Mastering Python for Finance

Built initially for scientific computing, Python quickly found its place in finance. Its flexibility and robustness can be easily incorporated into applications for mathematical studies, research, and software development.

With this book, you will learn about all the tools you need to successfully perform research studies and modeling, improve your trading strategies, and effectively manage risks. You will explore the various tools and techniques used in solving complex problems commonly faced in finance. You will learn how to price financial instruments such as stocks, options, interest rate derivatives, and futures using computational methods. Also, you will learn how you can perform data analytics on market indexes and use NoSQL to store tick data.

Who this book is written for

If you are an undergraduate or graduate student, a beginner to algorithmic development and research, or a software developer in the financial industry who is interested in using Python for quantitative methods in finance, this is the book for you. It would be helpful to have a bit of familiarity with basic Python usage, but no prior experience is required.

What you will learn from this book

- Perform interactive computing with IPython Notebook
- Solve linear equations of financial models and perform ordinary least squares regression
- Explore nonlinear modeling and solutions for optimum points using root-finding algorithms and solvers
- Discover different types of numerical procedures used in pricing options
- Model fixed-income instruments with bonds and interest rates
- Manage big data with NoSQL and perform analytics with Hadoop
- Build a high-frequency algorithmic trading platform with Python
- Create an event-driven backtesting tool and measure your strategies

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Python for Financial Applications'
- A synopsis of the book’s content
- More information on Mastering Python for Finance

About the Author

James Ma Weiming works with high-frequency, low-latency trading systems, writing his own programs and tools, most of which are open sourced. He is currently supporting veteran traders in the, trading pits of the Chicago Board of Trade devising strategies to game the market. He graduated from the Stuart School of Business at Illinois Institute of Technology with a master of science degree in finance.

He started his career in Singapore after receiving his bachelor's degree in computer engineering from Nanyang Technological University and diploma in information technology from Nanyang Polytechnic. During his career, he has worked in treasury operations handling foreign exchange and fixed income products. He also developed mobile applications with a company operating a funds and investments distribution platform.
Mastering Python for Finance

Python is widely practiced in various sectors of finance, such as banking, investment management, insurance, and even real estate, for building tools that help in financial modeling, risk management, and trading. Even big financial corporations embrace Python to build their infrastructure for position management, pricing, risk management, and trading systems.

Throughout this book, theories from academic financial studies will be introduced, accompanied by their mathematical concepts to help you understand their uses in practical situations. You will see how Python is applied to classical pricing models, linearity, and nonlinearity of finance, numerical procedures, and interest rate models, that form the foundations of complex financial models. You will learn about the root-finding methods and finite difference pricing for developing an implied volatility curve with options.

With the advent of advanced computing technologies, methods for the storing and handling of massive amounts of data have to be considered. Hadoop is a popular tool in big data. You will be introduced to the inner workings of Hadoop and its integration with Python to derive analytical insights on financial data. You will also understand how Python supports the use of NoSQL for storing non-structured data.

Many brokerage firms are beginning to offer APIs to customers to trade using their own customized trading software. Using Python, you will learn how to connect to a broker API, retrieve market data, generate trading signals, and send orders to the exchange. The implementation of the mean-reverting and trend-following trading strategies will be covered. Risk management, position tracking, and backtesting techniques will be discussed to help you manage the performance of your trading strategies.

The use of Microsoft Excel is pervasive in the financial industry, from bond trading to back-office operations. You will be taught how to create numerical pricing Component Object Model (COM) servers in Python that will enable your spreadsheets to compute and update model values on the fly.

What This Book Covers

Chapter 1, Python for Financial Applications, explores the aspects of Python in judging its suitability as a programming language in finance. The IPython Notebook is introduced as a beneficial tool to visualize data and to perform scientific computing.

Chapter 2, The Importance of Linearity in Finance, uses Python to solve systems of linear equations, perform integer programming, and apply matrix algebra to linear optimization of portfolio allocation.

Chapter 4, *Numerical Procedures*, explores trees, lattices, and finite differencing schemes for valuation of options.

Chapter 5, *Interest Rates and Derivatives*, discusses the bootstrapping process of the yield curve and covers some short rate models for pricing the interest rate derivatives with Python.

Chapter 6, *Interactive Financial Analytics with Python and VSTOXX*, discusses the volatility indexes. We will perform analytics on EURO STOXX 50 Index and VSTOXX data, and replicate the main index using options prices of the sub-indexes.

