Table of Contents

Preface ............................................................................................................ xiii

Part I. Python and Finance

1. Why Python for Finance ........................................................................... 3
   The Python Programming Language ...................................................... 3
   A Brief History of Python ....................................................................... 5
   The Python Ecosystem ........................................................................... 6
   The Python User Spectrum .................................................................... 7
   The Scientific Stack ............................................................................... 8
   Technology in Finance ......................................................................... 9
   Technology Spending ........................................................................... 9
   Technology as Enabler ........................................................................ 10
   Technology and Talent as Barriers to Entry ......................................... 11
   Ever-Increasing Speeds, Frequencies, and Data Volumes .................... 11
   The Rise of Real-Time Analytics .......................................................... 13
   Python for Finance ............................................................................... 14
   Finance and Python Syntax .................................................................. 14
   Efficiency and Productivity Through Python ....................................... 18
   From Prototyping to Production ......................................................... 23
   Data-Driven and AI-First Finance ....................................................... 24
   Data-Driven Finance ........................................................................... 24
   AI-First Finance .................................................................................. 28
   Conclusion ............................................................................................ 31
   Further Resources ............................................................................... 31
The Basics 90
Multiple Dimensions 94
Metainformation 97
Reshaping and Resizing 98
Boolean Arrays 101
Speed Comparison 103
Structured NumPy Arrays 105
Vectorization of Code 106
Basic Vectorization 107
Memory Layout 110
Conclusion 112
Further Resources 112

5. Data Analysis with pandas. ................................................................. 113
   The DataFrame Class 114
      First Steps with the DataFrame Class 114
      Second Steps with the DataFrame Class 119
   Basic Analytics 123
   Basic Visualization 126
   The Series Class 128
   GroupBy Operations 130
   Complex Selection 132
   Concatenation, Joining, and Merging 135
      Concatenation 136
      Joining 137
      Merging 139
   Performance Aspects 141
   Conclusion 143
   Further Reading 143

6. Object-Oriented Programming. ....................................................... 145
   A Look at Python Objects 149
      int 149
      list 150
      ndarray 151
      DataFrame 152
   Basics of Python Classes 154
   Python Data Model 159
   The Vector Class 163
   Conclusion 164
   Further Resources 164
Part III. Financial Data Science

7. Data Visualization .......................................................... 167
   Static 2D Plotting ...................................................... 168
      One-Dimensional Data Sets ...................................... 169
      Two-Dimensional Data Sets .................................... 176
      Other Plot Styles .................................................... 183
   Static 3D Plotting ...................................................... 191
   Interactive 2D Plotting .............................................. 195
      Basic Plots .......................................................... 195
      Financial Plots ...................................................... 199
   Conclusion .............................................................. 203
   Further Resources .................................................... 204

8. Financial Time Series ................................................... 205
   Financial Data .......................................................... 206
   Data Import ............................................................ 206
   Summary Statistics ................................................... 210
   Changes over Time .................................................... 212
   Resampling .............................................................. 215
   Rolling Statistics ..................................................... 217
      An Overview ......................................................... 218
      A Technical Analysis Example ................................. 220
   Correlation Analysis ................................................ 222
      The Data ............................................................. 222
      Logarithmic Returns .............................................. 224
      OLS Regression ................................................... 226
      Correlation .......................................................... 227
   High-Frequency Data .................................................. 228
   Conclusion .............................................................. 230
   Further Resources .................................................... 230

9. Input/Output Operations ................................................ 231
   Basic I/O with Python ................................................ 232
      Writing Objects to Disk ............................................ 232
      Reading and Writing Text Files ................................ 236
      Working with SQL Databases .................................... 239
      Writing and Reading NumPy Arrays ............................ 242
   I/O with pandas ....................................................... 244
      Working with SQL Databases .................................... 245
      From SQL to pandas ............................................... 247
      Working with CSV Files .......................................... 250
11. **Mathematical Tools.**

Approximation 312
Regression 313
Interpolation 324
Convex Optimization 328
Global Optimization 329
Local Optimization 331
Constrained Optimization 332
Integration 334
Numerical Integration 336
Integration by Simulation 337
Symbolic Computation 337
Basics 338
Equations 340
Integration and Differentiation 340
Differentiation 341
Conclusion 343
Further Resources 343

12. **Stochastics.**

Random Numbers 346
Simulation 352
Random Variables 353
Stochastic Processes 356
Variance Reduction 372
Valuation 375
European Options 376
American Options 380
Risk Measures 383
Value-at-Risk 383
Credit Valuation Adjustments 388
Python Script 392
Conclusion 394
Further Resources 395

13. **Statistics.**

Normality Tests 398
Benchmark Case 399
Real-World Data 409
Portfolio Optimization 415
The Data 416
The Basic Theory 417
### Part IV. Algorithmic Trading

14. **The FXCM Trading Platform** ........................................... 467  
   - Getting Started ......................................................... 469  
   - Retrieving Data .......................................................... 469  
     - Retrieving Tick Data .................................................. 470  
     - Retrieving Candles Data .............................................. 472  
   - Working with the API ................................................... 474  
     - Retrieving Historical Data ........................................... 475  
     - Retrieving Streaming Data ......................................... 477  
   - Placing Orders ........................................................... 478  
   - Account Information ................................................... 480  
   - Conclusion ..................................................................... 480  
   - Further Resources ........................................................ 481

15. **Trading Strategies** ....................................................... 483  
   - Simple Moving Averages .................................................. 484  
   - Data Import .................................................................... 485  
   - Trading Strategy ............................................................ 485  
   - Vectorized Backtesting .................................................... 487  
   - Optimization ................................................................... 489  
   - Random Walk Hypothesis ............................................... 491  
   - Linear OLS Regression ...................................................... 494  
     - The Data ..................................................................... 495  
     - Regression .................................................................... 497  
     - Clustering .................................................................... 499  
     - Frequency Approach ..................................................... 501
16. **Automated Trading** .................................................. 521

   Capital Management ........................................... 522
   The Kelly Criterion in a Binomial Setting ................. 522
   The Kelly Criterion for Stocks and Indices ............... 527
   ML-Based Trading Strategy .................................. 532
   Vectorized Backtesting ....................................... 533
   Optimal Leverage .............................................. 537
   Risk Analysis .................................................... 539
   Persisting the Model Object ................................. 543
   Online Algorithm .............................................. 544
   Infrastructure and Deployment .............................. 546
   Logging and Monitoring ..................................... 547
   Conclusion ...................................................... 550
   Python Scripts .................................................. 550
   Automated Trading Strategy ................................. 550
   Strategy Monitoring ......................................... 553
   Further Resources ............................................. 554

17. **Valuation Framework** .......................................... 557

   Fundamental Theorem of Asset Pricing .................. 558
   A Simple Example ............................................. 558
   The General Results ......................................... 559
   Risk-Neutral Discounting ................................... 560
   Modeling and Handling Dates ............................... 561
   Constant Short Rate ......................................... 563
   Market Environments ....................................... 565
   Conclusion ...................................................... 568
## Further Resources

### 18. Simulation of Financial Models

- Random Number Generation
- Generic Simulation Class
- Geometric Brownian Motion
  - The Simulation Class
  - A Use Case
- Jump Diffusion
  - The Simulation Class
  - A Use Case
- Square-Root Diffusion
  - The Simulation Class
  - A Use Case
- Conclusion
- Further Resources

### 19. Derivatives Valuation

- Generic Valuation Class
- European Exercise
  - The Valuation Class
  - A Use Case
- American Exercise
  - Least-Squares Monte Carlo
  - The Valuation Class
  - A Use Case
- Conclusion
- Further Resources

### 20. Portfolio Valuation

- Derivatives Positions
  - The Class
  - A Use Case
- Derivatives Portfolios
  - The Class
  - A Use Case
- Conclusion
- Further Resources

### 21. Market-Based Valuation

- Options Data
- Model Calibration
These days, Python is undoubtedly one of the major strategic technology platforms in the financial industry. When I started writing the first edition of this book in 2013, I still had many conversations and presentations in which I argued relentlessly for Python’s competitive advantages in finance over other languages and platforms. Toward the end of 2018, this is not a question anymore: financial institutions around the world now simply try to make the best use of Python and its powerful ecosystem of data analysis, visualization, and machine learning packages.

Beyond the realm of finance, Python is also often the language of choice in introductory programming courses, such as in computer science programs. Beyond its readable syntax and multiparadigm approach, a major reason for this is that Python has also become a first class citizen in the areas of artificial intelligence (AI), machine learning (ML), and deep learning (DL). Many of the most popular packages and libraries in these areas are either written directly in Python (such as scikit-learn for ML) or have Python wrappers available (such as TensorFlow for DL).

Finance itself is entering a new era, and two major forces are driving this evolution. The first is the programmatic access to basically all the financial data available—in general, this happens in real time and is what leads to data-driven finance. Decades ago, most trading or investment decisions were driven by what traders and portfolio managers could read in the newspaper or learn through personal conversations. Then came terminals that brought financial data in real time to the traders’ and portfolio managers’ desks via computers and electronic communication. Today, individuals (or teams) can no longer keep up with the vast amounts of financial data generated in even a single minute. Only machines, with their ever-increasing processing speeds and computational power, can keep up with the volume and velocity of financial data. This means, among other things, that most of today’s global equities trading volume is driven by algorithms and computers rather than by human traders.

The second major force is the increasing importance of AI in finance. More and more financial institutions try to capitalize on ML and DL algorithms to improve opera-
tions and their trading and investment performances. At the beginning of 2018, the first dedicated book on “financial machine learning” was published, which underscores this trend. Without a doubt, there are more to come. This leads to what might be called *AI-first finance*, where flexible, parameterizable ML and DL algorithms replace traditional financial theory—theory that might be elegant but no longer very useful in the new era of data-driven, AI-first finance.

Python is the right programming language and ecosystem to tackle the challenges of this era of finance. Although this book covers basic ML algorithms for unsupervised and supervised learning (as well as deep neural networks, for instance), the focus is on Python's data processing and analysis capabilities. To fully account for the importance of AI in finance—now and in the future—another book-length treatment is necessary. However, most of the AI, ML, and DL techniques require such large amounts of data that mastering data-driven finance should come first anyway.

This second edition of *Python for Finance* is more of an upgrade than an update. For example, it adds a complete part (Part IV) about algorithmic trading. This topic has recently become quite important in the financial industry, and is also quite popular with retail traders. It also adds a more introductory part (Part II) where fundamental Python programming and data analysis topics are presented before they are applied in later parts of the book. On the other hand, some chapters from the first edition have been deleted completely. For instance, the chapter on web techniques and packages (such as Flask) was dropped because there are more dedicated and focused books about such topics available today.

For the second edition, I tried to cover even more finance-related topics and to focus on Python techniques that are particularly useful for financial data science, algorithmic trading, and computational finance. As in the first edition, the approach is a practical one, in that implementation and illustration come before theoretical details and I generally focus on the big picture rather than the most arcane parameterization options of a certain class, method, or function.

Having described the basic approach for the second edition, it is worth emphasizing that this book is neither an introduction to Python programming nor to finance in general. A vast number of excellent resources are available for both. This book is located at the intersection of these two exciting fields, and assumes that the reader has some background in programming (not necessarily Python) as well as in finance. Such readers learn how to apply Python and its ecosystem to the financial domain.

The Jupyter Notebooks and codes accompanying this book can be accessed and executed via our Quant Platform. You can sign up for free at [http://py4fi.pqp.io](http://py4fi.pqp.io).

