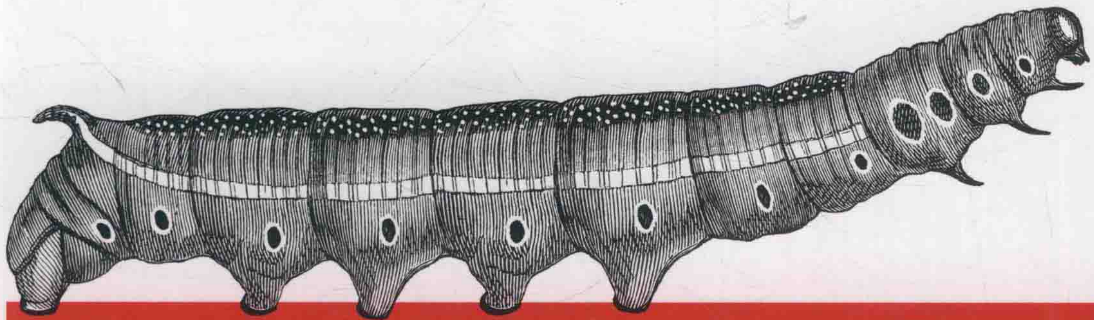
O'REILLY®



Foundations for Analytics with Python

Python数据分析基础（影印版）

東南大學出版社

Clinton W. Brownley 著

Python数据分析基础（影印版）

Foundations for Analytics with Python

Clinton W. Brownley 著

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

O'Reilly Media, Inc.授权东南大学出版社出版

南京 东南大学出版社

图书在版编目(CIP)数据

Python 数据分析基础:英文/(美)克林顿·W.布朗利(Clinton W. Brownley)著. —影印本. —南京:东南大学出版社,2017.2(2017.7重印)

书名原文:Foundations for Analytics with Python
ISBN 978-7-5641-7000-4

Ⅰ.①P… Ⅱ.①克… Ⅲ.①软件工具-程序设计-英文 Ⅳ.①TP311.561

中国版本图书馆 CIP 数据核字(2017)第 008143 号
图字:10-2015-254 号

Python 数据分析基础(影印版)

出版发行:东南大学出版社
地　　址:南京四牌楼 2 号　　邮编:210096
出 版 人:江建中
网　　址:http://www.seupress.com
电子邮件:press@seupress.com
印　　刷:江苏凤凰数码印务有限公司
开　　本:787 毫米×980 毫米　16 开本
印　　张:22
字　　数:431 千字
版　　次:2017 年 2 月第 1 版
印　　次:2017 年 7 月第 2 次印刷
书　　号:ISBN 978-7-5641-7000-4
定　　价:79.00 元

本社图书若有印装质量问题,请直接与营销部联系。电话(传真):025-83791830

For Aisha and Amaya,

"Education is the kindling of a flame,
not the filling of a vessel." —Socrates

May you always enjoy stoking the fire.

Preface

This book is intended for readers who deal with data in spreadsheets on a regular basis, but who have never written a line of code. The opening chapters will get you set up with the Python environment, and teach you how to get the computer to look at data and take simple actions with it. Soon, you'll learn to do things with data in spreadsheets (CSV files) and databases.

At first this will feel like a step backward, especially if you're a power user of Excel. Painstakingly telling Python how to loop through every cell in a column when you used to select and paste feels slow and frustrating (especially when you have to go back three times to find a typo). But as you become more proficient, you'll start to see where Python really shines, especially in automating tasks that you currently do over and over.

This book is written so that you can work through it from beginning to end and feel confident that you can write code that works and does what you expect at the end. It's probably a good idea to type out the code at first, so that you get accustomed to things like tabs and closing your parentheses and quotes, but all the code is available online (*https://github.com/cbrownley/foundations-for-analytics-with-python*) and you may wind up referring to those links to copy and paste as you do your own work in the future. That's fine! Knowing when to cut and paste is part of being an efficient programmer. Reading the book as you go through the examples will teach you why and how the code samples work.

Good luck on your journey to becoming a programmer!

Why Read This Book? Why Learn These Skills?

