

Pandas for Everyone

The Pearson Addison-Wesley Data & Analytics Series



Visit informit.com/awdataseries for a complete list of available publications.

The **Pearson Addison-Wesley Data & Analytics Series** provides readers with practical knowledge for solving problems and answering questions with data. Titles in this series primarily focus on three areas:

1. **Infrastructure:** how to store, move, and manage data
2. **Algorithms:** how to mine intelligence or make predictions based on data
3. **Visualizations:** how to represent data and insights in a meaningful and compelling way

The series aims to tie all three of these areas together to help the reader build end-to-end systems for fighting spam; making recommendations; building personalization; detecting trends, patterns, or problems; and gaining insight from the data exhaust of systems and user interactions.



Make sure to connect with us!
informit.com/connect

Pandas for Everyone

Python Data Analysis

Second Edition

Daniel Y. Chen

◆ Addison-Wesley

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Cover image: SkillUp / Shutterstock
Figure 3.7: The Matplotlib development team
Figure B1 (Appendix): GitHub, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2022948110

Copyright © 2023 Pearson Education, Inc.

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearson.com/permissions.

ISBN-13: 978-0-13-789115-3

ISBN-10: 0-13-789115-6

ScoutAutomatedPrintCode

Pearson's Commitment to Diversity, Equity, and Inclusion

Pearson is dedicated to creating bias-free content that reflects the diversity of all learners. We embrace the many dimensions of diversity, including but not limited to race, ethnicity, gender, socioeconomic status, ability, age, sexual orientation, and religious or political beliefs.

Education is a powerful force for equity and change in our world. It has the potential to deliver opportunities that improve lives and enable economic mobility. As we work with authors to create content for every product and service, we acknowledge our responsibility to demonstrate inclusivity and incorporate diverse scholarship so that everyone can achieve their potential through learning. As the world's leading learning company, we have a duty to help drive change and live up to our purpose to help more people create a better life for themselves and to create a better world.

Our ambition is to purposefully contribute to a world where:

- Everyone has an equitable and lifelong opportunity to succeed through learning.
- Our educational products and services are inclusive and represent the rich diversity of learners.
- Our educational content accurately reflects the histories and experiences of the learners we serve.
- Our educational content prompts deeper discussions with learners and motivates them to expand their own learning (and worldview).

While we work hard to present unbiased content, we want to hear from you about any concerns or needs with this Pearson product so that we can investigate and address them.

- Please contact us with concerns about any potential bias at <https://www.pearson.com/report-bias.html>.

This page intentionally left blank



*To all the teachers, advisors, and mentors I've had over the years.
And to my family: Mom, Dad, Eric, and Julia*



This page intentionally left blank

Contents

Foreword to Second Edition xxiii

Foreword to First Edition xxv

Preface xxvii

Breakdown of the Book xxviii

Part I xxviii

Part II xxix

Part III xxix

Part IV xxix

Part V xxx

Appendices xxx

How to Read This Book xxx

Newcomers xxx

Fluent Python Programmers xxxi

Instructors xxxi

Setup xxxi

Get the Data xxxi

Set up Python xxxi

Feedback, Please! xxxii

Acknowledgments xxxiii

About the Author xxxvii

Changes in the Second Edition xxxix

I Introduction 1

1 Pandas DataFrame Basics 3

1.1 Introduction 3

Learning Objectives 3

1.2 Load Your First Data Set 4

1.3 Look at Columns, Rows, and Cells 6

1.3.1 Select and Subset Columns
by Name 7

1.3.2 Subset Rows 11

1.3.3 Subset Rows by Row Number:
.iloc[] 13

1.3.4 Mix It Up 15

- 1.3.5 Subsetting Rows and Columns 21
- 1.4 Grouped and Aggregated Calculations 23
 - 1.4.1 Grouped Means 23
 - 1.4.2 Grouped Frequency Counts 27
- 1.5 Basic Plot 27
- Conclusion 28

2 Pandas Data Structures Basics 31

- Learning Objectives 31
- 2.1 Create Your Own Data 31
 - 2.1.1 Create a Series 31
 - 2.1.2 Create a DataFrame 32
- 2.2 The Series 33
 - 2.2.1 The Series Is ndarray-like 35
 - 2.2.2 Boolean Subsetting: Series 36
 - 2.2.3 Operations Are Automatically Aligned and Vectorized (Broadcasting) 39
- 2.3 The DataFrame 42
 - 2.3.1 Parts of a DataFrame 42
 - 2.3.2 Boolean Subsetting: DataFrames 43
 - 2.3.3 Operations Are Automatically Aligned and Vectorized (Broadcasting) 44
- 2.4 Making Changes to Series and DataFrames 45
 - 2.4.1 Add Additional Columns 45
 - 2.4.2 Directly Change a Column 47
 - 2.4.3 Modifying Columns with `.assign()` 50
 - 2.4.4 Dropping Values 52
- 2.5 Exporting and Importing Data 52
 - 2.5.1 Pickle 53
 - 2.5.2 Comma-Separated Values (CSV) 55

2.5.3	Excel	55
2.5.4	Feather	56
2.5.5	Arrow	58
2.5.6	Dictionary	58
2.5.7	JSON (JavaScript Objectd Notation)	59
2.5.8	Other Data Output Types	62
Conclusion		63

3 Plotting Basics 65

Learning Objectives		65
3.1	Why Visualize Data?	65
3.2	Matplotlib Basics	66
3.2.1	Figure Objects and Axes Subplots	67
3.2.2	Anatomy of a Figure	71
3.3	Statistical Graphics Using <code>matplotlib</code>	72
3.3.1	Univariate (Single Variable)	73
3.3.2	Bivariate (Two Variables)	74
3.3.3	Multivariate Data	76
3.4	Seaborn	78
3.4.1	Univariate	79
3.4.2	Bivariate Data	83
3.4.3	Multivariate Data	94
3.4.4	Facets	99
3.4.5	Seaborn Styles and Themes	105
3.4.6	How to Go Through Seaborn Documentation	109
3.4.7	Next-Generation Seaborn Interface	110
3.5	Pandas Plotting Method	111
3.5.1	Histogram	111
3.5.2	Density Plot	111
3.5.3	Scatter Plot	112

- 3.5.4 Hexbin Plot 113
- 3.5.5 Box Plot 114
- Conclusion 115

4 Tidy Data 117

- Learning Objectives 117
 - Note About This Chapter 117
- 4.1 Columns Contain Values, Not Variables 118
 - 4.1.1 Keep One Column Fixed 118
 - 4.1.2 Keep Multiple Columns Fixed 120
- 4.2 Columns Contain Multiple Variables 122
 - 4.2.1 Split and Add Columns Individually 123
 - 4.2.2 Split and Combine in a Single Step 125
- 4.3 Variables in Both Rows and Columns 126
- Conclusion 129

5 Apply Functions 131

- Learning Objectives 131
 - Note About This Chapter 131
- 5.1 Primer on Functions 131
- 5.2 Apply (Basics) 133
 - 5.2.1 Apply Over a Series 133
 - 5.2.2 Apply Over a DataFrame 135
- 5.3 Vectorized Functions 138
 - 5.3.1 Vectorize with NumPy 140
 - 5.3.2 Vectorize with Numba 140
- 5.4 Lambda Functions (Anonymous Functions) 141
- Conclusion 142

II Data Processing 143

6 Data Assembly 145

- Learning Objectives 145

6.1	Combine Data Sets	145
6.2	Concatenation	146
6.2.1	Review Parts of a DataFrame	146
6.2.2	Add Rows	147
6.2.3	Add Columns	150
6.2.4	Concatenate with Different Indices	151
6.3	Observational Units Across Multiple Tables	154
6.3.1	Load Multiple Files Using a Loop	157
6.3.2	Load Multiple Files Using a List Comprehension	158
6.4	Merge Multiple Data Sets	160
6.4.1	One-to-One Merge	162
6.4.2	Many-to-One Merge	163
6.4.3	Many-to-Many Merge	163
6.4.4	Check Your Work with Assert	166
	Conclusion	167
7	Data Normalization	169
	Learning Objectives	169
7.1	Multiple Observational Units in a Table (Normalization)	169
	Conclusion	173
8	Groupby Operations: Split-Apply-Combine	175
	Learning Objectives	175
8.1	Aggregate	176
8.1.1	Basic One-Variable Grouped Aggregation	176
8.1.2	Built-In Aggregation Methods	178
8.1.3	Aggregation Functions	179
8.1.4	Multiple Functions Simultaneously	182
8.1.5	Use a dict in <code>.agg()</code> / <code>.aggregate()</code>	182

- 8.2 Transform 184
 - 8.2.1 Z-Score Example 184
 - 8.2.2 Missing Value Example 186
- 8.3 Filter 188
- 8.4 The `pandas.core.groupby.DataFrameGroupBy` object 190
 - 8.4.1 Groups 190
 - 8.4.2 Group Calculations Involving Multiple Variables 191
 - 8.4.3 Selecting a Group 191
 - 8.4.4 Iterating Through Groups 192
 - 8.4.5 Multiple Groups 194
 - 8.4.6 Flattening the Results (`.reset_index()`) 194
- 8.5 Working With a MultiIndex 195
- Conclusion 199

III Data Types 201

9 Missing Data 203

- Learning Objectives 203
- 9.1 What Is a NaN Value? 203
- 9.2 Where Do Missing Values Come From? 205
 - 9.2.1 Load Data 205
 - 9.2.2 Merged Data 206
 - 9.2.3 User Input Values 207
 - 9.2.4 Reindexing 209
- 9.3 Working With Missing Data 210
 - 9.3.1 Find and Count Missing Data 210
 - 9.3.2 Clean Missing Data 212
 - 9.3.3 Calculations With Missing Data 215
- 9.4 Pandas Built-In NA Missing 216
- Conclusion 218

10 Data Types 219

Learning Objectives 219

10.1 Data Types 219

10.2 Converting Types 220

10.2.1 Converting to String
Objects 22010.2.2 Converting to Numeric
Values 221

10.3 Categorical Data 225

10.3.1 Convert to Category 225

10.3.2 Manipulating Categorical
Data 226

Conclusion 227

11 Strings and Text Data 229

Introduction 229

Learning Objectives 229

11.1 Strings 229

11.1.1 Subset and Slice
Strings 22911.1.2 Get the Last Character in a
String 231

11.2 String Methods 233

11.3 More String Methods 234

11.3.1 Join 234

11.3.2 Splitlines 235

11.4 String Formatting (F-Strings) 236

11.4.1 Formatting Numbers 238

11.5 Regular Expressions (RegEx) 239

11.5.1 Match a Pattern 240

11.5.2 Remember What Your RegEx
Patterns Are 243

11.5.3 Find a Pattern 244

11.5.4 Substitute a Pattern 245

11.5.5 Compile a Pattern 246

11.6 The regex Library 247

Conclusion 247

12 Dates and Times 249

- Learning Objectives 249
- 12.1 Python's `datetime` Object 249
- 12.2 Converting to `datetime` 250
- 12.3 Loading Data That Include Dates 253
- 12.4 Extracting Date Components 254
- 12.5 Date Calculations and `Timedeltas` 257
- 12.6 `Datetime` Methods 259
- 12.7 Getting Stock Data 261
- 12.8 Subsetting Data Based on Dates 263
 - 12.8.1 The `DatetimeIndex` Object 263
 - 12.8.2 The `TimedeltaIndex` Object 265
- 12.9 Date Ranges 266
 - 12.9.1 Frequencies 268
 - 12.9.2 Offsets 268
- 12.10 Shifting Values 270
- 12.11 Resampling 276
- 12.12 Time Zones 278
- 12.13 Arrow for Better Dates and Times 280
- Conclusion 280

IV Data Modeling 281

13 Linear Regression (Continuous Outcome Variable) 283

- 13.1 Simple Linear Regression 283
 - 13.1.1 With `statsmodels` 284
 - 13.1.2 With `scikit-learn` 285
- 13.2 Multiple Regression 287
 - 13.2.1 With `statsmodels` 287
 - 13.2.2 With `scikit-learn` 288
- 13.3 Models with Categorical Variables 289
 - 13.3.1 Categorical Variables in `statsmodels` 289
 - 13.3.2 Categorical Variables in `scikit-learn` 291

- 13.4 One-Hot Encoding in scikit-learn with
Transformer Pipelines 294

Conclusion 296

14 Generalized Linear Models 297

About This Chapter 297

- 14.1 Logistic Regression (Binary Outcome
Variable) 297

- 14.1.1 With `statsmodels` 299

- 14.1.2 With `sklearn` 300

- 14.1.3 Be Careful of scikit-learn
Defaults 302

- 14.2 Poisson Regression (Count Outcome
Variable) 304

- 14.2.1 With `statsmodels` 304

- 14.2.2 Negative Binomial
Regression for
Overdispersion 306

- 14.3 More Generalized Linear Models 308

Conclusion 309

15 Survival Analysis 311

- 15.1 Survival Data 311

- 15.2 Kaplan Meier Curves 312

- 15.3 Cox Proportional Hazard Model 314

- 15.3.1 Testing the Cox Model
Assumptions 315

Conclusion 317

16 Model Diagnostics 319

- 16.1 Residuals 319

- 16.1.1 Q-Q Plots 322

- 16.2 Comparing Multiple Models 324

- 16.2.1 Working with Linear
Models 324

- 16.2.2 Working with GLM
Models 327

- 16.3 k -Fold Cross-Validation 329

Conclusion 334

17 Regularization 335

- 17.1 Why Regularize? 335
- 17.2 LASSO Regression 337
- 17.3 Ridge Regression 338
- 17.4 Elastic Net 340
- 17.5 Cross-Validation 341
- Conclusion 343