My company (The Python Quants) and myself provide many more resources to master Python for financial data science, artificial intelligence, algorithmic trading, and computational finance. You can start by visiting the following sites:
From all the offerings that we have created over the last few years, I am most proud of our Certificate Program in Python for Algorithmic Trading. It provides over 150 hours of live and recorded instruction, over 1,200 pages of documentation, over 5,000 lines of Python code, and over 50 Jupyter Notebooks. The program is offered multiple times per year and we update and improve it with every cohort. The online program is the first of its kind, in that successful delegates obtain an official university certificate in cooperation with htw saar University of Applied Sciences.

In addition, I recently started The AI Machine, a new project and company to standardize the deployment of automated, algorithmic trading strategies. With this project, we want to implement in a systematic and scalable fashion what we have been teaching over the years in the field, in order to capitalize on the many opportunities in the algorithmic trading field. Thanks to Python—and data-driven and AI-first finance—this project is possible these days even for a smaller team like ours.

I closed the preface for the first edition with the following words:

I am really excited that Python has established itself as an important technology in the financial industry. I am also sure that it will play an even more important role there in the future, in fields like derivatives and risk analytics or high performance computing. My hope is that this book will help professionals, researchers, and students alike make the most of Python when facing the challenges of this fascinating field.

When I wrote these lines in 2014, I couldn't have predicted how important Python would become in finance. In 2018, I am even happier that my expectations and hopes have been so greatly surpassed. Maybe the first edition of the book played a small part in this. In any case, a big thank you is in order to all the relentless open source developers out there, without whom the success story of Python couldn't have been written.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
Indicates new terms, URLs, and email addresses.
Constant width

Used for program listings, as well as within paragraphs to refer to software packages, programming languages, file extensions, filenames, program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

Constant width italic

Shows text that should be replaced with user-supplied values or by values determined by context.

Using Code Examples

Supplemental material (in particular, Jupyter Notebooks and Python scripts/modules) is available for usage and download at http://py4fi.pqp.io.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you’re reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O’Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product’s documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: “Python for Finance, 2nd Edition, by Yves Hilpisch (O’Reilly). Copyright 2019 Yves Hilpisch, 978-1-492-02433-0.”
If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

O’Reilly Safari

Safari (formerly Safari Books Online) is a membership-based training and reference platform for enterprise, government, educators, and individuals.


For more information, please visit http://oreilly.com/safari.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O’Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at http://bit.ly/python-finance-2e.

To comment or ask technical questions about this book, send email to bookquestions@oreilly.com.

For more information about our books, courses, conferences, and news, see our website at http://www.oreilly.com.

Find us on Facebook: http://facebook.com/oreilly
Follow us on Twitter: http://twitter.com/oreillymedia
Watch us on YouTube: http://www.youtube.com/oreillymedia
Acknowledgments

I want to thank all those who helped to make this book a reality—in particular, the team at O’Reilly, who really improved my manuscript in many ways. I would like to thank the tech reviewers, Hugh Brown and Jake VanderPlas. The book benefited from their valuable feedback and their many suggestions. Any remaining errors, of course, are mine.

Michael Schwed, with whom I have been working closely for more than ten years, deserves a special thank you. Over the years, I have benefited in innumerable ways from his work, support, and Python know-how.

I also want to thank Jason Ramchandani and Jorge Santos of Refinitiv (formerly Thomson Reuters) for their continued support not only of my work but also of the open source community in general.

As with the first edition, the second edition of this book has tremendously benefited from the dozens of “Python for finance” talks I have given over the years, as well as the hundreds of hours of “Python for finance” trainings. In many cases the feedback from participants helped to improve my training materials, which often ended up as chapters or sections in this book.

Writing the first edition took me about a year. Overall, writing and upgrading the second edition also took about a year, which was quite a bit longer than I expected. This is mainly because the topic itself keeps me very busy travel- and business-wise, which I am very grateful for.

Writing books requires many hours in solitude and such hours cannot be spent with the family. Therefore, thank you to Sandra, Lilli, Henry, Adolf, Petra, and Heinz for all your understanding and support—not only with regard to writing this book.

I dedicate the second edition of this book, as the first one, to my lovely, strong, and compassionate wife Sandra. She has given new meaning over the years to what family is really about. Thank you.

— Yves

Saarland, November 2018
This part introduces Python for finance. It consists of two chapters:

- **Chapter 1** briefly discusses Python in general and argues in some detail why Python is well suited to addressing the technological challenges in the financial industry as well as in financial data analytics.

- **Chapter 2** is about Python infrastructure; it provides a concise overview of important aspects of managing a Python environment to get you started with interactive financial analytics and financial application development in Python.
CHAPTER 1

Why Python for Finance

Banks are essentially technology firms.  
—Hugo Banziger

The Python Programming Language

Python is a high-level, multipurpose programming language that is used in a wide range of domains and technical fields. On the Python website you find the following executive summary:

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python’s simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

This pretty well describes why Python has evolved into one of the major programming languages today. Nowadays, Python is used by the beginner programmer as well as by the highly skilled expert developer, at schools, in universities, at web companies, in large corporations and financial institutions, as well as in any scientific field.

Among other features, Python is:

Open source

Python and the majority of supporting libraries and tools available are open source and generally come with quite flexible and open licenses.
Interpreted
The reference CPython implementation is an interpreter of the language that translates Python code at runtime to executable byte code.

Multiparadigm
Python supports different programming and implementation paradigms, such as object orientation and imperative, functional, or procedural programming.

Multipurpose
Python can be used for rapid, interactive code development as well as for building large applications; it can be used for low-level systems operations as well as for high-level analytics tasks.

Cross-platform
Python is available for the most important operating systems, such as Windows, Linux, and macOS. It is used to build desktop as well as web applications, and it can be used on the largest clusters and most powerful servers as well as on such small devices as the Raspberry Pi.

Dynamically typed
Types in Python are in general inferred at runtime and not statically declared as in most compiled languages.

Indentation aware
In contrast to the majority of other programming languages, Python uses indentation for marking code blocks instead of parentheses, brackets, or semicolons.

Garbage collecting
Python has automated garbage collection, avoiding the need for the programmer to manage memory.

When it comes to Python syntax and what Python is all about, Python Enhancement Proposal 20—i.e., the so-called “Zen of Python”—provides the major guidelines. It can be accessed from every interactive shell with the command `import this`:

```python
In [1]: import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
```
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

A Brief History of Python

Although Python might still have the appeal of something new to some people, it has been around for quite a long time. In fact, development efforts began in the 1980s by Guido van Rossum from the Netherlands. He is still active in Python development and has been awarded the title of Benevolent Dictator for Life by the Python community. In July 2018, van Rossum stepped down from this position after decades of being an active driver of the Python core development efforts. The following can be considered milestones in the development of Python:

- **Python 0.9.0** released in 1991 (first release)
- **Python 1.0** released in 1994
- **Python 2.0** released in 2000
- **Python 2.6** released in 2008
- **Python 3.0** released in 2008
- **Python 3.1** released in 2009
- **Python 2.7** released in 2010
- **Python 3.2** released in 2011
- **Python 3.3** released in 2012
- **Python 3.4** released in 2014
- **Python 3.5** released in 2015
- **Python 3.6** released in 2016
- **Python 3.7** released in June 2018

It is remarkable, and sometimes confusing to Python newcomers, that there are two major versions available, still being developed and, more importantly, in parallel use since 2008. As of this writing, this will probably keep on for a little while since tons of code available and in production is still Python 2.6/2.7. While the first edition of this book was based on Python 2.7, this second edition uses Python 3.7 throughout.
The Python Ecosystem

A major feature of Python as an ecosystem, compared to just being a programming language, is the availability of a large number of packages and tools. These packages and tools generally have to be imported when needed (e.g., a plotting library) or have to be started as a separate system process (e.g., a Python interactive development environment). Importing means making a package available to the current namespace and the current Python interpreter process.

Python itself already comes with a large set of packages and modules that enhance the basic interpreter in different directions, known as the Python Standard Library. For example, basic mathematical calculations can be done without any importing, while more specialized mathematical functions need to be imported through the math module:

```
In [2]: 100 * 2.5 + 50
Out[2]: 300.0

In [3]: log(1)  
-------------------------------------------------------------------------
NameError                       Traceback (most recent call last)
<ipython-input-3-74f22a2fd43b> in <module>
-----> 1 log(1)  

NameError: name 'log' is not defined
```

Without further imports, an error is raised.

```
In [4]: import math  

In [5]: math.log(1)  
Out[5]: 0.0
```

1 Without further imports, an error is raised.

2 After importing the math module, the calculation can be executed.

While math is a standard Python module available with any Python installation, there are many more packages that can be installed optionally and that can be used in the very same fashion as the standard modules. Such packages are available from different (web) sources. However, it is generally advisable to use a Python package manager that makes sure that all libraries are consistent with each other (see Chapter 2 for more on this topic).

The code examples presented so far use interactive Python environments: IPython and Jupyter, respectively. These are probably the most widely used interactive Python environments at the time of this writing. Although IPython started out as just an enhanced interactive Python shell, it today has many features typically found in integrated development environments (IDEs), such as support for profiling and debug-
ging. Those features missing in IPython are typically provided by advanced text/code editors, like Vim, which can also be integrated with IPython. Therefore, it is not unusual to combine IPython with one's text/code editor of choice to form the basic toolchain for a Python development process.

IPython enhances the standard interactive shell in many ways. Among other things, it provides improved command-line history functions and allows for easy object inspection. For instance, the help text (docstring) for a function is printed by just adding a \? before or after the function name (adding ?? will provide even more information).

IPython originally came in two popular versions: a shell version and a browser-based version (the Notebook). The Notebook variant proved so useful and popular that it evolved into an independent, language-agnostic project now called Jupyter. Given this background, it is no surprise that Jupyter Notebook inherits most of the beneficial features of IPython—and offers much more, for example when it comes to visualization.

Refer to VanderPlas (2016, Chapter 1) for more details on using IPython.

The Python User Spectrum

Python does not only appeal to professional software developers; it is also of use for the casual developer as well as for domain experts and scientific developers.

Professional software developers find in Python all they might require to efficiently build large applications. Almost all programming paradigms are supported; there are powerful development tools available; and any task can, in principle, be addressed with Python. These types of users typically build their own frameworks and classes, also work on the fundamental Python and scientific stack, and strive to make the most of the ecosystem.

Scientific developers or domain experts are generally heavy users of certain packages and frameworks, have built their own applications that they enhance and optimize over time, and tailor the ecosystem to their specific needs. These groups of users also generally engage in longer interactive sessions, rapidly prototyping new code as well as exploring and visualizing their research and/or domain data sets.

Casual programmers like to use Python generally for specific problems they know that Python has its strengths in. For example, visiting the gallery page of matplotlib, copying a certain piece of visualization code provided there, and adjusting the code to their specific needs might be a beneficial use case for members of this group.

There is also another important group of Python users: beginner programmers, i.e., those that are just starting to program. Nowadays, Python has become a very popular language at universities, colleges, and even schools to introduce students to program-
A major reason for this is that its basic syntax is easy to learn and easy to understand, even for the non-developer. In addition, it is helpful that Python supports almost all programming styles.\(^2\)

### The Scientific Stack

There is a certain set of packages that is collectively labeled the *scientific stack*. This stack comprises, among others, the following packages:

**NumPy**

NumPy provides a multidimensional array object to store homogeneous or heterogeneous data; it also provides optimized functions/methods to operate on this array object.

**SciPy**

SciPy is a collection of subpackages and functions implementing important standard functionality often needed in science or finance; for example, one finds functions for cubic splines interpolation as well as for numerical integration.

**matplotlib**

This is the most popular plotting and visualization package for Python, providing both 2D and 3D visualization capabilities.