If you deal with data on a regular basis, then there are a lot of reasons for you to be excited about learning how to program. One benefit is that you can scale your data processing and analysis tasks beyond what would be feasible or practical to do manually. Perhaps you've already come across the problem of needing to process large files that contain so much data that it's impossible or impractical to open them. Even if you can open the files, processing them manually is time consuming and error prone, because any modifications you make to the data take a long time to update—and with so much data, it's easy to miss a row or column that you intended to change. Or perhaps you've come across the problem of needing to process a large number of files—so many files that it's impossible or impractical to process them manually. In some cases, you need to use data from dozens, hundreds, or even thousands of files. As the number of files increases, it becomes increasingly difficult to handle them manually. In both of these situations, writing a Python script to process the files solves your problem because Python scripts can process large files and lots of files quickly and efficiently.

Another benefit of learning to program is that you can automate repetitive data manipulation and analysis processes. In many cases, the operations we carry out on data are repetitive and time consuming. For example, a common data management process involves receiving data from a customer or supplier, extracting the data you want to retain, possibly transforming or reformatting the data, and then saving the data in a database or other data repository (this is the process known to data scientists as ETL—extract, transform, load). Similarly, a typical data analysis process involves acquiring the data you want to analyze, preparing the data for analysis, analyzing the data, and reporting the results. In both of these situations, once the process is established, it's possible to write Python code to carry out the operations. By creating a Python script to carry out the operations, you reduce a time-consuming, repetitive process down to the running of a script and free up your time to work on other impactful tasks.

On top of that, carrying out data processing and analysis operations in a Python script instead of manually reduces the chance of errors. When you process data manually, it's always possible to make a copy/paste error or a typo. There are lots of reasons why this might happen—you might be working so quickly that you miss the mistake, or you might be distracted or tired. Furthermore, the chance of errors increases when you're processing large files or lots of files, or when you're carrying out repetitive actions. Conversely, a Python script doesn't get distracted or tired. Once you debug your script and confirm that it processes the data the way you want it to, it will carry out the operations consistently and tirelessly.

Finally, learning to program is fun and empowering. Once you're familiar with the basic syntax, it's fun to try to figure out which pieces of syntax you need and how to fit them together to accomplish your overall data analysis goal. When it comes to code and syntax, there are lots of examples online that show you how to use specific pieces of syntax to carry out particular tasks. Online examples give you something to work with, but then you need to use your creativity and problem-solving skills to figure out how you need to modify the code you found online to suit your needs. The whole process of searching for the right code and figuring out how to make it work for you can be a lot of fun. Moreover, learning to program is incredibly empowering. For example, consider the situations I mentioned before, involving large files or lots of files. When you can't program, these situations are either incredibly time consuming or simply infeasible. Once you can program, you can tackle both situations relatively quickly and easily with Python scripts. Being able to carry out data processing and analysis tasks that were once laborious or impossible provides a tremendous rush of positive energy, so much so that you'll be looking for more opportunities to tackle challenging data processing tasks with Python.

Who Is This Book For?

This book is written for people who deal with data on a regular basis and have little to no programming experience. The examples in this book cover common data sources and formats, including text files, comma-separated values (CSV) files, Excel files, and databases. In some cases, these files contain so much data or there are so many files that it's impractical or impossible to open them or deal with them manually. In other cases, the process used to extract and use the data in the files is time consuming and error prone. In these situations, without the ability to program, you have to spend a lot of your time searching for the data you need, opening and closing files, and copying and pasting data.

Because you may never have run a script before, we'll start from the very beginning, exploring how to write code in a text file to create a Python script. We'll then review how to run our Python scripts in a Command Prompt window (for Windows users) and a Terminal window (for macOS users). (If you've done a bit of programming, you can skim Chapter 1 and move right into the data analysis parts in Chapter 2.)

Another way I've set out to make this book very user-friendly for new programmers is that instead of presenting code snippets that you'd need to figure out how to combine to carry out useful work, the examples in this book contain all of the Python code you need to accomplish a specific task. You might find that you're coming back to this book as a reference later on, and having all the code at hand will be really helpful then. Finally, following the adage "a picture is worth a thousand words," this book uses screenshots of the input files, Python scripts, Command Prompt and Terminal windows, and output files so you can literally see how to create the inputs, code, commands, and outputs.

