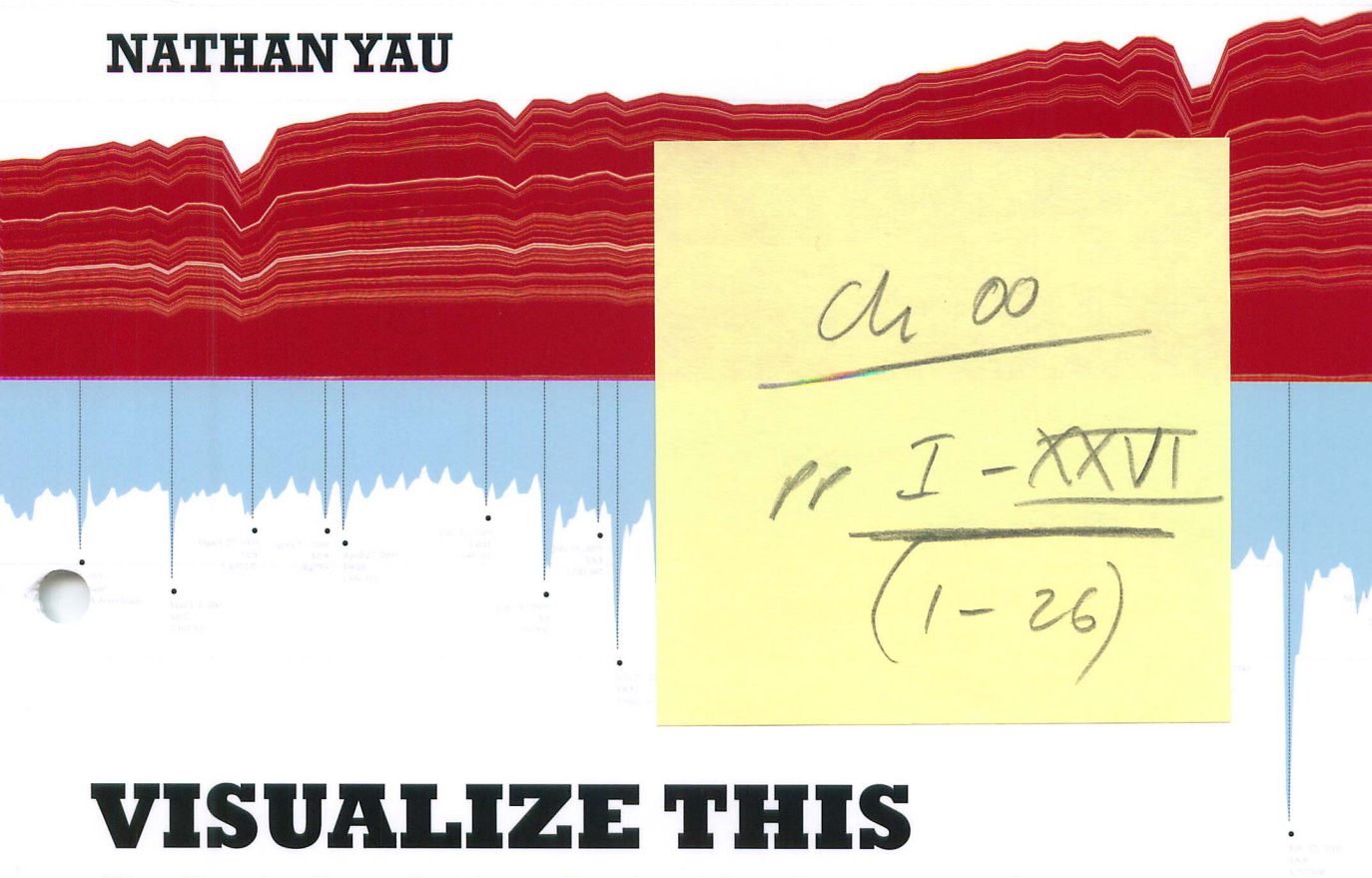
NATHAN YAU



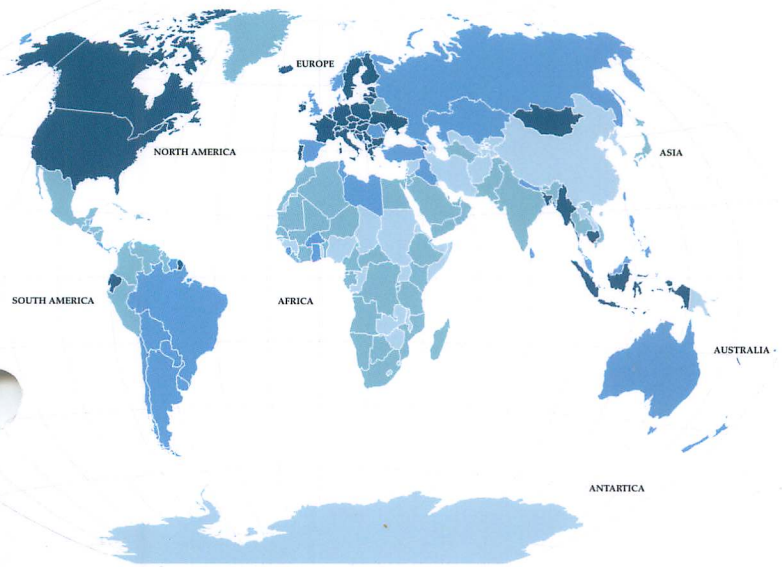
Ch 00

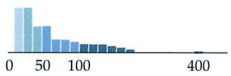
rr I - XXVI

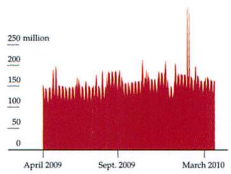
(1 - 26)