Chapter 7, *Big Data with Python*, walks you through the uses of Hadoop for big data and covers how to use Python to perform MapReduce operations. Data storage with NoSQL will also be covered.

Chapter 8, *Algorithmic Trading*, discusses a step-by-step approach to develop a mean-reverting and trend-following live trading infrastructure using Python and the API of a broker. Value-at-risk (VaR) for risk management will also be covered.

Chapter 9, *Backtesting*, discusses how to design and implement an event-driven backtesting system and helps you visualize the performance of our simulated trading strategy.

Chapter 10, *Excel with Python*, discusses how to build a Component Object Model (COM) server and client interface to communicate with Excel and to perform numerical pricing on the call and put options on the fly.
1
Python for Financial Applications

In this introductory chapter, we will explore the aspects of Python in order to judge its suitability as a programming language in finance. Notably, Python is widely practiced in various financial sectors, such as banking, investment management, insurance, and even in real estate for building tools that help in financial modeling, risk management, and trading. To help you get the most from the multitude of features that Python has to offer, we will introduce the IPython Notebook as a beneficial tool to help you visualize data and to perform scientific computing for presentation to end users.

In this chapter, we will cover the following topics:

- Benefits of Python over other programming languages for financial studies
- Features of Python for financial applications
- Implementing object-oriented design and functional design in Python
- Overview of IPython
- Getting IPython and IPython Notebook started
- Creating and saving notebook documents
- Various formats to export a notebook document
- Notebook document user interface
- Inserting Markdown language into a notebook document
- Performing calculations in Python in a notebook document
- Creating plots in a notebook document
- Various ways of displaying mathematical equations in a notebook document
- Inserting images and videos into a notebook document
- Working with HTML and pandas DataFrame in a notebook document
Is Python for me?

Today's financial programmers have a diverse choice of programming languages in implementing robust software solutions, ranging from C, Java, R, and MATLAB. However, each programming language was designed differently to accomplish specific tasks. Their inner workings, behavior, syntax, and performance affect the results of every user differently.

In this book, we will focus exclusively on the use of Python for analytical and quantitative finance. Originally intended for scientific computations, the Python programming language saw an increasingly widespread use in financial operations. In particular, pandas, a software library written for the Python programming language, was open sourced by an employee of AQR Capital Management to offer high-performance financial data management and quantitative analysis.

Even big financial corporations embrace Python to architect their infrastructure. Bank of America's Quartz platform uses Python for position management, pricing, and risk management. JP Morgan's Athena platform, a cross-market risk management and trading system, uses Python for flexibility in combination with C++ and Java.

The application of Python in finance is vast, and in this book, we will cover the fundamental topics in creating financial applications, such as portfolio optimization, numerical pricing, interactive analytics, big data with Hadoop, and more.

Here are some considerations on why you might use Python for your next financial application.

Free and open source

Python is free in terms of license. Documentation is widely available, and many Python online community groups are available, where one can turn in times of doubt. Because it is free and open source, anyone can easily view or modify the algorithms in order to adapt to customized solutions.

Being accessible to the public opens a whole new level of opportunities. Anyone can contribute existing enhancements or create new modules. For advanced users, interoperability between different programming languages is supported. A Python interpreter may be embedded in C and C++ programs. Likewise, with the appropriate libraries, Python may be integrated with other languages not limited to Fortran, Lisp, PHP, Lua, and more.

Python is available on all major operating systems, such as Windows, Unix, OS/2, Mac, among others.
High-level, powerful, and flexible

Python as a general-purpose, high-level programming language allows the user to focus on problem solving and leave low-level mechanical constructs such as memory management out of the picture.

The expressiveness of the Python programming language syntax helps quantitative developers in implementing prototypes quickly.

Python allows the use of object-oriented, procedural, as well as functional programming styles. Because of this flexibility, it is especially useful in implementing complex mathematical models containing multiple changeable parameters.

A wealth of standard libraries

By now, you should be familiar with the NumPy, SciPy, matplotlib, statsmodels, and pandas modules, as indispensable tools in quantitative analysis and data management.

Other libraries extend the functionalities of Python. For example, one may turn Python into a data visualization tool with the gnuplot package in visualizing mathematical functions and data interactively. With Tk-based GUI tools such as Tkinter, it is possible to turn Python scripts into GUI programs.

A widely popular shell for Python is IPython, which provides interactive computing and high-performance tools for parallel and distributed computing. With IPython Notebook, the rich text web interface of IPython, you can share code, text, mathematical expressions, plots, and other rich media with your target audience. IPython was originally intended for scientists to work with Python and data.

