰控制台session：

```
In [26]: %matplotlib
Using matplotlib backend: Qt4Agg
```

在Jupyter中，命令有所不同（图2-6）：

```
In [26]: %matplotlib inline
```

```
In [14]: %matplotlib inline
In [15]: import matplotlib.pyplot as plt
          plt.plot(np.random.randn(50).cumsum())
Out[15]: [<matplotlib.lines.Line2D at 0x7f828f0497f0>]
```

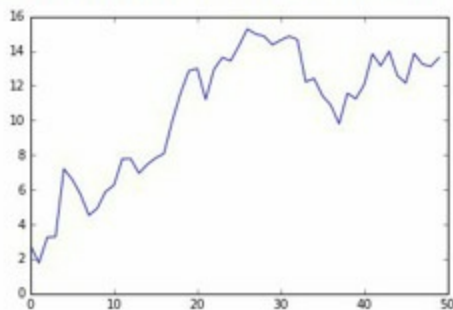


图2-6 Jupyter行内matplotlib作图

## 2.3 Python语法基础

---

在本节中，我将概述基本的Python概念和语言机制。在下一章，我将详细介绍Python的数据结构、函数和其它内建工具。

## 语言的语义

Python的语言设计强调的是可读性、简洁和清晰。有些人称Python为“可执行的伪代码”。



使用缩进，而不是括号

Python使用空白字符（**tab**和空格）来组织代码，而不是像其它语言，比如**R**、**C++**、**JAVA**和**Perl**那样使用括号。看一个排序算法的**for**循环：

```
for x in array:
    if x < pivot:
        less.append(x)
    else:
        greater.append(x)
```

冒号标志着缩进代码块的开始，冒号之后的所有代码的缩进量必须相同，直到代码块结束。不管是否喜欢这种形式，使用空白符是Python程序员开发的一部分，在我看来，这可以让python的代码可读性大大优于其它语言。虽然期初看起来很奇怪，经过一段时间，你就能适应了。

笔记：我强烈建议你使用四个空格作为默认的缩进，可以使用**tab**代替四个空格。许多文本编辑器的设置是使用制表位替代空格。某些人使用**tabs**或不同数目的空格数，常见的是使用两个空格。大多数情况下，四个空格是大多数人采用的方法，因此建议你也这样做。

你应该已经看到，Python的语句不需要用分号结尾。但是，分号却可以用来给同在一行的语句切分：

```
a = 5; b = 6; c = 7
```

**Python**不建议将多条语句放到一行，这会降低代码的可读性。

## 万物皆对象

Python语言的一个重要特性就是它的对象模型的一致性。每个数字、字符串、数据结构、函数、类、模块等等，都是在Python解释器的自有“盒子”内，它被认为是Python对象。每个对象都有类型（例如，字符串或函数）和内部数据。在实际中，这可以让语言非常灵活，因为函数也可以被当做对象使用。

## 注释

任何前面带有井号#的文本都会被Python解释器忽略。这通常被用来添加注释。有时，你会想排除一段代码，但并不删除。简便的方法就是将其注释掉：

```
results = []
for line in file_handle:
    # keep the empty lines for now
    # if len(line) == 0:
    #     continue
    results.append(line.replace('foo', 'bar'))
```

也可以在执行过的代码后面添加注释。一些人习惯在代码之前添加注释，前者这种方法有时也是有用的：

```
print("Reached this line") # Simple status report
```

## 函数和对象方法调用

你可以用圆括号调用函数，传递零个或多个参数，或者将返回值给一个变量：

```
result = f(x, y, z)
g()
```

几乎Python中的每个对象都有附加的函数，称作方法，可以用来访问对象的内容。可以用下面的语句调用：

```
obj.some_method(x, y, z)
```

函数可以使用位置和关键词参数：

```
result = f(a, b, c, d=5, e='foo')
```

后面会有更多介绍。

## 变量和参数传递

当在Python中创建变量（或名字），你就在等号右边创建了一个对这个变量的引用。考虑一个整数列表：

```
In [8]: a = [1, 2, 3]
```

假设将a赋值给一个新变量b：

```
In [9]: b = a
```

在有些方法中，这个赋值会将数据[1, 2, 3]也复制。在Python中，a和b实际上是同一个对象，即原有列表[1, 2, 3]（见图2-7）。你可以在a中添加一个元素，然后检查b：

```
In [10]: a.append(4)
```

```
In [11]: b
```

```
Out[11]: [1, 2, 3, 4]
```