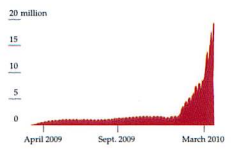
VISUALIZE THIS

The FlowingData Guide to Design, Visualization, and Statistics

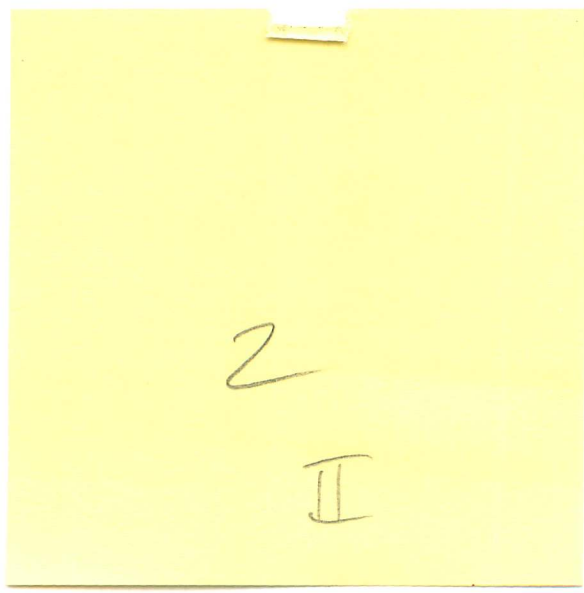




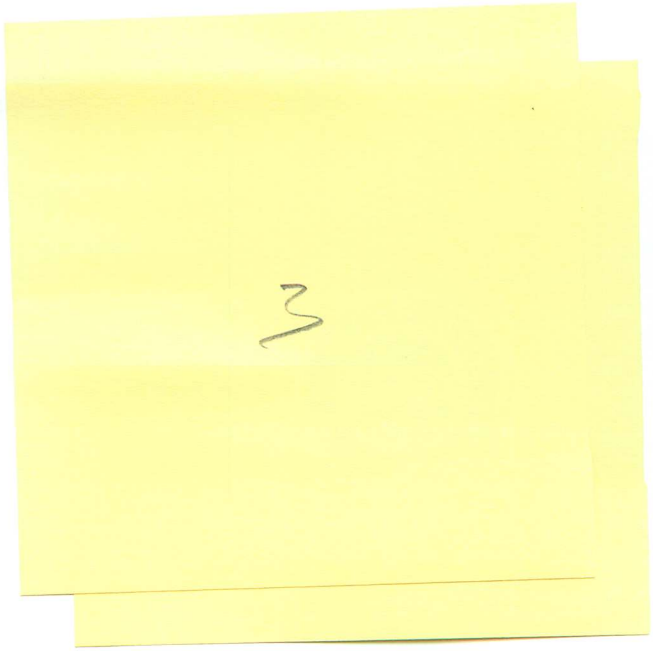


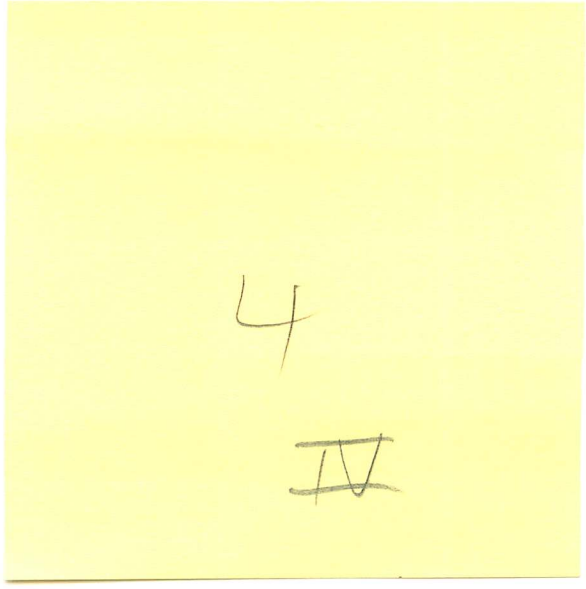


Sources: Mozilla, Internet World Stats; Design by: Nathan Yau, FlowingData


2
II

Visualize This




4
IV

Visualize This

The FlowingData Guide to Design, Visualization, and Statistics

Nathan Yau

Wiley Publishing, Inc.

Visualize This: The FlowingData Guide to Design, Visualization, and Statistics

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Published by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

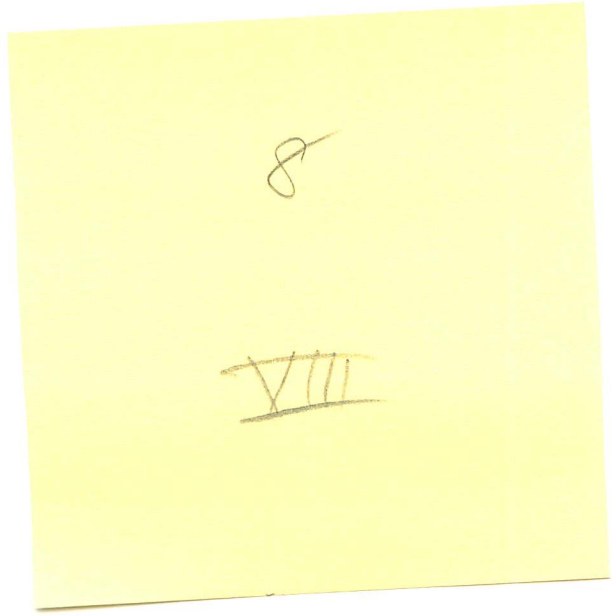
ISBN: 978-0-470-94488-2
ISBN: 978-1-118-14024-6 (ebk)
ISBN: 978-1-118-14026-0 (ebk)
ISBN: 978-1-118-14025-3 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2

Library of Congress Control Number: 2011928441

To my loving wife, Bea


8
VIII

About the Author

Since 2007, NATHAN YAU has written and created graphics for FlowingData, a site on visualization, statistics, and design. Working with groups such as *The New York Times*, CNN, Mozilla, and SyFy, Yau believes that data and information graphics, while great for analysis, are also perfect for telling stories with data.

Yau has a master's degree in statistics from the University of California, Los Angeles, and is currently a Ph.D. candidate with a focus on visualization and personal data.

About the Technical Editor

KIM REES is co-founder of Periscopic, a socially conscious information visualization firm. A prominent individual in the visualization community, Kim has over seventeen years of experience in the interactive industry. She has published papers in the *Parsons Journal of Information Mapping* and the InfoVIS 2010 Proceedings, and has spoken at the O'Reilly Strata Conference, WebVisions, AIGA Shift, and Portland Data Visualization. Kim received her bachelor of arts in Computer Science from New York University. Periscopic has been recognized in CommArts Insights, Adobe Success Stories, and awarded by the VAST Challenge, CommArts Web Picks, and the *Communication Arts Interactive Annual*. Recently, Periscopic's body of work was nominated for the Cooper-Hewitt National Design Awards.

Credits

Executive Editor
Carol Long

Senior Project Editor
Adaobi Obi Tulton

Technical Editor
Kim Rees

Senior Production Editor
Debra Banninger

Copy Editor
Apostrophe Editing Services

Editorial Director
Robyn B. Siesky

Editorial Manager
Mary Beth Wakefield

Freelancer Editorial Manager
Rosemarie Graham

Marketing Manager
Ashley Zurcher

Production Manager
Tim Tate

Vice President and Executive Group Publisher
Richard Swadley

Vice President and Executive Publisher
Barry Pruett

Associate Publisher
Jim Minatel

Project Coordinator, Cover
Katie Crocker

Compositor
Maureen Forys,
Happenstance Type-O-Rama

Proofreader
Nancy Carrasco

Indexer
Robert Swanson

Cover Image
Nathan Yau

Cover Designer
Ryan Sneed

Acknowledgments

THIS BOOK would not be possible without the work by the data scientists before me who developed and continue to create useful and open tools for everyone to use. The software from these generous developers makes my life much easier, and I am sure they will keep innovating.

My many thanks to FlowingData readers who helped me reach more people than I ever imagined. They are one of the main reasons why this book was written.

Thank you to Wiley Publishing, who let me write the book that I wanted to, and to Kim Rees for helping me produce something worth reading.

Finally, thank you to my wife for supporting me and to my parents who always encouraged me to find what makes me happy.

Contents

Introduction . . . . . *xv*

1 Telling Stories with Data **1**
More Than Numbers 2
What to Look For 8
Design 13
Wrapping Up 20

2 Handling Data **21**
Gather Data. 22
Formatting Data 38
Wrapping Up 52

3 Choosing Tools to Visualize Data. **53**
Out-of-the-Box Visualization. 54
Programming 62
Illustration 76
Mapping 80
Survey Your Options. 88
Wrapping Up 89

4 Visualizing Patterns over Time **91**
What to Look for over Time 92
Discrete Points in Time. 93
Continuous Data 118
Wrapping Up 132

5 Visualizing Proportions **135**
What to Look for in Proportions. 136
Parts of a Whole 136

Proportions over Time 161
Wrapping Up 178

6 Visualizing Relationships 179
What Relationships to Look For 180
Correlation 180
Distribution 200
Comparison 213
Wrapping Up 226

7 Spotting Differences 227
What to Look For 228
Comparing across Multiple Variables 228
Reducing Dimensions 258
Searching for Outliers 265
Wrapping Up 269

8 Visualizing Spatial Relationships 271
What to Look For 272
Specific Locations 272
Regions 285
Over Space and Time 302
Wrapping Up 325

9 Designing with a Purpose 327
Prepare Yourself 328
Prepare Your Readers 330
Visual Cues 334
Good Visualization 340
Wrapping Up 341

Index *343*

Introduction

Data is nothing new. People have been quantifying and tabulating things for centuries. However, while writing for FlowingData, my website on design, visualization, and statistics, I've seen a huge boom in just these past few years, and it keeps getting better. Improvements in technology have made it extremely easy to collect and store data, and the web lets you access it whenever you want. This wealth in data can, in the right hands, provide a wealth of information to help improve decision making, communicate ideas more clearly, and provide a more objective window looking in at how you look at the world and yourself.

A significant shift in release of government data came in mid-2009, with the United States' launch of Data.gov. It's a comprehensive catalog of data provided by federal agencies and represents transparency and accountability of groups and officials. The thought here is that you should know how the government spends tax dollars. Whereas before, the government felt more like a black box. A lot of the data on Data.gov was already available on agency sites scattered across the web, but now a lot of it is all in one place and better formatted for analysis and visualization. The United Nations has something similar with UNdata; the United Kingdom launched Data.gov.uk soon after, and cities around the world such as New York, San Francisco, and London have also taken part in big releases of data.

The collective web has also grown to be more open with thousands of Application Programming Interfaces (API) to encourage and entice developers to do something with all the available data. Applications such as Twitter and Flickr provide comprehensive APIs that enable completely different user interfaces from the actual sites. API-cataloging site ProgrammableWeb reports more than 2,000 APIs. New applications, such as Infochimps and Factual, also launched fairly recently and were specifically developed to provide structured data.