I'm going to go into detail to show how things work, as well as giving you some tools that you can put to use. This approach will help you build a solid basis for understanding "what's going on under the hood"—there will be times when you Google a solution to your problem and find useful code, and having done the exercises in this book, you'll have a good understanding of how code you find online works. This means you'll know both how to apply it in your situation and how to fix it if it breaks. As you'll build working code through these chapters, you may find that you'll use this book as a reference, or a "cookbook," with recipes to accomplish specific tasks. But remember, this is a "learn to cook" book; you'll be developing skills that you can generalize and combine to do all sorts of tasks.

Why Windows?

The majority of examples in this book show how to create and run Python scripts on Microsoft Windows. The focus on Windows is fairly straightforward: I want this book to help as many people as possible, and according to available estimates, the vast majority of desktop and laptop computers—especially in business analytics—run a Windows operating system. For instance, according to Net Applications, as of December 2014, Microsoft Windows occupies approximately 90% of the desktop and laptop operating system market. Because I want this book to appeal to desktop and laptop users, and the vast majority of these computers have a Windows operating system, I concentrate on showing how to create and run Python scripts on Windows.

Despite the book's emphasis on Windows, I also provide examples of how to create and run Python scripts on macOS, where appropriate. Almost everything that happens within Python itself will happen the same way no matter what kind of machine you're running it on. But where there are differences between operating systems, I'll give specific instructions for each. For instance, the first example in Chapter 1 illustrates how to create and run a Python script on both Microsoft Windows and macOS. Similarly, the first examples in Chapters 2 and 3 also illustrate how to create and run the scripts on both Windows and macOS. In addition, Chapter 8 covers both operating systems by showing how to create scheduled tasks on Windows and cron jobs on macOS. If you are a Mac user, use the first example in each chapter as a template for how to create a Python script, make it executable, and run the script. Then repeat the steps to create and run all of the remaining examples in each chapter.

Why Python?

There are many reasons to choose Python if your aim is to learn how to program in a language that will enable you to scale and automate data processing and analysis tasks. One notable feature of Python is its use of whitespace and indentation to denote line endings and blocks of code, in contrast to many other languages, which use extra characters like semicolons and curly braces for these purposes. This makes it relatively easy to see at first glance how a Python program is put together.

The extra characters found in other languages are troublesome for people who are new to programming, for at least two reasons. First, they make the learning curve longer and steeper. When you're learning to program, you're essentially learning a new language, and these extra characters are one more aspect of the language you need to learn before you can use the language effectively. Second, they can make the code difficult to read. Because in these other languages semicolons and curly braces denote blocks of code, people don't always use indentation to guide your eye around the blocks of code. Without indentation, these blocks of code can look like a jumbled mess.

Python sidesteps these difficulties by using whitespace and indentation, not semicolons and curly braces, to denote blocks of code. As you look through Python code, your eyes focus on the actual lines of code rather than the delimiters between blocks of code, because everything around the code is whitespace. Python code requires blocks of code to be indented, and indentation makes it easy to see where one block of code ends and another begins. Moreover, the Python community emphasizes code readability, so there is a culture of writing code that is comparatively easy to read and understand. All of these features make the learning curve shorter and shallower, which means you can get up and running and processing data with Python relatively quickly compared to many alternatives.

Another notable feature of Python that makes it ideal for data processing and analysis is the number of standard and add-in modules and functions that facilitate common data processing and analysis operations. Built-ins and standard library modules and functions come standard with Python, so when you download and install Python you immediately have access to these built-in modules and functions. You can read about all of the built-ins and standard modules in the Python Standard Library (PSL) (*https://docs.python.org/3/library*). Add-ins are other Python modules that you download and install separately so you can use the additional functions they provide. You can peruse many of the add-ins in the Python Package Index (PyPI) (*https://pypi.python.org/pypi*).

Some of the modules in the standard library provide functions for reading different file types (e.g., text, comma-separated values, JSON, HTML, XML, etc.); manipulating numbers, strings, and dates; using regular expression pattern matching; parsing comma-separated values files; calculating basic statistics; and writing data to different output file types and to disk. There are too many useful add-in modules to cover them all, but a few that we'll use or discuss in this book include the following:

`xlrd` *and* `xlwt`
: Provide functions for parsing and writing Microsoft Excel workbooks.