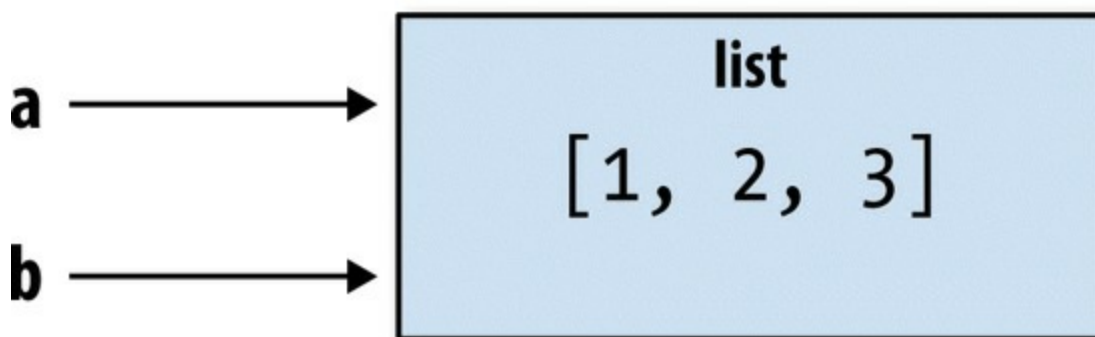


图2-7 对同一对象的双重引用

理解Python的引用的含义，数据是何时、如何、为何复制的，是非常重要的。尤其是当你用Python处理大

的数据集时。

笔记：赋值也被称作绑定，我们是把一个名字绑定给一个对象。变量名有时可能被称为绑定变量。

当你将对象作为参数传递给函数时，新的局域变量创建了对原始对象的引用，而不是复制。如果在函数里绑定一个新对象到一个变量，这个变动不会反映到上一层。因此可以改变可变参数的内容。假设有以下函数：

```
def append_element(some_list, element):  
    some_list.append(element)
```

然后有：

```
In [27]: data = [1, 2, 3]  
  
In [28]: append_element(data, 4)  
  
In [29]: data  
Out[29]: [1, 2, 3, 4]
```

## 动态引用，强类型

与许多编译语言（如**JAVA**和**C++**）对比，**Python**中的对象引用不包含附属的类型。下面的代码是没有问题的：

```
In [12]: a = 5

In [13]: type(a)
Out[13]: int

In [14]: a = 'foo'

In [15]: type(a)
Out[15]: str
```

变量是在特殊命名空间中的对象的名字，类型信息保存在对象自身中。一些人可能会说**Python**不是“类型化语言”。这是不正确的，看下面的例子：

```
In [16]: '5' + 5
-----
TypeError                                Traceback (most recent call last)
<ipython-input-16-f9dbf5f0b234> in <module>()
----> 1 '5' + 5
TypeError: must be str, not int
```

在某些语言中，例如**Visual Basic**，字符串‘5’可能被默认转换（或投射）为整数，因此会产生10。但在其它语言中，例如**JavaScript**，整数5个能被投射成字符串，结果是联结字符串‘55’。在这个方面，**Python**被认



为是强类型化语言，意味着每个对象都有明确的类型（或类），默许转换只会发生在特定的情况下，例如：

```
In [17]: a = 4.5

In [18]: b = 2

# String formatting, to be visited later
In [19]: print('a is {0}, b is {1}'.format(type(a), type(b)))
a is <class 'float'>, b is <class 'int'>

In [20]: a / b
Out[20]: 2.25
```

知道对象的类型很重要，最好能让函数可以处理多种类型的输入。你可以用`isinstance`函数检查对象是某个类型的实例：

```
In [21]: a = 5

In [22]: isinstance(a, int)
Out[22]: True
```

`isinstance`可以用类型元组，检查对象的类型是否在元组中：

```
In [23]: a = 5; b = 4.5

In [24]: isinstance(a, (int, float))
Out[24]: True

In [25]: isinstance(b, (int, float))
Out[25]: True
```

## 属性和方法

Python的对象通常都有属性（其它存储在对象内部的Python对象）和方法（对象的附属函数可以访问对象的内部数据）。可以用`obj.attribute_name`访问属性和方法：

```
In [1]: a = 'foo'
```

```
In [2]: a.<Press Tab>
```

a.capitalize	a.format	a.isupper	a.rindex
a.center	a.index	a.join	a.rjust
a.count	a.isalnum	a.ljust	a.rpartition
a.decode	a.isalpha	a.lower	a.rsplit
a.encode	a.isdigit	a.lstrip	a.rstrip
a.endswith	a.islower	a.partition	a.split
a.expandtabs	a.isspace	a.replace	a.splitlines
a.find	a.istitle	a.rfind	a.startswith