At the individual level, you can update friends on Facebook, share your location on Foursquare, or tweet what you're doing on Twitter, all with a few clicks on a mouse or taps on a keyboard. More specialized applications enable you to log what you eat, how much you

weigh, your mood, and plenty of other things. If you want to track something about yourself, there is probably an application to help you do it.

With all this data sitting around in stores, warehouses, and databases, the field is ripe for people to make sense of it. The data itself isn't all that interesting (to most people). It's the information that comes out of the data. People want to know what their data says, and if you can help them, you're going to be in high demand. There's a reason that Hal Varian, Google's chief economist, says that statistician is the sexy job of the next 10 years, and it's not just because statisticians are beautiful people. (Although we are quite nice to look at in that geek chic sort of way.)

Visualization

One of the best ways to explore and try to understand a large dataset is with visualization. Place the numbers into a visual space and let your brain or your readers' brains find the patterns. We're good at that. You can often find stories that you might never have found with just formal statistical methods.

John Tukey, my favorite statistician and the father of exploratory data analysis, was well versed in statistical methods and properties but believed that graphical techniques also had a place. He was a strong believer in discovering the unexpected through pictures. You can find out a lot about data just by visualizing it, and a lot of the time this is all you need to make an informed decision or to tell a story.

For example, in 2009, the United States experienced a significant increase in its unemployment rate. In 2007, the national average was 4.6 percent. In 2008, it had risen to 5.8 percent. By September 2009, however, it was 9.8 percent. These national averages tell only part of the story though. It's generalizing over an entire country. Were there any regions that had higher unemployment rates than others? Were there any regions that seemed to be unaffected?

The maps in Figure I-1 tell a more complete story, and you can answer the preceding questions after a glance. Darker-colored counties are areas that had relatively higher unemployment rates, whereas the lighter-colored counties had relatively lower rates. In 2009, you see a lot of regions with rates greater than 10 percent in the west and most areas in the east. Areas in the Midwest were not hit as hard (Figure I-2).

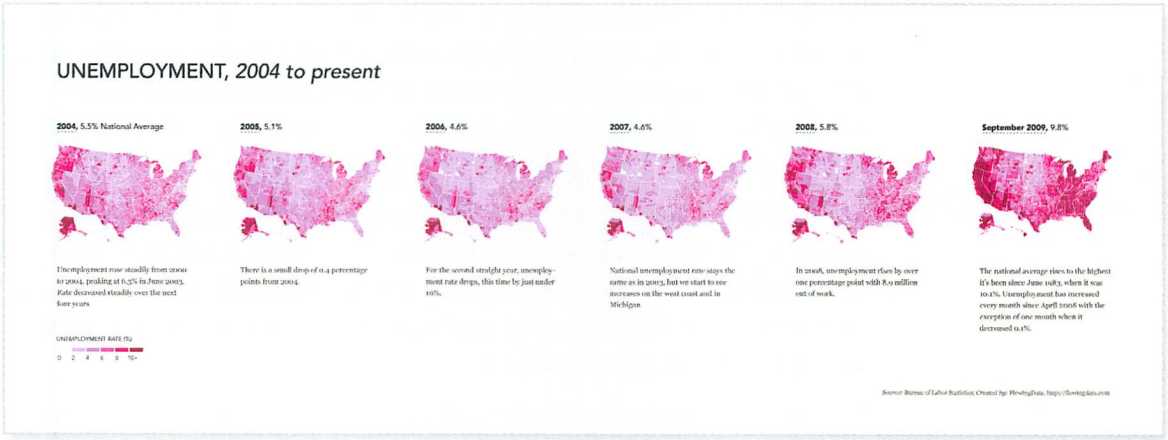


FIGURE I-1 Maps of unemployment in the United States from 2004 to 2009

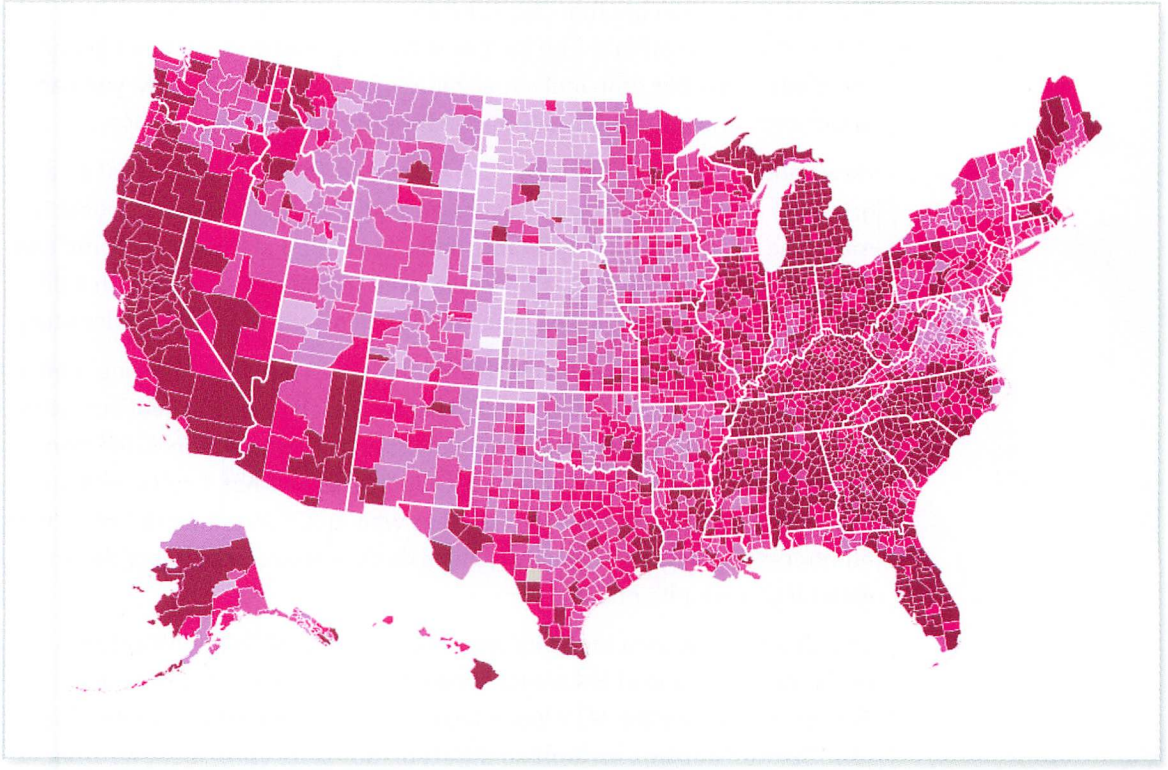


FIGURE I-2 Map of unemployment for 2009

You couldn't find these geographic and temporal patterns so quickly with just a spreadsheet, and definitely not with just the national averages. Also, although the county-level data is more complex, most people can still interpret the maps. These maps could in turn help policy makers decide where to allocate relief funds or other types of support.

The great thing about this is that the data used to produce these maps is all free and publicly available from the Bureau of Labor Statistics. Albeit the data was not incredibly easy to find from an outdated data browser, but the numbers are there at your disposal, and there is a lot sitting around waiting for some visual treatment.

The Statistical Abstract of the United States, for instance, exists as hundreds of tables of data (Figure I-3), but no graphs. That's an opportunity to provide a comprehensive picture of a country. Really interesting stuff. I graphed some of the tables a while back as a proof of concept, as shown in Figure I-4, and you get marriage and divorce rates, postal rates, electricity usage, and a few others. The former is hard to read and you don't get anything out of it other than individual values. In the graphical view, you can find trends and patterns easily and make comparisons at a glance.

News outlets, such as *The New York Times* and *The Washington Post* do a great job at making data more accessible and visual. They have probably made the best use of this available data, as related stories have come and passed. Sometimes data graphics are used to enhance a story with a different point of view, whereas other times the graphics tell the entire story.

Graphics have become even more prevalent with the shift to online media. There are now departments within news organizations that deal only with interactives or only graphics or only maps. *The New York Times*, for example, even has a news desk specifically dedicated to what it calls computer-assisted reporting. These are reporters who focus on telling the news with numbers. *The New York Times* graphics desk is also comfortable dealing with large amounts of data.

Visualization has also found its way into pop culture. Stamen Design, a visualization firm well known for its online interactives, has provided a Twitter tracker for the MTV Video Music Awards the past few years. Each year Stamen designs something different, but at its core, it shows what people are talking about on Twitter in real-time. When Kanye West had his little outburst during Taylor Swift's acceptance speech in 2009, it was obvious what people thought of him via the tracker.