`mysqlclient/MySQL-python/MySQLdb`
: Provide functions for connecting to MySQL databases and executing queries on tables in databases.

`pandas`
: Provides functions for reading different file types; managing, filtering, and transforming data; aggregating data and calculating basic statistics; and creating different types of plots.

`statsmodels`
: Provides functions for estimating statistical models, including linear regression models, generalized linear models, and classification models.

`scikit-learn`
: Provides functions for estimating statistical machine learning models, including regression, classification, and clustering, as well as carrying out data preprocessing, dimensionality reduction, and cross-validation.

If you're new to programming and you're looking for a programming language that will enable you to automate and scale your data processing and analysis tasks, then Python is an ideal choice. Python's emphasis on whitespace and indentation means the code is easier to read and understand, which makes the learning curve less steep than for other languages. And Python's built-in and add-in packages facilitate many common data manipulation and analysis operations, which makes it easy to complete all of your data processing and analysis tasks in one place.

Base Python and pandas

Pandas is an add-in module for Python that provides numerous functions for reading/writing, combining, transforming, and managing data. It also has functions for calculating statistics and creating graphs and plots. All of these functions simplify and reduce the amount of code you need to write to accomplish your data processing tasks. The module has become very popular among data analysts and others who use Python because it offers a lot of helpful functions, it's fast and powerful, and it simplifies and reduces the code you have to write to get your job done. Given its power and popularity, I want to introduce you to pandas in this book. To do so, I present pandas versions of the scripts in Chapters 2 and 3, I illustrate how to create graphs and plots with pandas in Chapter 6, and I demonstrate how to calculate various statistics with pandas in Chapter 7. I also encourage you to pick up a copy of Wes McKinney's book, *Python for Data Analysis* (O'Reilly) (*http://shop.oreilly.com/product/0636920023784.do*).[1]

At the same time, if you're new to programming, I also want you to learn basic programming skills. Once you learn these skills, you'll develop generally applicable problem-solving skills that will enable you to break down complex problems into smaller components, solve the smaller components, and then combine the components together to solve the larger problem. You'll also develop intuition for which data structures and algorithms you can use to solve different problems efficiently and

1 Wes McKinney is the original developer of the pandas module and his book is an excellent introduction to pandas, NumPy, and IPython (additional add-in modules you'll want to learn about as you broaden your knowledge of Python for data analysis).

effectively. In addition, there will be times when an add-in module like pandas doesn't have the functionality you need or isn't working the way you need it to. In these situations, if you don't have basic programming skills, you're stuck. Conversely, if you do have these skills you can create the functionality you need and solve the problem on your own. Being able to solve a programming problem on your own is exhilarating and incredibly empowering.

Because this book is for people who are new to programming, the focus is on basic, generally applicable programming skills. For instance, Chapter 1 introduces fundamental concepts such as data types, data containers, control flow, functions, `if-else` logic, and reading and writing files. In addition, Chapters 2 and 3 present two versions of each script: a base Python version and a pandas version. In each case, I present and discuss the base Python version first so you learn how to implement a solution on your own with general code, and then I present the pandas version. My hope is that you will develop fundamental programming skills from the base Python versions so you can use the pandas versions with a firm understanding of the concepts and operations pandas simplifies for you.

Anaconda Python

When it comes to Python, there are a variety of applications in which you can write your code. For example, if you download Python from Python.org, then your installation of Python comes with a graphical user interface (GUI) text editor called Idle. Alternatively, you can download IPython Notebook and write your code in an interactive, web-based environment. If you're working on macOS or you've installed Cygwin on Windows, then you can write your code in a Terminal window using one of the built-in text editors like Nano, Vim, or Emacs. If you're already familiar with one of these applications, then feel free to use it to follow along with the examples in this book.

However, in this section, I'm going to provide instructions for downloading and installing the free Anaconda Python distribution from Continuum Analytics because it has some advantages over the alternatives for a beginning programmer—and for the advanced programmer, too! The major advantage is that it comes with hundreds of the most popular add-in Python packages preinstalled so you don't have to experience the inevitable headaches of trying to install them and their dependencies on your own. For example, all of the add-in packages we use in this book come preinstalled in Anaconda Python.

