LEARNING MADE EASY

2nd Edition

Python® for Data Science

for dummies®
A Wiley Brand

Learn Python data analysis programming and statistics

Write code in the cloud with Google Colab™

Wrangle data and visualize information

John Paul Mueller
Luca Massaron

Authors of *Machine Learning for Dummies* and *Artificial Intelligence For Dummies*

Python® for Data Science

for dummies®
A Wiley Brand

Python® for Data Science

2nd Edition

by John Paul Mueller
and Luca Massaron

for dummies®
A Wiley Brand

Python® for Data Science For Dummies®, 2nd Edition

Published by: **John Wiley & Sons, Inc.,** 111 River Street, Hoboken, NJ 07030-5774, www.wiley.com

Library of Congress Control Number: 2018967877

ISBN: 978-1-119-54762-4; ISBN: 978-1-119-54766-2 (ebk); ISBN: 978-1-119-54764-8 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Contents at a Glance

Introduction 1

Part 1: Getting Started with Data Science and Python 7

CHAPTER 1: Discovering the Match between Data Science and Python 9
CHAPTER 2: Introducing Python's Capabilities and Wonders 21
CHAPTER 3: Setting Up Python for Data Science 39
CHAPTER 4: Working with Google Colab 59

Part 2: Getting Your Hands Dirty with Data 81

CHAPTER 5: Understanding the Tools 83
CHAPTER 6: Working with Real Data 99
CHAPTER 7: Conditioning Your Data 121
CHAPTER 8: Shaping Data 149
CHAPTER 9: Putting What You Know in Action 169

Part 3: Visualizing Information 183

CHAPTER 10: Getting a Crash Course in MatPlotLib 185
CHAPTER 11: Visualizing the Data 201

Part 4: Wrangling Data 227

CHAPTER 12: Stretching Python's Capabilities 229
CHAPTER 13: Exploring Data Analysis 251
CHAPTER 14: Reducing Dimensionality 275
CHAPTER 15: Clustering 295
CHAPTER 16: Detecting Outliers in Data 313

Part 5: Learning from Data 327

CHAPTER 17: Exploring Four Simple and Effective Algorithms 329
CHAPTER 18: Performing Cross-Validation, Selection, and Optimization 347
CHAPTER 19: Increasing Complexity with Linear and Nonlinear Tricks 371
CHAPTER 20: Understanding the Power of the Many 411

Part 6: The Part of Tens 429

CHAPTER 21: Ten Essential Data Resources 431
CHAPTER 22: Ten Data Challenges You Should Take 437

Index 447

Table of Contents

INTRODUCTION 1
- About This Book. 1
- Foolish Assumptions. 3
- Icons Used in This Book 4
- Beyond the Book 4
- Where to Go from Here 5

PART 1: GETTING STARTED WITH DATA SCIENCE AND PYTHON 7

CHAPTER 1: **Discovering the Match between Data Science and Python** 9
- Defining the Sexiest Job of the 21st Century. 11
  - Considering the emergence of data science 12
  - Outlining the core competencies of a data scientist 12
  - Linking data science, big data, and AI. 13
  - Understanding the role of programming 14
- Creating the Data Science Pipeline 14
  - Preparing the data. 15
  - Performing exploratory data analysis 15
  - Learning from data 15
  - Visualizing. 15
  - Obtaining insights and data products 16
- Understanding Python's Role in Data Science. 16
  - Considering the shifting profile of data scientists 16
  - Working with a multipurpose, simple, and efficient language 17
- Learning to Use Python Fast 18
  - Loading data. 19
  - Training a model 19
  - Viewing a result 19

CHAPTER 2: **Introducing Python's Capabilities and Wonders** 21
- Why Python?. 22
  - Grasping Python's Core Philosophy 23
  - Contributing to data science 23
  - Discovering present and future development goals 24

Working with Python 25
Getting a taste of the language 25
Understanding the need for indentation 26
Working at the command line or in the IDE 27
Performing Rapid Prototyping and Experimentation 31
Considering Speed of Execution 32
Visualizing Power 33
Using the Python Ecosystem for Data Science 35
Accessing scientific tools using SciPy 35
Performing fundamental scientific computing using NumPy 36
Performing data analysis using pandas 36
Implementing machine learning using Scikit-learn 36
Going for deep learning with Keras and TensorFlow 37
Plotting the data using matplotlib 38
Creating graphs with NetworkX 38
Parsing HTML documents using Beautiful Soup 38

CHAPTER 3: **Setting Up Python for Data Science** 39
Considering the Off-the-Shelf Cross-Platform Scientific Distributions 40
Getting Continuum Analytics Anaconda 40
Getting Enthought Deployment Manager 41
Getting WinPython 42
Installing Anaconda on Windows 42
Installing Anaconda on Linux 46
Installing Anaconda on Mac OS X 47
Downloading the Datasets and Example Code 48
Using Jupyter Notebook 49
Defining the code repository 50
Understanding the datasets used in this book 57

CHAPTER 4: **Working with Google Colab** 59
Defining Google Colab 60
Understanding what Google Colab does 60
Considering the online coding difference 61
Using local runtime support 63
Getting a Google Account 63
Creating the account 64
Signing in 64
Working with Notebooks 65
Creating a new notebook 65
Opening existing notebooks 66
Saving notebooks 68
Downloading notebooks 71

Performing Common Tasks 71
Creating code cells 71
Creating text cells 72
Creating special cells 73
Editing cells 74
Moving cells 75
Using Hardware Acceleration 75
Executing the Code 76
Viewing Your Notebook 76
Displaying the table of contents 77
Getting notebook information 77
Checking code execution 78
Sharing Your Notebook 79
Getting Help 80

PART 2: GETTING YOUR HANDS DIRTY WITH DATA 81

CHAPTER 5: **Understanding the Tools** 83
Using the Jupyter Console 84
Interacting with screen text 84
Changing the window appearance 86
Getting Python help 87
Getting IPython help 89
Using magic functions 90
Discovering objects 91
Using Jupyter Notebook 93
Working with styles 93
Restarting the kernel 94
Restoring a checkpoint 95
Performing Multimedia and Graphic Integration 96
Embedding plots and other images 96
Loading examples from online sites 96
Obtaining online graphics and multimedia 96

CHAPTER 6: **Working with Real Data** 99
Uploading, Streaming, and Sampling Data 100
Uploading small amounts of data into memory 101
Streaming large amounts of data into memory 102
Generating variations on image data 103
Sampling data in different ways 104
Accessing Data in Structured Flat-File Form 105
Reading from a text file 106
Reading CSV delimited format 107
Reading Excel and other Microsoft Office files 109

Sending Data in Unstructured File Form 111
Managing Data from Relational Databases 113
Interacting with Data from NoSQL Databases 115
Accessing Data from the Web 116

CHAPTER 7: **Conditioning Your Data** 121
Juggling between NumPy and pandas 122
Knowing when to use NumPy 122
Knowing when to use pandas 122
Validating Your Data 124
Figuring out what's in your data 124
Removing duplicates 126
Creating a data map and data plan 126
Manipulating Categorical Variables 129
Creating categorical variables 130
Renaming levels 131
Combining levels 132
Dealing with Dates in Your Data 133
Formatting date and time values 134
Using the right time transformation 135
Dealing with Missing Data 136
Finding the missing data 136
Encoding missingness 137
Imputing missing data 138
Slicing and Dicing: Filtering and Selecting Data 139
Slicing rows 140
Slicing columns 140
Dicing 141
Concatenating and Transforming 142
Adding new cases and variables 142
Removing data 144
Sorting and shuffling 145
Aggregating Data at Any Level 146

CHAPTER 8: **Shaping Data** 149
Working with HTML Pages 150
Parsing XML and HTML 150
Using XPath for data extraction 151
Working with Raw Text 153
Dealing with Unicode 153
Stemming and removing stop words 153
Introducing regular expressions 155
Using the Bag of Words Model and Beyond 158
Understanding the bag of words model 159

Working with n-grams. 161
Implementing TF-IDF transformations 162
Working with Graph Data. 165
Understanding the adjacency matrix 165
Using NetworkX basics 166

CHAPTER 9: **Putting What You Know in Action** 169
Contextualizing Problems and Data 170
Evaluating a data science problem 171
Researching solutions. 173
Formulating a hypothesis. 174
Preparing your data 175
Considering the Art of Feature Creation 175
Defining feature creation 175
Combining variables 176
Understanding binning and discretization 177
Using indicator variables 177
Transforming distributions 178
Performing Operations on Arrays. 178
Using vectorization 179
Performing simple arithmetic on vectors and matrices 179
Performing matrix vector multiplication .180
Performing matrix multiplication 181

PART 3: VISUALIZING INFORMATION 183

CHAPTER 10: **Getting a Crash Course in MatPlotLib**. 185
Starting with a Graph 186
Defining the plot 186
Drawing multiple lines and plots. 187
Saving your work to disk. 188
Setting the Axis, Ticks, Grids 189
Getting the axes. 189
Formatting the axes 190
Adding grids 191
Defining the Line Appearance 192
Working with line styles 193
Using colors 194
Adding markers 195
Using Labels, Annotations, and Legends 197
Adding labels 198
Annotating the chart. 198
Creating a legend. 199

CHAPTER 11: **Visualizing the Data** 201
Choosing the Right Graph 202
Showing parts of a whole with pie charts 202
Creating comparisons with bar charts 203
Showing distributions using histograms 205
Depicting groups using boxplots. 206
Seeing data patterns using scatterplots. 208
Creating Advanced Scatterplots 209
Depicting groups 209
Showing correlations 211
Plotting Time Series. 212
Representing time on axes 212
Plotting trends over time 214
Plotting Geographical Data 216
Using an environment in Notebook 217
Getting the Basemap toolkit 218
Dealing with deprecated library issues 218
Using Basemap to plot geographic data 220
Visualizing Graphs 221
Developing undirected graphs 222
Developing directed graphs 224