Table 126. **Marriages and Divorces—Number and Rate by State: 1990 to 2007**

[**2,443.5 represents 2,443,500.** By place of occurence. See Appendix III]

State	Marriages [1]						Divorces [3]					
	Number (1,000)			Rate per 1,000 population [2]			Number (1,000)			Rate per 1,000 population [2]		
	1990	2000	2007	1990	2000	2007	1990	2000	2007	1990	2000	2007
U.S. [4]	**2,443.5**	**2,329.0**	**2,204.6**	**9.8**	**8.3**	**7.3**	**1,182.0**	**(NA)**	**(NA)**	**4.7**	**4.1**	**3.6**
Alabama	43.1	45.0	42.4	10.6	10.3	9.2	25.3	23.5	19.8	6.1	5.4	4.3
Alaska [5]	5.7	5.6	5.8	10.2	8.9	8.4	2.9	2.7	3.0	5.5	4.4	4.3
Arizona [5]	36.8	38.7	39.5	10.0	7.9	6.2	25.1	21.6	24.5	6.9	4.4	3.9
Arkansas	36.0	41.1	33.7	15.3	16.0	11.9	16.8	17.9	16.8	6.9	6.9	5.9
California	237.1	196.9	225.8	7.9	5.9	6.2	128.0	(NA)	(NA)	4.3	(NA)	(NA)
Colorado	32.4	35.6	29.2	9.8	8.6	6.0	18.4	(NA)	21.2	5.5	(NA)	4.4
Connecticut	26.0	19.4	17.3	7.9	5.9	4.9	10.3	6.5	10.7	3.2	2.0	3.1
Delaware	5.6	5.1	4.7	8.4	6.7	5.5	3.0	3.2	3.9	4.4	4.2	4.5
District of Columbia	5.0	2.8	2.1	8.2	5.4	3.6	2.7	1.5	1.0	4.5	3.0	1.6
Florida	141.8	141.9	157.6	10.9	9.3	8.6	81.7	81.9	86.4	6.3	5.3	4.7
Georgia	66.8	56.0	64.0	10.3	7.1	6.7	35.7	30.7	(NA)	5.5	3.9	(NA)
Hawaii	18.3	25.0	27.3	16.4	21.2	21.3	5.2	4.6	(NA)	4.6	3.9	(NA)
Idaho	14.1	14.0	15.4	13.9	11.0	10.3	6.6	6.9	7.4	6.5	5.4	4.9
Illinois	100.6	85.5	75.3	8.8	7.0	5.9	44.3	39.1	32.8	3.8	3.2	2.6
Indiana	53.2	34.5	51.2	9.6	5.8	8.1	(NA)	(NA)	(NA)	(NA)	(NA)	(NA)
Iowa	24.9	20.3	20.1	9.0	7.0	6.7	11.1	9.4	7.8	3.9	3.3	2.6
Kansas	22.7	22.2	18.6	9.2	8.3	6.7	12.6	10.6	9.2	5.0	4.0	3.3
Kentucky	49.8	39.7	33.6	13.5	10.0	7.9	21.8	21.6	19.7	5.8	5.4	4.6
Louisiana	40.4	40.5	32.8	9.6	9.3	7.6	(NA)	(NA)	(NA)	(NA)	(NA)	(NA)
Maine	11.9	10.5	10.1	9.7	8.3	7.7	5.3	5.8	5.9	4.3	4.6	4.5
Maryland	46.3	40.0	35.5	9.7	7.7	6.3	16.1	17.0	17.4	3.4	3.3	3.1
Massachusetts	47.7	37.0	38.4	7.9	6.0	6.0	16.8	18.6	14.5	2.8	3.0	2.2
Michigan	76.1	66.4	59.1	8.2	6.7	5.9	40.2	39.4	35.5	4.3	4.0	3.5
Minnesota	33.7	33.4	29.8	7.7	6.9	5.7	15.4	14.8	(NA)	3.5	3.1	(NA)
Mississippi	24.3	19.7	15.7	9.4	7.1	5.4	14.4	14.4	14.2	5.5	5.2	4.9
Missouri	49.1	43.7	39.4	9.6	7.9	6.7	26.4	26.5	22.4	5.1	4.8	3.8
Montana	6.9	6.6	7.1	8.6	7.4	7.4	4.1	2.1	3.6	5.1	2.4	3.7
Nebraska	12.6	13.0	12.4	8.0	7.8	7.0	6.5	6.4	5.5	4.0	3.8	3.1
Nevada	120.6	144.3	126.4	99.0	76.7	49.3	13.3	18.1	16.6	11.4	9.6	6.5
New Hampshire	10.5	11.6	9.4	9.5	9.5	7.1	5.3	7.1	5.1	4.7	5.8	3.9
New Jersey [5]	58.7	50.4	45.4	7.6	6.1	5.2	23.6	25.6	25.7	3.0	3.1	3.0
New Mexico [5]	13.3	14.5	11.2	8.8	8.3	5.7	7.7	9.2	8.4	4.9	5.3	4.3
New York [5]	154.8	162.0	130.6	8.6	8.9	6.8	57.9	62.8	55.9	3.2	3.4	2.9
North Carolina	51.9	65.6	68.1	7.8	8.5	7.5	34.0	36.9	37.4	5.1	4.8	4.1
North Dakota	4.8	4.6	4.2	7.5	7.3	6.6	2.3	2.0	1.5	3.6	3.2	2.4
Ohio	98.1	88.5	70.9	9.0	7.9	6.2	51.0	49.3	37.9	4.7	4.4	3.3
Oklahoma	33.2	15.6	26.2	10.6	4.6	7.3	24.9	12.4	18.8	7.7	3.7	5.2
Oregon	25.3	26.0	29.4	8.9	7.8	7.8	15.9	16.7	14.8	5.5	5.0	4.0
Pennsylvania	84.9	73.2	71.1	7.1	6.1	5.7	40.1	37.9	35.3	3.3	3.2	2.8
Rhode Island	8.1	8.0	6.8	8.1	8.0	6.4	3.8	3.1	3.0	3.7	3.1	2.8
South Carolina	55.8	42.7	31.4	15.9	10.9	7.1	16.1	14.4	14.4	4.5	3.7	3.3
South Dakota	7.7	7.1	6.2	11.1	9.6	7.7	2.6	2.7	2.4	3.7	3.6	3.1
Tennessee	68.0	88.2	65.6	13.9	15.9	10.6	32.3	33.8	29.9	6.5	6.1	4.9
Texas	178.6	196.4	179.9	10.5	9.6	7.5	94.0	85.2	79.5	5.5	4.2	3.3
Utah	19.4	24.1	22.6	11.2	11.1	8.6	8.8	9.7	8.9	5.1	4.5	3.4
Vermont	6.1	6.1	5.3	10.9	10.2	8.6	2.6	5.1	2.4	4.5	8.6	3.8
Virginia	71.0	62.4	58.0	11.4	9.0	7.5	27.3	30.2	29.5	4.4	4.3	3.8
Washington	46.6	40.9	41.8	9.5	7.0	6.5	28.8	27.2	28.9	5.9	4.7	4.5
West Virginia	13.0	15.7	13.0	7.2	8.7	7.2	9.7	9.3	9.0	5.3	5.2	5.0
Wisconsin	38.9	36.1	32.2	7.9	6.8	5.8	17.8	17.6	16.1	3.6	3.3	2.9
Wyoming	4.9	4.9	4.8	10.7	10.3	9.3	3.1	2.8	2.9	6.6	5.9	5.5

NA Not available. [1] Data are counts of marriages performed, except as noted. [2] Based on total population residing in area: population enumerated as of April 1 for 1990 and 2000; estimated as of July 1 for all other years. [3] Includes annulments. [4] U.S. total for the number of divorces is an estimate which includes states not reporting. Beginning 2000, divorce rates based solely on the combined counts and populations for reporting states and the District of Columbia. The collection of detailed data of marriages and divorces was suspended in January 1996. [5] Some figures for marriages are marriage licenses issued.

Source: U.S. National Center for Health Statistics, National Vital Statistics Reports (NVSR), *Births, Marriages, Divorces, and Deaths: Provisional Data for 2007*, Vol. 56, No. 21, July 14, 2008 and prior reports.