**pandas**

pandas builds on NumPy and provides richer classes for the management and analysis of time series and tabular data; it is tightly integrated with matplotlib for plotting and PyTables for data storage and retrieval.

**scikit-learn**

scikit-learn is a popular machine learning (ML) package that provides a unified application programming interface (API) for many different ML algorithms, such as for estimation, classification, or clustering.

**PyTables**

PyTables is a popular wrapper for the HDF5 data storage package; it is a package to implement optimized, disk-based I/O operations based on a hierarchical database/file format.

---

1. Python, for example, is a major language used in the Master of Financial Engineering Program at Baruch College of the City University of New York. The first edition of this book is in use at a large number of universities around the world to teach Python for financial analysis and application building.

2. See [http://wiki.python.org/moin/BeginnersGuide](http://wiki.python.org/moin/BeginnersGuide), where you will find links to many valuable resources for both developers and non-developers getting started with Python.
Depending on the specific domain or problem, this stack is enlarged by additional packages, which more often than not have in common that they build on top of one or more of these fundamental packages. However, the least common denominator or basic building blocks in general are the NumPy ndarray class (see Chapter 4) and the pandas DataFrame class (see Chapter 5).

Taking Python as a programming language alone, there are a number of other languages available that can probably keep up with its syntax and elegance. For example, Ruby is a popular language often compared to Python. The language’s website describes Ruby as:

A dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write.

The majority of people using Python would probably also agree with the exact same statement being made about Python itself. However, what distinguishes Python for many users from equally appealing languages like Ruby is the availability of the scientific stack. This makes Python not only a good and elegant language to use, but also one that is capable of replacing domain-specific languages and tool sets like Matlab or R. It also provides by default anything that you would expect, say, as a seasoned web developer or systems administrator. In addition, Python is good at interfacing with domain-specific languages such as R, so that the decision usually is not about either Python or something else—it is rather about which language should be the major one.

**Technology in Finance**

With these “rough ideas” of what Python is all about, it makes sense to step back a bit and to briefly contemplate the role of technology in finance. This will put one in a position to better judge the role Python already plays and, even more importantly, will probably play in the financial industry of the future.

In a sense, technology per se is *nothing special* to financial institutions (as compared, for instance, to biotechnology companies) or to the finance function (as compared to other corporate functions, like logistics). However, in recent years, spurred by innovation and also regulation, banks and other financial institutions like hedge funds have evolved more and more into technology companies instead of being *just* financial intermediaries. Technology has become a major asset for almost any financial institution around the globe, having the potential to lead to competitive advantages as well as disadvantages. Some background information can shed light on the reasons for this development.

**Technology Spending**

Banks and financial institutions together form the industry that spends the most on technology on an annual basis. The following statement therefore shows not only that
technology is important for the financial industry, but that the financial industry is also really important to the technology sector:

FRAMINGHAM, Mass., June 14, 2018 – Worldwide spending on information technology (IT) by financial services firms will be nearly $500 billion in 2021, growing from $440 billion in 2018, according to new data from a series of Financial Services IT Spending Guides from International Data Corporation (IDC).

—IDC

In particular, banks and other financial institutions are engaging in a race to make their business and operating models digital:

Bank spending on new technologies was predicted to amount to 19.9 billion U.S. dollars in 2017 in North America.

The banks develop current systems and work on new technological solutions in order to increase their competitiveness on the global market and to attract clients interested in new online and mobile technologies. It is a big opportunity for global fintech companies which provide new ideas and software solutions for the banking industry.

—Statista

Large multinational banks today generally employ thousands of developers to maintain existing systems and build new ones. Large investment banks with heavy technological requirements often have technology budgets of several billion USD per year.

**Technology as Enabler**

The technological development has also contributed to innovations and efficiency improvements in the financial sector. Typically, projects in this area run under the umbrella of *digitalization*.

The financial services industry has seen drastic technology-led changes over the past few years. Many executives look to their IT departments to improve efficiency and facilitate game-changing innovation—while somehow also lowering costs and continuing to support legacy systems. Meanwhile, FinTech start-ups are encroaching upon established markets, leading with customer-friendly solutions developed from the ground up and unencumbered by legacy systems.

—PwC 19th Annual Global CEO Survey 2016

As a side effect of the increasing efficiency, competitive advantages must often be looked for in ever more complex products or transactions. This in turn inherently increases risks and makes risk management as well as oversight and regulation more and more difficult. The financial crisis of 2007 and 2008 tells the story of potential dangers resulting from such developments. In a similar vein, “algorithms and computers gone wild” represent a potential risk to the financial markets; this materialized dramatically in the so-called *flash crash of May 2010*, where automated selling led to...
large intraday drops in certain stocks and stock indices. Part IV covers topics related to the algorithmic trading of financial instruments.

**Technology and Talent as Barriers to Entry**

On the one hand, technology advances reduce cost over time, *ceteris paribus*. On the other hand, financial institutions continue to invest heavily in technology to both gain market share and defend their current positions. To be active today in certain areas in finance often brings with it the need for large-scale investments in both technology and skilled staff. As an example, consider the derivatives analytics space:

> Aggregated over the total software lifecycle, firms adopting in-house strategies for OTC [derivatives] pricing will require investments between $25 million and $36 million alone to build, maintain, and enhance a complete derivatives library.

—Ding (2010)

Not only is it costly and time-consuming to build a full-fledged derivatives analytics library, but you also need to have enough experts to do so. And these experts have to have the right tools and technologies available to accomplish their tasks. With the development of the Python ecosystem, such efforts have become more efficient and budgets in this regard can be reduced significantly today compared to, say, 10 years ago. Part V covers derivatives analytics and builds a small but powerful and flexible derivatives pricing library with Python and standard Python packages alone.

Another quote about the early days of Long-Term Capital Management (LTCM), formerly one of the most respected quantitative hedge funds—which, however, went bust in the late 1990s—further supports this insight about technology and talent:

> Meriwether spent $20 million on a state-of-the-art computer system and hired a crack team of financial engineers to run the show at LTCM, which set up shop in Greenwich, Connecticut. It was risk management on an industrial level.

—Patterson (2010)

The same computing power that Meriwether had to buy for millions of dollars is today probably available for thousands or can be rented from a cloud provider based on a flexible fee plan. Chapter 2 shows how to set up an infrastructure in the cloud for interactive financial analytics, application development, and deployment with Python. The budgets for such a professional infrastructure start at a few USD per month. On the other hand, trading, pricing, and risk management have become so complex for larger financial institutions that today they need to deploy IT infrastructures with tens of thousands of computing cores.

**Ever-Increasing Speeds, Frequencies, and Data Volumes**

The one dimension of the finance industry that has been influenced most by technological advances is the speed and frequency with which financial transactions are
decided and executed. Lewis (2014) describes so-called \textit{flash trading}—i.e., trading at the highest speeds possible—in vivid detail.

On the one hand, increasing data availability on ever-smaller time scales makes it necessary to react in real time. On the other hand, the increasing speed and frequency of trading makes the data volumes further increase. This leads to processes that reinforce each other and push the average time scale for financial transactions systematically down. This is a trend that had already started a decade ago:

Renaissance's Medallion fund gained an astonishing 80 percent in 2008, capitalizing on the market's extreme volatility with its lightning-fast computers. Jim Simons was the hedge fund world's top earner for the year, pocketing a cool $2.5 billion.

—Patterson (2010)

Thirty years' worth of daily stock price data for a single stock represents roughly 7,500 closing quotes. This kind of data is what most of today's finance theory is based on. For example, modern or mean-variance portfolio theory (MPT), the capital asset pricing model (CAPM), and value-at-risk (VaR) all have their foundations in daily stock price data.

In comparison, on a typical trading day during a single trading hour the stock price of Apple Inc. (AAPL) may be quoted around 15,000 times—roughly twice the number of quotes compared to available end-of-day closing quotes over 30 years (see the example in “Data-Driven and AI-First Finance” on page 24). This brings with it a number of challenges:

\textit{Data processing}  
It does not suffice to consider and process end-of-day quotes for stocks or other financial instruments; “too much” happens during the day, and for some instruments during 24 hours for 7 days a week.

\textit{Analytics speed}  
Decisions often have to be made in milliseconds or even faster, making it necessary to build the respective analytics capabilities and to analyze large amounts of data in real time.

\textit{Theoretical foundations}  
Although traditional finance theories and concepts are far from being perfect, they have been well tested (and sometimes well rejected) over time; for the millisecond and microsecond scales important as of today, consistent financial concepts and theories in the traditional sense that have proven to be somewhat robust over time are still missing.

All these challenges can in general only be addressed by modern technology. Something that might also be a little bit surprising is that the lack of consistent theories often is addressed by technological approaches, in that high-speed algorithms exploit
market microstructure elements (e.g., order flow, bid-ask spreads) rather than relying on some kind of financial reasoning.

The Rise of Real-Time Analytics

There is one discipline that has seen a strong increase in importance in the finance industry: financial and data analytics. This phenomenon has a close relationship to the insight that speeds, frequencies, and data volumes increase at a rapid pace in the industry. In fact, real-time analytics can be considered the industry’s answer to this trend.

Roughly speaking, “financial and data analytics” refers to the discipline of applying software and technology in combination with (possibly advanced) algorithms and methods to gather, process, and analyze data in order to gain insights, to make decisions, or to fulfill regulatory requirements, for instance. Examples might include the estimation of sales impacts induced by a change in the pricing structure for a financial product in the retail branch of a bank, or the large-scale overnight calculation of credit valuation adjustments (CVA) for complex portfolios of derivatives trades of an investment bank.

There are two major challenges that financial institutions face in this context:

**Big data**

Banks and other financial institutions had to deal with massive amounts of data even before the term “big data” was coined; however, the amount of data that has to be processed during single analytics tasks has increased tremendously over time, demanding both increased computing power and ever-larger memory and storage capacities.

**Real-time economy**

In the past, decision makers could rely on structured, regular planning as well as decision and (risk) management processes, whereas they today face the need to take care of these functions in real time; several tasks that have been taken care of in the past via overnight batch runs in the back office have now been moved to the front office and are executed in real time.

Again, one can observe an interplay between advances in technology and financial/business practice. On the one hand, there is the need to constantly improve analytics approaches in terms of speed and capability by applying modern technologies. On the other hand, advances on the technology side allow new analytics approaches that were considered impossible (or infeasible due to budget constraints) a couple of years or even months ago.

One major trend in the analytics space has been the utilization of parallel architectures on the central processing unit (CPU) side and massively parallel architectures on the general-purpose graphics processing unit (GPGPU) side. Current GPGPUs
have computing cores in the thousands, making necessary a sometimes radical rethinking of what parallelism might mean to different algorithms. What is still an obstacle in this regard is that users generally have to learn new programming paradigms and techniques to harness the power of such hardware.

**Python for Finance**

The previous section described selected aspects characterizing the role of technology in finance:

- Costs for technology in the finance industry
- Technology as an enabler for new business and innovation
- Technology and talent as barriers to entry in the finance industry
- Increasing speeds, frequencies, and data volumes
- The rise of real-time analytics

This section analyzes how Python can help in addressing several of the challenges these imply. But first, on a more fundamental level, a brief analysis of Python for finance from a language and syntax point of view.

**Finance and Python Syntax**

Most people who make their first steps with Python in a finance context may attack an algorithmic problem. This is similar to a scientist who, for example, wants to solve a differential equation, evaluate an integral, or simply visualize some data. In general, at this stage, little thought is given to topics like a formal development process, testing, documentation, or deployment. However, this especially seems to be the stage where people fall in love with Python. A major reason for this might be that Python syntax is generally quite close to the mathematical syntax used to describe scientific problems or financial algorithms.