PART 4: WRANGLING DATA 227

CHAPTER 12: **Stretching Python's Capabilities** 229
Playing with Scikit-learn 230
Understanding classes in Scikit-learn 230
Defining applications for data science 231
Performing the Hashing Trick 234
Using hash functions. 235
Demonstrating the hashing trick. 235
Working with deterministic selection 239
Considering Timing and Performance 240
Benchmarking with timeit 241
Working with the memory profiler 244
Running in Parallel on Multiple Cores 247
Performing multicore parallelism 248
Demonstrating multiprocessing 248

CHAPTER 13: **Exploring Data Analysis** 251
The EDA Approach. 252
Defining Descriptive Statistics for Numeric Data 253
Measuring central tendency 254
Measuring variance and range 255

Working with percentiles 256
Defining measures of normality 257
Counting for Categorical Data 259
Understanding frequencies 259
Creating contingency tables. 261
Creating Applied Visualization for EDA. 261
Inspecting boxplots 262
Performing t-tests after boxplots 263
Observing parallel coordinates 264
Graphing distributions 265
Plotting scatterplots 266
Understanding Correlation 268
Using covariance and correlation 268
Using nonparametric correlation 270
Considering the chi-square test for tables. 271
Modifying Data Distributions. 272
Using different statistical distributions 272
Creating a Z-score standardization. 273
Transforming other notable distributions 273

CHAPTER 14: **Reducing Dimensionality**. 275
Understanding SVD 276
Looking for dimensionality reduction 277
Using SVD to measure the invisible 279
Performing Factor Analysis and PCA 280
Considering the psychometric model. 280
Looking for hidden factors. 281
Using components, not factors 282
Achieving dimensionality reduction 282
Squeezing information with t-SNE 283
Understanding Some Applications 285
Recognizing faces with PCA 285
Extracting topics with NMF. 289
Recommending movies 291

CHAPTER 15: **Clustering** 295
Clustering with K-means. 297
Understanding centroid-based algorithms 298
Creating an example with image data 299
Looking for optimal solutions 301
Clustering big data 304
Performing Hierarchical Clustering 305
Using a hierarchical cluster solution. 307
Using a two-phase clustering solution 308
Discovering New Groups with DBScan. 310

CHAPTER 16: **Detecting Outliers in Data** 313
Considering Outlier Detection 314
Finding more things that can go wrong 315
Understanding anomalies and novel data. 316
Examining a Simple Univariate Method 317
Leveraging on the Gaussian distribution 319
Making assumptions and checking out 320
Developing a Multivariate Approach 322
Using principal component analysis 322
Using cluster analysis for spotting outliers 324
Automating detection with Isolation Forests. 325

PART 5: LEARNING FROM DATA 327

CHAPTER 17: **Exploring Four Simple and Effective Algorithms** 329
Guessing the Number: Linear Regression 329
Defining the family of linear models 330
Using more variables 331
Understanding limitations and problems 333
Moving to Logistic Regression 334
Applying logistic regression 335
Considering when classes are more 336
Making Things as Simple as Naïve Bayes. 337
Finding out that Naïve Bayes isn't so naïve 339
Predicting text classifications. 340
Learning Lazily with Nearest Neighbors 342
Predicting after observing neighbors 343
Choosing your k parameter wisely 344

CHAPTER 18: **Performing Cross-Validation, Selection, and Optimization** 347
Pondering the Problem of Fitting a Model. 348
Understanding bias and variance 349
Defining a strategy for picking models. 350
Dividing between training and test sets. 354
Cross-Validating 356
Using cross-validation on k folds. 357
Sampling stratifications for complex data 358
Selecting Variables Like a Pro 360
Selecting by univariate measures 360
Using a greedy search. 362
Pumping Up Your Hyperparameters 363
Implementing a grid search 364
Trying a randomized search 368

CHAPTER 19: **Increasing Complexity with Linear and Nonlinear Tricks** 371
Using Nonlinear Transformations. 372
Doing variable transformations 372
Creating interactions between variables 375
Regularizing Linear Models 379
Relying on Ridge regression (L2) 380
Using the Lasso (L1) 381
Leveraging regularization. 382
Combining L1 & L2: Elasticnet 382
Fighting with Big Data Chunk by Chunk 383
Determining when there is too much data 383
Implementing Stochastic Gradient Descent 383
Understanding Support Vector Machines 387
Relying on a computational method 387
Fixing many new parameters 390
Classifying with SVC. 392
Going nonlinear is easy 398
Performing regression with SVR 399
Creating a stochastic solution with SVM 401
Playing with Neural Networks 406
Understanding neural networks 407
Classifying and regressing with neurons 408

CHAPTER 20: **Understanding the Power of the Many** 411
Starting with a Plain Decision Tree 412
Understanding a decision tree 412
Creating trees for different purposes. 415
Making Machine Learning Accessible. 418
Working with a Random Forest classifier. 420
Working with a Random Forest regressor 421
Optimizing a Random Forest 422
Boosting Predictions 424
Knowing that many weak predictors win. 424
Setting a gradient boosting classifier 425
Running a gradient boosting regressor 426
Using GBM hyperparameters 427

PART 6: THE PART OF TENS 429

CHAPTER 21: **Ten Essential Data Resources** 431
Discovering the News with Subreddit 432
Getting a Good Start with KDnuggets. 432
Locating Free Learning Resources with Quora 432

Gaining Insights with Oracle's Data Science Blog 433
Accessing the Huge List of Resources on Data Science Central 433
Learning New Tricks from the Aspirational Data Scientist 434
Obtaining the Most Authoritative Sources at Udacity 435
Receiving Help with Advanced Topics at Conductrics 435
Obtaining the Facts of Open Source Data Science from Masters . . .436
Zeroing In on Developer Resources with Jonathan Bower 436

CHAPTER 22: **Ten Data Challenges You Should Take** 437
Meeting the Data Science London + Scikit-learn Challenge 438
Predicting Survival on the Titanic 438
Finding a Kaggle Competition that Suits Your Needs. 439
Honing Your Overfit Strategies 440
Trudging Through the MovieLens Dataset 440
Getting Rid of Spam E-mails. 441
Working with Handwritten Information. 442
Working with Pictures. 443
Analyzing Amazon.com Reviews 444
Interacting with a Huge Graph 444

INDEX 447

Introduction

Data is increasingly used for every possible purpose, and many of those purposes elude attention, but every time you get on the Internet, you generate even more. It's not just you, either; the growth of the Internet has been phenomenal, according to Internet World Stats (`https://www.internetworldstats.com/emarketing.htm`). Data science turns this huge amount of data into something useful — something that you use absolutely every day to perform an amazing array of tasks or to obtain services from someone else.

In fact, you've probably used data science in ways that you never expected. For example, when you used your favorite search engine this morning to look for something, it made suggestions on alternative search terms. Those terms are supplied by data science. When you went to the doctor last week and discovered the lump you found wasn't cancer, the doctor likely made her prognosis with the help of data science. In fact, you might work with data science every day and not even know it. *Python for Data Science For Dummies,* 2nd Edition not only gets you started using data science to perform a wealth of practical tasks but also helps you realize just how many places data science is used. By knowing how to answer data science problems and where to employ data science, you gain a significant advantage over everyone else, increasing your chances at promotion or that new job you really want.

About This Book

The main purpose of *Python for Data Science For Dummies,* 2nd Edition is to take the scare factor out of data science by showing you that data science is not only really interesting but also quite doable using Python. You might assume that you need to be a computer science genius to perform the complex tasks normally associated with data science, but that's far from the truth. Python comes with a host of useful libraries that do all the heavy lifting for you in the background. You don't even realize how much is going on, and you don't need to care. All you really need to know is that you want to perform specific tasks, and Python makes these tasks quite accessible.

Part of the emphasis of this book is on using the right tools. You start with Anaconda, a product that includes IPython and Jupyter Notebook — two tools that take the sting out of working with Python. You experiment with IPython in a fully interactive environment. The code you place in Jupyter Notebook (also called just Notebook throughout the book) is presentation quality, and you can mix a number of presentation elements right there in your document. It's not really like using a development environment at all. To make this book easier to use on alternative platforms, you also discover an online Interactive Development Environment application (IDE) named Google Colab that allows you to interact with most, but not quite all, of the book examples using your favorite tablet or (assuming that you can squint well enough) your smart phone.

You also discover some interesting techniques in this book. For example, you can create plots of all your data science experiments using MatPlotLib, and this book gives you all the details for doing that. This book also spends considerable time showing you available resources (such as packages) and how you can use Scikit-learn to perform some really interesting calculations. Many people would like to know how to perform handwriting recognition, and if you're one of them, you can use this book to get a leg up on the process.

Of course, you might still be worried about the whole programming environment issue, and this book doesn't leave you in the dark there, either. At the beginning, you find complete installation instructions for Anaconda, which are followed by the methods you need to get started with data science using Jupyter Notebook or Google Colab. The emphasis is on getting you up and running as quickly as possible, and to make examples straightforward and simple so that the code doesn't become a stumbling block to learning.

This second edition of the book provides you with updated examples using Python 3.*x* so that you're using the most modern version of Python while reading. In addition, you find a stronger emphasis on making examples simpler, but also making the environment more inclusive by adding material on deep learning. Consequently, you get a lot more out of this edition of the book as a result of the input provided by hundreds of readers before you.