18 Clustering 345

- 18.1 *k*-Means 345
 - 18.1.1 Dimension Reduction with PCA 347
- 18.2 Hierarchical Clustering 351
 - 18.2.1 Complete Clustering 352
 - 18.2.2 Single Clustering 352
 - 18.2.3 Average Clustering 353
 - 18.2.4 Centroid Clustering 353
 - 18.2.5 Ward Clustering 354
 - 18.2.6 Manually Setting the Threshold 355
- Conclusion 356

V Conclusion 357

19 Life Outside of Pandas 359

- 19.1 The (Scientific) Computing Stack 359
- 19.2 Performance 360
 - 19.2.1 Timing Your Code 360
 - 19.2.2 Profiling Your Code 360
 - 19.2.3 Concurrent Futures 360
- 19.3 Dask 360
- 19.4 Siuba 360
- 19.5 Ibis 361
- 19.6 Polars 361
- 19.7 PyJanitor 361
- 19.8 Pandera 361
- 19.9 Machine Learning 361
- 19.10 Publishing 362
- 19.11 Dashboards 362
- Conclusion 362

20 It's Dangerous To Go Alone! 363

- 20.1 Local Meetups 363
- 20.2 Conferences 363
- 20.3 The Carpentries 364
- 20.4 Podcasts 364
- 20.5 Other Resources 365
- Conclusion 365

Appendices 367**A Concept Maps 369****B Installation and Setup 373**

- B.1 Install Python 373
 - B.1.1 Anaconda 373
 - B.1.2 Miniconda 374
 - B.1.3 Uninstall Anaconda or Miniconda 374
 - B.1.4 Pyenv 374
- B.2 Install Python Packages 374
- B.3 Download Book Data 375

C Command Line 377

- C.1 Installation 377
 - C.1.1 Windows 377
 - C.1.2 Mac 377
 - C.1.3 Linux 378
- C.2 Basics 378

D Project Templates 379**E Using Python 381**

- E.1 Command Line and Text Editor 381
- E.2 Python and IPython 381
- E.3 Jupyter 382
- E.4 Integrated Development Environments (IDEs) 382

- F Working Directories 383**

- G Environments 385**
 - G.1 Conda Environments 385
 - G.2 Pyenv + Pipenv 387

- H Install Packages 389**
 - H.1 Updating Packages 390

- I Importing Libraries 391**

- J Code Style 393**
 - J.1 Line Breaks in Code 393

- K Containers: Lists, Tuples, and Dictionaries 395**
 - K.1 Lists 395
 - K.2 Tuples 396
 - K.3 Dictionaries 396

- L Slice Values 399**

- M Loops 401**

- N Comprehensions 403**

- O Functions 405**
 - O.1 Default Parameters 407
 - O.2 Arbitrary Parameters 407
 - O.2.1 *args 408
 - O.2.2 **kwargs 408

- P Ranges and Generators 409**

- Q Multiple Assignment 413**

R Numpy ndarray 415**S Classes 417****T SettingWithCopyWarning 419**

- T.1 Modifying a Subset of Data 419
- T.2 Replacing a Value 420
- T.3 More Resources 422

U Method Chaining 423**V Timing Code 427****W String Formatting 429**

- W.1 C-Style 429
- W.2 String Formatting: `.format()` Method 429
- W.3 Formatting Numbers 430

X Conditionals (if-elif-else) 433**Y New York ACS Logistic Regression Example 435**

- Y.0.1 With sklearn 439

Z Replicating Results in R 443

- Z.1 Linear Regression 444
- Z.2 Logistic Regression 446
- Z.3 Poisson Regression 447
 - Z.3.1 Negative Binomial Regression for Overdispersion 448

Index 451

This page intentionally left blank

Foreword to Second Edition

As the data science domain and educational landscape continues to evolve, there is an increasing need to train individuals to critically consider data both holistically and logically. Each year, given the advancement in computational power, magnitude of data, and data-informed decisions to make, more and more individuals are dipping their toes in the water of data science—and most are not aware of how messy their data sets are. Working with messy data is challenging, confusing, and not necessarily exciting, especially for newcomers. To continue to use data for informed decision-making, it is important to introduce concepts in data logic, planning, and purpose early in the stages of training best practices. The how, why, and lessons learned of teaching data science represent huge areas of exploration given the exponential increase in learners. There are numerous resources, MOOCs, Twitter threads, packages, cheat-sheets, and more out there for individuals to learn data science, either on their own or in a class. However, what is effective and what pathways are best for certain learner personas? Moreover, how does someone new to the field choose which educational resources mesh with their needs and background familiarity?

While spending many years as an educator for RStudio and The Carpentries, Dr. Daniel Chen recognized this need, and it has become his passion to introduce learners to core concepts to work with their data in more effective, reproducible, and reliable methods in an environment matching their comfort level with the field. I met Dan by semi-random chance and after a few conversations, we were well on our way with a dissertation topic stemming from these interests. With a shared passion in educating others in foundational data science methods and looking into those “hows” and “whys” of the ways in which we were teaching, we sought to understand our learners first and then create materials. It was a pleasure to work with Dan on his dissertation—and to see those insights incorporated here in *Pandas for Everyone, Second Edition*.

In the second edition, Dan takes learners step-by-step through practical scratch code examples for using Pandas. Using Pandas helps demystify Python data analysis, create organized manageable data sets, and, most importantly, have tidy data sets! It takes a special educator to get individuals (myself included!) excited about cleaning data, but that is what Dan does for his learners in *Pandas for Everyone*. Visualizing and modeling data are taught in easy-to-interpret style once learners become comfortable with manipulating and transforming their data sets, all of which is covered in sequential order. It is this mindset and presentation of materials that really makes this book for everyone—and aids the

learner in best practices while working with example data sets that mimic data sets they might use in real life. *Pandas for Everyone, Second Edition*, is a quick but detailed foray for new data scientists, instructors, and more to experience best practices and the massive potential of Pandas in a clear-cut format.

*—Anne M. Brown, PhD (she/her)
Assistant Professor
Data Services—University Libraries
Department of Biochemistry
Virginia Tech, Blacksburg, VA 24061*

Foreword to First Edition

With each passing year data becomes more important to the world, as does the ability to compute on this growing abundance of data. When deciding how to interact with data, most people make a decision between R and Python. This does not reflect a language war, but rather a luxury of choice where data scientists and engineers can work in the language with which they feel most comfortable. These tools make it possible for everyone to work with data for machine learning and statistical analysis. That is why I am happy to see what I started with *R for Everyone* extended to Python with *Pandas for Everyone*.

I first met Dan Chen when he stumbled into the “Introduction to Data Science” course while working toward a master’s in public health at Columbia University’s Mailman School of Public Health. He was part of a cohort of MPH students who cross-registered into the graduate school course and quickly developed a knack for data science, embracing statistical learning and reproducibility. By the end of the semester he was devoted to, and evangelizing, the merits of data science.

This coincided with the rise of Pandas, improving Python’s use as a tool for data science and enabling engineers already familiar with the language to use it for data science as well. This fortuitous timing meant Dan developed into a true multilingual data scientist, mastering both R and Pandas. This puts him in a great position to reach different audiences, as shown by his frequent and popular talks at both R and Python conferences and meetups. His enthusiasm and knowledge shine through and resonate in everything he does, from educating new users to building Python libraries. Along the way he fully embraces the ethos of the open-source movement.

As the name implies, this book is meant for everyone who wants to use Python for data science, whether they are veteran Python users, experienced programmers, statisticians, or entirely new to the field. For people brand new to Python the book contains a collection of appendixes for getting started with the language and for installing both Python and Pandas, and it covers the whole analysis pipeline, including reading data, visualization, data manipulation, modeling, and machine learning.

Pandas for Everyone is a tour of data science through the lens of Python, and Dan Chen is perfectly suited to guide that tour. His mixture of academic and industry experience lends valuable insights into the analytics process and how Pandas should be used to greatest effect. All this combines to make for an enjoyable and informative read for everyone.

—Jared Lander, series editor

This page intentionally left blank

Preface

My foray into teaching was in 2013 when I attended my first Software–Carpentry workshop, and I’ve been involved in teaching ever since. In 2019, I was lucky enough to be one of the RStudio (now Posit, PBC) interns with the education group. By then, data science education has already gained a tremendous amount of momentum. When I finished my internship, I needed a dissertation topic for my degree, and wanted to combine teaching with medicine. Luckily, I knew a librarian at the university, Andi Ogier, who connected me with Anne Brown, who was also interested in teaching data literacy skills in the health sciences. The rest is history. Anne became my PhD chair, and with the rest of my committee, Dave Higdon, Alex Hanlon, and Nikki Lewis, I got to do research on data science education in the medical and biomedical sciences.¹ The first edition of the book became a foundation for what data science topics were taught for the workshop component of the dissertation. The second edition of *Pandas for Everyone* incorporates many of the things I’ve learned while studying education and pedagogy.

Long story short, befriend a librarian. Their profession revolves around data.

In 2013, I didn’t even know the term “data science” existed. I was a master’s of public health (MPH) student in epidemiology at the time and was already captivated with the statistical methods beyond the *t*-test, ANOVA, and linear regression from my psychology and neuroscience undergraduate background. It was also in the fall of 2013 that I attended my first Software–Carpentry workshop and that I taught my first recitation section as a teaching assistant for my MPH program’s Quantitative Methods course (essentially a combination of a first-semester epidemiology and biostatistics course). I’ve been learning and teaching ever since.

I’ve come a long way since taking my first Introduction to Data Science course, which was taught by Rachel Schutt, PhD; Kayur Patel, PhD; and Jared Lander. They opened my eyes to what was possible. Things that were inconceivable (to me) were actually common practices, and anything I could think of was possible (although I now know that “possible” doesn’t mean “performs well”). The technical details of data science—the coding aspects—were taught by Jared in R. Jared’s friends and colleagues know how much of an aficionado he is of the R language.

At the time, I had been meaning to learn R, but the Python/R language war never breached my consciousness. On the one hand, I saw Python as just a programming language; on the other hand, I had no idea Python had an analytics stack (I’ve come a long way since then). When I learned about the SciPy stack and Pandas, I saw it as a bridge between what I knew how to do in Python from my undergraduate and high school days and what I had learned in my epidemiology studies and through my newly acquired data

1. You can learn more about my dissertation around data science education here: <https://github.com/chendaniely/dissertation>

science knowledge. As I became more proficient in R, I saw the similarities to Python. I also realized that a lot of the data cleaning tasks (and programming in general) involve thinking about how to get what you need—the rest is more or less syntax. It's important to try to imagine what the steps are and not get bogged down by the programming details. I've always been comfortable bouncing around the languages and never gave too much thought to which language was “better.” Having said that, this book is geared toward a newcomer to the Python data analytics world.

This book encapsulates all the people I've met, events I've attended, and skills I've learned over the past few years. One of the more important things I've learned (outside of knowing what things are called so Google can take me to the relevant StackOverflow page) is that reading the documentation is essential. As someone who has worked on collaborative lessons and written Python and R libraries, I can assure you that a lot of time and effort go into writing documentation. That's why I constantly refer to the relevant documentation page throughout this book. Some functions have so many parameters used for varying use cases that it's impractical to go through each of them. If that were the focus of this book, it might as well be titled *Loading Data Into Python*. But, as you practice working with data and become more comfortable with the various data structures, you'll eventually be able to make educated guesses about what the output of something will be, even though you've never written that particular line of code before. I hope this book gives you a solid foundation to explore on your own and be a self-guided learner.

I met a lot of people and learned a lot from them during the time I was putting this book together. A lot of the things I learned dealt with best practices, writing vectorized statements instead of loops, formally testing code, organizing project folder structures, and so on. I also learned a lot about teaching from actually teaching. Teaching really is the best way to learn material. Many of the things I've learned in the past few years have come to me when I was trying to figure them out to teach others. Once you have a basic foundation of knowledge, learning the next bit of information is relatively easy. Repeat the process enough times, and you'll be surprised how much you actually know. That includes knowing the terms to use for Google and interpreting the StackOverflow answers. The very best of us all search for our questions. Whether this is your first language or your fourth, I hope this book gives you a solid foundation to build upon and learn as well as a bridge to other analytics languages.

Breakdown of the Book

This book is organized into multiple parts plus a set of appendices.

Part I

Part I aims to be an introduction to Pandas using a realistic data set.

- Chapter 1: Starts by using Pandas to load a data set and begin looking at various rows and columns of the data. Here you will get a general sense of the syntax of Python and Pandas. The chapter ends with a series of motivating examples that illustrate what Pandas can do.

- Chapter 2: Dives deeper into what the Pandas `'DataFrame'` and `'Series'` objects are. This chapter also covers boolean subsetting, dropping values, and different ways to import and export data.
- Chapter 3: Covers plotting methods using `'matplotlib'`, `'seaborn'`, and `'pandas'` to create plots for exploratory data analysis.
- Chapter 4: Discusses Hadley Wickham's "Tidy Data" paper, which deals with reshaping and cleaning common data problems.
- Chapter 5: Focuses on applying functions over data, an important skill that encompasses many programming topics. Understanding how `'.apply()'` works will pave the way for more parallel and distributed coding when your data manipulations need to scale.