This can be illustrated by a financial algorithm, namely the valuation of a European call option by Monte Carlo simulation. The example considers a Black-Scholes-Merton (BSM) setup in which the option’s underlying risk factor follows a geometric Brownian motion.

Assume the following numerical *parameter values* for the valuation:

- Initial stock index level $S_0 = 100$
- Strike price of the European call option $K = 105$
- Time to maturity $T = 1$ year
- Constant, riskless short rate $r = 0.05$
• Constant volatility $\sigma = 0.2$

In the BSM model, the index level at maturity is a random variable given by Equation 1-1, with $z$ being a standard normally distributed random variable.

**Equation 1-1. Black-Scholes-Merton (1973) index level at maturity**

$$S_T = S_0 \exp \left( (r - \frac{1}{2} \sigma^2)T + \sigma \sqrt{T}z \right)$$

The following is an algorithmic description of the Monte Carlo valuation procedure:

1. Draw $I$ pseudo-random numbers $z(i), i \in \{1, 2, \ldots, I\}$, from the standard normal distribution.
2. Calculate all resulting index levels at maturity $S_T(i)$ for given $z(i)$ and Equation 1-1.
3. Calculate all inner values of the option at maturity as $h_T(i) = \max (S_T(i) - K, 0)$.
4. Estimate the option present value via the Monte Carlo estimator as given in Equation 1-2.

**Equation 1-2. Monte Carlo estimator for European option**

$$C_0 \approx e^{-rT} \frac{1}{T} \sum_{i=1}^{I} h_T(i)$$

This problem and algorithm must now be translated into Python. The following code implements the required steps:

```python
In [6]: import math
   import numpy as np

In [7]: S0 = 100.  
   K = 105.  
   T = 1.0  
   r = 0.05  
   sigma = 0.2  

In [8]: I = 100000  

In [9]: np.random.seed(1000)  

In [10]: z = np.random.standard_normal(I)  

In [11]: ST = S0 * np.exp((r - sigma ** 2 / 2) * T + sigma * math.sqrt(T) * z)
```
In [12]: hT = np.maximum(ST - K, 0)  

In [13]: C0 = math.exp(-r * T) * np.mean(hT)  

In [14]: print('Value of the European call option: {:5.3f}.'.format(C0))  
   Value of the European call option: 8.019.

1 NumPy is used here as the main package.
2 The model and simulation parameter values are defined.
3 The seed value for the random number generator is fixed.
4 Standard normally distributed random numbers are drawn.
5 End-of-period values are simulated.
6 The option payoffs at maturity are calculated.
7 The Monte Carlo estimator is evaluated.
8 The resulting value estimate is printed.

Three aspects are worth highlighting:

Syntax
The Python syntax is indeed quite close to the mathematical syntax, e.g., when it comes to the parameter value assignments.

Translation
Every mathematical and/or algorithmic statement can generally be translated into a single line of Python code.

Vectorization
One of the strengths of NumPy is the compact, vectorized syntax, e.g., allowing for 100,000 calculations within a single line of code.

This code can be used in an interactive environment like IPython or Jupyter Notebook. However, code that is meant to be reused regularly typically gets organized in so-called modules (or scripts), which are single Python files (technically text files) with the suffix .py. Such a module could in this case look like Example 1-1 and could be saved as a file named bsm_mcs_euro.py.
Example 1-1. Monte Carlo valuation of European call option

```python
# Monte Carlo valuation of European call option
# in Black-Scholes-Merton model
# bsm_mcs_euro.py
#
# Python for Finance, 2nd ed.
# (c) Dr. Yves J. Hilpisch
#
import math
import numpy as np

# Parameter Values
S0 = 100.  # initial index level
K = 105.  # strike price
T = 1.0   # time-to-maturity
r = 0.05  # riskless short rate
sigma = 0.2 # volatility
I = 100000 # number of simulations

# Valuation Algorithm
z = np.random.standard_normal(I) # pseudo-random numbers
# index values at maturity
ST = S0 * np.exp((r - 0.5 * sigma ** 2) * T + sigma * math.sqrt(T) * z)
hT = np.maximum(ST - K, 0) # payoff at maturity
C0 = math.exp(-r * T) * np.mean(hT) # Monte Carlo estimator

# Result Output
print('Value of the European call option \$%5.3f.\'$ % C0)
```

The algorithmic example in this subsection illustrates that Python, with its very syntax, is well suited to complement the classic duo of scientific languages, English and mathematics. It seems that adding Python to the set of scientific languages makes it more well rounded. One then has:

- **English** for writing and talking about scientific and financial problems, etc.
- **Mathematics** for concisely, exactly describing and modeling abstract aspects, algorithms, complex quantities, etc.
- **Python** for technically modeling and implementing abstract aspects, algorithms, complex quantities, etc.
Mathematics and Python Syntax

There is hardly any programming language that comes as close to mathematical syntax as Python. Numerical algorithms are therefore in general straightforward to translate from the mathematical representation into the Pythonic implementation. This makes prototyping, development, and code maintenance in finance quite efficient with Python.

In some areas, it is common practice to use pseudo-code and therewith to introduce a fourth language family member. The role of pseudo-code is to represent, for example, financial algorithms in a more technical fashion that is both still close to the mathematical representation and already quite close to the technical implementation. In addition to the algorithm itself, pseudo-code takes into account how computers work in principle.

This practice generally has its cause in the fact that with most (compiled) programming languages the technical implementation is quite “far away” from its formal, mathematical representation. The majority of programming languages make it necessary to include so many elements that are only technically required that it is hard to see the equivalence between the mathematics and the code.

Nowadays, Python is often used in a pseudo-code way since its syntax is almost analogous to the mathematics and since the technical “overhead” is kept to a minimum. This is accomplished by a number of high-level concepts embodied in the language that not only have their advantages but also come in general with risks and/or other costs. However, it is safe to say that with Python you can, whenever the need arises, follow the same strict implementation and coding practices that other languages might require from the outset. In that sense, Python can provide the best of both worlds: high-level abstraction and rigorous implementation.

Efficiency and Productivity Through Python

At a high level, benefits from using Python can be measured in three dimensions:

Efficiency

How can Python help in getting results faster, in saving costs, and in saving time?

Productivity

How can Python help in getting more done with the same resources (people, assets, etc.)?

Quality

What does Python allow one to do that alternative technologies do not allow for?
A discussion of these aspects can by nature not be exhaustive. However, it can highlight some arguments as a starting point.

**Shorter time-to-results**

A field where the efficiency of Python becomes quite obvious is interactive data analytics. This is a field that benefits tremendously from such powerful tools as IPython, Jupyter Notebook, and packages like pandas.

Consider a finance student who is writing their master’s thesis and is interested in S&P 500 index values. They want to analyze historical index levels for, say, a few years to see how the volatility of the index has fluctuated over time and hope to find evidence that volatility, in contrast to some typical model assumptions, fluctuates over time and is far from being constant. The results should also be visualized. The student mainly has to do the following:

- Retrieve index level data from the web
- Calculate the annualized rolling standard deviation of the log returns (volatility)
- Plot the index level data and the volatility results

These tasks are complex enough that not too long ago one would have considered them to be something for professional financial analysts only. Today, even the finance student can easily cope with such problems. The following code shows how exactly this works—without worrying about syntax details at this stage (everything is explained in detail in subsequent chapters):

```python
In [16]: import numpy as np  
    import pandas as pd  
    from pylab import plt, mpl

In [17]: plt.style.use('seaborn')  
    mpl.rcParams['font.family'] = 'serif'  
    %matplotlib inline

In [18]: data = pd.read_csv('..../source/tr_eikon_eod_data.csv',  
                          index_col=0, parse_dates=True)  
    data = pd.DataFrame(data['.SPX'])  
    data.dropna(inplace=True)  
    data.info()  
    <class 'pandas.core.frame.DataFrame'>  
    DatetimeIndex: 2138 entries, 2010-01-04 to 2018-06-29  
    Data columns (total 1 columns):  
      .SPX 2138 non-null float64  
    dtypes: float64(1)  
    memory usage: 33.4 KB

In [19]: data['rets'] = np.log(data / data.shift(1))  
    data['vola'] = data['rets'].rolling(252).std() * np.sqrt(252)
```
In [20]: data[['SPX', 'vola']].plot(subplots=True, figsize=(10, 6)); 

1. This imports NumPy and pandas.

2. This imports matplotlib and configures the plotting style and approach for Jupyter.

3. `pd.read_csv()` allows the retrieval of remotely or locally stored data sets in comma-separated values (CSV) form.

4. A subset of the data is picked and NaN (“not a number”) values eliminated.

5. This shows some metainformation about the data set.

6. The log returns are calculated in vectorized fashion (“no looping” on the Python level).

7. The rolling, annualized volatility is derived.

8. This finally plots the two time series.

Figure 1-1 shows the graphical result of this brief interactive session. It can be considered almost amazing that a few lines of code suffice to implement three rather complex tasks typically encountered in financial analytics: data gathering, complex and repeated mathematical calculations, as well as visualization of the results. The example illustrates that pandas makes working with whole time series almost as simple as doing mathematical operations on floating-point numbers.

Translated to a professional finance context, the example implies that financial analysts can—when applying the right Python tools and packages that provide high-level abstractions—focus on their domain and not on the technical intrinsicalities. Analysts can also react faster, providing valuable insights almost in real time and making sure they are one step ahead of the competition. This example of increased efficiency can easily translate into measurable bottom-line effects.
Ensuring high performance

In general, it is accepted that Python has a rather concise syntax and that it is relatively efficient to code with. However, due to the very nature of Python being an interpreted language, the prejudice persists that Python often is too slow for compute-intensive tasks in finance. Indeed, depending on the specific implementation approach, Python can be really slow. But it does not have to be slow—it can be highly performing in almost any application area. In principle, one can distinguish at least three different strategies for better performance:

Idioms and paradigms

In general, many different ways can lead to the same result in Python, but sometimes with rather different performance characteristics; “simply” choosing the right way (e.g., a specific implementation approach, such as the judicious use of data structures, avoiding loops through vectorization, or the use of a specific package such as pandas) can improve results significantly.

Compiling

Nowadays, there are several performance packages available that provide compiled versions of important functions or that compile Python code statically or dynamically (at runtime or call time) to machine code, which can make such functions orders of magnitude faster than pure Python code; popular ones are Cython and Numba.
Parallelization

Many computational tasks, in particular in finance, can significantly benefit from parallel execution; this is nothing special to Python but something that can easily be accomplished with it.

Performance Computing with Python

Python per se is not a high-performance computing technology. However, Python has developed into an ideal platform to access current performance technologies. In that sense, Python has become something like a glue language for performance computing technologies.