Another advantage is that it comes with an integrated development environment, or IDE, called Spyder. Spyder provides a convenient interface for writing, executing, and debugging your code, as well as installing packages and launching IPython Notebooks. It includes nice features such as links to online documentation, syntax coloring, keyboard shortcuts, and error warnings.

Another nice aspect of Anaconda Python is that it's cross-platform—there are versions for Linux, Mac, and Windows. So if you learn to use it on Windows but need to transition to a Mac at a later point, you'll still be able to use the same familiar interface.

One aspect of Anaconda Python to keep in mind while you're becoming familiar with Python and all of the available add-in packages is the syntax you use to install add-in packages. In Anaconda Python, you use the conda `install` command. For example, to install the add-in package `argparse`, you would type `conda install argparse`. This syntax is different from the usual `pip install` command you'd use if you'd installed Python from Python.org (if you'd installed Python from Python.org, then you'd install the `argparse` package with `python -m pip install argparse`). Anaconda Python also allows you to use the `pip install` syntax, so you can actually use either method, but it's helpful to be aware of this slight difference while you're learning to install add-in packages.

Installing Anaconda Python (Windows or Mac)

To install Anaconda Python, follow these steps:

1. Go to *http://continuum.io/downloads* (the website automatically detects your operating system—i.e., Windows or Mac).
2. Select "Windows 64-bit Python 3.5 Graphical Installer" (if you're using Windows) or "Mac OS X 64-bit Python 3.5 Graphical Installer" (if you're on a Mac).
3. Double-click the downloaded *.exe* (for Windows) or *.pkg* (for Mac) file.
4. Follow the installer's instructions.

Text Editors

Although we'll be using Anaconda Python and Spyder in this book, it's helpful to be familiar with some text editors that provide features for writing Python code. For instance, if you didn't want to use Anaconda Python, you could simply install Python from Python.org and then use a text editor like Notepad (for Windows) or TextEdit (for macOS). To use TextEdit to write Python scripts, you need to open TextEdit and change the radio button under TextEdit→Preferences from "Rich text" to "Plain text" so new files open as plain text. Then you'll be able to save the files with a *.py* extension.

An advantage of writing your code in a text editor is that there should already be one on your computer, so you don't have to worry about downloading and installing additional software. And as most desktops and laptops ship with a text editor, if you ever have to work on a different computer (e.g., one that doesn't have Spyder or a Terminal window), you'll be able to get up and running quickly with whatever text editor is available on the computer.

While writing your Python code in a text editor such as Notepad or TextEdit is completely acceptable and effective, there are other free text editors you can download that offer additional features, including code highlighting, adjustable tab sizes, and multi-line indenting and dedenting. These features (particularly code highlighting and multi-line indenting and dedenting) are incredibly helpful, especially while you're learning to write and debug your code.

Here is a noncomprehensive list of some free text editors that offer these features:

- Notepad++ (*http://notepad-plus-plus.org*) (Windows)
- Sublime Text (*http://www.sublimetext.com*) (Windows and Mac)
- jEdit (*http://www.jedit.org*) (Windows and Mac)
- TextWrangler (*http://www.barebones.com/products/textwrangler*) (Mac)

Again, I'll be using Anaconda Python and Spyder in this book, but feel free to use a text editor to follow along with the examples. If you download one of these editors, be sure to search online for the keystroke combination to use to indent and dedent multiple lines at a time. It'll make your life a lot easier when you start experimenting with and debugging blocks of code.

Download Book Materials

All of the Python scripts, input files, and output files presented in this book are available online at *https://github.com/cbrownley/foundations-for-analytics-with-python*.

It's possible to download the whole folder of materials to your computer, but it's probably simpler to just click on the filename and copy/paste the script into your text editor. (GitHub is a website for sharing and collaborating on code—it's very good at keeping track of different versions of a project and managing the collaboration process, but it has a pretty steep learning curve. When you're ready to start sharing your code and suggesting changes to other people's code, you might take a look at Chad Thompson's *Learning Git* (*http://shop.oreilly.com/product/110000769.do*) (Infinite Skills).)