Part II

Part II focuses on what happens after you load data and need to further process your data.

- Chapter 6: Focuses on combining data sets, either by concatenating them together or by merging disparate data.
- Chapter 7: Normalizes data for more robust data storage.
- Chapter 8: Describes `'.groupby()'` operations (i.e., split-apply-combine). These powerful concepts, like `'.apply()'`, are often needed to scale data. They are also great ways to efficiently aggregate, transform, or filter your data.

Part III

Part III covers the types of data stored in columns.

- Chapter 9: Covers what happens when there is missing data, how data are created to fill in missing data, and how to work with missing data, especially what happens when certain calculations are performed on them.
- Chapter 10: Deals with data types and how to convert from different types within `'DataFrame'` columns.
- Chapter 11: Introduces string manipulation, which is frequently needed as part of the data cleaning task because data are often encoded as text.
- Chapter 12: Explores Pandas's powerful date and time capabilities.

Part IV

With the data all cleaned and ready, the next step is to fit some models. Models can be used for exploratory purposes, not just for prediction, clustering, and inference. The goal of Part IV is not to teach statistics (there are plenty of books in that realm), but rather to show you how these models are fit and how they interface with Pandas. Part IV can be used as a bridge to fitting models in other languages.

- Chapter 13: Linear models are the simpler models to fit. This chapter covers fitting these models using the `'statsmodels'` and `'sklearn'` libraries.
- Chapter 14: Generalized linear models, as the name suggests, are linear models specified in a more general sense. They allow us to fit models with different response variables, such as binary data or count data.

- Chapter 15: Covers survival models, which is what you use when you have data censoring.
- Chapter 16: Since we have a core set of models that we can fit, the next step is to perform some model diagnostics to compare multiple models and pick the “best” one.
- Chapter 17: Regularization is a technique used when the models we are fitting are too complex or overfit our data.
- Chapter 18: Clustering is a technique we use when we don’t know the actual answer within our data, but we need a method to cluster or group “similar” data points together.

Part V

The book concludes with a few points about the larger Python ecosystem, and additional references.

- Chapter 19: Quickly summarizes the computation stack in Python, and starts down the path to code performance and scaling.
- Chapter 20: Provides some links and references on learning beyond the book.

Appendices

The appendices can be thought as a primer to Python programming. While they are not a complete introduction to Python, the various appendixes do supplement some of the topics throughout the book.

- Appendix A: Provides concept maps for the introductory chapters to help breakdown and relate concepts to one another.
- Appendixes B–J: These appendixes cover all the tasks related to running Python code—from installing Python, to using the command line to execute your scripts, and to organizing your code. They also cover creating Python environments and installing libraries.
- Appendixes K–Y: These appendixes cover general programming concepts that are relevant to Python and Pandas. They are supplemental references to the main part of the book.
- Appendix Z: Replicates some of the modeling code in R as a reference to compare similar results.

How to Read This Book

Whether you are a newcomer to Python or a fluent Python programmer, this book is meant to be read from the beginning. Educators, or people who plan to use the book for teaching, may also find the order of the chapters to be suitable for a workshop or class.

Newcomers

Absolute newcomers are encouraged to first look through Appendix A – Appendix J as they explain how to install Python and get it working. After taking these steps, readers will be ready to jump into the main body of the book. The earlier chapters make references to

the relevant appendixes as needed. The concept maps and learning objectives found at the beginning of the earlier chapters help organize and prepare the reader for what will be covered in the chapter, as well as point to the relevant appendixes to be read before continuing.

Fluent Python Programmers

Fluent Python programmers may find the first two chapters to be sufficient to get started and grasp the syntax of Pandas; they can then use the rest of the book as a reference. The objectives at the beginning of the earlier chapters point out which topics are covered in the chapter. The chapter on “tidy data” in Part I, and the chapters in Part III, will be particularly helpful in data manipulation.

Instructors

Instructors who want to use the book as a teaching reference may teach each chapter in the order presented. It should take approximately 45 minutes to 1 hour to teach each chapter. I have sought to structure the book so that chapters do not reference future chapters, so as to minimize the cognitive overload for students—but feel free to shuffle the chapters as needed.

The concept maps in Appendix A and the learning objectives provided in the earlier chapters should help contextualize how concepts are related to one another.

Setup

Everyone will have a different setup, so the best way to get the most updated set of instructions on setting up an environment to code through the book would be on the accompanying GitHub repository:

https://github.com/chendaniely/pandas_for_everyone

Otherwise, see Appendix B for information on how to install Python on your computer.

Get the Data

The easiest way to get all the data to code along the book is to download the ZIP file of the book’s repository here:

https://github.com/chendaniely/pandas_for_everyone

The book’s repository will have the latest instructors on how to download the book’s data, and more detailed instructors for how to get the book can be found in Appendix B.3.

Set Up Python

Appendix G and Appendix H cover environments and installing packages, respectively. There you will find the URLs and commands on how to setup Python to code along the book. Again, the book’s repository will always contain the latest set of instructions.

Feedback, Please!

Thank you for taking the time to go through this book. If you find any problems, issues, or mistakes within the book, please send me feedback! GitHub issues may be the best place to provide this information, but you can also email me at chendaniely@gmail.com. Just be sure to use the PFE or P4E tag in the beginning of the subject line so I can make sure your emails do not get flooded by various listserv emails. If there are topics that you feel should be covered in the book, please let me know. I will try my best to put up a notebook in the GitHub repository and to get it incorporated in a later printing or edition of the book.

Words of encouragement are appreciated.

Register your copy of *Pandas for Everyone, Second Edition*, on the InformIT site for convenient access to updates and/or corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account. Enter the product ISBN (9780137891153) and click Submit. Look on the Registered Products tab for an Access Bonus Content link next to this product, and follow that link to access any available bonus materials. If you would like to be notified of exclusive offers on new editions and updates, please check the box to receive email from us.

Acknowledgments

So many people have made this book happen, in addition to the folks from the first edition (see additional acknowledgments below).

The people who helped with the book logistics: Mary Roth and Debra Williams Cauley with the book production, Cody Huddleston and Gloria W with copy editing.

My PhD committee: Anne Brown, Dave Higdoen, Alex Hanlon, and Nikki Lewis, for getting me to think about teaching and pedagogy over the years and improving the book to make it better suited for teaching and learning.

All my students who gave me an opportunity to teach you and hone how to present materials.

Everyone who has sent me corrections over the years. In particular, Sam Johnson, who gave me a cover-to-cover set of improvements.

The creators, maintainers, and contributors to the Quarto scientific and technical publishing system for creating a tool to make the book much more maintainable.²

Finally, my friends and family who have helped me get through graduate school and provided feedback during the book writing.

Acknowledgments from the First Edition

Introduction to Data Science: The three people who paved the way for this book were my instructors in the “Introduction to Data Science” course at Columbia—Rachel Schutt, Kayur Patel, and Jared Lander. Without them, I wouldn’t even know what the term “data science” means. I learned so much about the field through their lectures and labs; everything I know and do today can be traced back to this class. The instructors were only part of the learning process. The people in my study group, where we fumbled through our homework assignments and applied our skills to the final project of summarizing scientific articles, made learning the material and passing the class possible. They were Niels Bantilan, Thomas Vo, Vivian Peng, and Sabrina Cheng (depicted in the figure here). Perhaps unsurprisingly, they also got me through my master’s program (more on that later).

2. Quarto: <https://quarto.org/docs/books>

One of the midnight doodles by Vivian Peng for our project group. We have Niels, our project leader, at the top; Thomas, me, and Sabrina in the middle row; and Vivian at the bottom.



Software-Carpentry: As part of the “Introduction to Data Science” course, I attended a Software-Carpentry workshop, where I was first introduced to Pandas. My first instructors were Justin Ely and David Warde-Farley. Since then I’ve been involved in the community, thanks to Greg Wilson, and still remember the first class I helped teach, led by Aron Ahmadi and Randal S. Olson. The many workshops that I’ve taught since then, and the fellow instructors whom I’ve met, gave me the opportunity to master the knowledge and skills I know and practice today, and to disseminate them to new learners, which has cumulated into this book.

Software-Carpentry also introduced me to the NumFOCUS, PyData, and the Scientific Python communities, where all my (Python) heroes can be found. There are too many to list here. My connection to the R world is all thanks to Jared Lander.

Columbia University Mailman School of Public Health: My undergraduate study group evolved into a set of lifelong friends during my master’s program. The members of this group got me through the first semester of the program in which epidemiology and biostatistics were first taught. The knowledge I learned in this program later transferred into my knowledge of machine learning. Thanks go to Karen Lin, Sally Cheung, Grace Lee, Wai Yee (Krystal) Khine, Ashley Harper, and Jacquie Cheung. A second set of thanks to go to my old study group alumni: Niels Bantilan, Thomas Vo, and Sabrina Cheng.

To my instructors, Katherine Keyes and Martina Pavlicova, thanks for being exemplary teachers in epidemiology, and biostatistics, respectively. Thanks also to Dana March Palmer, for whom I was a TA and who gave me my first teaching experience. Mark Orr served as my thesis advisor while I was at Mailman. The department of epidemiology had a subset of faculty who did computational and simulation modeling, under the leadership of Sandro Galea, the department chair at the time. After graduation, I got my first job as a data analyst with Jacqueline Merrill at the Columbia University School of Nursing.

Getting to Mailman was a life-altering event. I never would have considered entering an MPH program if it weren’t for Ting Ting Guo. As an advisor, Charlotte Glasser was a

tremendous help to me in planning out my frequent undergraduate major changes and postgraduate plans.

Virginia Tech: The people with whom I work at the Social and Decision Analytics Laboratory (SDAL) have made Virginia Tech one of the most enjoyable places where I've worked. A second thanks to Mark Orr, who got me here. The administrators of the lab, Kim Lyman and Lori Conerly, make our daily lives that much easier. Sallie Keller and Stephanie Shipp, the director and the deputy lab director, respectively, create a collaborative work environment. The rest of the lab members, past and present (in no particular order)—David Higdon, Gizem Korkmaz, Vicki Lancaster, Mark Orr, Bianca Pires, Aaron Schroeder, Ian Crandell, Joshua Goldstein, Kathryn Ziemer, Emily Molfino, and Ana Aizcorbe—also work hard at making my graduate experience fun. It's also been a pleasure to train and work with the summer undergraduate and graduate students in the lab through the Data Science for the Public Good program. I've learned a lot about teaching and implementing good programming practices. Finally, Brian Goode adds to my experience progressing through the program by always being available to talk about various topics.

The people down in Blacksburg, Virginia, where most of the book was written, have kept me grounded during my coursework. My PhD cohort—Alex Song Qi, Amogh Jalihal, Brittany Boribong, Bronson Weston, Jeff Law, and Long Tian—have always found time for me, and for one another, and offered opportunities to disconnect from the PhD grind. I appreciate their willingness to work to maintain our connections, despite being in an interdisciplinary program where we don't share many classes together, let alone labs.

Brian Lewis and Caitlin Rivers helped me initially get settled in Blacksburg and gave me a physical space to work in the Network Dynamics and Simulation Science Laboratory. Here, I met Gloria Kang, Pyrros (Alex) Telionis, and James Schlitt, who have given me creative and emotional outlets the past few years. NDSSL has also provided and/or been involved with putting together some of the data sets used in the book.

Last but not least, Dennie Munson, my program liaison, can never be thanked enough for putting up with all my shenanigans.

Book Publication Process: Debra Williams Cauley, thank you so much for giving me this opportunity to contribute to the Python and data science community. I've grown tremendously as an educator during this process, and this adventure has opened more doors for me than the number of times I've missed deadlines. A second thanks to Jared Lander for recommending me and putting me up for the task.

Even more thanks go to Gloria Kang, Jacquie Cheung, and Jared Lander for their feedback during the writing process. I also want to thank Chris Zahn for all the work in reviewing the book from cover to cover, and Kaz Sakamoto and Madison Arnsbarger for providing feedback and reviews. Through their many conversations with me, M Pacer, Sebastian Raschka, Andreas Müller, and Tom Augspurger helped me make sure I covered my bases, and did things “properly.”

Thanks to all the people involved in the post-manuscript process: Julie Nahil (production editor), Jill Hobbs (copy editor), Rachel Paul (project manager and proofreader), Jack Lewis (indexer), and SPi Global (compositor). Y'all have been a pleasure

to work with. More importantly, you polished my writing when it needed a little help and made sure the book was formatted consistently.

Family: My immediate and extended family have always been close. It is always a pleasure when we are together for holidays or random cookouts. It's always surprising how the majority of the 50-plus of us manage to regularly get together throughout the year. I am extremely lucky to have the love and support from this wonderful group of people.

To my younger siblings, Eric and Julia: It's hard being an older sibling! The two of you have always pushed me to be a better person and role model, and you bring humor, joy, and youth into my life.

A second thanks to my sister for providing the drawings in the preface and the appendix.

Last but not least, thank you, Mom and Dad, for all your support over the years. I've had a few last-minute career changes, and you have always been there to support my decisions, financially, emotionally, and physically—including helping me relocate between cities. Thanks to the two of you, I've always been able to pursue my ambitions while knowing full well I can count on your help along the way. This book is dedicated to you.