FIGURE I-3 Table from the Statistical Abstract of the United States

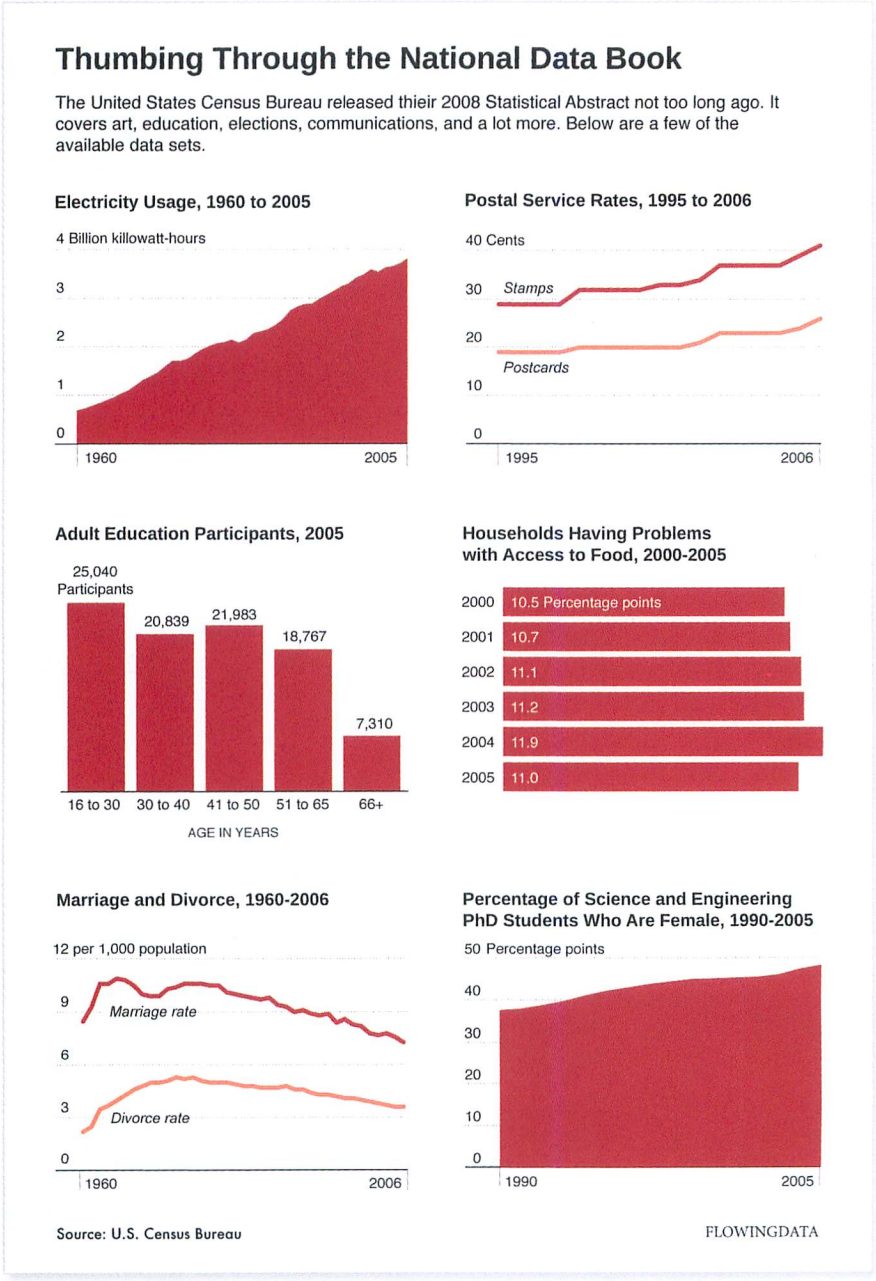


FIGURE I-4 A graphical view of data from the Statistical Abstract of the United States

At this point, you enter a realm of visualization less analytical and more about feeling. The definition of visualization starts to get kind of fuzzy. For a long time, visualization was about quantitative facts. You should recognize patterns with your tools, and they should aid your analysis in some way. Visualization isn't just about getting the cold hard facts. Like in the case of Stamen's tracker, it's almost more about the entertainment factor. It's a way for viewers to watch the awards show and interact with others in the process. Jonathan Harris' work is another great example. Harris designs his work, such as *We Feel Fine* and *Whale Hunt*, around stories rather than analytical insight, and those stories revolve around human emotion over the numbers and analytics.

Charts and graphs have also evolved into not just tools but also as vehicles to communicate ideas—and even tell jokes. Sites such as GraphJam and Indexed use Venn diagrams, pie charts, and the like to represent pop songs or show that a combination of red, black, and white equals a Communist newspaper or a panda murder. Data Underload, a data comic of sorts that I post on FlowingData, is my own take on the genre. I take everyday observations and put it in chart form. The chart in Figure I-5 shows famous movie quotes listed by the American Film Institute. It's totally ridiculous but amusing (to me, at least).

So what is visualization? Well, it depends on who you talk to. Some people say it's strictly traditional graphs and charts. Others have a more liberal view where anything that displays data is visualization, whether it is data art or a spreadsheet in Microsoft Excel. I tend to sway more toward the latter, but sometimes find myself in the former group, too. In the end, it doesn't actually matter all that much. Just make something that works for your purpose.

▶ Find more Data Underload on FlowingData at `http://datafl.ws/underload`

Whatever you decide visualization is, whether you're making charts for your presentation, analyzing a large dataset, or reporting the news with data, you're ultimately looking for truth. At some point in time, lies and statistics became almost synonymous, but it's not that the numbers lie. It's the people who use the numbers who lie. Sometimes it's on purpose to serve an agenda, but most of the time it's inadvertent. When you don't know how to create a graph properly or communicate with data in an unbiased way, false junk is likely to sprout. However, if you learn proper visualization techniques and how to work with data, you can state your points confidently and feel good about your findings.