This subsection sticks to a simple, but still realistic, example that touches upon all three strategies (later chapters illustrate the strategies in detail). A quite common task in financial analytics is to evaluate complex mathematical expressions on large arrays of numbers. To this end, Python itself provides everything needed:

```
In [21]: import math
   ...: loops = 2500000
   ...: a = range(1, loops)
   ...: def f(x):
   ...:     return 3 * math.log(x) + math.cos(x) ** 2
   ...: timeit r = [f(x) for x in a]
1.59 s ± 41.2 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

The Python interpreter needs about 1.6 seconds in this case to evaluate the function f() 2,500,000 times. The same task can be implemented using NumPy, which provides optimized (i.e., precompiled) functions to handle such array-based operations:

```
In [22]: import numpy as np
   ...: a = np.arange(1, loops)
   ...: timeit r = 3 * np.log(a) + np.cos(a) ** 2
87.9 ms ± 1.73 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Using NumPy considerably reduces the execution time to about 88 milliseconds. However, there is even a package specifically dedicated to this kind of task. It is called numexpr, for “numerical expressions.” It compiles the expression to improve upon the performance of the general NumPy functionality by, for example, avoiding in-memory copies of ndarray objects along the way:

```
In [23]: import numexpr as ne
   ...: ne.set_num_threads(1)
   ...: f = '3 * log(a) + cos(a) ** 2'
   ...: timeit r = ne.evaluate(f)
50.6 ms ± 4.2 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```
Using this more specialized approach further reduces execution time to about 50 milliseconds. However, `numexpr` also has built-in capabilities to parallelize the execution of the respective operation. This allows us to use multiple threads of a CPU:

```
In [24]: ne.set_num_threads(4)
%timeit r = ne.evaluate(f)
22.8 ms ± 1.76 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Parallelization brings execution time further down to below 23 milliseconds in this case, with four threads utilized. Overall, this is a performance improvement of more than 90 times. Note, in particular, that this kind of improvement is possible without altering the basic problem/algorithm and without knowing any detail about compiling or parallelization approaches. The capabilities are accessible from a high level even by non-experts. However, one has to be aware, of course, of which capabilities and options exist.

This example shows that Python provides a number of options to make more out of existing resources—i.e., to increase productivity. With the parallel approach, three times as many calculations can be accomplished in the same amount of time as compared to the sequential approach—in this case simply by telling Python to use multiple available CPU threads instead of just one.

**From Prototyping to Production**

Efficiency in interactive analytics and performance when it comes to execution speed are certainly two benefits of Python to consider. Yet another major benefit of using Python for finance might at first sight seem a bit subtler; at second sight, it might present itself as an important strategic factor for financial institutions. It is the possibility to use Python end-to-end, from prototyping to production.

Today’s practice in financial institutions around the globe, when it comes to financial development processes, is still often characterized by a separated, two-step process. On the one hand, there are the quantitative analysts (“quants”) responsible for model development and technical prototyping. They like to use tools and environments like Matlab and R that allow for rapid, interactive application development. At this stage of the development efforts, issues like performance, stability, deployment, access management, and version control, among others, are not that important. One is mainly looking for a proof of concept and/or a prototype that exhibits the main desired features of an algorithm or a whole application.

Once the prototype is finished, IT departments with their developers take over and are responsible for translating the existing prototype code into reliable, maintainable, and performant production code. Typically, at this stage there is a paradigm shift in that compiled languages, such as C++ or Java, are used to fulfill the requirements for deployment and production. Also, a formal development process with professional tools, version control, etc., is generally applied.
This two-step approach has a number of generally unintended consequences:

**Inefficiencies**
Prototype code is not reusable; algorithms have to be implemented twice; redundant efforts take time and resources; risks arise during translation

**Diverse skill sets**
Different departments show different skill sets and use different languages to implement “the same things”; people not only program but also speak different languages

**Legacy code**
Code is available and has to be maintained in different languages, often using different styles of implementation

Using Python, on the other hand, enables a **streamlined** end-to-end process from the first interactive prototyping steps to highly reliable and efficiently maintainable production code. The communication between different departments becomes easier. The training of the workforce is also more streamlined in that there is only one major language covering all areas of financial application building. It also avoids the inherent inefficiencies and redundancies when using different technologies in different steps of the development process. All in all, Python can provide a **consistent technological framework** for almost all tasks in financial analytics, financial application development, and algorithm implementation.

**Data-Driven and AI-First Finance**

Basically all the observations regarding the relationship of technology and the financial industry first formulated in 2014 for the first edition of this book still seem pretty current and important in August 2018, at the time of updating this chapter for the second edition of the book. However, this section comments on two major trends in the financial industry that are about to reshape it in a fundamental way. These two trends have mainly crystallized themselves over the last few years.

**Data-Driven Finance**

Some of the most important financial theories, such as MPT and CAPM, date as far back as to the 1950s and 1960s. However, they still represent a cornerstone in the education of students in such fields as economics, finance, financial engineering, and business administration. This might be surprising since the empirical support for most of these theories is meager at best, and the evidence is often in complete contrast to what the theories suggest and imply. On the other hand, their popularity is understandable since they are close to humans’ expectations of how financial markets might behave and since they are elegant mathematical theories resting on a number of appealing, if in general too simplistic, assumptions.
The scientific method, say in physics, starts with data, for example from experiments or observations, and moves on to hypotheses and theories that are then tested against the data. If the tests are positive, the hypotheses and theories might be refined and properly written down, for instance, in the form of a research paper for publication. If the tests are negative, the hypotheses and theories are rejected and the search begins anew for ones that conform with the data. Since physical laws are stable over time, once such a law is discovered and well tested it is generally there to stay, in the best case, forever.

The history of (quantitative) finance in large parts contradicts the scientific method. In many cases, theories and models have been developed “from scratch” on the basis of simplifying mathematical assumptions with the goal of discovering elegant answers to central problems in finance. Among others, popular assumptions in finance are normally distributed returns for financial instruments and linear relationships between quantities of interest. Since these phenomena are hardly ever found in financial markets, it should not come as a surprise that empirical evidence for the elegant theories is often lacking. Many financial theories and models have been formulated, proven, and published first and have only later been tested empirically. To some extent, this is of course due to the fact that financial data back in the 1950s to the 1970s or even later was not available in the form that it is today even to students getting started with a bachelor’s in finance.

The availability of such data to financial institutions has drastically increased since the early to mid-1990s, and nowadays even individuals doing financial research or getting involved in algorithmic trading have access to huge amounts of historical data down to the tick level as well as real-time tick data via streaming services. This allows us to return to the scientific method, which starts in general with the data before ideas, hypotheses, models, and strategies are devised.

A brief example shall illustrate how straightforward it has become today to retrieve professional data on a large scale even on a local machine, making use of Python and a professional data subscription to the Eikon Data APIs. The following example retrieves tick data for the Apple Inc. stock for one hour during a regular trading day. About 15,000 tick quotes, including volume information, are retrieved. While the symbol for the stock is AAPL, the Reuters Instrument Code (RIC) is AAPL.O:

```python
In [26]: import eikon as ek ①

In [27]: data = ek.get_timeseries('AAPL.O', fields='*',
                                     start_date='2018-10-18 16:00:00',
                                     end_date='2018-10-18 17:00:00',
                                     interval='tick') ②

In [28]: data.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 35350 entries, 2018-10-18 16:00:00.000000 to 2018-10-18
```

Data-Driven and AI-First Finance | 25
Data columns (total 2 columns):
VALUE   35285 non-null float64
VOLUME  35350 non-null float64
dtypes: float64(2)
memory usage: 828.5 KB

In [29]: data.tail() ③
Out[29]:
   VALUE   VOLUME
Date
2018-10-18 16:59:59.433 217.13   10.0
2018-10-18 16:59:59.433 217.13   12.0
2018-10-18 16:59:59.439 217.13  231.0
2018-10-18 16:59:59.754 217.14  100.0
2018-10-18 16:59:59.888 217.13  100.0

① Eikon Data API usage requires a subscription and an API connection.

② Retrieves the tick data for the Apple Inc. (AAPL.O) stock.

③ Shows the last five rows of tick data.

The Eikon Data APIs give access not only to structured financial data, such as historical price data, but also to unstructured data such as news articles. The next example retrieves metadata for a small selection of news articles and shows the beginning of one of the articles as full text:

In [30]: news = ek.get_news_headlines('R:AAPL.O Language:LEN',
                                       date_from='2018-05-01',
                                       date_to='2018-06-29',
                                       count=7) ①

In [31]: news ①
Out[31]:

   versionCreated    text
2018-06-28 23:00:00.000 RPT-FOCUS-AI ambulances and robot doctors: Chl...
2018-06-28 21:23:26.526 Why Investors Should Love Apple’s (AAPL) TV En...
2018-06-28 19:48:32.627 Reuters Insider - Trump: We're reclaiming our ...
Retrieves metadata for a small selection of news articles.

Retrieves the full text of a single article, delivered as an HTML document.

Imports the BeautifulSoup HTML parsing package and ...

... extracts the contents as plain text (a str object).

Prints the beginning of the news article.
Although just scratching the surface, these two examples illustrate that structured and unstructured historical financial data is available in a standardized, efficient way via Python wrapper packages and data subscription services. In many circumstances, similar data sets can be accessed for free even by individuals who make use of, for instance, trading platforms such as the one by FXCM Group, LLC, that is introduced in Chapter 14 and also used in Chapter 16. Once the data is on the Python level—indeoendent from the original source—the full power of the Python data analytics ecosystem can be harnessed.

**Data-Driven Finance**

Data is what drives finance these days. Even some of the largest and often most successful hedge funds call themselves “data-driven” instead of “finance-driven.” More and more offerings are making huge amounts of data available to large and small institutions and individuals. Python is generally the programming language of choice to interact with the APIs and to process and analyze the data.

**AI-First Finance**

With the availability of large amounts of financial data via programmatic APIs, it has become much easier and more fruitful to apply methods from artificial intelligence (AI) in general and from machine and deep learning (ML, DL) in particular to financial problems, such as in algorithmic trading.

Python can be considered a first-class citizen in the AI world as well. It is often the programming language of choice for AI researchers and practitioners alike. In that sense, the financial domain benefits from developments in diverse fields, sometimes not even remotely connected to finance. As one example consider the TensorFlow open source package for deep learning, which is developed and maintained by Google Inc. and used by (among others) its parent company Alphabet Inc. in its efforts to build, produce, and sell self-driving cars.

Although for sure not even remotely related to the problem of automatically, algorithmically trading stock, TensorFlow can, for example, be used to predict movements in financial markets. Chapter 15 provides a number of examples in this regard.

One of the most widely used Python packages for ML is scikit-learn. The code that follows shows how, in a highly simplified manner, classification algorithms from ML can be used to predict the direction of future market price movements and to base an algorithmic trading strategy on those predictions. All the details are explained in Chapter 15, so the example is therefore rather concise. First, the data import and the preparation of the features data (directional lagged log return data):
In [36]: import numpy as np
   import pandas as pd

In [37]: data = pd.read_csv('..//..//source/tr_eikon_eod_data.csv',
   index_col=0, parse_dates=True)
   data = pd.DataFrame(data['AAPL.O'])
   data['Returns'] = np.log(data / data.shift())
   data.dropna(inplace=True)

In [38]: lags = 6

In [39]: cols = []
   for lag in range(1, lags + 1):
      col = 'lag_{}'.format(lag)
      data[col] = np.sign(data['Returns'].shift(lag))
      cols.append(col)
   data.dropna(inplace=True)

1. Selects historical end-of-day data for the Apple Inc. stock (AAPL.O).
2. Calculates the log returns over the complete history.
3. Generates DataFrame columns with directional lagged log return data (+1 or -1).

Next, the instantiation of a model object for a support vector machine (SVM) algorithm, the fitting of the model, and the prediction step. Figure 1-2 shows that the prediction-based trading strategy, going long or short on Apple Inc. stock depending on the prediction, outperforms the passive benchmark investment in the stock itself:

In [40]: from sklearn.svm import SVC

In [41]: model = SVC(gamma='auto')

In [42]: model.fit(data[cols], np.sign(data['Returns']))
Out[42]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
   decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
   max_iter=-1, probability=False, random_state=None, shrinking=True,
   tol=0.001, verbose=False)

In [43]: data['Prediction'] = model.predict(data[cols])

In [44]: data['Strategy'] = data['Prediction'] * data['Returns']

In [45]: data[['Returns', 'Strategy']].cumsum().apply(np.exp).plot(  
   figsize=(10, 6));

1. Instantiates the model object.
2. Fits the model, given the features and the label data (all directional).
Uses the fitted model to create the predictions (in-sample), which are the positions of the trading strategy at the same time (long or short).