About the Author

Daniel Y. Chen, PhD, MPH, completed his PhD at Virginia Tech in Genetics, Bioinformatics, and Computational Biology (GBCB). His dissertation was on data science education in the medical and biomedical sciences. He completed a Master's of Public Health in Epidemiology at Columbia University Mailman School of Public Health, where he studied how attitudes toward behaviors diffuse and spread in social networks. In a past life, he studied psychology and neuroscience at the Macaulay Honors College at CUNY Hunter College and worked in a bench laboratory doing microscopy work looking at proteins in the brain associated with learning and memory.

Daniel currently works as a Postdoctoral Research and Teaching Fellow at the University of British Columbia and as a Data Science Educator at Posit, PBC (formerly, RStudio, PBC). He has been involved with The Carpentries as an instructor, instructor trainer, and community maintainer lead.

This page intentionally left blank

Changes in the Second Edition

The second edition mainly updates all the code and libraries to the latest versions at the time of writing. Most of the code from the first edition was unaffected. Bits of the plotting code and machine learning data modeling code ended up changing over the years and were updated.

From a pedagogical perspective, the main Pandas chapters have also been updated with proper learning objectives, and the introductory chapters have accompanying concept maps to help educators plan a learning path, and for learners to visualize how concepts are related to one another. These were all topics I've learned about while doing my dissertation, and I hope they become useful for learners and educators. The book also includes access to online bonus chapters on geopandas, Dask, and creating interactive graphics with Altair.

I've also rearranged the chapters in the second edition based on my experiences when I teach workshops. Part I of the book contains the most important bits of information that I aim to cover in my workshops. The rest of the book can be thought of as data processing details after the more fundamental topics are covered. The chapters that have big changes from the first edition have a section in the chapter's introduction on the details of what has changed.

Many of the libraries and tools mentioned in the conclusion chapters of the book will also have freely available chapters to accompany this book to help you extend your learning.

This page intentionally left blank

Tidy Data

Hadley Wickham, PhD,¹ one of the more prominent members of the R community, introduced the concept of **tidy data** in a *Journal of Statistical Software* paper.² Tidy data is a framework to structure data sets so they can be easily analyzed and visualized. It can be thought of as a goal one should aim for when cleaning data. Once you understand what tidy data is, that knowledge will make your data analysis, visualization, and collection much easier.

What is tidy data? Hadley Wickham's paper defines it as meeting the following criteria: (1) Each row is an observation, (2) Each column is a variable, and (3) Each type of observational unit forms a table.

The newer definition from the R4DS book³ focuses on an individual data set (i.e., table):

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

This chapter goes through the various ways to tidy data using examples from Wickham's paper.

Learning Objectives

The concept map for this chapter can be found in Figure A.4.

- Identify the components of tidy data
- Identify common data errors
- Use functions and methods to process and tidy data

Note About This Chapter

Data used in this chapter will have NaN missing values when they are loaded into Pandas (Chapter 9). In the raw CSV files, they will appear as empty values. I typically try to avoid

1. Hadley Wickham, PhD: <http://hadley.nz>

2. Tidy Data paper: <http://vita.had.co.nz/papers/tidy-data.pdf>

3. R For Data Science Book: <https://r4ds.had.co.nz/tidy-data.html>

forward referencing in the book, but I felt that the concept of tidy data warranted a much earlier place in the book because it is so fundamental to how we should be thinking about data technically (as opposed to ethically), that the chapter was moved toward the front of the book without having to cover more detailed data processing steps first. I could have changed the data sets such that there were no missing values, but opted not to do so because (1) it would no longer follow the data used in Wickam’s “Tidy Data” paper, and (2) it would be a less realistic data set.

4.1 Columns Contain Values, Not Variables

Data can have columns that contain values instead of variables. This is usually a convenient format for data collection and presentation.

4.1.1 Keep One Column Fixed

We’ll use data on income and religion in the United States from the Pew Research Center to illustrate how to work with columns that contain values, rather than variables.

```
import pandas as pd
pew = pd.read_csv('data/pew.csv')
```

When we look at this data set, we can see that not every column is a variable. The values that relate to income are spread across multiple columns. The format shown is a great choice when presenting data in a table, but for data analytics, the table should be reshaped so that we have `religion`, `income`, and `count` variables.

```
# show only the first few columns
print(pew.iloc[:, 0:5])
```

	religion	<\$10k	\$10-20k	\$20-30k	\$30-40k
0	Agnostic	27	34	60	81
1	Atheist	12	27	37	52
2	Buddhist	27	21	30	34
3	Catholic	418	617	732	670
4	Don't know/refused	15	14	15	11
..
13	Orthodox	13	17	23	32
14	Other Christian	9	7	11	13
15	Other Faiths	20	33	40	46
16	Other World Religions	5	2	3	4
17	Unaffiliated	217	299	374	365

```
[18 rows x 5 columns]
```

This view of the data is also known as “wide” data. To turn it into the “long” tidy data format, we will have to unpivot/melt/gather (depending on which statistical programming language we use) our dataframe.

Note

I usually use the terminology from the R world of using “pivot” to refer to going from wide data to long data and vice versa. I usually will specify the direction with “pivot longer” to go from wide data to long data, and “pivot wider” to go from long data to wide data.

In this chapter “pivot longer” will refer to the dataframe `.melt()` method, and “pivot wider” will refer to the dataframe `.pivot()` method.

Pandas DataFrames have a method called `.melt()` that will reshape the dataframe into a tidy format and it takes a few parameters:

- `id_vars` is a container (list, tuple, ndarray) that represents the variables that will remain as is.
- `value_vars` identifies the columns you want to melt down (or unpivot). By default, it will melt all the columns not specified in the `id_vars` parameter.
- `var_name` is a string for the new column name when the `value_vars` is melted down. By default, it will be called `variable`.
- `value_name` is a string for the new column name that represents the values for the `var_name`. By default, it will be called `value`.

```
# we do not need to specify a value_vars since we want to pivot
# all the columns except for the 'religion' column
pew_long = pew.melt(id_vars='religion')
```

```
print(pew_long)
```

	religion	variable	value
0	Agnostic	<\$10k	27
1	Atheist	<\$10k	12
2	Buddhist	<\$10k	27
3	Catholic	<\$10k	418
4	Don't know/refused	<\$10k	15
..
175	Orthodox	Don't know/refused	73
176	Other Christian	Don't know/refused	18
177	Other Faiths	Don't know/refused	71
178	Other World Religions	Don't know/refused	8
179	Unaffiliated	Don't know/refused	597

```
[180 rows x 3 columns]
```

Note

The `.melt()` method also exists as a pandas function, `pd.melt()`

The below two lines of code are equivalent:

```
# melt method
pew_long = pew.melt(id_vars='religion')

# melt function
pew_long = pd.melt(pew, id_vars='religion')
```

Internally, the `.melt()` method redirects the function call to the Pandas `pd.melt()` function. The `.melt()` method notation is there to make the Pandas API more consistent, and also allows us to method-chain (Appendix U).

We can change the defaults so that the melted/unpivoted columns are named.

```
pew_long = pew.melt(
    id_vars="religion", var_name="income", value_name="count"
)
```

```
print(pew_long)
```

	religion	income	count
0	Agnostic	<\$10k	27
1	Atheist	<\$10k	12
2	Buddhist	<\$10k	27
3	Catholic	<\$10k	418
4	Don't know/refused	<\$10k	15
..
175	Orthodox	Don't know/refused	73
176	Other Christian	Don't know/refused	18
177	Other Faiths	Don't know/refused	71
178	Other World Religions	Don't know/refused	8
179	Unaffiliated	Don't know/refused	597

```
[180 rows x 3 columns]
```

4.1.2 Keep Multiple Columns Fixed

Not every data set will have one column to hold still while you unpivot the rest of the columns. As an example, consider the Billboard data set.

```
billboard = pd.read_csv('data/billboard.csv')

# look at the first few rows and columns
print(billboard.iloc[0:5, 0:16])
```

	year	artist	track	time	date.entered	\
0	2000	2 Pac	Baby Don't Cry (Keep...)	4:22	2000-02-26	
1	2000	2Ge+her	The Hardest Part Of ...	3:15	2000-09-02	
2	2000	3 Doors Down	Kryptonite	3:53	2000-04-08	

3	2000	3 Doors Down						Loser	4:24	2000-10-21
4	2000	504 Boyz					Wobble Wobble	3:35	2000-04-15	

	wk1	wk2	wk3	wk4	wk5	wk6	wk7	wk8	wk9	wk10	wk11
0	87	82.0	72.0	77.0	87.0	94.0	99.0	NaN	NaN	NaN	NaN
1	91	87.0	92.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	81	70.0	68.0	67.0	66.0	57.0	54.0	53.0	51.0	51.0	51.0
3	76	76.0	72.0	69.0	67.0	65.0	55.0	59.0	62.0	61.0	61.0
4	57	34.0	25.0	17.0	17.0	31.0	36.0	49.0	53.0	57.0	64.0

You can see here that each week has its own column. Again, there is nothing *wrong* with this form of data. It may be easy to enter the data in this form, and it is much quicker to understand what it means when the data is presented in a table. However, there may be a time when you will need to melt the data. For example, if you wanted to create a faceted plot of the weekly ratings, the facet variable would need to be a column in the dataframe.

```
# use a list to reference more than 1 variable
billboard_long = billboard.melt(
  id_vars=["year", "artist", "track", "time", "date.entered"],
  var_name="week",
  value_name="rating",
)
print(billboard_long)
```

	year	artist	track	time
0	2000	2 Pac	Baby Don't Cry (Keep...	4:22
1	2000	2Ge+her	The Hardest Part Of ...	3:15
2	2000	3 Doors Down	Kryptonite	3:53
3	2000	3 Doors Down	Loser	4:24
4	2000	504 Boyz	Wobble Wobble	3:35
...
24087	2000	Yankee Grey	Another Nine Minutes	3:10
24088	2000	Yearwood, Trisha	Real Live Woman	3:55
24089	2000	Ying Yang Twins	Whistle While You Tw...	4:19
24090	2000	Zombie Nation	Kernkraft 400	3:30
24091	2000	matchbox twenty	Bent	4:12

	date.entered	week	rating
0	2000-02-26	wk1	87.0
1	2000-09-02	wk1	91.0
2	2000-04-08	wk1	81.0
3	2000-10-21	wk1	76.0
4	2000-04-15	wk1	57.0
...
24087	2000-04-29	wk76	NaN
24088	2000-04-01	wk76	NaN

```
24089    2000-03-18  wk76    NaN
24090    2000-09-02  wk76    NaN
24091    2000-04-29  wk76    NaN
```

```
[24092 rows x 7 columns]
```

4.2 Columns Contain Multiple Variables

Sometimes columns in a data set may represent multiple variables. This format is commonly seen when working with health data, for example. To illustrate this situation, let's look at the Ebola data set.

```
ebola = pd.read_csv('data/country_timeseries.csv')
print(ebola.columns)
```

```
Index(['Date', 'Day', 'Cases_Guinea', 'Cases_Liberia',
       'Cases_SierraLeone', 'Cases_Nigeria', 'Cases_Senegal',
       'Cases_UnitedStates', 'Cases_Spain', 'Cases_Mali',
       'Deaths_Guinea', 'Deaths_Liberia', 'Deaths_SierraLeone',
       'Deaths_Nigeria', 'Deaths_Senegal', 'Deaths_UnitedStates',
       'Deaths_Spain', 'Deaths_Mali'],
      dtype='object')
```

```
# print select rows and columns
print(ebola.iloc[:5, [0, 1, 2, 10]])
```

```
      Date Day  Cases_Guinea  Deaths_Guinea
0  1/5/2015  289         2776.0          1786.0
1  1/4/2015  288         2775.0          1781.0
2  1/3/2015  287         2769.0          1767.0
3  1/2/2015  286             NaN             NaN
4 12/31/2014  284         2730.0          1739.0
```

The column names `Cases_Guinea` and `Deaths_Guinea` actually contain two variables. The individual status (cases and deaths, respectively) as well as the country name, Guinea. The data is also arranged in a wide format that needs to be reshaped (with the `.melt()` method).

First, let's fix the problem we know how to fix, by melting the data into long format.

```
ebola_long = ebola.melt(id_vars=['Date', 'Day'])
```

```
print(ebola_long)
```

```
      Date Day  variable  value
0  1/5/2015  289  Cases_Guinea  2776.0
1  1/4/2015  288  Cases_Guinea  2775.0
2  1/3/2015  287  Cases_Guinea  2769.0
3  1/2/2015  286  Cases_Guinea    NaN
4 12/31/2014  284  Cases_Guinea  2730.0
...      ...  ...      ...      ...
```

```

1947  3/27/2014    5  Deaths_Mali  NaN
1948  3/26/2014    4  Deaths_Mali  NaN
1949  3/25/2014    3  Deaths_Mali  NaN
1950  3/24/2014    2  Deaths_Mali  NaN
1951  3/22/2014    0  Deaths_Mali  NaN

```

```
[1952 rows x 4 columns]
```

Conceptually, the column of interest can be split based on the underscore in the column name, `_`. The first part will be the new status column, and the second part will be the new country column. This will require some string parsing and splitting in Python (more on this in Chapter 11). In Python, a string is an object, similar to how Pandas has `Series` and `DataFrame` objects. Chapter 2 showed how `Series` can have methods such as `.mean()`, and `DataFrames` can have methods such as `.to_csv()`. Strings have methods as well. In this case, we will use the `.split()` method that takes a string and “splits” it up based on a given delimiter. By default, `.split()` will split the string based on a space, but we can pass in the underscore, `_`, in our example. To get access to the string methods, we need to use the `.str` attribute. `.str` is a special type of attribute that Pandas calls an “accessor” because it can “access” string methods (see Chapter 11 for more on strings). Access to the Python string methods and allow us to work across the entire column. This will be the key to parting out the multiple bits of information stored in each value.