To make absorbing the concepts even easier, this book uses the following conventions:

- Text that you're meant to type just as it appears in the book is in **bold**. The exception is when you're working through a step list: Because each step is bold, the text to type is not bold.
- When you see words in *italics* as part of a typing sequence, you need to replace that value with something that works for you. For example, if you

see "Type ***Your Name*** and press Enter," you need to replace *Your Name* with your actual name.

- Web addresses and programming code appear in `monofont`. If you're reading a digital version of this book on a device connected to the Internet, note that you can click the web address to visit that website, like this: `http://www.dummies.com`.
- When you need to type command sequences, you see them separated by a special arrow, like this: File ➪ New File. In this example, you go to the File menu first and then select the New File entry on that menu.

Foolish Assumptions

You might find it difficult to believe that we've assumed anything about you — after all, we haven't even met you yet! Although most assumptions are indeed foolish, we made these assumptions to provide a starting point for the book.

You need to e familiar with the platform you want to use because the book doesn't offer any guidance in this regard. (Chapter 3 does, however, provide Anaconda installation instructions, and Chapter 4 gets you started with Google Colab.) To provide you with maximum information about Python concerning how it applies to data science, this book doesn't discuss any platform-specific issues. You really do need to know how to install applications, use applications, and generally work with your chosen platform before you begin working with this book.

You must know how to work with Python. This edition of the book no longer contains a Python primer because you can find such as wealth of tutorials online (see `https://www.w3schools.com/python/` and `https://www.tutorialspoint.com/python/` as examples).

This book isn't a math primer. Yes, you see lots of examples of complex math, but the emphasis is on helping you use Python and data science to perform analysis tasks rather than teaching math theory. Chapters 1 and 2 give you a better understanding of precisely what you need to know to use this book successfully.

This book also assumes that you can access items on the Internet. Sprinkled throughout are numerous references to online material that will enhance your learning experience. However, these added sources are useful only if you actually find and use them.

Icons Used in This Book

As you read this book, you see icons in the margins that indicate material of interest (or not, as the case may be). This section briefly describes each icon in this book.

TIP

Tips are nice because they help you save time or perform some task without a lot of extra work. The tips in this book are time-saving techniques or pointers to resources that you should try in order to get the maximum benefit from Python or in performing data science–related tasks.

WARNING

We don't want to sound like angry parents or some kind of maniacs, but you should avoid doing anything that's marked with a Warning icon. Otherwise, you might find that your application fails to work as expected, you get incorrect answers from seemingly bulletproof equations, or (in the worst-case scenario) you lose data.

TECHNICAL STUFF

Whenever you see this icon, think advanced tip or technique. You might find these tidbits of useful information just too boring for words, or they could contain the solution you need to get a program running. Skip these bits of information whenever you like.

REMEMBER

If you don't get anything else out of a particular chapter or section, remember the material marked by this icon. This text usually contains an essential process or a bit of information that you must know to work with Python or to perform data science–related tasks successfully.

Beyond the Book

This book isn't the end of your Python or data science experience — it's really just the beginning. We provide online content to make this book more flexible and better able to meet your needs. That way, as we receive e-mail from you, we can address questions and tell you how updates to either Python or its associated add-ons affect book content. In fact, you gain access to all these cool additions:

- **Cheat sheet:** You remember using crib notes in school to make a better mark on a test, don't you? You do? Well, a cheat sheet is sort of like that. It provides you with some special notes about tasks that you can do with Python, IPython, IPython Notebook, and data science that not every other person knows. You can find the cheat sheet by going to `www.dummies.com`, searching this book's title, and scrolling down the page that appears. The cheat sheet contains really neat information such as the most common programming mistakes that cause people woe when using Python.

- **Updates:** Sometimes changes happen. For example, we might not have seen an upcoming change when we looked into our crystal ball during the writing of this book. In the past, this possibility simply meant that the book became outdated and less useful, but you can now find updates to the book by searching this book's title at `www.dummies.com`.

  In addition to these updates, check out the blog posts with answers to reader questions and demonstrations of useful book-related techniques at `http://blog.johnmuellerbooks.com/`.

- **Companion files:** Hey! Who really wants to type all the code in the book and reconstruct all those plots manually? Most readers would prefer to spend their time actually working with Python, performing data science tasks, and seeing the interesting things they can do, rather than typing. Fortunately for you, the examples used in the book are available for download, so all you need to do is read the book to learn Python for data science usage techniques. You can find these files at `www.dummies.com`. Search this book's title, and on the book's product page, scroll down to the section titled, "Have This Book?" Click the link in that section called "Additional Book Resources," and you'll be redirected to the Downloads section of the book's product page on `wiley.com`.

Where to Go from Here

It's time to start your Python for data science adventure! If you're completely new to Python and its use for data science tasks, you should start with Chapter 1 and progress through the book at a pace that allows you to absorb as much of the material as possible.

If you're a novice who's in an absolute rush to get going with Python for data science as quickly as possible, you can skip to Chapter 3 with the understanding that you may find some topics a bit confusing later. Skipping to Chapter 5 is okay if you already have Anaconda (the programming product used in the book) installed, but be sure to at least skim Chapter 3 so that you know what assumptions we made when writing this book. If you plan to use your tablet to work with this book, be certain to review Chapter 4 so that you understand the limitations presented by Google Colab in running the example code; not all of the examples work in this IDE. Make sure to install Anaconda with Python version 3.6.5 installed to obtain the best results from the book's source code.

Readers who have some exposure to Python and have Anaconda installed can save reading time by moving directly to Chapter 5. You can always go back to earlier chapters as necessary when you have questions. However, you should understand how each technique works before moving to the next one. Every technique, coding example, and procedure has important lessons for you, and you could miss vital content if you start skipping too much information.

1

Getting Started with Data Science and Python

IN THIS PART . . .

Understanding how Python can make data science easier.

Defining the Python features commonly used for data science.

Creating a Python setup of your own.

Working with Google Colab on alternative devices.

IN THIS CHAPTER

» Discovering the wonders for data science

» Exploring how data science works

» Creating the connection between Python and data science

» Getting started with Python

Chapter 1 Discovering the Match between Data Science and Python

Data science may seem like one of those technologies that you'd never use, but you'd be wrong. Yes, data science involves the use of advanced math techniques, statistics, and big data. However, data science also involves helping you make smart decisions, creating suggestions for options based on previous choices, and making robots see objects. In fact, people use data science in so many different ways that you literally can't look anywhere or do anything without feeling the effects of data science on your life. In short, data science is the person behind the partition in the experience of the wonderment of technology. Without data science, much of what you accept as typical and expected today wouldn't even be possible. This is the reason that being a data scientist is the sexiest job of the twenty-first century.

REMEMBER

To make data science doable by someone who's less than a math genius, you need tools. You could use any of a number of tools to perform data science tasks, but Python is uniquely suited to making it easier to work with data science. For one thing, Python provides an incredible number of math-related libraries that help you perform tasks with a less-than-perfect understanding of precisely what is going on. However, Python goes further by supporting multiple coding styles (programming paradigms) and doing other things to make your job easier. Therefore, yes, you could use other languages to write data science applications, but Python reduces your workload, so it's a natural choice for those who really don't want to work hard, but rather to work smart.

This chapter gets you started with Python. Even though this book isn't designed to provide you with a complete Python tutorial, exploring some basic Python issues will reduce the time needed for you to get up to speed. (If you do need a good starting tutorial, please get *Beginning Programming with Python For Dummies*, 2nd Edition, by John Mueller (Wiley). You'll find that the book provides pointers to tutorials and other aids as needed to fill in any gaps that you may have in your Python education.

CHOOSING A DATA SCIENCE LANGUAGE

There are many different programming languages in the world — and most were designed to perform tasks in a certain way or even make it easier for a particular profession's work to be done with greater ease. Choosing the correct tool makes your life easier. It's akin to using a hammer to drive a screw rather than a screwdriver. Yes, the hammer works, but the screwdriver is much easier to use and definitely does a better job. Data scientists usually use only a few languages because they make working with data easier. With this in mind, here are the top languages for data science work in order of preference:

- **Python (general purpose):** Many data scientists prefer to use Python because it provides a wealth of libraries, such as NumPy, SciPy, MatPlotLib, pandas, and Scikit-learn, to make data science tasks significantly easier. Python is also a precise language that makes it easy to use multi-processing on large datasets — reducing the time required to analyze them. The data science community has also stepped up with specialized IDEs, such as Anaconda, that implement the Jupyter Notebook concept, which makes working with data science calculations significantly easier (Chapter 3 demonstrates how to use Jupyter Notebook, so don't worry about it in this chapter). Besides all of these things in Python's favor, it's also an excellent language for creating glue code with languages such as C/C++ and Fortran. The Python documentation actually shows how to create the required extensions. Most Python users rely on the language to see patterns, such as allowing a robot to see a group of pixels as an object. It also sees use for all sorts of scientific tasks.
- **R (special purpose statistical):** In many respects, Python and R share the same sorts of functionality but implement it in different ways. Depending on which source you view, Python and R have about the same number of proponents, and some people use Python and R interchangeably (or sometimes in tandem). Unlike

Python, R provides its own environment, so you don't need a third-party product such as Anaconda. However, R doesn't appear to mix with other languages with the ease that Python provides.