Calculates the log returns of the trading strategy given the prediction values and the benchmark log returns.

Plots the performance of the ML-based trading strategy compared to the performance of the passive benchmark investment.

![Figure 1-2. ML-based algorithmic trading strategy vs. passive benchmark investment in Apple Inc. stock](image)

The simplified approach taken here does not account for transaction costs, nor does it separate the data set into training and testing subsets. However, it shows how straightforward the application of ML algorithms to financial data is, at least in a technical sense; practically, a number of important topics need to be considered (see López de Prado (2018)).

**AI-First Finance**

AI will reshape finance in a way that other fields have been reshaped already. The availability of large amounts of financial data via programmatic APIs functions as an enabler in this context. Basic methods from AI, ML, and DL are introduced in Chapter 13 and applied to algorithmic trading in Chapters 15 and 16. A proper treatment of *AI-first finance*, however, would require a book fully dedicated to the topic.
AI in finance, as a natural extension of data-driven finance, is for sure a fascinating and exciting field, both from a research and a practitioner’s point of view. Although this book uses several methods from AI, ML, and DL in different contexts, overall the focus lies—in line with the subtitle of the book—on the fundamental Python techniques and approaches needed for data-driven finance. These are, however, equally important for AI-first finance.

**Conclusion**

Python as a language—and even more so as an ecosystem—is an ideal technological framework for the financial industry as whole and the individual working in finance alike. It is characterized by a number of benefits, like an elegant syntax, efficient development approaches, and usability for prototyping as well as production. With its huge amount of available packages, libraries, and tools, Python seems to have answers to most questions raised by recent developments in the financial industry in terms of analytics, data volumes and frequency, compliance and regulation, as well as technology itself. It has the potential to provide a single, powerful, consistent framework with which to streamline end-to-end development and production efforts even across larger financial institutions.

In addition, Python has become the programming language of choice for artificial intelligence in general and machine and deep learning in particular. Python is therefore the right language for data-driven finance as well as for AI-first finance, two recent trends that are about to reshape finance and the financial industry in fundamental ways.

**Further Resources**

The following books cover several aspects only touched upon in this chapter in more detail (e.g., Python tools, derivatives analytics, machine learning in general, and machine learning in finance):


When it comes to algorithmic trading, the author’s company offers a range of online training programs that focus on Python and other tools and techniques required in this rapidly growing field:
Sources referenced in this chapter are, among others, the following:

- Ding, Cubillas (2010). “Optimizing the OTC Pricing and Valuation Infrastructure.” Celent.
Symbols
% character, 71
%time function, 276
%timeit function, 276
* (multiplication) operator, 150, 161
+ (addition) operator, 150, 161
2D plotting
    interactive, 195-203
    matplotlib import and customization, 168
    one-dimensional data sets, 169-176
    other plot styles, 183-191
    two-dimensional data sets, 176-183
3D plotting, 191-194
    __abs__ method, 160
    __add__ method, 161
    __bool__ method, 160
    __getitem__ method, 161
    __init__ method, 155, 159
    __iter__ method, 162
    __len__ method, 161
    __mul__ method, 161
    __repr__ method, 160
    __sizeof__ method, 150
{} (curly braces), 71

A
absolute differences, calculating, 212
absolute price data, 442
abstraction, 147
acknowledgments, xviii
adaptive quadrature, 336
addition (+) operator, 150, 161
aggregation, 148, 158
AI-first finance, 28
algorithmic trading
    automated trading, 521-554
    FXCM trading platform, 467-481
    trading strategies, 483-520
algorithms (see also financial algorithms)
    Fibonacci numbers, 286-289
    for supervised learning, 448
    for unsupervised learning, 444
    prime numbers, 282-285
    the number pi, 290-293
Amazon Web Services (AWS), 50
American options, 376, 380, 607-614
anonymous functions, 80
antithetic paths, 573
antithetic variates, 373
append() method, 136
appending, using pandas, 136
apply() method, 142, 218
approximation
    interpolation technique, 324-328
    main focus of, 312
    package imports and customizations, 312
    regression technique, 313-324
arbitrary-precision floats, 65
array module, 88
arrays (see also NumPy)
    handling with pure Python code, 86-90
    I/O with PyTables, 262
    Python array class, 88-90
    writing and reading NumPy arrays, 242
artificial intelligence (AI), 28
Asian payoff, 606
attributes, in object-oriented programming, 145
attributions, xvi
automated trading
  capital management, 522-532
  infrastructure and deployment, 546
  logging and monitoring, 547-550
  ML-based trading strategy, 532-543
  online algorithm, 544
  Python scripts, 550-554
  risk management, 547
average_cy1() function, 280
average_nb() function, 279
average_np() function, 278
average_py() function, 277

B
Bayesian statistics
  Bayesian regression, 430
  Bayes’ formula, 429
  concept of, 398
  real-world data application, 435
  updating estimates over time, 439
Benevolent Dictator for Life, 5
Bermudan exercise, 380, 607
big data, 13, 231
binomial trees
  Cox, Ross, and Rubinstein pricing model, 294
  Cython implementation, 297
  Numba implementation, 297
  NumPy implementation, 295
  Python implementation, 294
bit_length() method, 62
Black-Scholes-Merton (BSM), 14, 299, 353, 356, 369, 673-676
Booleans, 66
boxplots, 188
Brownian motion, 299, 354, 356, 399, 491
bsm_functions.py module, 378

C
call options, 375
callback functions, 477
candles data, 472
capital asset pricing model, 398
capital management
  Kelly criterion for stocks and indices, 527-532
  Kelly criterion in binomial settings, 522-526
capital market line, 425
changes over time, calculating, 212-215
charts and graphs (see data visualization)
  Chi square distribution, 351
  Cholesky decomposition, 365
  class attributes, 145
classes
  building custom, 154-159
  in object-oriented programming, 145
classification problems, 448, 504-511
cloud instances
  basics of, 34
  benefits of, 56
  files required, 51
  installation script for Python and Jupyter Notebook, 53
  Jupyter Notebook configuration file, 52
  major tools used, 50
  RSA public and private keys, 51
  script to orchestrate Droplet setup, 55
  selecting appropriate hardware architecture, 273
  service providers, 50
code examples, obtaining and using, xvi
coin tossing game, 522
collection operators, 66
compilation
  dynamic compiling, 276, 279
  packages to speed up algorithms, 308
  static, 280
complex selection, using pandas, 132-135
composition, 148
compressed tables, 260
concatenation, using pandas, 135
conda
  basic package management with, 37-41
  Miniconda installation, 35
  virtual environment management with, 41-44
constant short rate, 563
constant volatility, 365
constants, 565
containers, 34 (see also Docker containers)
contingent claims, valuation of, 375
control structures, 78
convex optimization
  constrained optimization, 332
  global minimum representation, 328
  global optimization, 329
  local optimization, 331
use cases for, 328
   correlation analysis
data for, 222
direct correlation measures, 227
logarithmic returns, 224
OLS regression, 226
count() method, 76
counter-based looping, 78
covariance matrix, 416
covariances, 398
Cox, Ross, and Rubinstein pricing model, 294, 359
count() method, 76
counter-based looping, 78
count() method, 76
counter-based looping, 78
covariance matrix, 416
covariances, 398
DAX 30 stock index, 637
decision trees (DTs), 452
deep learning (DL), 28, 454
deep neural networks (DNNs)
   benefits and drawbacks of, 454
   feature transforms, 457
   trading strategies and, 512-519
   train-test splits and, 459
   with scikit-learn, 454
   with TensorFlow, 455
delta, 599
derivatives analytics
   derivatives valuation, 595-616
   DX analytics package, 556, 617
   DX pricing library, 555
   market-based valuation, 637-657
   portfolio valuation, 617-636
   simulation of financial models, 571-592
   valuation framework, 557-569
derivatives portfolios
   class to model, 622-626
   use cases for, 626-633
derivatives positions
   class to model, 618
   use cases for, 620
derivatives valuation
   American exercise, 607-614
   European exercise, 600-607
   generic valuation class, 596-600
derivatives_portfolio class, 627, 634
work with DataFrame objects, 115-118, 152
work with ndarray objects, 119-123, 151, 170
DataFrame() function, 119
date-time information (see also financial time series data)
   financial plots, 199-203
   managing with pandas, 119-123
   modeling and handling dates, 561
   NumPy functionality for handling, 665-667
   pandas functionality for handling, 668-670
   parsing with regular expressions, 74
   plotting, 667
   Python datetime module, 659-665
datetime module, 659-665
datetime64 information, 667
DateTimeIndex objects, 120, 668
date_range() function, 121
DAX 30 stock index, 637
decision trees (DTs), 452
depth learning (DL), 28, 454
depth neural networks (DNNs)
   benefits and drawbacks of, 454
   feature transforms, 457
   trading strategies and, 512-519
   train-test splits and, 459
   with scikit-learn, 454
   with TensorFlow, 455
delta, 599
derivatives analytics
   derivatives valuation, 595-616
   DX analytics package, 556, 617
   DX pricing library, 555
   market-based valuation, 637-657
   portfolio valuation, 617-636
   simulation of financial models, 571-592
   valuation framework, 557-569
derivatives portfolios
   class to model, 622-626
   use cases for, 626-633
derivatives positions
   class to model, 618
   use cases for, 620
derivatives valuation
   American exercise, 607-614
   European exercise, 600-607
   generic valuation class, 596-600
   derivatives_portfolio class, 627, 634

D

data visualization
   interactive 2D plotting, 195-203
   packages for, 167
   static 2D plotting, 168-191
   static 3D plotting, 191-194
   using pandas, 126
Data-Driven Documents (D3.js) standard, 167, 195
data-driven finance, 24
DataFrame class
   benefits of, 114
   major features of, 115
derivatives_position class, 634
describe() function, 123, 211
deserialization, 233
df.iplot() method, 196
diachronic interpretation, 429
dict objects, 81, 235
diff() function, 213
digitalization, 10
DigitalOcean, 50
dir function, 63
discretization error, 356
diversification, 416
Docker containers
  basics of, 45
  benefits of, 50
  building an Ubuntu and Python Docker image, 46-50
  Docker images versus Docker containers, 45
double-precision standard, 64
downsampling, 215
Droplets, 50, 55
DST (Daylight Saving Time), 663
dst() method, 663
DX (Derivatives analytiX) pricing library, 555
DX analytics package, 556, 617
dx.constant_short_rate class, 564, 617
dx.derivatives_portfolio, 626
dx.geometric_brownian_motion class, 582, 602, 617
dx.jump_diffusion class, 583, 617
dx.market_environment class, 565, 577, 617, 621
dx.square_root_diffusion class, 588, 617
dx.valuation_class class, 599
dx.valuation_mcs_american class, 611, 618
dx.valuation_mcs_european class, 602, 618
dx_frame.py module, 568
dx_simulation.py, 591
dynamic compiling, 276, 279
dynamic simulation, 356
dynamically typed languages, 62

E
early exercise premium, 382
Editor, 50
efficient frontier, 421, 424
efficient markets hypothesis (EMH), 399, 492
Eikon Data API, 25
elif control element, 79
e encapsulation, 148, 156
estimation of Greeks, 599
estimation problems, 448
Euler scheme, 357, 360, 583
European options, 375, 600-607, 673-676
eval() method, 142
event-based backtesting, 537
ewma_cy() function, 307
ewma_nb() function, 307
ewma_py() function, 306
Excel files, I/O with pandas, 251
.exectemany() method, 246
execution time, estimating for loops, 276
expected portfolio return, 418
expected portfolio variance, 418
exponentially weighted moving average (EWMA)
  Cython implementation, 307
  equation for, 304
  Numba implementation, 307
  Python implementation, 305