Overview of Chapters

Chapter 1, Python Basics

We'll begin by exploring how to create and run a Python script. This chapter focuses on basic Python syntax and the elements of Python that you need to know for later chapters in the book. For example, we'll discuss basic data types such as numbers and strings and how you can manipulate them. We'll also cover the main data containers (i.e., lists, tuples, and dictionaries) and how you use them to store and manipulate your data, as well as how to deal with dates, as dates often appear in business analysis. This chapter also discusses programming concepts such as control flow, functions, and exceptions, as these are important elements for including business logic in your code and gracefully handling errors. Finally, the chapter explains how to get your computer to read a text file, read multiple text files, and write to a CSV-formatted output file. These are important techniques for accessing input data and retaining specific output data that I expand on in later chapters in the book.

Chapter 2, Comma-Separated Values (CSV) Files

This chapter covers how to read and write CSV files. The chapter starts with an example of parsing a CSV input file "by hand," without Python's built-in `csv` module. It transitions to an illustration of potential problems with this method of parsing and then presents an example of how to avoid these potential problems by parsing a CSV file with Python's `csv` module. Next, the chapter discusses how to use three different types of conditional logic to filter for specific rows from the input file and write them to a CSV output file. Then the chapter presents two different ways to filter for specific columns and write them to the output file. After covering how to read and parse a single CSV input file, we'll move on to discussing how to read and process multiple CSV files. The examples in this section include presenting summary information about each of the input files, concatenating data from the input files, and calculating basic statistics for each of the input files. The chapter ends with a couple of examples of less common procedures, including selecting a set of contiguous rows and adding a header row to the dataset.

Chapter 3, Excel Files

Next, we'll cover how to read Excel workbooks with a downloadable, add-in module called `xlrd`. This chapter starts with an example of introspecting an Excel workbook (i.e., presenting how many worksheets the workbook contains, the names of the worksheets, and the number of rows and columns in each of the worksheets). Because Excel stores dates as numbers, the next section illustrates how to use a set of functions to format dates so they appear as dates instead of as numbers. Next, the chapter discusses how to use three different types of conditional logic to filter for specific rows from a single worksheet and write them to a CSV output file. Then the chapter presents two different ways to filter for specific columns and write them to the output file. After covering how to read and parse a single worksheet, the chapter moves on to discuss how to read and process all worksheets in a workbook and a subset of worksheets in a workbook. The examples in these sections show how to filter for specific rows and columns in the worksheets. After discussing how to read and parse any number of worksheets in a single workbook, the chapter moves on to review how to read and process multiple workbooks. The examples in this section include presenting summary information about each of the workbooks, concatenating data from the workbooks, and calculating basic statistics for each of the workbooks. The chapter ends with a couple of examples of less common procedures, including selecting a set of contiguous rows and adding a header row to the dataset.

Chapter 4, Databases
: Here, we'll cover how to carry out basic database operations in Python. The chapter starts with examples that use Python's built-in `sqlite3` module so that you don't have to install any additional software. The examples illustrate how to carry out some of the most common database operations, including creating a database and table, loading data in a CSV input file into a database table, updating records in a table using a CSV input file, and querying a table. When you use the `sqlite3` module, the database connection details are slightly different from the ones you would use to connect to other database systems like MySQL, PostgreSQL, and Oracle. To show this difference, the second half of the chapter demonstrates how to interact with a MySQL database system. If you don't already have MySQL on your computer, the first step is to download and install MySQL. From there, the examples mirror the `sqlite3` examples, including creating a database and table, loading data in a CSV input file into a database table, updating records in a table using a CSV input file, querying a table, and writing query results to a CSV output file. Together, the examples in the two halves of this chapter provide a solid foundation for carrying out common database operations in Python.

Chapter 5, Applications
: This chapter contains three examples that demonstrate how to combine techniques presented in earlier chapters to tackle three different problems that are representative of some common data processing and analysis tasks. The first application covers how to find specific records in a large collection of Excel and CSV files. As you can imagine, it's a lot more efficient and fun to have a computer search for the records you need than it is to search for them yourself. Opening, searching in, and closing dozens of files isn't fun, and the task becomes more and more challenging as the number of files increases. Because the problem involves searching through CSV and Excel files, this example utilizes a lot of the material covered in Chapters 2 and 3.