也可以用`getattr`函数，通过名字访问属性和方法：

```
In [27]: getattr(a, 'split')  
Out[27]: <function str.split>
```

在其它语言中，访问对象的名字通常称作“反射”。本书不会大量使用`getattr`函数和相关的`hasattr`和`setattr`函数，使用这些函数可以高效编写原生的、可重复使用的代码。

## 鸭子类型

经常地，你可能不关心对象的类型，只关心对象是否有某些方法或用途。这通常被称为“鸭子类型”，来自“走起来像鸭子、叫起来像鸭子，那么它就是鸭子”的说法。例如，你可以通过验证一个对象是否遵循迭代协议，判断它是可迭代的。对于许多对象，这意味着它有一个 `__iter__` 魔术方法，其它更好的判断方法是使用 `iter` 函数：

```
def isiterable(obj):
    try:
        iter(obj)
        return True
    except TypeError: # not iterable
        return False
```

这个函数会返回字符串以及大多数Python集合类型为 `True`：

```
In [29]: isiterable('a string')
Out[29]: True

In [30]: isiterable([1, 2, 3])
Out[30]: True

In [31]: isiterable(5)
Out[31]: False
```

我总是用这个功能编写可以接受多种输入类型的函数。常见的例子是编写一个函数可以接受任意类型的序列（`list`、`tuple`、`ndarray`）或是迭代器。你可先检验对象

是否是列表（或是Numpy数组），如果不是的话，将其转变成列表：

```
if not isinstance(x, list) and iterable(x):  
    x = list(x)
```

## 引入

在Python中，模块就是一个有`.py`扩展名、包含Python代码的文件。假设有以下模块：

```
# some_module.py
PI = 3.14159

def f(x):
    return x + 2

def g(a, b):
    return a + b
```

如果想从同目录下的另一个文件访问`some_module.py`中定义的变量和函数，可以：

```
import some_module
result = some_module.f(5)
pi = some_module.PI
```

或者：

```
from some_module import f, g, PI
result = g(5, PI)
```

使用`as`关键词，你可以给引入起不同的变量名：

```
import some_module as sm
from some_module import PI as pi, g as gf

r1 = sm.f(pi)
r2 = gf(6, pi)
```

## 二元运算符和比较运算符

大多数二元数学运算和比较都不难想到：

```
In [32]: 5 - 7  
Out[32]: -2
```

```
In [33]: 12 + 21.5  
Out[33]: 33.5
```

```
In [34]: 5 <= 2  
Out[34]: False
```

表2-3列出了所有的二元运算符。

要判断两个引用是否指向同一个对象，可以使用`is`方法。`is not`可以判断两个对象是不同的：

```
In [35]: a = [1, 2, 3]
```

```
In [36]: b = a
```

```
In [37]: c = list(a)
```

```
In [38]: a is b  
Out[38]: True
```

```
In [39]: a is not c  
Out[39]: True
```

因为`list`总是创建一个新的Python列表（即复制），我们可以断定`c`是不同于`a`的。使用`is`比较与`==`运算符不同，如下：

```
In [40]: a == c
Out[40]: True
```

`is`和`is not`常用来判断一个变量是否为`None`，因为只有一个`None`的实例：

```
In [41]: a = None

In [42]: a is None
Out[42]: True
```

运算	说明
$a + b$	a 加 b
$a - b$	a 减 b
$a * b$	a 乘以 b
$a / b$	a 除以 b
$a // b$	a 除以 b，结果只取整数部分
$a ** b$	a 的 b 次幂
$a \& b$	a 或 b 都为 True，则为 True；对于整数，取逐位 AND
$a   b$	a 或 b 有一个为 True，则为 True；对于整数，取逐位 OR
$a \wedge b$	对于布尔，a 或 b 有一个为 True，则为 True，二者都为 True，为 False；对于整数，取逐位 EXCLUSIVE-OR
$a == b$	a 等于 b，则为 True
$a != b$	a 不等于 b，则为 True
$a < b, a \leq b$	a 小于（或小于等于）b，则为 True
$a > b, a \geq b$	a 大于（或大于等于）b，则为 True
$a \text{ is } b$	a 和 b 引用同一个 Python 对象，则为 True
$a \text{ is not } b$	a 和 b 引用不同的 Python 对象，则为 True

表2-3 二元运算符

## 可变与不可变对象

Python中的大多数对象，比如列表、字典、NumPy数组，和用户定义的类型（类），都是可变的。意味着这些对象或包含的值可以被修改：

```
In [43]: a_list = ['foo', 2, [4, 5]]
```

```
In [44]: a_list[2] = (3, 4)
```

```
In [45]: a_list
```

```
Out[45]: ['foo', 2, (3, 4)]
```