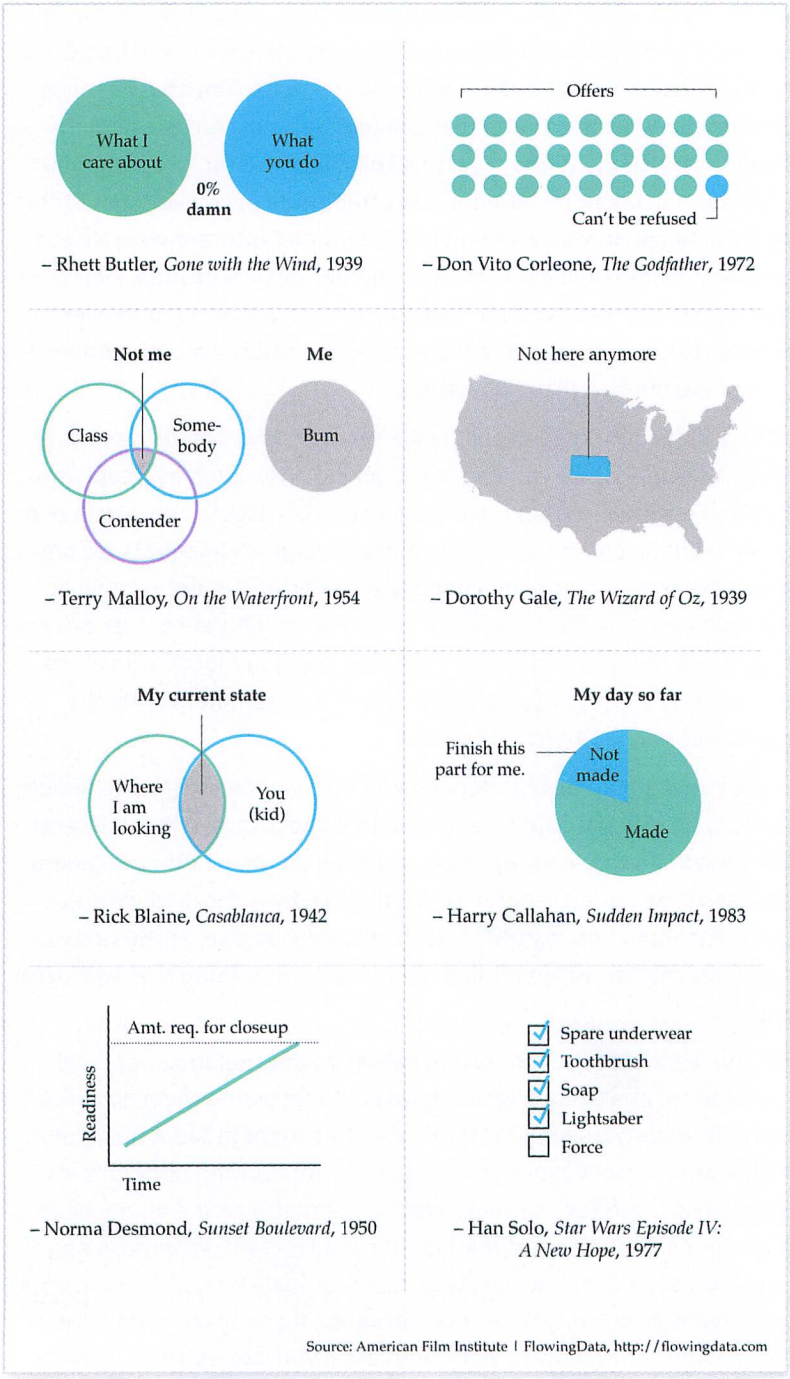


FIGURE I-5 Movie quotes in graph form

Learning Data

I got my start in statistics during my freshman year in college. It was a required introductory course toward my unrelated electrical engineering degree. Unlike some of the horror stories I've heard, my professor was extremely enthusiastic about his teaching and clearly enjoyed the topic. He quickly walked up and down the stairs of the lecture hall as he taught. He waved his hands wildly as he spoke and got students involved as he walked by. To this day, I don't think I've ever had such an excited teacher or professor, and it's undoubtedly something that drew me into the area of data and eventually what led to graduate school in statistics four years later.

Through all my undergraduate studies, statistics was data analysis, distributions, and hypothesis testing, and I enjoyed it. It was fun looking at a dataset and finding trends, patterns, and correlations. When I started graduate school though, my views changed, and things got even more interesting.

Statistics wasn't just about hypothesis testing (which turns out isn't all that useful in a lot of cases) and pattern-finding anymore. Well, no, I take that back. Statistics was still about those things, but there was a different feel to it. Statistics is about storytelling with data. You get a bunch of data, which represents the physical world, and then you analyze that data to find not just correlations, but also what's going on around you. These stories can then help you solve real-world problems, such as decreasing crime, improving healthcare, and moving traffic on the freeway, or it can simply help you stay more informed.

A lot of people don't make that connection between data and real life. I think that's why so many people tell me they "hated that course in college" when I tell them I'm in graduate school for statistics. I know you won't make that same mistake though, right? I mean, you're reading this book after all.

How do you learn the necessary skills to make use of data? You can get it through courses like I did, but you can also learn on your own through experience. That's what you do during a large portion of graduate school anyway.

It's the same way with visualization and information graphics. You don't have to be a graphic designer to make great graphics. You don't need a statistics PhD either. You just need to be eager to learn, and like almost everything in life, you have to practice to get better.

I think the first data graphics I made were in the fourth grade. They were for my science fair project. My project partner and I pondered (very deeply I am sure) what surface snails move on the fastest. We put snails on rough and smooth surfaces and timed them to see how long it took them to go a specific distance. So the data at hand was the times for different surfaces, and I made a bar graph. I can't remember if I had the insight to sort from least to greatest, but I do remember struggling with Excel. The next year though when we studied what cereal red flour beetles preferred, the graphs were a snap. After you learn the basic functionality and your way around the software, the rest is quite easy to pick up. If that isn't a great example of learning from experience, then I don't know what is. Oh, and by the way, the snails moved fastest on glass, and the red flour beetles preferred Grape Nuts, in case you were wondering.

This is basic stuff we're talking about here, but it's essentially the same process with any software or programming language you learn. If you've never written a line of code, R, many statisticians' computing environment of choice, can seem intimidating, but after you work through some examples, you start to quickly get the hang of things. This book can help you with that.

I say this because that's how I learned. I remember when I first got into more of the design aspects of visualization. It was the summer after my second year in graduate school, and I had just gotten the great news that I was going to be a graphics editor intern at *The New York Times*. Up until then, graphics had always been a tool for analysis (with the occasional science fair bar graph) to me, and aesthetics and design didn't matter so much, if at all. Data and its role in journalism didn't occur to me.

So to prepare, I read all the design books I could and went through a guide on Adobe Illustrator because I knew that's what *The New York Times* used. It wasn't until I actually started making graphics though when I truly started learning. When you learn by doing, you're forced to pick up what is necessary, and your skills evolve as you deal with more data and design more graphics.

How to Read This Book

This book is example-driven and written to give you the skills to take a graphic from start to finish. You can read it cover to cover, or you can pick your spots if you already have a dataset or visualization in mind. The chapters are organized so that the examples are self-contained. If you're new to data, the early chapters should be especially useful to you. They cover how to approach your data, what you should look for, and the tools available to you. You can see where to find data and how to format and prepare it for visualization. After that, the visualization techniques are split by data type and what type of story you're looking for. Remember, always let the data do the talking.

Whatever way you decide to read this book, I highly recommend reading the book with a computer in front of you, so that you can work through examples step-by-step and check out sources referred to in notes and references. You can also download code and data files and interact with working demos at `www.wiley.com/visualizethis` and `http://book.flowingdata.com`.

Just to make things completely clear, here's a flowchart in Figure I-6 to help you figure what spots to pick. Have fun!