- **SQL (database management):** The most important thing to remember about Structured Query Language (SQL) is that it focuses on data rather than tasks. Businesses can't operate without good data management — the data is the business. Large organizations use some sort of relational database, which is normally accessible with SQL, to store their data. Most Database Management System (DBMS) products rely on SQL as their main language, and DBMS usually has a large number of data analysis and other data science features built in. Because you're accessing the data natively, there is often a significant speed gain in performing data science tasks this way. Database Administrators (DBAs) generally use SQL to manage or manipulate the data rather than necessarily perform detailed analysis of it. However, the data scientist can also use SQL for various data science tasks and make the resulting scripts available to the DBAs for their needs.
- **Java (general purpose):** Some data scientists perform other kinds of programming that require a general purpose, widely adapted and popular, language. In addition to providing access to a large number of libraries (most of which aren't actually all that useful for data science, but do work for other needs), Java supports object orientation better than any of the other languages in this list. In addition, it's strongly typed and tends to run quite quickly. Consequently, some people prefer it for finalized code. Java isn't a good choice for experimentation or ad hoc queries.
- **Scala (general purpose):** Because Scala uses the Java Virtual Machine (JVM) it does have some of the advantages and disadvantages of Java. However, like Python, Scala provides strong support for the functional programming paradigm, which uses lambda calculus as its basis (see *Functional Programmming For Dummies,* by John Mueller [Wiley] for details). In addition, Apache Spark is written in Scala, which means that you have good support for cluster computing when using this language; — think huge dataset support. Some of the pitfalls of using Scala are that it's hard to set up correctly, it has a steep learning curve, and it lacks a comprehensive set of data science specific libraries.

Defining the Sexiest Job of the 21st Century

At one point, the world viewed anyone working with statistics as a sort of accountant or perhaps a mad scientist. Many people consider statistics and analysis of data boring. However, data science is one of those occupations in which the more you learn, the more you want to learn. Answering one question often spawns more questions that are even more interesting than the one you just answered.

However, the thing that makes data science so sexy is that you see it everywhere and used in an almost infinite number of ways. The following sections provide you with more details on why data science is such an amazing field of study.

Considering the emergence of data science

Data science is a relatively new term. William S. Cleveland coined the term in 2001 as part of a paper entitled "Data Science: An Action Plan for Expanding the Technical Areas of the Field of Statistics." It wasn't until a year later that the International Council for Science actually recognized data science and created a committee for it. Columbia University got into the act in 2003 by beginning publication of the *Journal of Data Science.*

REMEMBER

However, the mathematical basis behind data science is centuries old because data science is essentially a method of viewing and analyzing statistics and probability. The first essential use of statistics as a term comes in 1749, but statistics are certainly much older than that. People have used statistics to recognize patterns for thousands of years. For example, the historian Thucydides (in his History of the Peloponnesian War) describes how the Athenians calculated the height of the wall of Platea in fifth century BC by counting bricks in an unplastered section of the wall. Because the count needed to be accurate, the Athenians took the average of the count by several solders.

The process of quantifying and understanding statistics is relatively new, but the science itself is quite old. An early attempt to begin documenting the importance of statistics appears in the ninth century when Al-Kindi wrote *Manuscript on Deciphering Cryptographic Messages.* In this paper, Al-Kindi describes how to use a combination of statistics and frequency analysis to decipher encrypted messages. Even in the beginning, statistics saw use in practical application of science to tasks that seemed virtually impossible to complete. Data science continues this process, and to some people it might actually seem like magic.

Outlining the core competencies of a data scientist

As is true of anyone performing most complex trades today, the data scientist requires knowledge of a broad range of skills to perform the required tasks. In fact, so many different skills are required that data scientists often work in teams. Someone who is good at gathering data might team up with an analyst and someone gifted in presenting information. It would be hard to find a single person with all the required skills. With this in mind, the following list describes areas in which a data scientist could excel (with more competencies being better):

- **Data capture:** It doesn't matter what sort of math skills you have if you can't obtain data to analyze in the first place. The act of capturing data begins by managing a data source using database management skills. However, raw data isn't particularly useful in many situations — you must also understand the data domain so that you can look at the data and begin formulating the sorts of questions to ask. Finally, you must have data-modeling skills so that you understand how the data is connected and whether the data is structured.
- **Analysis:** After you have data to work with and understand the complexities of that data, you can begin to perform an analysis on it. You perform some analysis using basic statistical tool skills, much like those that just about everyone learns in college. However, the use of specialized math tricks and algorithms can make patterns in the data more obvious or help you draw conclusions that you can't draw by reviewing the data alone.
- **Presentation:** Most people don't understand numbers well. They can't see the patterns that the data scientist sees. It's important to provide a graphical presentation of these patterns to help others visualize what the numbers mean and how to apply them in a meaningful way. More important, the presentation must tell a specific story so that the impact of the data isn't lost.

Linking data science, big data, and AI

Interestingly enough, the act of moving data around so that someone can perform analysis on it is a specialty called Extract, Transformation, and Loading (ETL). The ETL specialist uses programming languages such as Python to extract the data from a number of sources. Corporations tend not to keep data in one easily accessed location, so finding the data required to perform analysis takes time. After the ETL specialist finds the data, a programming language or other tool transforms it into a common format for analysis purposes. The loading process takes many forms, but this book relies on Python to perform the task. In a large, real-world operation, you might find yourself using tools such as Informatica, MS SSIS, or Teradata to perform the task.

REMEMBER

Data science isn't necessarily a means to an end; it may instead be a step along the way. As a data scientist works through various datasets and finds interesting facts, these facts may act as input for other sorts of analysis and AI applications. For example, consider that your shopping habits often suggest what books you might like or where you might like to go for a vacation. Shopping or other habits can also help others understand other, sometimes less benign, activities as well. *Machine Learning For Dummies* and *AI For Dummies*, both by John Mueller and Luca Massaron (Wiley) help you understand these other uses of data science. For now, consider the fact that what you learn in this book can have a definite effect on a career path that will go many other places.

Understanding the role of programming

A data scientist may need to know several programming languages in order to achieve specific goals. For example, you may need SQL knowledge to extract data from relational databases. Python can help you perform data loading, transformation, and analysis tasks. However, you might choose a product such as MATLAB (which has its own programming language) or PowerPoint (which relies on VBA) to present the information to others. (If you're interested to see how MATLAB compares to the use of Python, you can get my book, *MATLAB For Dummies*, published by John Wiley & Sons, Inc.) The immense datasets that data scientists rely on often require multiple levels of redundant processing to transform into useful processed data. Manually performing these tasks is time consuming and error prone, so programming presents the best method for achieving the goal of a coherent, usable data source.

Given the number of products that most data scientists use, it may not be possible to use just one programming language. Yes, Python can load data, transform it, analyze it, and even present it to the end user, but it works only when the language provides the required functionality. You may have to choose other languages to fill out your toolkit. The languages you choose depend on a number of criteria. Here are the things you should consider:

- How you intend to use data science in your code (you have a number of tasks to consider, such as data analysis, classification, and regression)
- Your familiarity with the language
- The need to interact with other languages
- The availability of tools to enhance the development environment
- The availability of APIs and libraries to make performing tasks easier

Creating the Data Science Pipeline

Data science is partly art and partly engineering. Recognizing patterns in data, considering what questions to ask, and determining which algorithms work best are all part of the art side of data science. However, to make the art part of data science realizable, the engineering part relies on a specific process to achieve specific goals. This process is the data science pipeline, which requires the data scientist to follow particular steps in the preparation, analysis, and presentation of the data. The following sections help you understand the data science pipeline better so that you can understand how the book employs it during the presentation of examples.

Preparing the data

The data that you access from various sources doesn't come in an easily packaged form, ready for analysis — quite the contrary. The raw data not only may vary substantially in format, but you may also need to transform it to make all the data sources cohesive and amenable to analysis. Transformation may require changing data types, the order in which data appears, and even the creation of data entries based on the information provided by existing entries.

Performing exploratory data analysis

The math behind data analysis relies on engineering principles in that the results are provable and consistent. However, data science provides access to a wealth of statistical methods and algorithms that help you discover patterns in the data. A single approach doesn't ordinarily do the trick. You typically use an iterative process to rework the data from a number of perspectives. The use of trial and error is part of the data science art.

Learning from data

As you iterate through various statistical analysis methods and apply algorithms to detect patterns, you begin learning from the data. The data might not tell the story that you originally thought it would, or it might have many stories to tell. Discovery is part of being a data scientist. In fact, it's the fun part of data science because you can't ever know in advance precisely what the data will reveal to you.

REMEMBER

Of course, the imprecise nature of data and the finding of seemingly random patterns in it means keeping an open mind. If you have preconceived ideas of what the data contains, you won't find the information it actually does contain. You miss the discovery phase of the process, which translates into lost opportunities for both you and the people who depend on you.

Visualizing

Visualization means seeing the patterns in the data and then being able to react to those patterns. It also means being able to see when data is not part of the pattern. Think of yourself as a data sculptor — removing the data that lies outside the patterns (the outliers) so that others can see the masterpiece of information beneath. Yes, you can see the masterpiece, but until others can see it, too, it remains in your vision alone.

Obtaining insights and data products

The data scientist may seem to simply be looking for unique methods of viewing data. However, the process doesn't end until you have a clear understanding of what the data means. The insights you obtain from manipulating and analyzing the data help you to perform real-world tasks. For example, you can use the results of an analysis to make a business decision.

In some cases, the result of an analysis creates an automated response. For example, when a robot views a series of pixels obtained from a camera, the pixels that form an object have special meaning and the robot's programming may dictate some sort of interaction with that object. However, until the data scientist builds an application that can load, analyze, and visualize the pixels from the camera, the robot doesn't see anything at all.

Understanding Python's Role in Data Science

Given the right data sources, analysis requirements, and presentation needs, you can use Python for every part of the data science pipeline. In fact, that's precisely what you do in this book. Every example uses Python to help you understand another part of the data science equation. Of all the languages you could choose for performing data science tasks, Python is the most flexible and capable because it supports so many third-party libraries devoted to the task. The following sections help you better understand why Python is such a good choice for many (if not most) data science needs.

