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”，重点却是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，使用Python和Ruby进行网站建设工作非常流行。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原因，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成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）对新想法做研究、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，因此许多人也愿意对此做一些取舍。但是，在那些延迟要求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，能更加便捷地完成重塑、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，或处理来源不同的索引不同的数据，所造成的错误。
- 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，它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以外的语言。

对我个人而言，我的大部分Python工作都要用到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 2016, 11:41:13)
[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、Ubantu、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）为类似以下的内容：

```
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；
- 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，我建议你去参加一个。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存储；

- 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

  创建交互式和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。报告本书错误的最好方法是O'Reilly的errata页面，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。