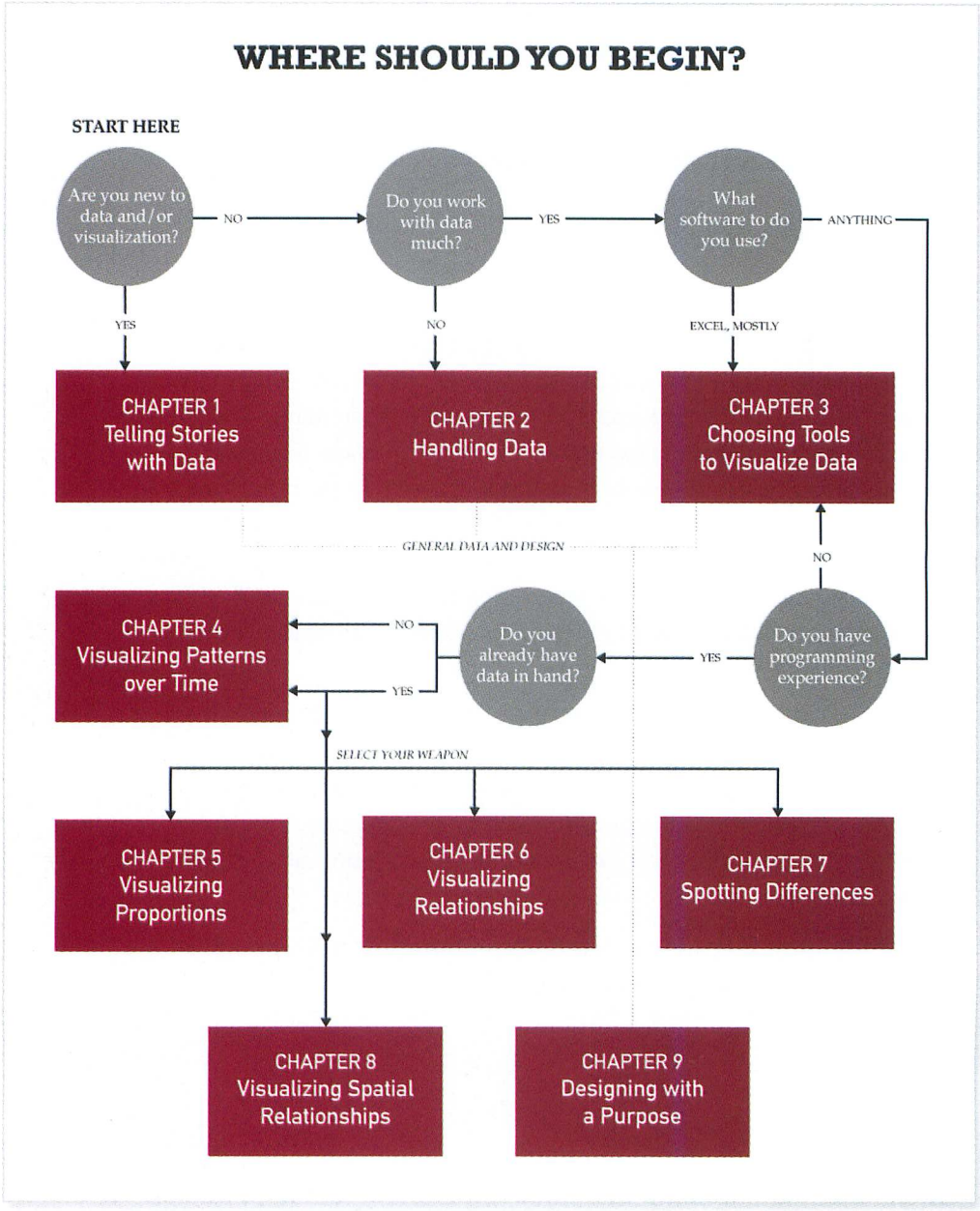


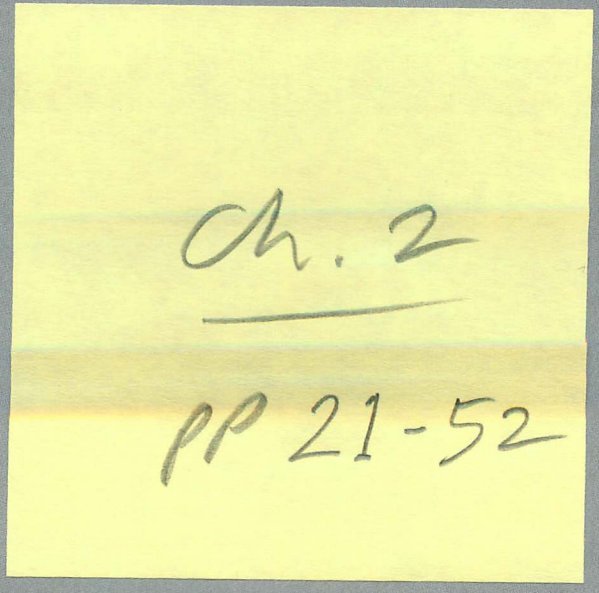
FIGURE I-6 Where to start reading this book

Handling Data

Before you start working on the visual part of any visualization, you actually need data. The data is what makes a visualization interesting. If you don't have interesting data, you just end up with a forgettable graph or a pretty but useless picture. Where can you find good data? How can you access it?

When you have your data, it needs to be formatted so that you can load it into your software. Maybe you got the data as a comma-delimited text file or an Excel spreadsheet, and you need to convert it to something such as XML, or vice versa. Maybe the data you want is accessible point-by-point from a web application, but you want an entire spreadsheet.

Learn to access and process data, and your visualization skills will follow.



Gather Data

Data is the core of any visualization. Fortunately, there are a lot of places to find it. You can get it from experts in the area you're interested in, a variety of online applications, or you can gather it yourself.

Provided by Others

This route is common, especially if you're a freelance designer or work in a graphics department of a larger organization. This is a good thing a lot of the time because someone else did all the data gathering work for you, but you still need to be careful. A lot of mistakes can happen along the way before that nicely formatted spreadsheet gets into your hands.

When you share data with spreadsheets, the most common mistake to look for is typos. Are there any missing zeros? Did your client or data supplier mean six instead of five? At some point, data was read from one source and then input into Excel or a different spreadsheet program (unless a delimited text file was imported), so it's easy for an innocent typo to make its way through the vetting stage and into your hands.

You also need to check for context. You don't need to become an expert in the data's subject matter, but you should know where the original data came from, how it was collected, and what it's about. This can help you build a better graphic and tell a more complete story when you design your graphic. For example, say you're looking at poll results. When did the poll take place? Who conducted the poll? Who answered? Obviously, poll results from 1970 are going to take on a different meaning from poll results from the present day.

Finding Sources

If the data isn't directly sent to you, it's your job to go out and find it. The bad news is that, well, that's more work on your shoulders, but the good news is that's it's getting easier and easier to find data that's relevant and machine-readable (as in, you can easily load it into software). Here's where you can start your search.

SEARCH ENGINES

How do you find anything online nowadays? You Google it. This is a no-brainer, but you'd be surprised how many times people email me asking if I know where to find a particular dataset and a quick search provided relevant results. Personally, I turn to Google and occasionally look to Wolfram|Alpha, the computational search engine.

▶ See Wolfram|Alpha at `http://wolframalpha.com`. The search engine can be especially useful if you're looking for some basic statistics on a topic.

DIRECT FROM THE SOURCE

If a direct query for "data" doesn't provide anything of use, try searching for academics who specialize in the area you're interested in finding data for. Sometimes they post data on their personal sites. If not, scan their papers and studies for possible leads. You can also try emailing them, but make sure they've actually done related studies. Otherwise, you'll just be wasting everyone's time.

You can also spot sources in graphics published by news outlets such as *The New York Times*. Usually data sources are included in small print somewhere on the graphic. If it's not in the graphic, it should be mentioned in the related article. This is particularly useful when you see a graphic in the paper or online that uses data you're interested in exploring. Search for a site for the source, and the data might be available.

This won't always work because finding contacts seems to be a little easier when you email saying that you're a reporter for the so-and-so paper, but it's worth a shot.

UNIVERSITIES

As a graduate student, I frequently make use of the academic resources available to me, namely the library. Many libraries have amped up their technology resources and actually have some expansive data archives. A number of statistics departments also keep a list of data files, many of which are publicly accessible. Albeit, many of the datasets made available by these departments are intended for use with course labs and homework. I suggest visiting the following resources:

- Data and Story Library (DASL) (`http://lib.stat.cmu.edu/DASL/`)—An online library of data files and stories that illustrate the use of basic statistics methods, from Carnegie Mellon

- Berkeley Data Lab (http://sunsite3.berkeley.edu/wikis/datalab/)—Part of the University of California, Berkeley library system
- UCLA Statistics Data Sets (www.stat.ucla.edu/data/)—Some of the data that the UCLA Department of Statistics uses in their labs and assignments

GENERAL DATA APPLICATIONS

A growing number of general data-supplying applications are available. Some applications provide large data files that you can download for free or for a fee. Others are built with developers in mind with data accessible via Application Programming Interface (API). This lets you use data from a service, such as Twitter, and integrate the data with your own application. Following are a few suggested resources:

- Freebase (www.freebase.com)—A community effort that mostly provides data on people, places, and things. It's like Wikipedia for data but more structured. Download data dumps or use it as a backend for your application.
- Infochimps (http://infochimps.org)—A data marketplace with free and for-sale datasets. You can also access some datasets via their API.
- Numbrary (http://numbrary.com)—Serves as a catalog for (mostly government) data on the web.
- AggData (http://aggdata.com)—Another repository of for-sale datasets, mostly focused on comprehensive lists of retail locations.
- Amazon Public Data Sets (http://aws.amazon.com/publicdatasets)—There's not a lot of growth here, but it does host some large scientific datasets.
- Wikipedia (http://wikipedia.org)—A lot of smaller datasets in the form of HTML tables on this community-run encyclopedia.

TOPICAL DATA

Outside more general data suppliers, there's no shortage of subject-specific sites offering loads of free data.

Following is a small taste of what's available for the topic of your choice.

Geography

Do you have mapping software, but no geographic data? You're in luck. Plenty of shapefiles and other geographic file types are at your disposal.

- TIGER (`www.census.gov/geo/www/tiger/`)—From the Census Bureau, probably the most extensive detailed data about roads, railroads, rivers, and ZIP codes you can find
- OpenStreetMap (`www.openstreetmap.org/`)—One of the best examples of data and community effort
- Geocommons (`www.geocommons.com/`)—Both data and a mapmaker
- Flickr Shapefiles (`www.flickr.com/services/api/`)—Geographic boundaries as defined by Flickr users

Sports

People love sports statistics, and you can find decades' worth of sports data. You can find it on *Sports Illustrated* or team organizations' sites, but you can also find more on sites dedicated to the data specifically.

- Basketball Reference (`www.basketball-reference.com/`)—Provides data as specific as play-by-play for NBA games.
- Baseball DataBank (`http://baseball-databank.org/`)—Super basic site where you can download full datasets.
- databaseFootball (`www.databasefootball.com/`)—Browse data for NFL games by team, player, and season.

World

Several noteworthy international organizations keep data about the world, mainly health and development indicators. It does take some sifting though, because a lot of the datasets are quite sparse. It's not easy to get standardized data across countries with varied methods.

- Global Health Facts (`www.globalhealthfacts.org/`)—Health-related data about countries in the world.

