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.

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2022948110

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

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 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 `'sklean'` 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 appendices 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 appendices 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.

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 Ahmadia 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 though 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 form 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

4

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 book[3] 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 `DataFrame`s 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 `DataFrame`s 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 _ understore.

```
# 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 objects[4]
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()` method[5]), 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

Numbers

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z