4.2.1 Split and Add Columns Individually

We can use the `.str` accessor to make a call to the `.split()` method and pass in the `_` underscore.

```

# get the variable column
# access the string methods
# and split the column based on a delimiter
variable_split = ebola_long.variable.str.split('_')

print(variable_split[:5])

```

```

0    [Cases, Guinea]
1    [Cases, Guinea]
2    [Cases, Guinea]
3    [Cases, Guinea]
4    [Cases, Guinea]
Name: variable, dtype: object

```

After we split on the underscore, the values are returned in a list. We can tell it’s a list by:

1. Knowing about the `.split()` method on base Python string objects⁴
2. Visually seeing the square brackets in the output, []
3. Getting the `type()` of one of the items in the `Series`

4. String `.split()` documentation: <https://docs.python.org/3/library/stdtypes.html#str.split>

```

# the entire container
print(type(variable_split))

<class 'pandas.core.series.Series'>

# the first element in the container
print(type(variable_split[0]))

<class 'list'>

```

Now that the column has been split into various pieces, the next step is to assign those pieces to a new column. First, however, we need to extract all the 0-index elements for the `status` column and the 1-index elements for the `country` column. To do so, we need to access the string methods again, and then use the `.get()` method to “get” the index we want for each row.

```

status_values = variable_split.str.get(0)
country_values = variable_split.str.get(1)

print(status_values)

```

```

0      Cases
1      Cases
2      Cases
3      Cases
4      Cases
...
1947  Deaths
1948  Deaths
1949  Deaths
1950  Deaths
1951  Deaths
Name: variable, Length: 1952, dtype: object

```

Now that we have the vectors we want, we can add them to our dataframe.

```

ebola_long['status'] = status_values
ebola_long['country'] = country_values

print(ebola_long)

```

	Date	Day	variable	value	status	country
0	1/5/2015	289	Cases_Guinea	2776.0	Cases	Guinea
1	1/4/2015	288	Cases_Guinea	2775.0	Cases	Guinea
2	1/3/2015	287	Cases_Guinea	2769.0	Cases	Guinea
3	1/2/2015	286	Cases_Guinea	NaN	Cases	Guinea
4	12/31/2014	284	Cases_Guinea	2730.0	Cases	Guinea


```

...      ...      ...
1947    3/27/2014    5    Deaths_Mali    NaN    Deaths    Mali
1948    3/26/2014    4    Deaths_Mali    NaN    Deaths    Mali
1949    3/25/2014    3    Deaths_Mali    NaN    Deaths    Mali
1950    3/24/2014    2    Deaths_Mali    NaN    Deaths    Mali
1951    3/22/2014    0    Deaths_Mali    NaN    Deaths    Mali

```

```
[1952 rows x 6 columns]
```

4.2.2 Split and Combine in a Single Step

We can actually do the above steps in a single step. If we look at the `.str.split()` method documentation (you can find this by looking by going to the Pandas API documentation `> Series > String Handling (.str.) > .split() method5`), there is a parameter named `expand` that defaults to `False`, but when we set it to `True`, it will return a `DataFrame` where each result of the split is in a separate column, instead of a `Series` of `list` containers.

```

# reset our ebola_long data
ebola_long = ebola.melt(id_vars=['Date', 'Day'])

# split the column by _ into a dataframe using expand
variable_split = ebola_long.variable.str.split('_', expand=True)

print(variable_split)

```

```

      0      1
0    Cases  Guinea
1    Cases  Guinea
2    Cases  Guinea
3    Cases  Guinea
4    Cases  Guinea
...    ...    ...
1947  Deaths  Mali
1948  Deaths  Mali
1949  Deaths  Mali
1950  Deaths  Mali
1951  Deaths  Mali

```

```
[1952 rows x 2 columns]
```

From here, we can actually use the Python and Pandas multiple assignment feature (Appendix Q), to directly assign the newly split columns into the original `DataFrame`. Since our output `variable_split` returned a `DataFrame` with two columns, we can assign two new columns to our `ebola_long` `DataFrame`.

5. `Series.str.split()` method documentation: <https://pandas.pydata.org/docs/reference/api/pandas.Series.str.split.html#pandas.Series.str.split>

```
| ebola_long[['status', 'country']] = variable_split
```

```
| print(ebola_long)
```

	Date	Day	variable	value	status	country
0	1/5/2015	289	Cases_Guinea	2776.0	Cases	Guinea
1	1/4/2015	288	Cases_Guinea	2775.0	Cases	Guinea
2	1/3/2015	287	Cases_Guinea	2769.0	Cases	Guinea
3	1/2/2015	286	Cases_Guinea	NaN	Cases	Guinea
4	12/31/2014	284	Cases_Guinea	2730.0	Cases	Guinea
...
1947	3/27/2014	5	Deaths_Mali	NaN	Deaths	Mali
1948	3/26/2014	4	Deaths_Mali	NaN	Deaths	Mali
1949	3/25/2014	3	Deaths_Mali	NaN	Deaths	Mali
1950	3/24/2014	2	Deaths_Mali	NaN	Deaths	Mali
1951	3/22/2014	0	Deaths_Mali	NaN	Deaths	Mali

```
[1952 rows x 6 columns]
```

You can also opt to do this as a concatenation (`pd.concat()`) function call as well (Chapter 6).

4.3 Variables in Both Rows and Columns

At times, data will be formatted so that variables are in both rows and columns – that is, in some combination of the formats described in previous sections of this chapter. Most of the methods needed to tidy up such data have already been presented (`.melt()` and some string parsing with the `.str.` accessor attribute). What is left to show is what happens if a column of data actually holds two variables instead of one variable. In this case, we will have to “pivot” the variable into separate columns, i.e., go from long data to wide data.

```
| weather = pd.read_csv('data/weather.csv')
| print(weather.iloc[:5, :11])
```

	id	year	month	element	d1	d2	d3	d4	d5	d6	d7
0	MX17004	2010	1	tmax	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	MX17004	2010	1	tmin	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	MX17004	2010	2	tmax	NaN	27.3	24.1	NaN	NaN	NaN	NaN
3	MX17004	2010	2	tmin	NaN	14.4	14.4	NaN	NaN	NaN	NaN
4	MX17004	2010	3	tmax	NaN	NaN	NaN	NaN	32.1	NaN	NaN

The weather data include minimum (`tmin`) and maximum (`tmax`) temperatures recorded for each day (`d1`, `d2`, ..., `d31`) of the month (`month`). The `element` column contains variables that need to be pivoted wider to become new columns, and the day variables need to be melted into row values.

Again, there is nothing wrong with the data in the current format. It is simply not in a shape amenable to analysis, although this kind of formatting can be helpful when presenting data in reports. Let's first fix the day values.

```
weather_melt = weather.melt(
    id_vars=["id", "year", "month", "element"],
    var_name="day",
    value_name="temp",
)
```

```
| print(weather_melt)
```

	id	year	month	element	day	temp
0	MX17004	2010	1	tmax	d1	NaN
1	MX17004	2010	1	tmin	d1	NaN
2	MX17004	2010	2	tmax	d1	NaN
3	MX17004	2010	2	tmin	d1	NaN
4	MX17004	2010	3	tmax	d1	NaN
...
677	MX17004	2010	10	tmin	d31	NaN
678	MX17004	2010	11	tmax	d31	NaN
679	MX17004	2010	11	tmin	d31	NaN
680	MX17004	2010	12	tmax	d31	NaN
681	MX17004	2010	12	tmin	d31	NaN

```
[682 rows x 6 columns]
```

Next, we need to pivot up the variables stored in the element column.

```
weather_tidy = weather_melt.pivot_table(
    index=['id', 'year', 'month', 'day'],
    columns='element',
    values='temp'
)
```

```
| print(weather_tidy)
```

	element	tmax	tmin
id	year month day		
MX17004	2010 1 d30	27.8	14.5
	2 d11	29.7	13.4
	d2	27.3	14.4
	d23	29.9	10.7
	d3	24.1	14.4
...

11	d27	27.7	14.2
	d26	28.1	12.1
	d4	27.2	12.0
12	d1	29.9	13.8
	d6	27.8	10.5

[33 rows x 2 columns]

Looking at the pivoted table, we notice that each value in the `element` column is now a separate column. We can leave this table in its current state, but we can also flatten the hierarchical columns.

```
weather_tidy_flat = weather_tidy.reset_index()
print(weather_tidy_flat)
```

element	id	year	month	day	tmax	tmin
0	MX17004	2010	1	d30	27.8	14.5
1	MX17004	2010	2	d11	29.7	13.4
2	MX17004	2010	2	d2	27.3	14.4
3	MX17004	2010	2	d23	29.9	10.7
4	MX17004	2010	2	d3	24.1	14.4
..
28	MX17004	2010	11	d27	27.7	14.2
29	MX17004	2010	11	d26	28.1	12.1
30	MX17004	2010	11	d4	27.2	12.0
31	MX17004	2010	12	d1	29.9	13.8
32	MX17004	2010	12	d6	27.8	10.5

[33 rows x 6 columns]

Likewise, we can apply these methods without the intermediate dataframe:

```
weather_tidy = (
    weather_melt
    .pivot_table(
        index=['id', 'year', 'month', 'day'],
        columns='element',
        values='temp')
    .reset_index()
)
print(weather_tidy)
```

element	id	year	month	day	tmax	tmin
0	MX17004	2010	1	d30	27.8	14.5
1	MX17004	2010	2	d11	29.7	13.4
2	MX17004	2010	2	d2	27.3	14.4
3	MX17004	2010	2	d23	29.9	10.7
4	MX17004	2010	2	d3	24.1	14.4

```

..      ....  ....  ....  ....  ....
28      MX17004  2010      11  d27  27.7  14.2
29      MX17004  2010      11  d26  28.1  12.1
30      MX17004  2010      11   d4  27.2  12.0
31      MX17004  2010      12   d1  29.9  13.8
32      MX17004  2010      12   d6  27.8  10.5

```

```
[33 rows x 6 columns]
```

Conclusion

This chapter explored how we can reshape data into a format that is conducive to data analysis, visualization, and collection. We applied the concepts in Hadley Wickham’s “Tidy Data” paper to show the various functions and methods to reshape our data. This is an important skill because some functions need data to be organized into a certain shape, tidy or not, to work. Knowing how to reshape your data is an important skill for both the data scientist and the analyst.

This page intentionally left blank

Index

Symbols

- * operator, specifying model interactions, 324
- { }(curly brackets), dictionary syntax, 397
- % (percent) operator, calling magic commands, 427
- + (plus) operator, adding covariates to linear models, 324
- () (round brackets)
 - line breaks, 393–394
 - tuple syntax, 396
- [] (square brackets)
 - dictionary values, 397
 - getting first character of string, 230
 - list syntax, 395–396

Numbers

- 2D density plot, 88–89

A

- Aggregation (or aggregate)
 - of built-in methods, 178–179
 - of calculations, 23
 - of functions, 179–182
 - multiple functions simultaneously, 182–184
 - one-variable grouped aggregation, 176–177
 - options for applying functions in and aggregate methods, 182–184
 - overview of, 176
 - saving groupby object without running aggregate, transform, or filter, 190–191

- AIC (Akaike information criteria), 327, 329

Alignment

- DataFrame, 44–45
- Series, 39–42

Anaconda

- command prompt, 381–382
- installers for, 373–374
- Miniconda, 374
- package installation, 389–390
- Python distribution, 385
- Spyder IDE, 382
- uninstalling, 374

AnacondaCon conference, 364

ANOVA (analysis of variance), 326–327

Anscombe’s quartet

- for data visualization, 65–66, 70–71
- plotting with facets, 99–100

Apache Arrow, 58, 280

apply

- concept map for, 372
- creating/using functions, 131–132
- functions across rows or columns of data, 133
- lambda functions, 141–142
- numba library and, 140–141
- over a DataFrame, 135–138
- over a Series, 133–135
- overview of, 131
- primer on, 131–132
- summary/conclusion, 142
- vectorized functions, 138–141

*args, function parameter, 408

Arrays

- scientific computing stack, 359
- sklearn library and, 286–287
- working with, 415–416

Arrow, 58
 for dates and times, 280

`assert`, checking data assembly with, 166

`assign`, modifying columns with, 50–52

Assignment
 multiple, 413–414
 passing/reassigning values, 395–396

`astype` method
 converting column to categorical type, 225–226
 converting to numeric values, 221–222
 converting values to strings, 220

Attributes
 class, 417
 dot notation and, 10–11
 Series, 35

Average cluster algorithm, in hierarchical clustering, 353–354

Axes, plotting, 67–71

B

Bar plots, 89–91

Bash shell, 377–378

BIC (Bayesian information criteria), 327, 329

“The Big Book of Python,” 365

“The Big Book of R,” 365

Binary
 feather format for saving, 56–57
 logistic regression for binary response variable, 297
 serialize and save data in binary format, 53