其它的，例如字符串和元组，是不可变的：

```
In [46]: a_tuple = (3, 5, (4, 5))
```

```
In [47]: a_tuple[1] = 'four'
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-47-b7966a9ae0f1> in <module>()  
----> 1 a_tuple[1] = 'four'  
TypeError: 'tuple' object does not support item assignment
```

记住，可以修改一个对象并不意味着就要修改它。这被称为副作用。例如，当写一个函数，任何副作用都要在文档或注释中写明。如果可能的话，我推荐避免副作用，采用不可变的方式，即使要用到可变对象。



## 标量类型

Python的标准库中有一些内建的类型，用以处理数值数据、字符串、布尔值，和日期时间。这些单值类型被称为标量类型，本书中称其为标量。表2-4列出了主要的标量。日期和时间处理会另外讨论，因为它们是标准库的`datetime`模块提供的。

类型	说明
None	Python 的空值（只存在一个 None 对象的实例）
str	字符串类型，存有 Unicode（UTF-8 编码）字符串
bytes	原生 ASCII 字节（或 Unicode 编码为字节）
float	双精度（64 位）浮点数（注意没有 double 类型）
bool	True 或 False 值
int	任意精度整数

表2-4 Python的标量

## 数值类型

Python的主要数值类型是`int`和`float`。`int`可以存储任意大的数：

```
In [48]: ival = 17239871
```

```
In [49]: ival ** 6
```

```
Out[49]: 26254519291092456596965462913230729701102721
```

浮点数使用Python的`float`类型。每个数都是双精度（64位）的值。也可以用科学计数法表示：

```
In [50]: fval = 7.243
```

```
In [51]: fval2 = 6.78e-5
```

不能得到整数的除法会得到浮点数：

```
In [52]: 3 / 2
```

```
Out[52]: 1.5
```

要获得C-风格的整除（去掉小数部分），可以使用底除运算符`//`：

```
In [53]: 3 // 2
```

```
Out[53]: 1
```

## 字符串

许多人是因为Python强大而灵活的字符串处理而使用Python的。你可以用单引号或双引号来写字符串：

```
a = 'one way of writing a string'
b = "another way"
```

对于有换行符的字符串，可以使用三引号，`'''`或`"""`都行：

```
c = """
This is a longer string that
spans multiple lines
"""
```

字符串`c`实际包含四行文本，`"""`后面和`lines`后面的换行符。可以用`count`方法计算`c`中的新的行：

```
In [55]: c.count('\n')
Out[55]: 3
```

Python的字符串是不可变的，不能修改字符串：

```
In [56]: a = 'this is a string'

In [57]: a[10] = 'f'
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-57-5ca625d1e504> in <module>()
----> 1 a[10] = 'f'
TypeError: 'str' object does not support item assignment

In [58]: b = a.replace('string', 'longer string')
```

# 有关此电子书试读版的说明

本人可以帮助你找到你要的高清PDF电子书，计算机类，文学，艺术，设计，医学，理学，经济，金融等等。

质量都很清晰，为方便读者阅读观看，每本100%。都带可跳转的书签索引和目录。

一般情况下，出版半年左右就会有PDF 极个别的书出PDF 时间要长一些

如看到试读版信息.说明已经有完整版，需求完整版即可联系我。

请添加 **QQ 312082710** 和**微信**

312082710或扫描微信二维码添加



PDF 代找说明：

本人已经帮助了上万人找到了他们需要的PDF ,其实网上有很多PDF,

大家如果在网上不到的话，可以联系我**QQ 312082710**或**微信312082710**

大部分我都可以找到，而且每本100%带书签索引目录。

因PDF 电子书都有版权，请不要随意传播，如果您有经济购买能力，请尽量购买正版。

提供各种书籍的高清PDF 电子版代找服务，如果你找不到自己想要的书的PDF 电子版，我们可以帮您找到，如有需要，请联系QQ 2028969416微信2028969416 备用 QQ 202896416  
若以上联系方式失效，您可通过以下电子邮件获取有效联系方式。邮箱：

[312082710@qq.com](mailto:312082710@qq.com)

若您没有QQ 通讯工具，请点击下面链接与客服取得联系。&

<https://item.taobao.com/item.htm?id=565681036651>

声明：本人只提供代找服务，每本100%索引|书签和目录，因寻找和后期制作pdf 电子书有一定难度

仅收取代找费用。如因PDF 产生的版权纠纷，与本人无关，我们仅仅只是帮助你寻找到你要的PDF 而已。