Considering the shifting profile of data scientists

Some people view the data scientist as an unapproachable nerd who performs miracles on data with math. The data scientist is the person behind the curtain in an Oz-like experience. However, this perspective is changing. In many respects, the world now views the data scientist as either an adjunct to a developer or as a new type of developer. The ascendance of applications of all sorts that can learn is the essence of this change. For an application to learn, it has to be able to manipulate large databases and discover new patterns in them. In addition, the application must be able to create new data based on the old data — making an informed prediction of sorts. The new kinds of applications affect people in ways that would have seemed like science fiction just a few years ago. Of course, the most noticeable of these applications define the behaviors of robots that will interact far more closely with people tomorrow than they do today.

From a business perspective, the necessity of fusing data science and application development is obvious: Businesses must perform various sorts of analysis on the huge databases it has collected — to make sense of the information and use it to predict the future. In truth, however, the far greater impact of the melding of these two branches of science — data science and application development — will be felt in terms of creating altogether new kinds of applications, some of which aren't even possibly to imagine with clarity today. For example, new applications could help students learn with greater precision by analyzing their learning trends and creating new instructional methods that work for that particular student. This combination of sciences might also solve a host of medical problems that seem impossible to solve today — not only in keeping disease at bay, but also by solving problems, such as how to create truly usable prosthetic devices that look and act like the real thing.

Working with a multipurpose, simple, and efficient language

Many different ways are available for accomplishing data science tasks. This book covers only one of the myriad methods at your disposal. However, Python represents one of the few single-stop solutions that you can use to solve complex data science problems. Instead of having to use a number of tools to perform a task, you can simply use a single language, Python, to get the job done. The Python difference is the large number scientific and math libraries created for it by third parties. Plugging in these libraries greatly extends Python and allows it to easily perform tasks that other languages could perform, but with great difficulty.

TIP

Python's libraries are its main selling point; however, Python offers more than reusable code. The most important thing to consider with Python is that it supports four different coding styles:

- **Functional:** Treats every statement as a mathematical equation and avoids any form of state or mutable data. The main advantage of this approach is having no side effects to consider. In addition, this coding style lends itself better than the others to parallel processing because there is no state to consider. Many developers prefer this coding style for recursion and for lambda calculus.
- **Imperative:** Performs computations as a direct change to program state. This style is especially useful when manipulating data structures and produces elegant, but simple, code.

- **Object-oriented:** Relies on data fields that are treated as objects and manipulated only through prescribed methods. Python doesn't fully support this coding form because it can't implement features such as data hiding. However, this is a useful coding style for complex applications because it supports encapsulation and polymorphism. This coding style also favors code reuse.
- **Procedural:** Treats tasks as step-by-step iterations where common tasks are placed in functions that are called as needed. This coding style favors iteration, sequencing, selection, and modularization.

Learning to Use Python Fast

It's time to try using Python to see the data science pipeline in action. The following sections provide a brief overview of the process you explore in detail in the rest of the book. You won't actually perform the tasks in the following sections. In fact, you don't install Python until Chapter 3, so for now, just follow along in the text. This book uses a specific version of Python and an IDE called Jupyter Notebook, so please wait until Chapter 3 to install these features (or skip ahead, if you insist, and install them now). Don't worry about understanding every aspect of the process at this point. The purpose of these sections is to help you gain an understanding of the flow of using Python to perform data science tasks. Many of the details may seem difficult to understand at this point, but the rest of the book will help you understand them.

REMEMBER

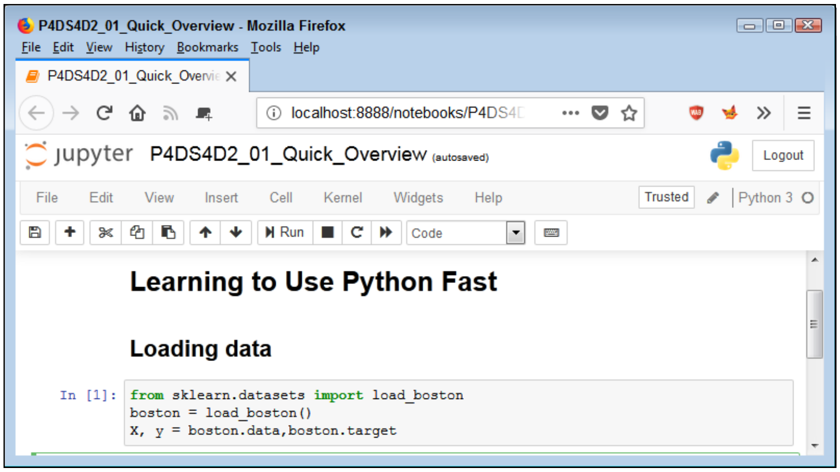
The examples in this book rely on a web-based application named Jupyter Notebook. The screenshots you see in this and other chapters reflect how Jupyter Notebook looks in Firefox on a Windows 7 system. The view you see will contain the same data, but the actual interface may differ a little depending on platform (such as using a notebook instead of a desktop system), operating system, and browser. Don't worry if you see some slight differences between your display and the screenshots in the book.

TIP

You don't have to type the source code for this chapter in by hand. In fact, it's a lot easier if you use the downloadable source (see the Introduction for details on downloading the source code). The source code for this chapter appears in the `P4DS4D2_01_Quick_Overview.ipynb` source code file.

Loading data

Before you can do anything, you need to load some data. The book shows you all sorts of methods for performing this task. In this case, Figure 1-1 shows how to load a dataset called Boston that contains housing prices and other facts about houses in the Boston area. The code places the entire dataset in the `boston` variable and then places parts of that data in variables named `X` and `y`. Think of variables as you would storage boxes. The variables are important because they make it possible to work with the data.


P4DS4D2_01_Quick_Overview - Mozilla Firefox
File Edit View History Bookmarks Tools Help
P4DS4D2_01_Quick_Overvie
localhost:8888/notebooks/P4DS4
jupyter P4DS4D2_01_Quick_Overview (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Learning to Use Python Fast
Loading data

```
In [1]: from sklearn.datasets import load_boston
        boston = load_boston()
        X, y = boston.data,boston.target
```

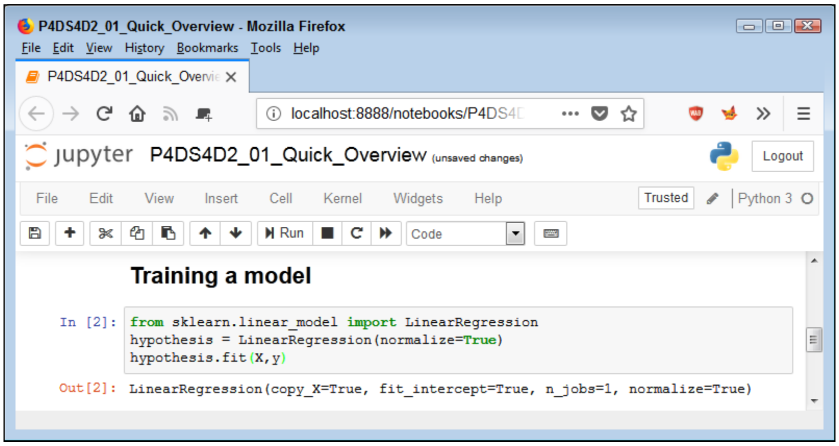
FIGURE 1-1: Loading data into variables so that you can manipulate it.

Training a model

Now that you have some data to work with, you can do something with it. All sorts of algorithms are built into Python. Figure 1-2 shows a linear regression model. Again, don't worry precisely how this works; later chapters discuss linear regression in detail. The important thing to note in Figure 1-2 is that Python lets you perform the linear regression using just two statements and to place the result in a variable named `hypothesis`.

Viewing a result

Performing any sort of analysis doesn't pay unless you obtain some benefit from it in the form of a result. This book shows all sorts of ways to view output, but Figure 1-3 starts with something simple. In this case, you see the coefficient output from the linear regression analysis.

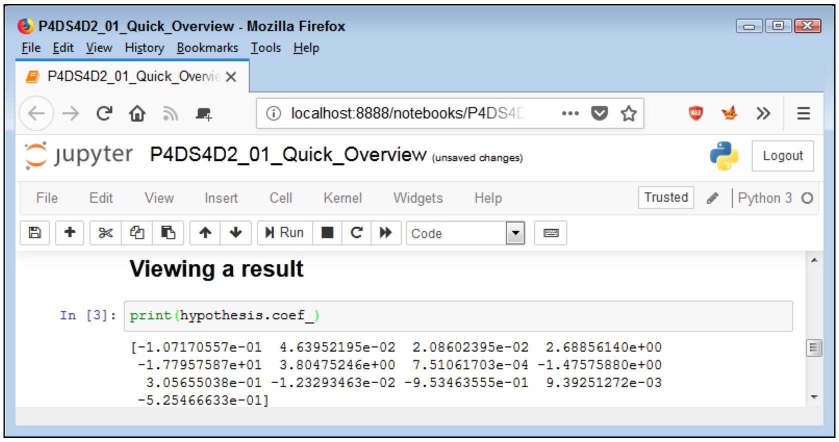


Training a model

```
In [2]: from sklearn.linear_model import LinearRegression
        hypothesis = LinearRegression(normalize=True)
        hypothesis.fit(X,y)

Out[2]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=True)
```

FIGURE 1-2: Using the variable content to train a linear regression model.