Objected-oriented versus functional programming

If you are working as a programmer in the finance industry, chances are that your program will be built for handling thousands or millions of dollars' worth in transactions. It is crucial that your programs are absolutely free of errors. More often than not, bugs arise due to unforeseen circumstances. As financial software systems and models become larger and more complex, practicing good software design is crucial. While writing the Python code, you may want to consider the object-oriented approach or the functional approach to structure your code for better readability.
The object-oriented approach

As the demand for clarity, speed, and flexibility in your program increases, it is important to keep your code readable, manageable, and lean. One popular technical approach to building software systems is by applying the object-oriented paradigm. Consider the following example of displaying a greeting message as a class:

```python
class Greeting(object):
    def __init__(self, my_greeting):
        self.my_greeting = my_greeting

    def say_hello(self, name):
        print "%s %s" % (self.my_greeting, name)
```

We created a class called `Greeting` that is capable of accepting an input argument in its constructor. For this example, we will define our greeting as "Hello". The `say_hello` function is invoked with an input name and prints our greeting messages as follows:

```python
>>> greeting = Greeting("Hello")
>>> greeting.say_hello("World")
>>> greeting.say_hello("Dog")
>>> greeting.say_hello("Cat")
Hello World
Hello Dog
Hello Cat
```

The functional approach

We can achieve the same `Greeting` functionality using the functional approach. Functional programming is a programming paradigm, where computer programs are structured and styled in such a way that they can be evaluated as mathematical functions. These pseudo mathematical functions avoid changing state data, while increasing reusability and brevity.

In Python, a function object can be assigned to a variable and, like any other variables, can be passed into functions as an argument as well as return its value.
Let's take a look at the following code that gives us the same output:

```python
from functools import partial

def greeting(my_greeting, name):
    print "%s %s" % (my_greeting, name)
```

Here, we defined a function named `greeting` that takes in two arguments. Using the `partial` function of the `functools` module, we treated the function, `greeting`, as an input variable, along with our variable greeting message as `Hello`.

```python
>>> say_hello_to = partial(greeting, "Hello")
>>> say_hello_to("World")
>>> say_hello_to("Dog")
>>> say_hello_to("Cat")
```

We assigned the `say_hello_to` variable as its return value, and reused this variable in printing our greetings with three different names by executing it as a function that accepts input arguments.

**Which approach should I use?**

There is no clear answer to this question. We have just demonstrated that Python supports both the object-oriented approach and the functional approach. We can see that in certain circumstances the functional approach in software development is brevity to a large extent. Using the `say_hello_to` function provides better readability over `greeting.say_hello()`. It boils down to the programmer's decision as to what works best in making the code more readable and having ease of maintenance during the software life cycle while collaborating with fellow developers.

As a general rule of thumb, in large and complex software systems representing objects as classes helps in code management between team members. By working with classes, the scope of work can be more easily defined, and system requirements can be easily scaled using object-oriented design. When working with financial mathematical models, using functional programing helps to keep the code working in the same fashion as its accompanying mathematical concepts.

**Which Python version should I use?**

The code examples in this book have been tested in Python 2.7 but are optimized to run on Python 3. Many of the third-party Python modules mentioned in this book require at least Python 2.7, and some do not have support for Python 3 as yet. To achieve the best compatibility, it is recommended that you install Python 2.7 on your workstation.
If you have not installed Python on your workstation, you can find out more about Python from the official source at https://www.python.org. However, in order to build financial applications from the examples, you are required to use a number of additional third-party Python modules such as NumPy, SciPy, and pandas. It is recommended that you obtain an all-in-one installer to ease the installation procedures. The following are some popular installers that include hundreds of supported packages:

- Anaconda by Continuum Analytics at https://store.continuum.io/cshop/anaconda
- Canopy by Enthought at https://store.enthought.com

Introducing IPython
IPython is an interactive shell with high-performance tools used for parallel and distributed computing. With the IPython Notebook, you can share code, text, mathematical expressions, plots, and other rich media with your target audience.

In this section, we will learn how to get started and run a simple IPython Notebook.

Getting IPython
Depending on how you have installed Python on your machine, IPython might have been included in your Python environment. Please consult the IPython official documentation for the various installation methods most comfortable for you. The official page is available at http://ipython.org.