Bivariate statistics
 in `matplotlib`, 74–76
 in `seaborn`, 83–94

Booleans (`bool`)
 subsetting `DataFrame`, 43
 subsetting `Series`, 36–39

Boxplots
 for bivariate statistics, 75–76

 creating, 114–115

Broadcasting, Pandas support for, 40–41, 44–45

C

Calculations
 `datetime`, 257–258
 involving multiple variables, 191
 with missing data (values), 215–216
 of multiple functions simultaneously, 182–184
 timing execution of, 360, 427–428

Carpentries, 364

CAS (computer algebra systems), 359

category
 converting column to, 225–226
 manipulating categorical data, 226
 overview of, 225
 representing categorical variables, 221
 `sklearn` library used with categorical variables, 291–293
 `statsmodels` library used with categorical variables, 289–291

Centroid cluster algorithm, in hierarchical clustering, 353–354

Chaining methods, 423–425

Characters
 formatting strings of, 430
 getting first character of string, 230
 getting last character of string, 231–233
 slicing multiple letters of string, 230
 strings as series of, 229

Classes, 417–418

Clustering
 average cluster algorithm, 353–354
 centroid cluster algorithm, 353–354
 complete cluster algorithm, 352
 dimension reduction using PCA, 347–351
 hierarchical clustering, 351–356
 k-means, 345–351
 manually setting threshold for, 355–356
 overview of, 345

- Clustering (*continued*)
 - single cluster algorithm, 352–353
 - summary/conclusion, 356
 - ward cluster algorithm, 354–355
- Code
 - profiling, 360
 - reuse, 405
 - style, 393–394
 - timing execution of, 360, 427–428
- `coerce`, 224–225
- Colon (:), use in slicing syntax, 15, 399–400
- Colors, multivariate statistics in `seaborn`, 95–97
- Columns
 - adding, 45–47
 - concatenation generally, 150–151
 - concatenation with different indices, 153–154
 - converting to `category`, 225–226
 - directly changing, 47–50
 - dot notation to pull values of, 10–11
 - dropping values, 52
 - methods of indexing, 11
 - modifying with `assign`, 50–52
 - rows and columns both containing variables, 126–129
 - selecting, 15–16
 - single value returns, 8–9
 - slicing, 18–21
 - subsetting by name, 7–8
 - subsetting by range, 16–18
 - subsetting generally, 21–23
 - subsetting using slicing syntax, 15–16
- Columns, with multiple variables
 - overview of, 122–123
 - split and add individually, 123–125
 - split and combine in single step, 125–126
- Columns, with values not variables
 - keeping multiple columns fixed, 120–122
 - keeping one column fixed, 118–120
 - overview of, 118
- Command line
 - basic commands, 378
 - Linux, 378
 - Mac, 377
 - overview of, 377
 - Windows, 377
- Comma-separated values. *See* CSV (comma-separated values)
- `compile`, pattern compilation, 246–247
- Complete cluster algorithm, in hierarchical clustering, 352
- Comprehensions
 - function comprehension, 403–404
 - list comprehension, 158–160
 - overview of, 401–402
- Computer algebra systems (CAS), 359
- Concatenation (`concat`)
 - adding columns, 150–151
 - adding rows, 147–150
 - dataframe parts and, 146–147
 - with different indices, 151–154
 - `ignore_index` parameter after, 149–150
 - observational units across multiple tables, 154–160
 - overview of, 146
 - split and combine in single step, 125–126
- Concept maps, 369–372
- `concurrent.features`, 360
- `conda`
 - creating environments, 385–387
 - `install`, 374
 - managing packages, 389
 - `update`, 390
- Conditional statements, 433–434
- Conferences, 363–364
- Confidence interval, in linear regression example, 285
- Containers
 - `join` method and, 234–235
 - looping over contents, 401–402
 - overview, 395
 - types of, 229
- Conversion, of data types
 - to `category`, 225–226
 - to `datetime`, 250–253

Conversion, of data types (*continued*)
 to `numeric`, 221–225
 to `string`, 220–221

Counting
`groupby` count, 197–199
 missing data (values), 210–212
 Poisson regression and, 304–308

Count (bar) plot, for univariate statistics, 81–83

Covariates
 adding to linear models, 324
 multiple linear regression with three covariates, 320–322

Cox proportional hazards model
 survival analysis, 314–316
 testing assumptions, 315–316

`C` `printf` style formatting, 429

`cProfile`, profiling code, 360

`create` (environments), 385–387

Cross-validation
 model diagnostics, 329–333
 regularization techniques, 341–343

`cross_val_scores`, 332–333

CSV (comma-separated values)
 for data storage, 55
 importing CSV files, 55
 loading multiple files using list comprehension, 158–160

Cumulative sum (`cumsum`), 199

`cython`, performance-related library, 360

D

Dash, 362

Dashboards, 362

`Dask` library, 360

Data assembly
 adding rows, 147–150
 checking your work on, 166
 combining data sets, 145
 concatenation, 146–154
 concatenation with different indices, 151–154

`dataframe` parts and, 146–147

`ignore_index` parameter after concatenation, 149–150

loading multiple files using list comprehension, 158–160

loading multiple files using list comprehension, 158–160

many-to-many merges, 163–166

many-to-one merges, 163

merging multiple data sets, 160–166

observational units across multiple tables, 154–160

one-to-one merges, 162–163

overview of, 145

summary/conclusion, 167

tidy data, 167

DataFrame
 adding columns, 45–47
 aggregation, 182–183
 alignment and vectorization, 44–45
`apply` function(s), 135–138
 basic plots, 27–28
 boolean subsetting, 43
 as class, 417–418
 concatenation, 149
 concept map for basics in, 369
 converting to Arrow objects, 58
 converting to dictionary objects, 58–59
 creating, 32–33
 defined, 3
 directly changing columns, 47–50
 exporting, 56
 grouped and aggregated calculations, 23–27
 grouped frequency counts, 27
 grouped means, 23–26
 histogram, 111
 loading first data set, 4–6
 methods, 43
`ndarray` save method, 53
 overview of, 3, 42
 parts of, 42–43
 single value returns, 8–9

- DataFrame (*continued*)
 - slicing columns, 18–21
 - subsetting columns by name, 7–8
 - subsetting columns by range, 16–18
 - subsetting columns using slicing syntax, 15–16
 - subsetting rows and columns, 21–23
 - subsetting rows by index label, 11–13
 - subsetting rows by row number, 13–14
 - summary/conclusion, 28–29
 - type function for checking, 5
 - writing CSV files (`to_csv` method), 55
- Data models, 281–282
 - diagnostics (*See* Model diagnostics)
 - generalized linear (*See* GLM (generalized linear models))
 - linear (*See* Linear models)
- Data normalization
 - multiple observational units in a table, 169–170
 - overview, 169
- Data sets
 - cleaning data, 416
 - combining, 145
 - downloading for this book, 375
 - equality tests for missing data, 203–204
 - exporting/importing data (*See* Exporting/importing data)
 - Indemics (Interactive Epidemic Simulation), 196
 - lists for data storage, 395–396
 - loading, 4–6
 - many-to-many merges, 163–166
 - many-to-one merges, 163
 - merging, 160–166
 - one-to-one merges, 162–163
 - tidy data, 117
- Data structures
 - adding columns, 45–47
 - concept map for, 370
 - creating, 31–33
 - CSV (comma-separated values), 55
 - DataFrame alignment and vectorization, 44–45
 - DataFrame boolean subsetting, 43
 - DataFrame generally, 42–43
 - directly changing columns, 47–50
 - dropping values, 52
 - Excel and, 55–56
 - exporting/importing data, 52
 - feather format, 56–57
 - making changes to, 45
 - overview of, 31
 - pickle data, 53–54
 - Series alignment and vectorization, 39–42
 - Series boolean subsetting, 36–39
 - Series generally, 33–35
 - Series methods, 35–37
 - Series similarity with ndarray, 35–36
 - summary/conclusion, 63
- Data types (`dtype`)
 - category dtype, 225
 - converting to `category`, 225–226
 - converting to `datetime`, 250–253
 - converting to `numeric`, 221–225
 - converting to `string`, 220–221
 - getting list of types stored in column, 225–226
 - manipulating categorical data, 226
 - overview of, 219
 - Series attributes, 35
 - specifying from `numpy` library, 221
 - summary/conclusion, 227
 - `to_numeric` function, 222–225
 - viewing list of, 219–220
- `date_range` function, 266–269
- `datetime`
 - adding columns to data structures, 45–47
 - Arrow with, 280
 - calculations, 257–258
 - converting to, 250–253
 - directly changing columns, 48–49
 - extracting date components (year, month, day), 254–257
 - frequencies, 268

datetime (*continued*)

- getting stock-related data, 261–263
- loading date related data, 253–254
- methods, 259–261
- object, 249–250
- offsets, 268–269
- overview of, 249
- ranges, 266–269
- resampling, 276–278
- shifting values, 270–276
- subsetting data based on dates, 263–266
- summary/conclusion, 280
- time zones, 278–279

DatetimeIndex, 263–265, 268

Day, extracting date components from **datetime** object, 254–257

Daylight savings time, 278

def keyword, use with functions, 405–406

Density plots

- 2D density plot, 88–89
- plot.kde** function, 111–112
- for univariate statistics, 80

Diagnostics. *See* Model diagnostics

Dictionaries (**dict**)

- creating **DataFrame**, 32–33
- objects to converting **DataFrame** objects to, 58–59
- overview of, 396–398
- passing method to, 182–183

Directories, working, 383–384

distplot, creating histograms, 81–82

dmatrices function, **statsmodels** library, 331–333

Docstrings (**docstring**), function

- documentation, 132, 405

Dot notation, to pull a column of values, 10–11

dropna parameter

- counting missing values, 210–212
- dropping missing values, 214–215

Dropping (**drop**)

- data structure values, 52
- missing data (values), 214–215

dtype. *See* Data types (**dtype**)

E

EAFP (easier to ask for forgiveness than for permissions), 191

Elastic net, regularization technique, 340–341

elif, 433–434

else, 433–434

Environments

- creating, 385–388
- deleting, 387
- Pipenv**, 387–388
- Pyenv**, 387

Equality tests, for missing data, 203–204

errors parameter, numeric, 223–224

EuroSciPy conference, 364

Excel

- DataFrame** and, 56
- Series** and, 56

Exporting/importing data

- Arrow, 58
- CSV (comma-separated values), 55
- dictionary, 58–59
- Excel, 55–56
- feather format, 56–57
- JSON, 59–62
- methods, 63
- output types, 62–63
- overview of, 52
- pickle** data, 53–54

F

Facets, plotting, 99–104

Feather format, interface with R language, 56–57

Files

- loading multiple using list comprehension, 158–160
- working directories and, 383

fillna method, 212–213

Filter (**filter**), **groupby** operations, 188–189

Find
 missing data (values), 210–212
 patterns, 244–245

findall, patterns, 244–245

Fizz Buzz, 433–434

float/float64, 221

Folders
 project organization, 379
 working directories and, 383

for loop. *See* Loops (for loop)

format method, 236

Formats/formatting
 date formats, 252
 serialize and save data in binary format, 53
 strings (string), 236–239, 429–431

Formatted literal strings (f-strings), 236–239

formula API, in `statsmodels` library, 284–285

freq parameter, 268

Frequency
 datetime, 268
 grouped frequency counts, 27
 offsets, 268–269
 resampling converting between, 276–278

f-strings, 236–238

f-strings (formatted literal strings), 236–239

Functions
 across rows or columns of data, 133
 aggregation, 179–182
 apply over `DataFrame`, 135–138
 apply over `Series`, 133–135
 arbitrary parameters, 407–408
 calculating multiple simultaneously, 182–184
 comprehensions and, 403–404
 creating/using, 131–132
 custom, 180–181
 default parameters, 407
 groupby, 178
`**kwargs`, 408
 lambda, 141–142

options for applying in and aggregate methods, 182–184
 overview of, 405–408
 regular expressions (RegEx), 240
 vectorized, 138–141
 z-score example of transforming data, 184–186

G

Ganssle, Paul, 280

Gapminder data set, 4

Generalized linear models (GLM). *See also* Linear regression models
 logistic regression, 446–447
 model diagnostics, 327–329
 more GLM options, 308–309
 negative binomial regression, 306–308, 448–449
 overview of, 297
 Poisson regression, 304–308, 447–449
`sklearn` library for logistic regression, 300–304
`statsmodels` library for logistic regression, 299–300
`statsmodels` library for Poisson regression, 304–306
 summary/conclusion, 309
 survival analysis, 311–317
 testing Cox model assumptions, 315–316

Generators
 converting to list, 16–17
 overview of, 409–411

get
 dictionary values with, 397–398
 selecting groups, 191–192

Git for Windows, 377

github, 365

GLM (generalized linear models). *See* Generalized linear models

glm function, in `statsmodels` library, 306, 308–309

Going it alone, 363–365

Groupby (groupby)
 aggregation, 176–184
 aggregation functions, 179–182
 applying functions in and **aggregate** methods, 182–184
 built-in aggregation methods, 178–179
 calculations generally, 24–25
 calculations involving multiple variables, 191
 calculations of means, 23–26
 compared with SQL, 175
 filtering, 188–189
 flattening results, 194–195
 frequency counts, 27
 iterating through groups, 192–194
 methods and functions, 178
 missing value example, 186–188
 multiple groups, 194
 one-variable grouped aggregation, 176–177
 overview of, 175
 saving without running **aggregate**, **transform**, or **filter** methods, 190–191
 selecting groups, 192
 summary/conclusion, 199–200
 transform, 184–188
 working with **multiIndex**, 195–199
z-score example of transforming data, 184–186