F
fat tails, 385, 413
feature transforms, 457
Fibonacci numbers, 286-289
fib_rec_py1() function, 286
filter() function, 80
finance
  AI-first finance, 28
  data-driven, 24
  role of Python in, 14-24
  role of technology in, 9-14
financial algorithms (see also algorithms; automated trading; trading strategies)
  Black-Scholes-Merton (BSM), 14, 299, 353, 356, 369, 673-676
  Cox, Ross, and Rubinstein pricing model, 294, 359
  first-best versus best solutions, 308
  Least-Squares Monte Carlo (LSM), 381, 608
  online algorithm, 544
  simulation of financial models, 571-592
  support vector machine (SVM), 29, 460
financial and data analytics
  challenges of, 13
  definition of, 13
selecting appropriate hardware architecture, 273
write once, retrieve multiple times, 267
financial indicators, 217
financial instruments
custom modeling using Python classes, 154-159
symbols for (RICs), 209
financial studies, 217
financial theory, 398
financial time series data
changes over time, 212-215
correlation analysis using pandas, 222-227
data import using pandas, 206-209
definition and examples of, 205
high frequency data using pandas, 228
package imports and customizations, 206
recursive pandas algorithms for, 304-308
resampling, 215
rolling statistics using pandas, 217-222
statistical analysis of real-world data,
409-415
summary statistics using pandas, 210-212
tools for, 205
find_MAP() function, 432
first in, first out (FIFO) principle, 235
first-best solution, 308
fixed Gaussian quadrature, 336
flash trading, 12
floats, 63
flow control, 68
for loops, 78
foresight bias, avoiding, 217
format() function, 71
frequency approach, 501-503
frequency distribution, 631
full truncation, 360
functional programming, 80
Fundamental Theorem of Asset Pricing, 558-560
FXCM trading platform
getting started, 469
retrieving prepackaged historical data
candles data, 475
initial steps, 474
placing orders, 478
streaming data, 477
fxcmpy package, 469
G
Gaussian mixture, 444, 447
Gaussian Naive Bayes (GNB), 449, 504
gbm_mcs_dyna() function, 377
gbm_mcs_stat() function, 376
generate_paths() method, 580
generate_payoff() method, 600
generate_time_grid() method, 574
generic simulation class, 574-577
generic valuation class, 596-600
gen_paths() function, 399
geometric Brownian motion, 356, 399, 577-582
get_info() method, 619
get_instrument_values() method, 575
get_price() method, 156
get_year_deltas() function, 562
graphs and charts (see data visualization)
Greeks, estimation of, 599
Greenwich Mean Time (GMT), 662
GroupBy operations, 130
H
hard disk drives (HDDs), 231
HDF5 database standard, 252, 264
Heston stochastic volatility model, 365
hidden layers, 454
high frequency data, 228
histograms, 186, 225
hit ratio, 500
hybrid disk drives, 231
I
idioms and paradigms, 308
IEEE 754, 64
if control element, 79
immutable objects, 76
import this command, 4
importing, definition of, 6
index() method, 123, 211
inheritance, 147
input/output (I/O) operations
compatibility issues, 236
role in financial analyses, 231
with pandas
    from SQL to pandas, 247
    working with CSV files, 250
    working with Excel files, 251
    working with SQL databases, 245
with PyTables
    out-of-memory computations, 264
    working with arrays, 262
    working with compressed tables, 260
    working with tables, 253
with Python
    reading and writing text files, 236
    working with SQL databases, 239
    writing and reading NumPy arrays, 242
    writing objects to disk, 232
with TsTables
    data retrieval, 270
    data storage, 269
    sample data, 267
instance attributes, 145
instantiation, in object-oriented programming, 146
integers, 62, 149
integrated development environments (IDEs), 6
integration
    integration by simulation, 337
    integration interval, 335
    numerical integration, 336
    package imports and customizations, 334
    use cases for, 334
interactive 2D plotting
    basic plots, 195-199
    financial plots, 199-203
    packages for, 195
interpolation technique
    basic idea of, 324
    linear splines interpolation, 324
    potential drawbacks of, 328
    sci.splrep() and sci.splev() functions, 325
IPython
    benefits and history of, 6
    exiting, 48
    GBM simulation class, 580
    installing, 39
    interactive data analytics and, 19
    tab completion capabilities, 62
    with Python 2.7 syntax, 42
is_prime() function, 283, 285
is_prime_cy2() function, 285
is_prime_nb() function, 285
iterative algorithms, 287
joining, using pandas, 137
jump diffusion, 369, 582-586
Jupyter
    downloading, xvi
    Jupyter Notebook
        basics of, 50
        configuration file, 52
        history of, 6
        installation script, 53
        security measures, 53
k-means clustering algorithm, 444, 446, 499-501
Kelly criterion
    for stocks and indices, 527-532
    in binomial settings, 522-526
kernel density estimator (KDE), 225
key-value stores, 81
keyword module, 66
kurtosis test, 405
lambda functions, 80
LaTeX typesetting, 190, 339
Least-Squares Monte Carlo (LSM), 381, 608
least-squares regression, 321
left join, 137
leverage effect, 365
linear regression, 314
linear splines interpolation, 324
list comprehensions, 79
lists
    constructing arrays with, 86
    defining, 76
    expanding and reducing, 77
    looping over, 79
    in market environment, 565
    in object-oriented programming, 150
    operations and methods, 78
LLVM (low level virtual machine), 279
log returns, calculating, 214, 224
log-normal distribution, 354, 399
logical operators, 67
logistic regression (LR), 451, 504
longest drawdown period, 540
Longstaff-Schwartz model, 608
loops
  Cython, 280
  estimating execution time, 276
  Numba, 279
  NumPy, 278
  Python, 277
loss level, 388

M
machine learning (ML)
  adoption of in financial industry, 28
  basics of, 398
  packages for, 444
  supervised learning, 448-461
  types covered, 444
  unsupervised learning, 444-447
  \texttt{map()} function, 80
market environments, 565, 574
market-based valuation
  model calibration, 641-650
  options data, 638-640
  Python code for, 654
Markov chain Monte Carlo (MCMC) sampling, 432, 437
Markov property, 356
Markowitz, Harry, 397, 415
martingale approach, 560
martingale measure, 375, 558, 578
mathematical tools
  adoption of applied mathematics in financial industry, 311
  approximation, 312-328
  convex optimization, 328-334
  integration, 334-337
  mathematics and Python syntax, 18
  symbolic computation, 337-343
matplotlib
  basics of, 8
  benefits of, 167
  boxplot generation using, 188
  date-time information, 667
  histogram generation using, 186, 225
  matplotlib gallery, 189
  NumPy data structures and, 171
  pandas wrapper around, 126
  scatter plot generation using, 184, 246
  static 2D plotting using, 168-191
  maximization of long-term wealth, 522
  maximization of the Sharpe ratio, 421
  maximum drawdown, 540
McKinney, Wes, 205
\texttt{mcs\_pi\_py()} function, 292
\texttt{mcs\_simulation\_cy()} function, 302
\texttt{mcs\_simulation\_nb()} function, 302
\texttt{mcs\_simulation\_np()} function, 301
\texttt{mcs\_simulation\_py()} function, 300
mean return, 398
\texttt{mean()} method, 129
mean-reverting processes, 359
mean-squared error (MSE), 646
mean-variance portfolio selection, 420
memory layout, 110
memoryless process, 356
merging, using pandas, 139
methods, in object-oriented programming, 145
Miniconda, 35
minimization function, 421
minimization of portfolio variance, 423
\texttt{minimize()} function, 421
\texttt{min\_func\_sharpe()} function, 423
ML-based trading strategy
  optimal leverage, 537
  overview of, 532
  persisting model object, 543
  risk analysis, 539-543
  vectorized backtesting, 533-537
MLPClassifier algorithm class, 454
Modern Portfolio Theory (MPT), 415 (see also portfolio optimization)
modularization, 147, 617
moment matching, 374, 573
Monte Carlo simulation, 14, 290, 299-304, 337, 352, 375
multiplication (\texttt{*}) operator, 150, 161
multiprocessing module, 276, 285, 303
mutable objects, 77

N
noisy data, 319
nonredundancy, 148
\texttt{norm.pdf()} function, 403
normal distribution, 398
normal log returns, 399
normality tests  
benchmark case, 399-409  
real-world data, 409-415  
role of in finance, 397, 398  
skewness, kurtosis, and normality, 405
normality_tests() function, 405  
normalization, 214  
normalized price data, 442  
normaltest(), 405
now() function, 662  
np.allclose() function, 234  
np.arange() function, 242, 666  
np.concatenate() function, 373  
np.dot() function, 419  
np.exp() function, 215  
np.linspace() function, 312  
np.meshgrid() function, 192  
np.polyfit(), 313, 325  
np.polyval(), 313, 325  
np.sum() function, 142  
np.lognormal() function, 354  
np.standard_normal() function, 354
Numba  
binomial trees using, 297  
exponentially weighted moving average (EWMA), 307  
looping in, 279  
Monte Carlo simulation using, 302  
potential drawbacks of, 279  
prime number algorithm, 283
numerical integration, 336
NumPy  
basics of, 8, 85  
binomial trees using, 295  
data structures covered, 85  
date-time information, 665-667  
datetime64 information, 667  
handling arrays of data with Python, 86-90  
looping in, 278  
Monte Carlo simulation using, 301  
regular NumPy arrays  
Boolean arrays, 101  
built-in methods, 91  
mathematical operations, 92  
metainformation, 97  
multiple dimensions, 94  
NumPy dtype objects, 97  
numpy.ndarray class, 90, 151, 170  
reshaping and resizing, 98
speed comparison, 103  
universal functions, 92  
structured NumPy arrays, 105  
universal functions applied to pandas, 126  
vectorization of code, 106-112  
writing and reading NumPy arrays, 242
numpy.random subpackage, 346, 572
NUTS() function, 432

O
object relational mappers, 239
object-oriented programming (OOP)  
benefits and drawbacks of, 145  
dx.derivatives_portfolio class, 626  
example class implementation, 146  
features of, 147  
Python classes, 154-159  
Python data model, 159-163  
Python objects, 149-154  
terminology used in, 145  
Vector class, 163
objects, in object-oriented programming, 145
online algorithm, 544
OpenSSL, 51
optimal decision step, 609
optimal fraction f *, 523
optimal stopping problem, 380, 608
option pricing theory, 399
opts object, 422
ordinary least-squares (OLS) regression, 226, 494-498
out-of-memory computations, 264
overfitting, 491