Viewing a result

```
In [3]: print(hypothesis.coef_)

        [-1.07170557e-01  4.63952195e-02  2.08602395e-02  2.68856140e+00
         -1.77957587e+01  3.80475246e+00  7.51061703e-04 -1.47575880e+00
          3.05655038e-01 -1.23293463e-02 -9.53463555e-01  9.39251272e-03
         -5.25466633e-01]
```

FIGURE 1-3: Outputting a result as a response to the model.

TIP

One of the reasons that this book uses Jupyter Notebook is that the product helps you to create nicely formatted output as part of creating the application. Look again at Figure 1-3 and you see a report that you could simply print and offer to a colleague. The output isn't suitable for many people, but those experienced with Python and data science will find it quite usable and informative.

IN THIS CHAPTER

- » Delving into why Python came about
- » Getting a quick start with Python
- » Considering Python's special features
- » Defining and exploring the power of Python for the data scientist

Chapter 2

Introducing Python's Capabilities and Wonders

All computers run on just one language — machine code. However, unless you want to learn how to talk like a computer in 0s and 1s, machine code isn't particularly useful. You'd never want to try to define data science problems using machine code. It would take an entire lifetime (if not longer) just to define one problem. Higher-level languages make it possible to write a lot of code that humans can understand quite quickly. The tools used with these languages make it possible to translate the human-readable code into machine code that the machine understands. Therefore, the choice of languages depends on the human need, not the machine need. With this in mind, this chapter introduces you to the capabilities that Python provides that make it a practical choice for the data scientist. After all, you want to know why this book uses Python and not another language, such as Java or C++. These other languages are perfectly good choices for some tasks, but they're not as suited to meet data science needs.

The chapter begins with a short history of Python so that you know a little about why developers created Python in the first place. You also see some simple Python examples to get a taste for the language. As part of exploring Python in this chapter, you discover all sorts of interesting features that Python provides. Python gives you access to a host of libraries that are especially suited to meet the needs of the data scientist. In fact, you use a number of these libraries throughout the book as you work through the coding examples. Knowing about these libraries in advance will help you understand the programming examples and why the book shows how to perform tasks in a certain way.

REMEMBER

Even though this chapter does show examples of working with Python, you don't really begin using Python in earnest until Chapter 6. This chapter provides you with an overview so that you can better understand what Python can do. Chapter 3 shows how to install the particular version of Python used for this book. Chapters 4 and 5 are about tools you can use, with Chapter 4 emphasizing Google's Colab, an alternative environment for coding. In short, if you don't quite understand an example in this chapter, don't worry: You get plenty of additional information in later chapters.

Why Python?

Python is the vision of a single person, Guido van Rossum. You might be surprised to learn that Python has been around a long time — Guido started the language in December 1989 as a replacement for the ABC language. Not much information is available as to the precise goals for Python, but it does retain ABC's ability to create applications using less code. However, it far exceeds the ability of ABC to create applications of all types, and in contrast to ABC, boasts four programming styles. In short, Guido took ABC as a starting point, found it limited, and created a new language without those limitations. It's an example of creating a new language that really is better than its predecessor.

Python has gone through a number of iterations and currently has two development paths. The 2.*x* path is backward compatible with previous versions of Python, while the 3.*x* path isn't. The compatibility issue is one that figures into how data science uses Python because a few of the libraries won't work with 3.*x*. However, this issue is slowly being resolved, and you should use 3.*x* for all new development because the end of the line is coming for the 2.*x* versions (see `https://pythonclock.org/` for details). As a result, this edition of the book uses 3.*x* code. In addition, some versions use different licensing because Guido was working at various companies during Python's development. You can see a listing of the versions and their respective licenses at `https://docs.python.org/3/license.html`. The Python Software Foundation (PSF) owns all current versions of Python, so unless you use an older version, you really don't need to worry about the licensing issue.

USING THE RIGHT LANGUAGE FOR THE JOB

Computer languages provide a means for humans to write down instructions in a systematic and understandable way. Computers don't actually understand computer languages — a computer relies on machine-code for instructions. The reason languages are so important is that humans don't characteristically understand machine language, so the conversion from something humans understand to something machines understand is essential. Python provides a specific set of features that make writing data science applications easier. As with any other language, it provides the right toolset for some situations and not for others. Use Python (or any other language) when it provides the functionality you need to accomplish a task. If you start finding the language getting in the way, it's time to choose a different language because in the end, the computer doesn't care which language you use. Computer languages are for people, not the other way around.

Grasping Python's Core Philosophy

Guido actually started Python as a skunkworks project. The core concept was to create Python as quickly as possible, yet create a language that is flexible, runs on any platform, and provides significant potential for extension. Python provides all these features and many more. Of course, there are always bumps in the road, such as figuring out just how much of the underlying system to expose. You can read more about the Python design philosophy at `http://python-history.blogspot.com/2009/01/pythons-design-philosophy.html`. The history of Python at `http://python-history.blogspot.com/2009/01/introduction-and-overview.html` also provides some useful information.

Contributing to data science

Because this is a book about data science, you're probably wondering how Python contributes toward better data science and what the word *better* actually means in this case. Knowing that a lot of organizations use Python doesn't help you because it doesn't really say much about how they use Python, and if you want to match your choice of language to your particular need, answering how they use Python becomes important.

One such example appears at `https://www.datasciencegraduateprograms.com/python/`. In this case, the article talks about Forecastwatch.com (`https://forecastwatch.com/`), which actually does watch the weather and try to make predictions better. Every day, Forecastwatch.com compares 36,000 forecasts with the actual weather that people experience and then uses the results to create better forecasts. Trying to aggregate and make sense of the weather data for 800 U.S. cities is daunting, so Forecastwatch.com needed a language that could do what it needed with the least amount of fuss. Here are the reasons Forecast.com chose Python:

- **Library support:** Python provides support for a large number of libraries, more than any one organization will ever need. According to `https://www.python.org/about/success/forecastwatch/`, Forecastwatch.com found the regular expression, thread, object serialization, and gzip data compression libraries especially useful.
- **Parallel processing:** Each of the forecasts is processed as a separate thread so that the system can work through them quickly. The thread data includes the web page URL that contains the required forecast, along with category information, such as city name.
- **Data access:** This huge amount of data can't all exist in memory, so Forecast.com relies on a MySQL database accessed through the MySQLdb (`https://sourceforge.net/projects/mysql-python/`) library, which is one of those few libraries that hasn't moved on to Python 3.*x* yet. However, the associated website promises the required support soon.
- **Data display:** Originally, PHP produced the Forecastwatch.com output. However, by using Quixote (`https://www.mems-exchange.org/software/quixote/`), which is a display framework, Forecastwatch.com was able to move everything to Python.

Discovering present and future development goals

The original development (or design) goals for Python don't quite match what has happened to the language since Guido initially thought about it. Guido originally intended Python as a second language for developers who needed to create one-off code but who couldn't quite achieve their goals using a scripting language. The original target audience for Python was the C developer. You can read about these original goals in the interview at `http://www.artima.com/intv/pyscale.html`.

You can find a number of applications written in Python today, so the idea of using it solely for scripting didn't come to fruition. In fact, you can find listings of Python applications at `https://www.python.org/about/apps/` and `https://www.python.org/about/success/`. As of this writing, Python is the fourth-ranked language in the world (see `https://www.tiobe.com/tiobe-index/`). It continues to move up the scale because developers see it as one of the best ways to create modern applications, many of which rely on data science.

Naturally, with all these success stories to go on, people are enthusiastic about adding to Python. You can find lists of Python Enhancement Proposals (PEPs) at `http://legacy.python.org/dev/peps/`. These PEPs may or may not see the light of day, but they prove that Python is a living, growing language that will continue to provide features that developers truly need to create great applications of all types, not just those for data science.

Working with Python

This book doesn't provide you with a full Python tutorial. (However, you can get a great start with *Beginning Programming with Python For Dummies,* 2nd Edition, by John Paul Mueller (Wiley). For now, it's helpful to get a brief overview of what Python looks like and how you interact with it, as in the following sections.

TIP

You don't have to type the source code for this chapter manually. You'll find using the downloadable source a lot easier (see the Introduction for details on downloading the source code). The source code for this chapter appears in the `P4DS4D2_02_Using_Python.ipynb` source code file.

Getting a taste of the language

Python is designed to provide clear language statements but to do so in an incredibly small space. A single line of Python code may perform tasks that another language usually takes several lines to perform. For example, if you want to display something on-screen, you simply tell Python to print it, like this:

```
print("Hello There!")
```

TIP

This is an example of a 3.*x* `print()` function. (The 2.*x* version of Python includes both a function form of `print`, which requires parentheses, and a statement form of `print`, which omits the parentheses.) The "Why Python?" section of this chapter mentions some differences between the 2.*x* path and the 3.*x* path. If you use the `print()` function without parentheses in 3.*x*, you get an error message:

```
    File "<Jupyter-input-1-fe18535d9681>", line 1
    print "Hello There!"
                       ^
SyntaxError: Missing parentheses in call to 'print'. Did you
   mean print("Hello There!")?
```

The point is that you can simply tell Python to output text, an object, or anything else using a simple statement. You don't really need too much in the way of advanced programming skills. When you want to end your session using a command line environment such as IDLE, you simply type `quit()` and press Enter. This book relies on a much better environment, Jupyter Notebook, which really does make your code look as though it came from someone's notebook.

Understanding the need for indentation

Python relies on indentation to create various language features, such as conditional statements. One of the most common errors that developers encounter is not providing the proper indentation for code. You see this principle in action later in the book, but for now, always be sure to pay attention to indentation as you work through the book examples. For example, here is an `if` statement (a conditional that says that if something meets the condition, perform the code that follows) with proper indentation.

```
if 1 < 2:
   print("1 is less than 2")
```

WARNING

The `print` statement must appear indented below the conditional statement. Otherwise, the condition won't work as expected, and you might see an error message, too.