Groups
 iterating through, 192–194
 selecting, 191–192
 working with multiple, 194

Guido, Sarah, 241

H

Hendryx-Parker, Calvin, 387

hexbin plot
 bivariate statistics in **seaborn**, 87–88
plt.hexbin function, 113–114

Hierarchical clustering
 average cluster algorithm, 353–354

centroid cluster algorithm, 353–354
 complete cluster algorithm, 352
 manually setting threshold for, 355–356
 overview of, 351–352
 single cluster algorithm, 352–353
 ward cluster algorithm, 354–355

Histograms
 creating using **plot.hist** functions, 111
 of model residuals, 323
 for univariate statistics in **matplotlib**, 73–74
 for univariate statistics in **seaborn**, 79–83

I

Ibis, 361

id, unique identifiers, 220

IDEs (integrated development environments), Python, 382

if, 433–434

ignore_index parameter, after concatenation, 149–150

iloc
 indexing rows or columns, 11
Series attributes, 35
 subsetting rows and columns, 21–23
 subsetting rows by number, 13–14

Importing (import). *See also* **Exporting/importing data**
itertools library, 410–411
 libraries, 391–392
 loading first data set, 4–5
matplotlib library, 66–72
pandas, 415

Indemics (Interactive Epidemic Simulation)
 data set, 208

Indices
 beginning and ending indices in ranges, 399
 concatenate columns with different indices, 153–154
 concatenate rows with different indices, 151–153

Indices (*continued*)

- date ranges, 267–268
- issues with absolute, 22
- out of bounds notification, 138
- reindexing as source of missing values, 209–210
- subsetting columns by index position
 - break, 8
- subsetting date based on, 263–266
- subsetting rows by index label, 11–13
- working with `multiIndex`, 195–199

`inplace` parameter, functions and methods, 49–50

Installation

- of Anaconda, 373–374
- from command line, 377–378
- Python packages, 374

Integers (`int`/`int64`)

- converting to `string`, 220–221
- vectors with integers (scalars), 40

integrated development environments (IDEs), 382

Interactive Epidemic Simulation (Indemics) data set, 196

Interpolation, in filling missing data, 213–214

IPython (`ipython`)

- `ipython` command, 381–382
- magic commands, 427

Iteration. *See* Loops (for loop)

`iTerm2`, 377

`itertools` library, 410–411

J

JavaScript Objectd notation, 59–62

`join`

- merges and, 160
- string methods, 234–235

`jointplot`, creating `seaborn` scatterplot, 85–88

JSON data, 59–62

Jupyter, 360

`jupyter` command, 382

JupyterCon, 364

Jupyter Days, 364

K

`KaplanMeierFitter`, `lifelines` library, 312–313

KDE plot, of bivariate statistics, 89–90

`keep_default_na` parameter, specifying NaN values, 205

Kelleher, Adam, 241

Kelleher, Andrew, 241

Keys, creating `DataFrame`, 32–33

Key–value pairs, 397–398

Key–value stores, 408

Keywords

- `lambda` keyword, 142
- passing keyword argument, 134–135

k -fold cross validation, 329–333

k -means

- clustering, 345–351
- using PCA, 349–351

`**kwargs`, 408

L

L1 regularization, 337–338, 341

L2 regularization, 338–341

`lambda` functions, applying, 141–142

Lander, Jared, 241

LASSO regression, 337–338, 341

Leap years/leap seconds, 278

Learning resources, for self-directed learners, 363–365

Libraries. *See also* by individual types

- importing, 391–392
- performance libraries, 360

`lifelines` library, 311–313

- `CoxPHFitter` class, 314–315
- `KaplanMeierFitter` class, 312–313

Linear regression models. *See also* GLM (generalized linear models)

- with categorical variables, 289–293

Linear regression models. See also GLM (generalized linear models) (*continued*)

- cross-validation, 341–343
- elastic net, 340–341
- LASSO regression regularization, 337–338
- model diagnostics, 324–327
- multiple regression, 287–289
- one-hot encoding in, 294–295
- R^2 (coefficient of determination)
 - regression score function, 332
- reasons for regularization, 335–337
- replicating results in R, 444–446
- residuals, 320–322
- restoring labels in `sklearn` models, 293
- ridge regression, 338–340
- simple linear regression, 283–287
- `sklearn` library for multiple regression, 288–289
- `sklearn` library for simple linear regression, 285–287
- `statsmodels` library for multiple regression, 287–288
- `statsmodels` library for simple linear regression, 284–285
- summary/conclusion, 296

Line breaks, 393–394

Linux

- command line, 378
- installing Anaconda, 373–374
- running `python` and `ipython` commands, 382
- viewing working directory, 383

List comprehension, 158–160

Lists (`list`)

- comprehensions and, 403–404
- converting generator to, 16–17, 409–410
- creating `Series`, 31–32
- of data types, 219–220
- loading multiple files using comprehension, 158–160
- loading multiple files using list comprehension, 158–160
- looping, 401–402

- multiple assignment, 413–414
- overview of, 395–396
- single value returns, 9–10

`lmp1ot`

- creating scatterplots, 85
- with hue parameter, 96–97

Loading data

- `datetime` data, 253–254
- as source of missing data, 205–206

`loc`

- indexing rows or columns, 11–13
- `Series` attributes, 35
- subsetting rows and columns, 21–23
- subsetting rows or columns, 15–16

Logic, three-valued, 203–204

Logistic regression

- example of, 435–441
- overview of, 297–304
- replicating results in R, 446–447
- `sklearn` library for, 300–304
- `statsmodels` library for, 299–300
- working with GLM models, 328–329

`logit` function, performing logistic regression, 299–300

Loops (for `loop`)

- comprehensions and, 403–404
- overview of, 401–402
- through groups, 192–194
- through lists, 401–402

M

Mac

- command line, 377–378
- installing Anaconda, 373
- `pwd` command for viewing working directory, 383
- running `python` and `ipython` commands, 382

Machine learning models, 285, 361–362

Machine Learning Operations (MLOps), 362

Many-to-many merges, 163–166

Many-to-one merges, 163

- Markham, Kevin, 422
- `match`, pattern matching, 240–243
- `matplotlib` library
 - axes subplots, 67–71
 - bivariate statistics, 74–76
 - figure anatomy, 71–72
 - figure objects, 67–71
 - multivariate statistics, 76–78
 - overview of, 66–72
 - statistical graphics, 72–73
 - univariate statistics, 73–74
- Matrices, 331–333, 415–416
- Mean (`mean`)
 - custom functions, 180–181
 - group calculations involving multiple variables, 191
 - grouped means, 23–26
 - `numpy` library, 179
 - `Series` in identifying, 37–38
- Meetups, 363
- `melt` function
 - converting wide data into tidy data, 118–120
 - line breaks, 393–394
 - rows and columns both containing variables, 126–127
- Merges (`merge`)
 - many-to-many, 163–166
 - many-to-one, 163
 - of multiple data sets, 160–166
 - one-to-one, 162–163
 - as source of missing data, 206–207
- Methods
 - built-in aggregation methods, 178–179
 - chaining, 423–425
 - class, 418
 - `datetime`, 259–261
 - export, 62–63
 - `Series`, 35–37
 - string, 233–236
- Miniconda, 374
- Mirjalili, Vahid, 241
- Missing data (`NaN` values)
 - built-in `Na` value, 218
 - calculations with, 215–216
 - cleaning, 212–215
 - concatenation and, 148–149, 153
 - date range for filling in, 272–273
 - dropping, 214–215
 - fill forward or fill backward, 212–213
 - finding and counting, 210–212
 - interpolation in filling, 213–214
 - loading data as source of, 205–206
 - merged data as source of, 206–207
 - overview of, 203
 - recoding or replacing (`fillna` method), 212
 - reindexing causing, 209–210
 - sources of, 205–210
 - specifying with `na_values` parameter, 205–206
 - summary/conclusion, 218
 - transform example, 186–188
 - user input creating, 207–208
 - what is a `NaN` value, 203–204
 - working with, 210–216
- MLOps (Machine Learning Operations), 362
- Model diagnostics
 - comparing multiple models, 324–329
 - k*-fold cross validation, 329–333
 - overview of, 319
 - q–q plots, 322–324
 - residuals, 319–324
 - summary/conclusion, 334
 - working with GLM models, 327–329
 - working with linear models, 324–327
- Models
 - data, 281–282
 - generalized linear (*See* GLM (generalized linear models))
 - linear (*See* Linear models)
- Month, extracting date components from `datetime` object, 254–257
- Müller, Andreas, 241
- Multiple assignment, 413–414

Multiple regression
 with categorical variables, 289–293
 overview of, 287
 residuals, 320–322
`sklearn` library for, 288–289
`statsmodels` library for, 287–288

Multivariate statistics
 in `matplotlib`, 76–78
 in `seaborn`, 94–99

N

`na_filter` parameter, specifying NaN values, 205–206

Name, subsetting columns by, 7–8

NaN. *See* Missing data (NaN values)

Na value, missing data with built-in, 218

`na_values` parameter, specifying NaN values, 205–206

`ndarray`
 restoring labels in `sklearn` models, 293
`Series` similarity with, 35–36
 working with matrices and arrays, 415–416

Negative binomial regression, 306–308, 448–449
 replicating results in R, 448–449

Negative numbers, slicing values from end of container, 230–231

New York ACS logistic regression example, 435–441

Normal distribution
 of data, 336
 q-q plots and, 322–324

Normalization, data, 169–173

`numba` library
 performance-related libraries, 360
 timing execution of statements or expressions, 360
`vectorize` decorator from, 140–141

Numbers (`numeric`)
 converting variables to numeric values, 221–225
 formatting number strings, 238–239, 430–431

negative numbers, 230–231
`to_numeric` function, 222–225

`numpy` library
 broadcasting support, 44–45
 exporting/importing data, 53–55
 mean, 179
`ndarray`, 415–416
 performance and, 360
 restoring labels in `sklearn` models, 293
`Series` similarity with `numpy.ndarray`, 35
`sklearn` library taking `numpy` arrays, 286–287
 specifying `dtype` from, 220–221
`vectorize`, 140

`nunique` method, grouped frequency counts, 27

O

Object-oriented languages, 417

Objects
 classes, 417–418
 converting to `datetime`, 250–253
`datetime`, 249–250
 figure, plotting, 67–71
 lists as, 395–396
 plots and plotting using Pandas objects, 111–115

Observational units
 across multiple tables, 154–160
 in a table, 169–173

Odds ratios, performing logistic regression, 300

Offsets, frequency, 268–269

One-to-one merges, 162–163

OSX. *See* Mac

Overdispersion of data, negative binomial regression for, 306–308, 448–449

P

Packages
 benefits of isolated environments, 385–386

- Packages (*continued*)
 - Installing, 389–390
 - updating, 390
- `pairgrid`, bivariate statistics, 93–94
- Pairwise relationships (`pairplot`)
 - bivariate statistics, 93–94
 - with hue parameter, 98
- `pandera`, 361
- Panel, 362
- Parameters
 - arbitrary function parameters, 407–408
 - default function parameters, 407
 - functions taking, 406–407
- passing/reassigning values, 395–396
- `patsy` library, 331–333
- Patterns. *See also* Regular expressions (`regex`)
 - compiling, 246–247
 - matching, 240–243
 - substituting, 245–246
- PCA (principal component analysis), 347–351
- `pd`
 - alias for `pandas`, 5
 - reading `pickle` data, 53–54
- PEP8 (Python Enhancement Proposal 8), 393
- Performance
 - avoiding premature optimization, 360
 - profiling code, 360
 - timing your code, 360, 427–428
- `pickle` data, 53–54
- Pipeline, 294–295
- `Pipenv`, 387–388
- `pip install`, 374, 389–390
- Pivot/unpivot
 - columns containing multiple variables, 122–126
 - converting wide data into tidy data, 119–120
 - keeping multiple columns fixed, 120–122
 - rows and columns both containing variables, 127–128
- Placeholders, formatting strings, 238, 430
- Plots/plotting (`plot`)
 - basic plots, 27–28
 - bivariate statistics in `matplotlib`, 74–76
 - bivariate statistics in `seaborn`, 83–94
 - concept map for, 371
 - creating boxplots (`plot.box`), 113–115
 - creating density plots (`plot.kde`), 111–112
 - creating scatterplots (`plot.scatter`), 112–113
 - linear regression residuals, 320–322
 - `matplotlib` library, 66–72
 - multivariate statistics in `matplotlib`, 76–78
 - multivariate statistics in `seaborn`, 94–99
 - overview of, 65
 - Pandas objects and, 111–115
 - q–q plots, 322–324
 - `seaborn` library, 78
 - statistical graphics, 72–73
 - summary/conclusion, 115
 - themes and styles in `seaborn`, 105–108
 - univariate statistics in `matplotlib`, 73–74
 - univariate statistics in `seaborn`, 79–83
- `PLOT_TYPE` functions, 111
- `plt.hexbin` function, 113–114
- Podcast resources, for self-directed learners, 364–365
- Point representation, Anscombe’s data set, 67
- `poisson` function, in `statsmodels` library, 304–306
- Poisson regression
 - negative binomial regression as alternative to, 306–308, 448–449
 - overview of, 304
 - replicating results in R, 447–449
 - `statsmodels` library for, 304–306
- Polars, 360
- Principal component analysis (PCA), 347–351
- Project templates, 379, 383
- Pryke, Benjamin, 422
- PyCon conference, 364