IPython can be downloaded from https://github.com/ipython. To install IPython, unpack the packages to a folder. From the terminal, navigate to the top-level source directory and run the following command:

$ python setup.py install

Using pip
The pip tool is a great way to install Python packages automatically. Think of it as a package manager for Python. For example, to install IPython without having to download all the source files, just run the following command in the terminal:

$ pip install ipython

To get pip to work in the terminal, it has to be installed as a Python module. Instructions for downloading and installing pip can be found at https://pypi.python.org/pypi/pip.
The IPython Notebook

The IPython Notebook is the web-based interactive computing interface of IPython used for the whole computation process of developing, documenting, and executing the code. This section covers some of the common features in IPython Notebook that you may consider using for building financial applications.

Here is a screenshot of the IPython Notebook in Windows OS:

![IPython Notebook Screenshot]

Notebooks allow in-browser editing and executing of the code with the outputs attached to the code that generated them. It has the capability of displaying rich media, including images, videos, and HTML components.

Its in-browser editor allows the Markdown language that can provide rich text and commentary for the code.

Mathematical notations can be included with the use of LaTeX, rendered natively by MathJax. With the ability to import Python modules, publication-quality figures can be included inline and rendered using the matplotlib library.
Notebook documents

Notebook documents are saved with the .ipynb extension. Each document contains everything related to an interactive session, stored internally in the JSON format. Since JSON files are represented in plain text, this allows notebooks to be version controlled and easily sharable.

Notebooks can be exported to a range of static formats, including HTML, LaTeX, PDF, and slideshows.

Notebooks can also be available as static web pages and served to the public via URLs using nbviewer (IPython Notebook Viewer) without users having to install Python. Conversion is handled using the nbconvert service.

Running the IPython Notebook

Once you have IPython successfully installed in the terminal, type the following command:

```
$ ipython notebook
```

This will start the IPython Notebook server service, which runs in the terminal. By default, the service will automatically open your default web browser and navigate to the landing page. To access the notebook program manually, enter the http://localhost:8888 URL address in your web browser.

![By default, a notebook runs in port 8888. To infer the correct notebook address, check the log output from the terminal.]

The landing page of the notebook web application is called the dashboard, which shows all notebooks currently available in the notebook directory. By default, this directory is the one from which the notebook server was started.

Creating a new notebook

Click on New Notebook from the dashboard to create a new notebook. You can navigate to the File | New menu option from within an active notebook:
Here, you will be presented with the notebook name, a menu bar, a toolbar, and an empty code cell.

The menu bar presents different options that may be used to manipulate the way the notebook functions.

The toolbar provides shortcuts to frequently used notebook operations in the form of icons.

**Notebook cells**

Each logical section in a notebook is known as a cell. A cell is a multi-line text input field that accepts plain text. A single notebook document contains at least one cell and can have multiple cells.

To execute the contents of a cell, from the menu bar, go to Cell | Run, or click on the Play button from the toolbar, or use the keyboard shortcut Shift + Enter.
Each cell can be formatted as a **Code**, **Markdown**, **Raw NBConvert**, or heading cell:

**Code cell**

By default, each cell starts off as a code cell, which executes the Python code when you click on the **Run** button. Cells with a rounded rectangular box and gray background accept text inputs. The outputs of the executed box are displayed in the white space immediately below the text input.

**Markdown cell**

Markdown cells accept the Markdown language that provides a simple way to format plain text into rich text. It allows arbitrary HTML code for formatting.

Mathematical notations can be displayed with standard LaTeX and AMS-LaTeX (the `amsmath` package). Surround the LaTeX equation with $ to display inline mathematics, and $$ to display equations in a separate block. When executed, MathJax will render Latex equations with high-quality typography.

**Raw NBConvert cell**

Raw cells provide the ability to write the output directly and are not evaluated by the notebook.

**Heading cells**

Cells may be formatted as a heading cell, from level 1 (top level) to level 6 (paragraph). These are useful for the conceptual structure of your document or to construct a table of contents.
Simple exercises with IPython Notebook

Let's get started by creating a new notebook and populating it with some content. We will insert the various types of objects to demonstrate the various tasks.