UNDERSTANDING THE ANACONDA PACKAGE

The book approaches Anaconda as a product. In fact, you do install and interact with Anaconda as you would any other single product. However, Anaconda is actually a compilation of several open source applications. You can use these applications individually or in cooperation with each other to achieve specific coding goals. In most of the book, you use a single application, Jupyter Notebook, to accomplish tasks. However, you want to know about the other applications bundled in Anaconda to get the best use out of the product as a whole.

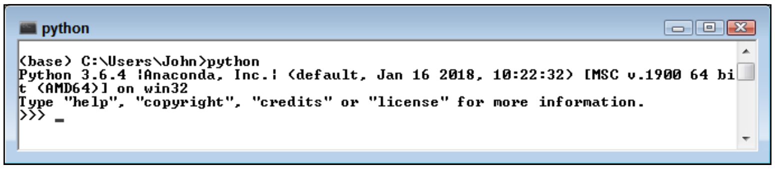
A large number of data scientists rely on the Anaconda product bundling, which is why this book uses it. However, you might find that some of the open source products come in a newer form when downloaded separately. For example, Jupyter Notebook actually comes in a newer form than found in the Anaconda bundle (`http://jupyter.org/`). Because of the differences in Jupyter Notebook versions you need to install the version of Jupyter Notebook specified for this book, which means using the Anaconda package, rather than a separate download.

Working at the command line or in the IDE

Anaconda is a product that makes using Python even easier. It comes with a number of utilities that help you work with Python in a variety of ways. The vast majority of this book relies on Jupyter Notebook, which is part of the Anaconda installation you create in Chapter 3. You saw this editor used in Chapter 1 and you see it again later in the book. In fact, this book doesn't use any of the other Anaconda utilities much at all. However, they do exist, and sometimes they're helpful in playing with Python. The following sections provide a brief overview of the other Anaconda utilities for creating Python code. You may want to experiment with them as you work through various coding techniques in the book.

Creating new sessions with Anaconda Command Prompt

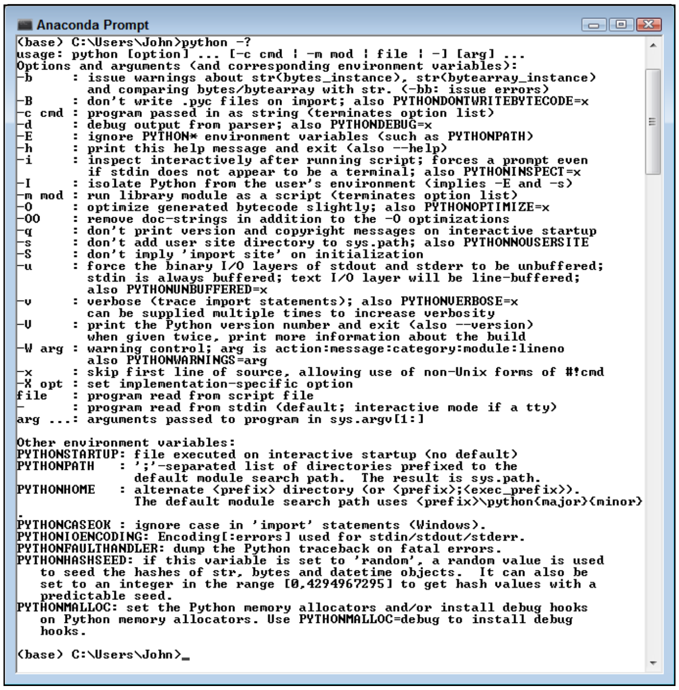
Only one of the Anaconda utilities provides direct access to the command line, Anaconda Prompt. When you start this utility, you see a command prompt at which you can type commands. The main advantage of this utility is that you can start an Anaconda utility with any of the switches it provides to modify that utility's standard environment. Of course, you start many of the utilities using the Python interpreter that you access using the `python.exe` command. (If you have both Python 3.6 and Python 2.7 installed on your system and open a regular command prompt or terminal window, you may see the Python 2.7 version start instead of the Python 3.6 version, so it's always best to open an Anaconda Command Prompt to ensure that you get the right version of Python.) So you could simply type **python** and press Enter to start a copy of the Python interpreter should you wish to do so. Figure 2-1 shows how the plain Python interpreter looks.



```
(base) C:\Users\John>python
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bi
t (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

FIGURE 2-1: A view of the plain Python interpreter.

You quit the interpreter by typing **quit()** and pressing Enter. Once back at the command line, you can discover the list of `python.exe` command-line switches by typing **python -?** and pressing Enter. Figure 2-2 shows just some of the ways in which you can change the Python interpreter environment.



Anaconda Prompt

```
(base) C:\Users\John>python -?
usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...
Options and arguments (and corresponding environment variables):
-b     : issue warnings about str(bytes_instance), str(bytearray_instance)
         and comparing bytes/bytearray with str. (-bb: issue errors)
-B     : don't write .pyc files on import; also PYTHONDONTWRITEBYTECODE=x
-c cmd : program passed in as string (terminates option list)
-d     : debug output from parser; also PYTHONDEBUG=x
-E     : ignore PYTHON* environment variables (such as PYTHONPATH)
-h     : print this help message and exit (also --help)
-i     : inspect interactively after running script; forces a prompt even
         if stdin does not appear to be a terminal; also PYTHONINSPECT=x
-I     : isolate Python from the user's environment (implies -E and -s)
-m mod : run library module as a script (terminates option list)
-O     : optimize generated bytecode slightly; also PYTHONOPTIMIZE=x
-OO    : remove doc-strings in addition to the -O optimizations
-q     : don't print version and copyright messages on interactive startup
-s     : don't add user site directory to sys.path; also PYTHONNOUSERSITE
-S     : don't imply 'import site' on initialization
-u     : force the binary I/O layers of stdout and stderr to be unbuffered;
         stdin is always buffered; text I/O layer will be line-buffered;
         also PYTHONUNBUFFERED=x
-v     : verbose (trace import statements); also PYTHONVERBOSE=x
         can be supplied multiple times to increase verbosity
-V     : print the Python version number and exit (also --version)
         when given twice, print more information about the build
-W arg : warning control; arg is action:message:category:module:lineno
         also PYTHONWARNINGS=arg
-x     : skip first line of source, allowing use of non-Unix forms of #!cmd
-X opt : set implementation-specific option
file   : program read from script file
-      : program read from stdin (default; interactive mode if a tty)
arg ...: arguments passed to program in sys.argv[1:]

Other environment variables:
PYTHONSTARTUP: file executed on interactive startup (no default)
PYTHONPATH   : ';'-separated list of directories prefixed to the
               default module search path.  The result is sys.path.
PYTHONHOME   : alternate <prefix> directory (or <prefix>;<exec_prefix>).
               The default module search path uses <prefix>\python{major}{minor}
.
PYTHONCASEOK : ignore case in 'import' statements (Windows).
PYTHONIOENCODING: Encoding[:errors] used for stdin/stdout/stderr.
PYTHONFAULTHANDLER: dump the Python traceback on fatal errors.
PYTHONHASHSEED: if this variable is set to 'random', a random value is used
   to seed the hashes of str, bytes and datetime objects.  It can also be
   set to an integer in the range [0,4294967295] to get hash values with a
   predictable seed.
PYTHONMALLOC: set the Python memory allocators and/or install debug hooks
   on Python memory allocators. Use PYTHONMALLOC=debug to install debug
   hooks.

(base) C:\Users\John>_
```

FIGURE 2-2: The Python interpreter includes all sorts of command-line switches.

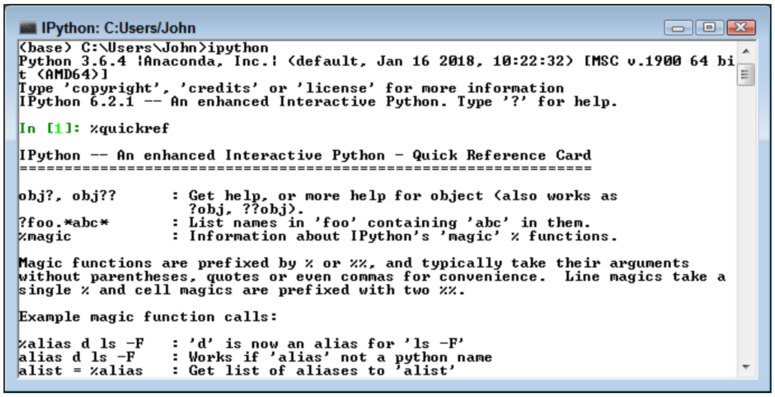
If you want, you can create a modified form of any of the utilities provided by Anaconda by starting the interpreter with the correct script. The scripts appear in the `scripts` subdirectory. For example, type **python Anaconda3/scripts/Jupyter-script.py** and press Enter to start the Jupyter environment without using the graphical command for your platform. You can also add command-line arguments to further modify the script's behavior. When working with this script, you can get information about Jupyter by using the following command-line arguments:

- `--version`: Obtains the version of Jupyter Notebook in use.
- `--config-dir`: Displays the configuration directory for Jupyter Notebook (where the configuration information is stored).
- `--data-dir`: Displays the storage location of Jupyter application data, rather than projects. Your projects generally appear in your user folder.
- `--runtime-dir`: Shows the location of the Jupyter runtime files, which is normally a subdirectory of the data directory.
- `--paths`: Creates a list of paths that Jupyter is configured to use.

Consequently, if you want to obtain a list of Jupyter paths, you type **python Anaconda3/scripts/Jupyter-script.py --paths** and press Enter. The `scripts` subdirectory also contains a wealth of executable files. Often, these files are compiled versions of scripts and may execute more quickly as a result. If you want to start the Jupyter Notebook browser environment, you can either type **python Anaconda3/scripts/Jupyter-notebook-script.py** and press Enter to use the script version or execute `Jupyter-notebook.exe`. The result is the same in either case.

Entering the IPython environment

The Interactive Python (IPython) environment provides enhancements to the standard Python interpreter. To start this environment, you use the `IPython` command, rather than the standard `Python` command, at the Anaconda Prompt. The main purpose of the environment shown in Figure 2-3 is to help you use Python with less work. Note that the Python version is the same as when using the `Python` command, but that the IPython version is different and that you see a different prompt. To see these enhancements (as shown in the figure), type **%quickref** and press Enter.