PyData, 364
 pyenv, 374
 Pyenv, 387–388
 pyjani tor, 361
 Python
 Anaconda distribution, 385
 assert, 166
 command line and text editor, 381
 comparing Pandas types with, 7
 conferences, 364
 enhanced features in Pandas, 3
 IDEs (integrated development environments), 382
 ipython command, 381–382
 jupyter command, 382
 as object-oriented languages, 417
 running from command line, 377–378
 scientific computing stack, 350
 ways to use, 381–382
 working with objects, 5
 as zero-indexed languages, 399
 Python Enhancement Proposal 8 (PEP8), 393

Q

q-q plots, model diagnostics, 322–324

R

random-state method, directly changing columns, 47–48
 range, 409–410
 Ranges (range)
 beginning and ending indices, 399
 date ranges, 266–269
 filling in missing values, 272–273
 overview of, 409–411
 passing range of values, 395–396
 subsetting columns, 16–18
 Raschka, Sebastian, 241
 R ecosystem, 362
 replicating results in, 443–449

Regex. *See* Regular expressions (regex)
 regplot, creating scatterplot, 83–85
 Regression
 keeping labels in sklearn models, 293
 LASSO regression regularization, 337–338
 logistic regression, 297–304, 446–447
 more GLM options, 308–309
 multiple regression, 287–289
 negative binomial regression, 306–308, 448–449
 New York ACS example, 435–441
 Poisson regression, 304–308, 447–449
 reasons for regularization, 335–337
 ridge regression regularization, 338–340
 simple linear regression, 283–287
 sklearn library for logistic regression, 300–304
 sklearn library for multiple regression, 288–289
 sklearn library for simple linear regression, 285–287
 statsmodels library for logistic regression, 299–300
 statsmodels library for multiple regression, 287–288
 statsmodels library for Poisson regression, 304–306
 statsmodels library for simple linear regression, 284–285
 Regular expressions (RegEx)
 functions in re, 240
 overview of, 239
 pattern compilation, 246–247
 pattern matching, 240–243
 pattern substitution, 245–246
 regex library, 247
 special characters, 240
 syntax, special characters, and functions, 240
 Regularization
 cross-validation, 341–343
 elastic net, 340–341
 LASSO regression, 337–338

- Regularization (*continued*)
 - overview of, 335
 - reasons for, 335–337
 - ridge regression, 338–340
 - summary/conclusion, 343
 - reindex method, reindexing as source of missing values, 209–210
 - re module, 240–243, 247
 - Resampling, `datetime`, 276–278
 - Residuals, model diagnostics, 319–324
 - Residual sum of squares (RSS), 326–327
 - Resources, 363–365
 - Ridge
 - regression elastic net and, 341
 - regularization techniques, 338–340
 - R language, interface with (`to_feather` method), 56–57
 - Rows
 - concatenation generally, 145–147
 - concatenation with different indices, 151–153
 - methods of indexing, 11
 - multiple observational units in a table, 169–173
 - removing row numbers from output, 55
 - rows and columns both containing variables, 126–129
 - subsetting multiple, 13
 - subsetting rows and columns, 21–23
 - subsetting rows by index label, 11–13
 - subsetting rows by row number, 13–14
 - RSS (residual sum of squares), 326–327
 - Rug plots, for univariate statistics, 80–81
-
- ## S
-
- Scalars, 40
 - Scatterplots
 - for bivariate statistics, 74–75
 - `matplotlib` example, 69
 - for multivariate statistics, 77–78
 - `plot.scatter` function, 112–113
 - Scientific computing stack, 350
 - SciPy conference, 364
 - `scipy` library
 - hierarchical clustering, 351
 - performance libraries, 360
 - scientific computing stack, 359
 - Scripts
 - project templates for running, 383
 - running Python from command line, 377–378
 - `seaborn`
 - Anscombe’s quartet for data visualization, 65–66
 - bivariate statistics, 83–94
 - multivariate statistics, 94–99
 - overview of, 78
 - themes and styles, 105–108
 - `tips` data set, 187
 - `titanic` data set, 297–299
 - univariate statistics, 79–83
 - Searches. *See* Find
 - Semicolon (;), types of delimiters, 55
 - Serialization, serialize and save data in binary format, 53
 - Series
 - adding columns, 45–47
 - aggregation functions, 183–184
 - alignment and vectorization, 39–42
 - apply function(s) over, 133–135
 - attributes, 35
 - boolean subsetting, 36–39
 - categorical attributes or methods, 226
 - as class, 417–418
 - creating, 31–32
 - defined, 3
 - directly changing columns, 47–50
 - exporting/importing data, 53
 - exporting to Excel (`to_excel` method), 56
 - histogram, 111
 - methods, 35–37
 - overview of, 33–35
 - similarity with `ndarray`, 35–36
 - single value returns, 8–9
 - writing CSV files (`to_csv` method), 55

- SettingWithCopyWarning, 419–422
- Shape
 - DataFrame attributes, 5
 - Series attributes, 35
- Shape, in plotting, 97–98
- Shell scripts, running Python from
 - command line, 377–378
- Shiny for Python, 362
- Simple linear regression
 - overview of, 283
 - sklearn library, 285–287
 - statsmodels library, 284–285
- Single cluster algorithm, in hierarchical clustering, 352–353
- Siuba, 360
- Size, in plotting, 77–78
- size attribute, Series, 35
- sklearn library
 - defaults in, 302–304
 - importing PCA function, 347–348
 - keeping labels in sklearn models, 293
 - k-fold cross validation, 330–331
 - KMeans function, 345–347
 - for logistic regression, 300–304
 - logistic regression example, 439–441
 - for multiple regression, 288–289
 - one-hot encoding with, 294–295
 - for simple linear regression, 285–287
 - splitting data into training and testing sets, 335–336
 - transformer pipelines in, 294–295
- Slicing
 - colon (:) use in slicing syntax, 15, 399–400
 - columns, 18–21
 - string from beginning or to end, 232
 - strings, 230–231
 - strings incrementally, 232–233
 - subsetting columns, 15–16
 - subsetting multiple rows and columns, 22–23
 - values, 399–400
- snakevis, profiling code, 360
- sns.distplot, creating histograms, 81
- Sns.set_style function, 105–108
- Special characters, regular expressions, 240
- Split–apply–combine, 175
- splitlines method, strings, 235–236
- split method
 - split and add columns individually, 123–125
 - split and combine in single step, 125–126
- Spyder IDE, 382
- SQL
 - comparing Pandas to, 162
 - groupy compared with SQL GROUP BY, 175
- Square brackets ([])
 - getting first character of string, 230
 - list syntax, 395–396
- Statistical graphics
 - bivariate statistics in matplotlib, 74–76
 - bivariate statistics in seaborn, 83–94
 - matplotlib library, 66–72
 - multivariate statistics in matplotlib, 76–78
 - multivariate statistics in seaborn, 94–99
 - overview of, 72–73
 - seaborn library, 78
 - univariate statistics in matplotlib, 73–74
 - univariate statistics in seaborn, 79–83
- Statistics
 - basic plots, 27–28
 - grouped and aggregated calculations, 23–27
 - grouped frequency counts, 27
 - grouped means, 23–26
- statsmodels library
 - for logistic regression, 299–300
 - for multiple regression, 287–288
 - for Poisson regression, 304–306
 - for simple linear regression, 284–285
- Stocks/stock prices, 261–263
- Storage
 - of information in dictionaries, 396–398
 - lists for data storage, 395–396

str accessor, 123
 Streamlit, 362
strftime, for date formats, 252–253
Strings (string)
 accessing methods, 123
 converting values to, 220–221
 formatting, 236–239, 429–431
 getting last character in, 231–233
 methods, 233–236
 overview of, 229
 pattern compilation, 246–247
 pattern matching, 240–243
 pattern substitution, 245–246
 regular expressions (regex) and, 239–240, 247
 subset and slice, 229–231
 summary/conclusion, 247
str.replace, pattern substitution, 245–246
Styles, **seaborn**, 105–108
Subplot syntax, 68
Subsets/subsetting
 columns by index position break, 8
 columns by name, 7–8
 columns by range, 16–18
 columns generally, 21–23
 columns using slicing syntax, 15–16
 data by dates, 263–266
 DataFrame boolean subsetting, 43
 lists, 395–396
 modifying with
 SettingWithCopyWarning, 419–420
 multiple rows, 13
 rows by index label, 11–13
 rows by row number, 13–14
 rows generally, 21–23
 strings, 229–231
 tuples, 396
sum
 cumulative (**cumsum**), 199
 custom functions, 180
Summarization. *See* Aggregation (or aggregate)

Survival analysis, 311–317
 Cox proportional hazards model, 314–316
 data for, 311–312
 Kaplan Meier curves, 312–314
 overview, 311
 summary/conclusion, 317
SyPy, 359

T

Tables
 observational units across multiple, 154–160
 observational units in, 169–173
Tab separated values (TSV), 55, 253
tail, returning last row, 13
T attribute, **Series**, 35
Templates, project, 379, 383
Terminal application, Mac, 377
Text. *See also* Characters; **Strings (string)**
 function documentation (**docstring**), 132
 overview of, 229
Themes, **seaborn**, 105–109
Three-valued logic, 203–204
Tidy data
 columns containing multiple variables, 122–126
 columns containing values not variables, 118–122
 concept map for, 372
 data assembly, 167
 data normalization, 169–173
 definition of, 117
 keeping multiple columns fixed, 120–122
 keeping one column fixed, 118–120
 overview of, 117
 rows and columns both containing variables, 126–129
 split and add columns individually, 123–125

Tidy data (*continued*)
 split and combine in single step, 125–126
 summary/conclusion, 129

`tidyverse`, 360

Time. *See* `datetime`

`TimeDeltaIndex`, 265–266

`timedelta` object
 date calculations, 257–258
 subsetting date based data, 265–266

`timeit` function, timing execution of
 statements or expressions, 360, 427–428

Time zones, 278–279

`tips` data set, `seaborn` library, 187, 283

`titanic` data set, 297–299

`to_csv` method, 55

`to_datetime` function, 250–253

`to_dict` method, 58–59

`to_excel` method, 56

`to_feather` method, 57

`to_numeric` function, 222–225

Transform (`transform`)
 applying to data, 323–324
 missing value example of transforming
 data, 186–188
 overview of, 184
z-score example of transforming data,
 184–186

Transformer pipelines, 294–295

True, 434

TSV (tab separated values), 55, 253

Tuples (`tuple`), 396

2D density plot, 88–89

`type` function, working with Python
 objects, 5

U

Unique identifiers, 220

Univariate statistics
 in `matplotlib`, 73–74
 in `seaborn`, 79–83

Updates, package, 390

User input, as source of missing data,
 207–208

V

`value_counts` method, 27, 211–212

Values (`value`)
 columns containing values not variables
 (*See* Columns, with values not variables)
 converting to strings, 220–221
 creating `DataFrame` values, 34
 directly changing columns, 47–50
 dropping, 52
 functions taking, 406–407
 missing (*See* Missing data (NaN values))
 multiple assignment of list of, 413–414
 passing/reassigning, 395–396
 replacing with
 `SettingWithCopyWarning`, 420–421
 `Series` attributes, 35
 shifting `datetime` values, 270–276
 slicing, 399–400

VanderPlas, Jake, 359

Variables
 adding covariates to linear models, 324
 bi-variable statistics (*See* Bivariate
 statistics)
 calculations involving multiple, 191
 columns containing multiple (*See*
 Columns, with multiple variables)
 columns containing values not variables
 (*See* Columns, with values not variables)
 converting to numeric values, 221–225
 multiple assignment, 413–414
 multiple linear regression with three
 covariates, 320–322
 multiple variable statistics (*See*
 Multivariate statistics)
 one-variable grouped aggregation,
 176–177
 rows and columns both containing,
 126–129
 single variable statistics (*See* Univariate
 statistics)
`sklearn` library used with categorical
 variables, 291–293
`statsmodels` library used with
 categorical variables, 289–291

Vectors (`vectorize`)

- applying vectorized function, 138–141
- with common index labels (automatic alignment), 41–42
- `DataFrame` alignment and vectorization, 44–45
- `Series` alignment and vectorization, 39–42
- `Series` referred to as vectors, 35
- timing, 427–428
- using `numba` library, 140–141
- using `numpy` library, 140
- vectors of different length, 40–41
- vectors of same length, 39–40
- vectors with integers (scalars), 40

Violin plots

- bivariate statistics, 91–93
- creating scatterplots, 91–93
- with hue parameter, 96–97

Visualization

- Anscombe’s quartet for data visualization, 65–66
- using plots for, 27–28
- value of, 65–66

Voilà, 362

W

Ward cluster algorithm, in hierarchical clustering, 354–355

Wickham, Hadley, 99, 117

“Wide” data, converting into tidy data, 118–120

Windows

- Anaconda command prompt, 381–382
- `cd` command for viewing working directory, 383
- command line, 377
- installing Anaconda, 373

X

`xarray` library, 359

XGBoost, 361

Y

Year, extracting date components from `datetime` object, 254–257

Z

Zero-indexed languages, 399

z-score, transforming data, 184–186