P
package managers  
basics of, 34  
conda basic operations, 37-41  
Miniconda installation, 35
pandas  
basic analytics, 123-126  
basic visualization, 126  
basics of, 8  
benefits of, 113  
calculating changes over time using, 212-215
complex selection, 132-135
concatenation, 135
 correlation analysis using, 222-227
data formats supported, 244
data structures covered, 113
DataFrame class, 114-123, 152
date-time information, 668-670
development of, 205
error tolerance of, 126
GroupBy operations, 130
handling high frequency data using, 228
import-export functions and methods, 245
importing financial data using, 206-209
joining, 137
merging, 139
multiple options provided by, 143
NumPy universal functions and, 126
performance aspects, 141
recursive function implementations, 304-308
rolling statistics using, 218
Series class, 128
summary statistics using, 210-212
working with CSV files in, 250
working with Excel files in, 251
working with SQL databases in, 245
paradigms and idioms, 308
parallel processing, 285
parallelization, 303, 308
parameters, in object-oriented programming, 146
pct_change() function, 213
pd.concat() function, 136
pd.date_range() function, 668
pd.read_csv() function, 206, 245, 251
percentage change, calculating, 213
perfect foresight, 217
performance Python
algorithms, 281-293
approaches to speed up tasks, 275, 308
binomial trees, 294-298
ensuring high performance, 21
loops, 276-281
Monte Carlo simulation, 299-304
recursive pandas algorithms, 304-308
supposed Python shortcomings, 275
pi (π), 290
pickle.dump() function, 233
pickle.load() function, 233
plot() method, 126, 129
plotly
basic plots, 195
benefits of, 167, 195
Getting Started with Plotly for Python
guide, 195
local or remote rendering, 195
plotting types available, 198
plot_option_stats() function, 605
plt.axis() function, 173
plt.boxplot() function, 188
plt.hist() function, 186
plt.legend() function, 177
plt.plot() function, 169, 177
plt.plot_surface() function, 193
plt.scatter() function, 184
plt.setp() function, 189
plt.subplot() function, 181
plt.title() function, 174
plt.xlabel() function, 174
plt.xlim() function, 173
plt.ylabel() function, 174
plt.ylim() function, 173
Poisson distribution, 351
polymorphism, 148
portfolio optimization
basic theory behind, 417
capital market line, 425
efficient frontier, 424
minimal risk through diversification, 416
normally distributed returns and, 415
optimal portfolios, 421
pioneering work of Harry Markowitz, 397
portfolio theory, 398, 415
portfolio valuation
derivatives portfolios
class to model, 622-626
use cases for, 626-633
derivatives positions
class to model, 618
use cases for, 620
wrapper module for, 634
port_ret() function, 420
port_vol() function, 420
present_value() method, 599
price movements, predicting direction of, 504
pricing library, 555
prime numbers
definition of, 282
multiprocessing module and, 285
testing for with Cython, 284
testing for with Numba, 283
Index | 687
testing for with Python, 282
print() function, 71
print_statistics() function, 355, 402
private instance attributes, 157
probability density function (PDF), 403
probability of default, 388
pseudo-code, 18
pseudo-random numbers, 346, 372
put options, 375
PyMC3, 430
PyTables
basics of, 8
benefits of, 252
out-of-memory computations, 264
working with arrays, 262
working with compressed tables, 260
working with tables, 253
Python data model
benefits of, 163
eexample model implementation, 159-163
tasks and constructs supported by, 159
Python data structures
built-in structures, 75
control structures, 78
dicts, 81, 235
functional programming, 80
lists, 76, 150
sets, 82
structures covered, 61
tuples, 75
Python data types
Booleans, 66
dynamically versus statically typed languages, 62
floats, 63
integers, 62, 149
printing and string replacements, 71
regular expressions and, 74
strings, 69
types covered, 61
Python Enhancement Proposal 20, 4
Python for Algorithmic Trading certificate program, xv
Python infrastructure
cloud instances, 50-56
Docker containers, 45-50
package managers, 35-41
tools and strategies available, 34
version selection and deployment, 33
virtual environment managers, 41-44
Python programming language (see also object-oriented programming)
adoption of in financial industry, xiii
benefits of, 18
ecosystem for, 6, 308
efficiency and productivity through, 18-23
ensuring high performance, 21
executive summary and features, 3
from prototyping to production, 23
history of, 5
scientific stack, 8
syntax, 4, 14-18
user spectrum, 7
Python Quant Platform, xiv
The Python Quants GmbH, 556
Python Standard Library, 6
pytz module, 664
Q
Quant Platform, 556
quantile-quantile (QQ) plots, 404
R
rand() function, 346
random access memory (RAM), 231
random numbers
generating random number to different distribution laws, 349
normal distribution in finance, 350
numpy.random subpackage for, 346
simple random number generation, 347
standard normally distributed, 572
visualization of generation, 348
visualization of random number generation from various distributions, 351
random variables, 353
random walk hypothesis (RWH), 440, 491-494
randomized train-test split, 511
range() method, 78
re module, 74
real-time analytics, 13
real-time data, 477
real-time economy, 13
recombining trees, 294
recursive function implementations, 286, 304-308
reduce() function, 80
regression technique
individual basis functions, 317
least-squares approach, 321
linear regression, 314
monomials as basis functions, 313
multiple dimensions and, 321
noisy data and, 319
np.polyval() function, 314
ordinary least-squares (OLS) regression, 226, 494-498
parameters of polyfit() function, 313
task of, 313
unsorted data, 320
regular expressions, 74
relational databases, 239
relative return data, 442
Relative Strength Index (RSI), 199
relevant markets, 622
replace() method, 70
resampling, 215
reusability, 148
Reuters Instrument Codes (RICs), 209
risk management
automated trading, 547
credit valuation adjustments (CVA), 388
FXCM trading platform, 468
minimizing portfolio risk, 416
valuation classes for, 595
value-at-risk (VaR), 383
risk-neutral discounting
constant short rate, 563
modeling and handling dates, 560
risk-neutral investors, 523
risk-neutral valuation approach, 560
riskless assets, 426
rolling statistics
deriving using pandas, 218
financial time series example, 217
technical analysis example, 220
Romberg integration, 336
RSA public and private keys, 51
SciPy
basics of, 8, 39
documentation, 343, 463
scipy.integrate package, 334
scipy.integrate subpackage, 336
scipy.optimize.minimize() function, 333
scipy.stats subpackage, 355, 402
sco.fmin() function, 331
sco.fsolve() function, 427
scs.describe() function, 356, 402
scs.scoreatpercentile() function, 385
Secure Shell (SSH), 50
Secure Sockets Layer (SSL), 50
dynamic simulation, 356
random variables, 353
stochastic processes, 356
value of in finance, 352
variance reduction, 372
simulation
generic simulation class, 574-577
geometric Brownian motion, 577-582
jump diffusion, 582-586
overview of, 614
random number generation, 572
square-root diffusion, 587-590
wrapper module for, 591
skewness test, 405
slicing, 77
sn_random_numbers() function, 572
solid state disks (SSDs), 231
SQLAlchemy, 239
SQLite3, 239
square-root diffusion, 359, 587-590
stacking, 99
standard normally distributed random numbers, 572
static coupling, 276
statically typed languages, 62
statistical learning, 398
statistics
Bayesian statistics, 429-443
machine learning (ML), 444-461
normality tests, 398-409
portfolio optimization, 415-428
value of in finance, 397
stochastic differential equation (SDE), 299, 356
stochastic processes
definition of, 356
geometric Brownian motion, 356, 399
jump diffusion, 369
square-root diffusion, 359
stochastic volatility, 365
stochastic volatility models, 365
stochastics
Python script, 392
random numbers, 346-352
risk measures, 383-391
simulation, 352-375
use cases for, 345
valuation, 375-382
str() function, 69
streaming data, 477
strike values, 191, 376
strings
parsing date-time information, 74
printing and string replacements, 71
string methods, 69
text representation with, 69
Unicode strings, 71
Structured Query Language (SQL) databases
from SQL to pandas, 247
working with in pandas, 245
working with in Python, 239
sum() method, 142
summary statistics, 210-212
supervised learning
classification versus estimation problems, 448
data for, 448
decision trees (DTs), 452
deep neural networks (DNNs), 454-461
definition of, 448
Gaussian Naive Bayes (GNB), 449, 504
logistic regression (LR), 451, 504
support vector machine (SVM) algorithm, 29, 460, 504
sy.diff() function, 341
Symbol class, 338
symbolic computation
differentiation, 341
equations, 340
integration and differentiation, 340
Symbol class, 338
SymPy library for, 337
SymPy, 337-343
T
tables
compressed tables with PyTables, 260
data retrieval with TsTables, 270
data storage with TsTables, 269
I/O with PyTables, 253
tail risk, 383
technical analysis, rolling statistics using pandas, 220
technology in finance
advances in speed and frequency, 11
potential of, 9
real-time analytics, 13
technology and talent as barriers to entry, 11
technology as enabler, 10
technology spending, 9
TensorFlow, 28, 455, 515-519
Terminal, 50
text files
compatibility issues, 236
I/O with Python, 232
reading and writing with Python, 236
text/code editors, 7, 50
tick data, 228, 470
time indices, 120
time-to-results, improved with Python, 19
timedelta objects, 661
times-to-maturity, 191
Timestamp object, 668
today() function, 662
Index

V

valuation
American options, 380
derivatives valuation, 595-616
European options, 14, 376
market-based valuation, 637-657
portfolio valuation, 617-636
valuation of contingent claims, 375
valuation framework
Fundamental Theorem of Asset Pricing, 558-560
market environments, 565-567
risk-neutral discounting, 560-564
value-at-risk (VaR), 383, 542
van Rossum, Guido, 5
variance of the returns, 398
variance reduction, 372, 573
vectorization of code
benefits of, 308
increased memory footprint with, 278
speeding up typical tasks with, 278
with NumPy, 106-112
with NumPy looping, 278
vectorized backtesting approach, 483, 487, 533-537
vega, 599
view_init() method, 194
Vim, 7
virtual environment managers, 34, 41-44
volatility clusters, spotting, 224
volatility processes, 359
volatility surfaces, 191

Z
Zen of Python, 4
zero-based numbering, 76

Index | 691
About the Author

Dr. Yves J. Hilpisch is founder and managing partner of The Python Quants, a group focusing on the use of open source technologies for financial data science, artificial intelligence, algorithmic trading, and computational finance. He is also founder and CEO of The AI Machine, a company focused on harnessing the power of artificial intelligence for algorithmic trading via a proprietary strategy execution platform. He is the author of two other books:


Yves lectures on computational finance at the CQF Program and on algorithmic trading at the EPAT Program. He is also the director of the first online training program leading to a University Certificate in Python for Algorithmic Trading.

Yves wrote the financial analytics library DX Analytics and organizes meetups, conferences, and bootcamps about Python for quantitative finance and algorithmic trading in London, Frankfurt, Berlin, Paris, and New York. He has given keynote speeches at technology conferences in the United States, Europe, and Asia.

Colophon

The animal on the cover of *Python for Finance* is a Hispaniolan solenodon. The Hispaniolan solenodon (*Solenodon paradoxus*) is an endangered mammal that lives on the Caribbean island of Hispaniola, which comprises Haiti and the Dominican Republic. It’s particularly rare in Haiti and a bit more common in the Dominican Republic.

Solenodons are known to eat arthropods, worms, snails, and reptiles. They also consume roots, fruit, and leaves on occasion. A solenodon weighs a pound or two and has a foot-long head and body plus a ten-inch tail, give or take. This ancient mammal looks somewhat like a big shrew. It’s quite furry, with reddish-brown coloring on top and lighter fur on its undersides, while its tail, legs, and prominent snout lack hair.

It has a rather sedentary lifestyle and often stays out of sight. When it does come out, its movements tend to be awkward, and it sometimes trips when running. However, being a night creature, it has developed an acute sense of hearing, smell, and touch. Its own distinctive scent is said to be “goatlike.”

It excretes toxic saliva from a groove in the second lower incisor and uses it to paralyze and attack its invertebrate prey. As such, it is one of few venomous mammals. Sometimes the venom is released when fighting among each other, and can be fatal to the solenodon itself. Often, after initial conflict, they establish a dominance relation-
ship and get along in the same living quarters. Families tend to live together for a long time. Apparently, it only drinks while bathing.

Many of the animals on O’Reilly covers are endangered; all of them are important to the world. To learn more about how you can help, go to animals.oreilly.com.

The cover image is from Wood’s Illustrated Natural History. The cover fonts are URW Typewriter and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag’s Ubuntu Mono.