```
(base) C:\Users\John>ipython
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bi
t (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: %quickref

IPython -- An enhanced Interactive Python - Quick Reference Card
================================================================

obj?, obj??      : Get help, or more help for object (also works as
                   ?obj, ??obj).
?foo.*abc*       : List names in 'foo' containing 'abc' in them.
%magic           : Information about IPython's 'magic' % functions.

Magic functions are prefixed by % or %%, and typically take their arguments
without parentheses, quotes or even commas for convenience.  Line magics take a
single % and cell magics are prefixed with two %%.

Example magic function calls:

%alias d ls -F   : 'd' is now an alias for 'ls -F'
alias d ls -F    : Works if 'alias' not a python name
alist = %alias   : Get list of aliases to 'alist'
```

FIGURE 2-3: The Jupyter environment is easier to use than the standard Python interpreter.

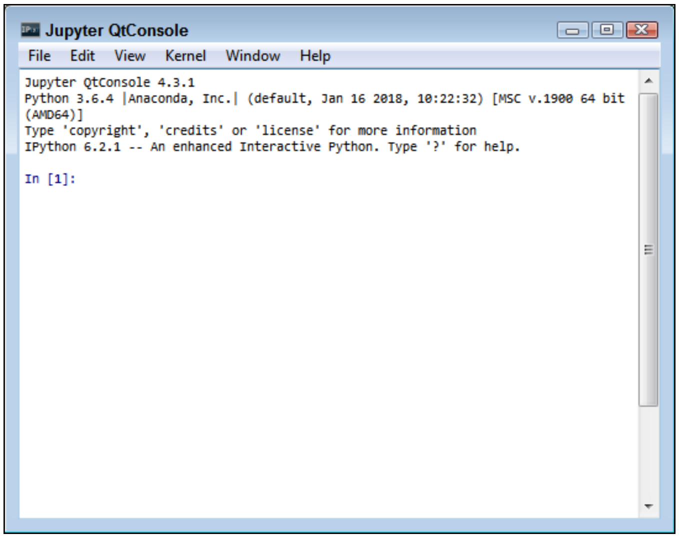
One of the more interesting additions to IPython is a fully functional clear screen (`cls`) command. You can't clear the screen easily when working in the Python interpreter, which means that things tend to get a bit messy after a while. It's also possible to perform tasks such as searching for variables using wildcard matches. Later in the book, you see how to use the magic functions to perform tasks such as capturing the amount of time it takes to perform a task for the purpose of optimization.

Entering Jupyter QTConsole environment

Trying to remember Python commands and functions is hard — and trying to remember the enhanced Jupyter additions is even harder. In fact, some people would say that the task is impossible (and perhaps they're right). This is where the Jupyter QTConsole comes into play. It adds a graphical user interface (GUI) on top of Jupyter that makes using the enhancements that Jupyter provides a lot easier.

TIP

You may think that QTConsole is missing if you used previous versions of Anaconda, but it's still present. Only the direct access method is gone. To start QTConsole, open an Anaconda Prompt, type **Jupyter QTConsole**, and press Enter. You see QTConsole start, as shown in Figure 2-4. Of course, you give up a little screen real estate to get this feature, and some hardcore programmers don't like the idea of using a GUI, so you have to choose what sort of environment to work with when programming.



```
Jupyter QtConsole 4.3.1
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bit
(AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

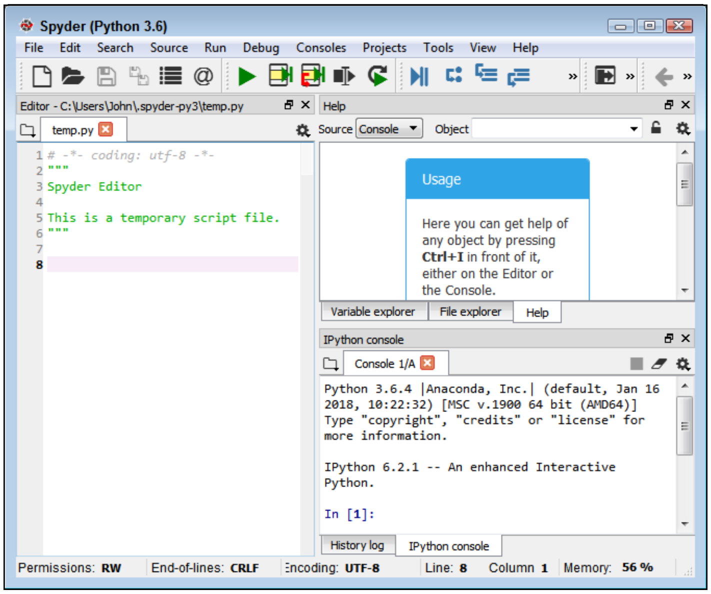
FIGURE 2-4: Use the QTConsole to make working with Jupyter easier.

Some of the enhanced commands appear in menus across the top of the window. All you need to do is choose the command you want to use. For example, to restart the kernel, you choose Kernel ➪ Restart Current Kernel. You also have the same access to IPython commands. For example, type **%magic** and press Enter to see a list of magic commands.

Editing scripts using Spyder

Spyder is a fully functional Integrated Development Environment (IDE). You use it to load scripts, edit them, run them, and perform debugging tasks. Figure 2-5 shows the default windowed environment.

The Spyder IDE is much like any other IDE that you might have used in the past. The left side contains an editor in which you type code. Any code you create is placed in a script file, and you must save the script before running it. The upper-right window contains various tabs for inspecting objects, exploring variables, and interacting with files. The lower-right window contains the Python console, a history log, and the Jupyter console. Across the top, you see menu options for performing all the tasks that you normally associate with working with an IDE.



```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

```
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16
2018, 10:22:32) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for
more information.

IPython 6.2.1 -- An enhanced Interactive
Python.

In [1]:
```

Permissions: **RW** End-of-lines: **CRLF** Encoding: **UTF-8** Line: **8** Column **1** Memory: **56 %**

FIGURE 2-5: Spyder is a traditional style IDE for developers who need one.

Performing Rapid Prototyping and Experimentation

Python is all about creating applications quickly and then experimenting with them to see how things work. The act of creating an application design in code without necessarily filling in all the details is *prototyping*. Python uses less code than other languages to perform tasks, so prototyping goes faster. The fact that many of the actions you need to perform are already defined as part of libraries that you load into memory makes things go faster still.

Data science doesn't rely on static solutions. You may have to try multiple solutions to find the particular solution that works best. This is where experimentation comes into play. After you create a prototype, you use it to experiment with various algorithms to determine which algorithm works best in a particular situation. The algorithm you use varies depending on the answers you see and the data you use, so there are too many variables to consider for any sort of canned solution.

TECHNICAL STUFF

The prototyping and experimentation process occurs in several phases. As you go through the book, you discover that these phases have distinct uses and appear in a particular order. The following list shows the phases in the order in which you normally perform them.

1. **Building a data pipeline.** To work with the data, you must create a pipeline to it. It's possible to load some data into memory. However, after the dataset gets to a certain size, you need to start working with it on disk or by using other means to interact with it. The technique you use for gaining access to the data is important because it impacts how fast you get a result.
2. **Performing the required shaping.** The shape of the data — the way in which it appears and its characteristics (such as data type), is important in performing analysis. To perform an apples-to-apples comparison, like data has to be shaped the same. However, just shaping the data the same isn't enough. The shape has to be correct for the algorithms you employ to analyze it. Later chapters (starting with Chapter 7) help you understand the need to shape data in various ways.
3. **Analyzing the data.** When analyzing data, you seldom employ a single algorithm and call it good enough. You can't know which algorithm will produce the same results at the outset. To find the best result from your dataset, you experiment on it using several algorithms. This practice is emphasized in the later chapters of the book when you start performing serious data analysis.
4. **Presenting a result.** A picture is worth a thousand words, or so they say. However, you need the picture to say the correct words or your message gets lost. Using the MATLAB-like plotting functionality provided by the `matplotlib` library, you can create multiple presentations of the same data, each of which describes the data graphically in different ways. To ensure that your meaning really isn't lost, you must experiment with various presentation methods and determine which one works best.

Considering Speed of Execution

Computers are known for their prowess in crunching numbers. Even so, analysis takes considerable processing power. The datasets are so large that you can bog down even an incredibly powerful system. In general, the following factors control the speed of execution for your data science application:

» **Dataset size:** Data science relies on huge datasets in many cases. Yes, you can make a robot see objects using a modest dataset size, but when it comes to making business decisions, larger is better in most situations. The application type determines the size of your dataset in part, but dataset size also relies on the size of the source data. Underestimating the effect of dataset size is deadly in data science applications, especially those that need to operate in real time (such as self-driving cars).