Creating a notebook with heading and Markdown cells

We will begin by creating a new notebook by performing the following steps:

1. Click on New Notebook from the dashboard to create a new notebook. If from within an active notebook, navigate to the File | New menu option.
2. In the input field of the first cell, enter a page title for this notebook. In this example, type in Welcome to Hello World.
3. From the options toolbar menu, go to Cells | Cell Type and select Heading 1. This will format the text we have entered as the page title. The changes, however, will not be immediate at this time.
4. From the options toolbar menu, go to Insert | Insert Cell Below. This will create another input cell below our current cell.
5. In this example, we will insert the following piece of text that contains the Markdown code:

```
Text Examples

This is an example of an *italic* text.

This is an example of a **bold*** text.

This is an example of a list item:
- Item #1
- Item #2
- Item #3

---

#heading 1
##heading 2
###heading 3
####heading 4
#####heading 5
######heading 6
```
6. From the toolbar, select **Markdown** instead of **Code**.

7. To run your code, go to **Cell | Run All**. This option will run all the Python commands and format your cells as required.

When the current cell is executed successfully, the notebook will focus on the next cell below, ready for your next input. If no cell is available, one will be automatically created and will receive the input focus.

This will give us the following output:

```
Welcome to Hello World

Text Examples

This is an example of an italic text.
This is an example of a **bold** text.
This is an example of a list item:
- Item #1
- Item #2
- Item #3

heading 1
heading 2
heading 3
heading 4
heading 5
heading 6
```

**Saving notebooks**

Go to **File** and click on **Save** and **Checkpoint**. Our notebook will be saved as an `.ipynb` file.
Mathematical operations in cells
Let’s perform a simple mathematical calculation in the notebook; let’s add the numbers 3 and 5 and assign the result to the `answer` variable by typing in the code cell:

```python
answer = 3 + 5
```

From the options menu, go to Insert | Insert Cell Below to add a new code cell at the bottom. We want to output the result by typing in the following code in the next cell:

```python
print answer
```

Next, go to Cell | Run All. Our answer is printed right below the current cell.

Displaying graphs
The `matplotlib` module provides a MATLAB-like plotting framework in Python. With the `matplotlib.pyplot` function, charts can be plotted and rendered as graphic images for display in a web browser.

Let’s demonstrate a simple plotting functionality of the IPython Notebook. In a new cell, paste the following code:

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

x = np.linspace(0, 2*math.pi)
plt.plot(x, np.sin(x), label=r'$\sin(x)$')
plt.plot(x, np.cos(x), 'ro', label=r'$\cos(x)$')
plt.title(r'Two plots in a graph')
plt.legend()
```

The first three lines of the code contain the required import statements. Note that the NumPy, math, and matplotlib packages are required for the code to work in the IPython Notebook.
In the next statement, the variable \( x \) represents our \( x \) axis values from 0 through 7 in real numbers. The following statement plots the \( \sin \) function for every value of \( x \). The next \texttt{plot} command plots the \( \cos \) function for every value of \( x \) as a dotted line. The last two lines of the code print the title and legend respectively.

Running this cell gives us the following output:

![Graph showing \( \sin(x) \) and \( \cos(x) \)]

---

**Inserting equations**

What is TeX and LaTeX? **TeX** is the industry standard document markup language for math markup commands. **LaTeX** is a variant of TeX that separates the document structure from the content.

Mathematical equations can be displayed using LaTeX in the Markdown parser. The IPython Notebook uses MathJax to render LaTeX surrounded with \$$\$$ inside Markdown.

For this example, we will display a standard normal cumulative distribution function by typing in the following command in the cell:

\[
N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{z^2}{2}} \, dz
\]

Select **Markdown** from the toolbar and run the current cell. This will transform the current cell into its respective equation output:

\[
N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{z^2}{2}} \, dz
\]
Besides using the MathJax typesetting, another way of displaying the same equation is using the `math` function of the IPython display module, as follows:

```python
from IPython.display import Math
Math(r'N(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{z^2}{2}}\, dz')
```

The preceding code will display the same equation, as shown in the following screenshot:

Notice that, since this cell is run as a normal code cell, the output equation is displayed immediately below the code cell.

We can also display equations inline with text. For example, we will use the following code with a single `$` wrapping around the LaTeX expression:

```
This expression $\sqrt{3x-1}+(1+x)^2$ is an example of a TeX inline equation
```

Run this cell as the Markdown cell. This will transform the current cell into the following:

```
This expression $\sqrt{3x - 1} + (1 + x)^2$ is an example of a TeX inline equation.
```

### Displaying images

To work with images, such as JPEG and PNG, use the `Image` class of the IPython display module. Run the following code to display a sample image:

```python
from IPython.display import Image
Image(url='http://python.org/images/python-logo.gif')
```
On running the code cell, it will display the following output:

```
In [15]: from IPython.display import Image
      | Image(url='http://python.org/images/python-logo.gif')
Out[15]:
```

![Python logo](http://python.org/images/python-logo.gif)

### Inserting YouTube videos

The `lib.display` module of the IPython package contains a `YouTubeVideo` function, where you can embed videos hosted externally on YouTube into your notebook. For example, run the following code:

```
from IPython.lib.display import YouTubeVideo

# An introduction to Python by Google.
YouTubeVideo('tKTzoB2Vjuk')
```

The video will be displayed below the code, as shown in the following screenshot:
Working with HTML

Notebook allows HTML representations to be displayed. One common use of HTML is the ability to display data with tables. The following code outputs a table with two columns and three rows, including a header row:

```python
from IPython.display import HTML

table = """""""
<tr>
<th>Header 1</th>
<th>Header 2</th>
</tr>
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
"""

HTML(table)
```

The `HTML` function will render HTML tags as a string in its input argument. We can see the final output as follows:

<table>
<thead>
<tr>
<th></th>
<th>Header 1</th>
<th>Header 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>row 1</td>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2</td>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</tbody>
</table>

The pandas DataFrame object as an HTML table

In a notebook, pandas allow DataFrame objects to be represented as HTML tables.

In this example, we will retrieve the stock market data from Yahoo! Finance and store them in a pandas DataFrame object with the help of the `pandas.io.data.web.DataReader` function. Let's use the `AAPL` ticker symbol as the first argument, `yahoo` as its second argument, the start date of the market data as the third argument, and the end date as the last argument:

```python
import pandas.io.data as web
import datetime
```
Python for Financial Applications

```
start = datetime.datetime(2014, 1, 1)
end = datetime.datetime(2014, 12, 31)
df = web.DataReader("AAPL", 'yahoo', start, end)
df.head()
```

With the `df.head()` command, the first five rows of the DataFrame object that contains the market data is displayed as an HTML table in the notebook:

```
In [78]: import pandas.io.data as web
In [79]: start = datetime.datetime(2014, 1, 1)
   ...: end = datetime.datetime(2014, 12, 31)
In [80]: df = web.DataReader("AAPL", 'yahoo', start, end)
   ...: df.head()
Out[80]:
                    Open  High   Low  Close  Volume  Adj Close
Date
2014-01-02     555.68  557.03  552.02  553.13  58671200   77.39
2014-01-03     552.86  553.70  540.43  540.98   98116900   75.69
2014-01-06     537.45  546.80  533.60  543.93  103152700   76.10
2014-01-07     544.32  545.96  537.92  540.04   79302300   75.56
2014-01-08     538.81  545.56  538.69  543.46   64632400   76.04
```

**Notebook for finance**

You are now ready to place your code in a chronological order and present the key financial information, such as plots and data to your audience. Many industry practitioners use the IPython Notebook as their preferred editor for financial model development in helping them to visualize data better.

You are strongly encouraged to explore the powerful features the IPython Notebook has to offer that best suit your modeling needs. A gallery of interesting notebook projects used in scientific computing can be found at https://github.com/ipython/ipython/wiki/A-gallery-of-interesting-IPython-Notebooks.
Summary

In this chapter, we discussed how Python might be suitable for certain areas of finance and also discussed its advantages for our software applications. We also considered the functional programming paradigm and the object-oriented programming paradigm that are supported in Python, and saw how we can achieve brevity in our applications. There is no clear rule as to how one approach may be favored over the other. Ultimately, Python gives programmers the flexibility to structure their code to the best interests of the project at hand.

We were introduced to IPython, the interactive computing shell for Python, and explored its usefulness in scientific computing and rich media presentation. We worked on simple exercises on our web browser with the IPython Notebook, and learned how to create a new notebook document, insert text with the Markdown language, performed simple calculations, plotted graphs, displayed mathematical equations, inserted images and videos, rendered HTML, and learned how to use pandas to fetch the stock market data from Yahoo! Finance as a DataFrame object before presenting its content as an HTML table. This will help us visualize data and perform rich media presentations to our audience.

Python is just one of the many powerful programming languages that can be considered in quantitative finance studies, not limited to Julia, R, MATLAB, and Java. You should be able to present key concepts more effectively in the Python language. These concepts, once mastered, can easily be applied to any language you choose when creating your next financial application.

In the next chapter, we will explore linear models in finance and techniques used in portfolio management.
Where to buy this book

You can buy Mastering Python for Finance from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.