  The second application covers how to group or "bin" data into unique categories and calculate statistics for each of the categories. The specific example is parsing a CSV file of customer service package purchases that shows when customers paid for particular service packages (i.e., Bronze, Silver, or Gold), organizing the data into unique customer names and packages, and adding up the amount of time each customer spent in each package. The example uses two building blocks, creating a function and storing data in a dictionary, which are introduced in Chapter 1 but aren't used in Chapters 2, 3, and 4. It also introduces another new technique: keeping track of the previous row you processed and the row you're currently processing, in order to calculate a statistic based on values in the two rows. These two techniques—grouping or binning data with a dictionary and keeping track of the current row and the previous row—are very powerful capabilities that enable you to handle many common analysis tasks that involve events over time.

The third application covers how to parse a text file, group or bin data into categories, and calculate statistics for the categories. The specific example is parsing a MySQL error log file, organizing the data into unique dates and error messages, and counting the number of times each error message appeared on each date. The example reviews how to parse a text file, a technique that briefly appears in Chapter 1. The example also shows how to store information separately in both a list and a dictionary in order to create the header row and the data rows for the output file. This is a reminder that you can parse text files with basic string operations and another good example of how to use a nested dictionary to group or bin data into unique categories.

Chapter 6, Figures and Plots

In this chapter, you'll learn how to create common statistical graphs and plots in Python with four plotting libraries: `matplotlib`, `pandas`, `ggplot`, and `seaborn`. The chapter begins with `matplotlib` because it's a long-standing package with lots of documentation (in fact, `pandas` and `seaborn` are built on top of `matplotlib`). The `matplotlib` section illustrates how to create histograms and bar, line, scatter, and box plots. The `pandas` section discusses some of the ways `pandas` simplifies the syntax you need to create these plots and illustrates how to create them with `pandas`. The `ggplot` section notes the library's historical relationship with R and the Grammar of Graphics and illustrates how to use `ggplot` to build some common statistical plots. Finally, the `seaborn` section discusses how to create standard statistical plots as well as plots that would be more cumbersome to code in `matplotlib`.

Chapter 7, Descriptive Statistics and Modeling

Here, we'll look at how to produce standard summary statistics and estimate regression and classification models with the `pandas` and `statsmodels` packages. `pandas` has functions for calculating measures of central tendency (e.g., mean, median, and mode), as well as for calculating dispersion (e.g., variance and standard deviation). It also has functions for grouping data, which makes it easy to calculate these statistics for different groups of data. The `statsmodels` package has functions for estimating many types of regression and classification models. The chapter illustrates how to build multivariate linear regression and logistic classification models based on data in pandas DataFrames and then use the models to predict output values for new input data.

Chapter 8, Scheduling Scripts to Run Automatically
: This chapter covers how to schedule your scripts to run automatically on a routine basis on both Windows and macOS. Until this chapter, we ran the scripts manually on the command line. Running a script manually on the command line is convenient when you're debugging the script or running it on an ad hoc basis. However, it can be a nuisance if your script needs to run on a routine basis (e.g., daily, weekly, monthly, or quarterly), or if you need to run lots of scripts on a routine basis. On Windows, you create scheduled tasks to run scripts automatically on a routine basis. On macOS, you create cron jobs, which perform the same actions. This chapter includes several screenshots to show you how to create and run scheduled tasks and cron jobs. By scheduling your scripts to run on a routine basis, you don't ever forget to run a script and you can scale beyond what's possible when you're running scripts manually on the command line.

Chapter 9, Where to Go from Here
: The final chapter covers some additional built-in and add-in Python modules and functions that are important for data processing and analysis tasks, as well as some additional data structures that will enable you to efficiently handle a variety of complex programming problems you may run into as you move beyond the topics covered in this book. Built-ins are bundled into the Python installation, so they are immediately available to you when you install Python. The built-in modules discussed in this chapter include `collections`, `random`, `statistics`, `itertools`, and `operator`. The built-in functions include `enumerate`, `filter`, `reduce`, and `zip`. Add-in modules don't come with the Python installation, so you have to download and install them separately. The add-in modules discussed in this chapter include NumPy, SciPy, and Scikit-Learn. We also take a look at some additional data structures that can help you store, process, or analyze your data more quickly and efficiently, such as stacks, queues, trees, and graphs.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords. Also used for module and package names,