- UNdata (http://data.un.org/)—Aggregator of world data from a variety of sources
- World Health Organization (www.who.int/research/en/)—Again, a variety of health-related datasets such as mortality and life expectancy
- OECD Statistics (http://stats.oecd.org/)—Major source for economic indicators
- World Bank (http://data.worldbank.org/)—Data for hundreds of indicators and developer-friendly

Government and Politics

There has been a fresh emphasis on data and transparency in recent years, so many government organizations supply data, and groups such as the Sunlight Foundation encourage developers and designers to make use of it. Government organizations have been doing this for awhile, but with the launch of data.gov, much of the data is available in one place. You can also find plenty of nongovernmental sites that aim to make politicians more accountable.

- Census Bureau (www.census.gov/)—Find extensive demographics here.
- Data.gov (http://data.gov/)—Catalog for data supplied by government organizations. Still relatively new, but has a lot of sources.
- Data.gov.uk (http://data.gov.uk/)—The Data.gov equivalent for the United Kingdom.
- DataSF (http://datasf.org/)—Data specific to San Francisco.
- NYC DataMine (http://nyc.gov/data/)—Just like the above, but for New York.
- Follow the Money (www.followthemoney.org/)—Big set of tools and datasets to investigate money in state politics.
- OpenSecrets (www.opensecrets.org/)—Also provides details on government spending and lobbying.

no longer available

Data Scraping

Often you can find the exact data that you need, except there's one problem. It's not all in one place or in one file. Instead it's in a bunch of HTML pages or on multiple websites. What should you do?

The straightforward, but most time-consuming method would be to visit every page and manually enter your data point of interest in a spreadsheet. If you have only a few pages, sure, no problem.

What if you have a thousand pages? That would take too long—even a hundred pages would be tedious. It would be much easier if you could automate the process, which is what *data scraping* is for. You write some code to visit a bunch of pages automatically, grab some content from that page, and store it in a database or a text file.

NOTE

Although coding is the most flexible way to scrape the data you need, you can also try tools such as Needlebase and Able2Extract PDF converter. Use is straightforward, and they can save you time.

EXAMPLE: SCRAPE A WEBSITE

The best way to learn how to scrape data is to jump right into an example. Say you wanted to download temperature data for the past year, but you can't find a source that provides all the numbers for the right time frame or the correct city. Go to almost any weather website, and at the most, you'll usually see only temperatures for an extended 10-day forecast. That's not even close to what you want. You want actual temperatures from the past, not predictions about future weather.

▶ Visit Weather Underground at `http://wunderground.com`.

Fortunately, the Weather Underground site does provide historic temperatures; however, you can see only one day at a time.

To make things more concrete, look up temperature in Buffalo. Go to the Weather Underground site and search for **BUF** in the search box. This should take you to the weather page for Buffalo Niagara International, which is the airport in Buffalo (see Figure 2-1).

FIGURE 2-1 Temperature in Buffalo, New York, according to Weather Underground

The top of the page provides the current temperature, a 5-day forecast, and other details about the current day. Scroll down toward the middle of the page to the History & Almanac panel, as shown in Figure 2-2. Notice the drop-down menu where you can select a specific date.

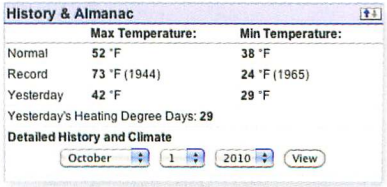
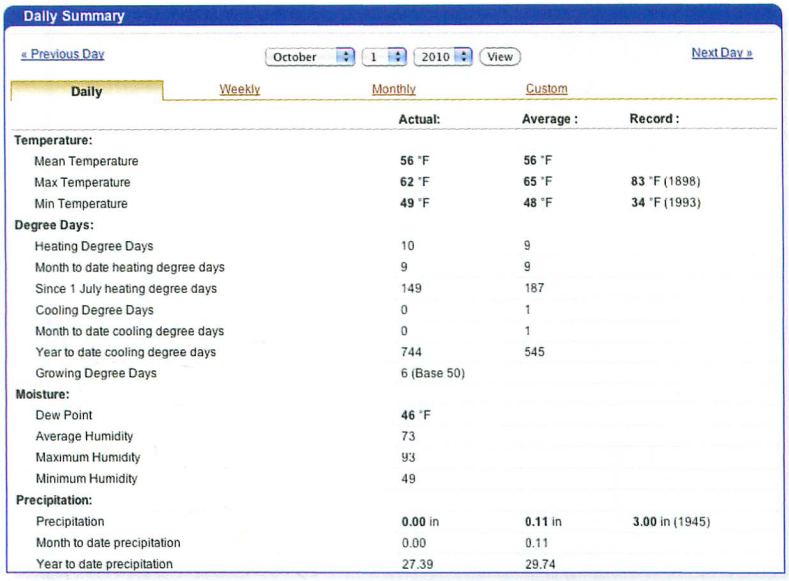


FIGURE 2-2 Drop-down menu to see historical data for a selected date

Adjust the menu to show October 1, 2010, and click the View button. This takes you to a different view that shows you details for your selected date (see Figure 2-3).



Daily Summary

« Previous Day | October 1 2010 View | Next Day »

Daily | Weekly | Monthly | Custom

	Actual:	Average :	Record :
Temperature:			
Mean Temperature	**56** °F	**56** °F	
Max Temperature	**62** °F	**65** °F	**83** °F (1898)
Min Temperature	**49** °F	**48** °F	**34** °F (1993)
Degree Days:			
Heating Degree Days	10	9	
Month to date heating degree days	9	9	
Since 1 July heating degree days	149	187	
Cooling Degree Days	0	1	
Month to date cooling degree days	0	1	
Year to date cooling degree days	744	545	
Growing Degree Days	6 (Base 50)		
Moisture:			
Dew Point	**46** °F		
Average Humidity	73		
Maximum Humidity	93		
Minimum Humidity	49		
Precipitation:			
Precipitation	**0.00** in	**0.11** in	**3.00** in (1945)
Month to date precipitation	0.00	0.11	
Year to date precipitation	27.39	29.74	

FIGURE 2-3 Temperature data for a single day

There's temperature, degree days, moisture, precipitation, and plenty of other data points, but for now, all you're interested in is maximum temperature per day, which you can find in the second column, second row down. On October 1, 2010, the maximum temperature in Buffalo was 62 degrees Fahrenheit.

Getting that single value was easy enough. Now how can you get that maximum temperature value every day, during the year 2009? The easy-and-straightforward way would be to keep changing the date in the drop-down. Do that 365 times and you're done.

Wouldn't that be fun? No. You can speed up the process with a little bit of code and some know-how, and for that, turn to the Python programming language and Leonard Richardson's Python library called Beautiful Soup.

You're about to get your first taste of code in the next few paragraphs. If you have programming experience, you can go through the following

relatively quickly. Don't worry if you don't have any programming experience though—I'll take you through it step-by-step. A lot of people like to keep everything within a safe click interface, but trust me. Pick up just a little bit of programming skills, and you can open up a whole bag of possibilities for what you can do with data. Ready? Here you go.

First, you need to make sure your computer has all the right software installed. If you work on Mac OS X, you should have Python installed already. Open the Terminal application and type **python** to start (see Figure 2-4).

▶ Visit `http://python.org` to download and install Python. Don't worry; it's not too hard.

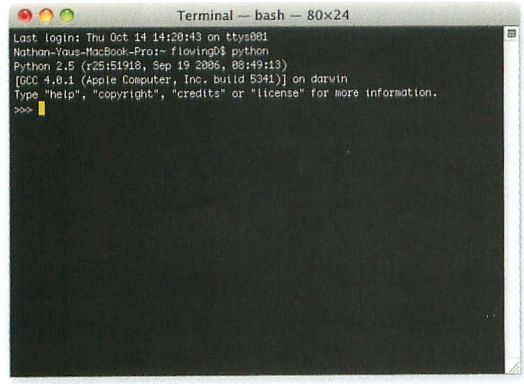


FIGURE 2-4 Starting Python in OS X

▶ Visit `www.crummy.com/software/BeautifulSoup/` to download Beautiful Soup. Download the version that matches the version of Python that you use.

If you're on a Windows machine, you can visit the Python site and follow the directions on how to download and install.

Next, you need to download Beautiful Soup, which can help you read web pages quickly and easily. Save the Beautiful Soup Python (`.py`) file in the directory that you plan to save your code in. If you know your way around Python, you can also put Beautiful Soup in your library path, but it'll work the same either way.

After you install Python and download Beautiful Soup, start a file in your favorite text or code editor, and save it as `get-weather-data.py`. Now you can code.

The first thing you need to do is load the page that shows historical weather information. The URL for historical weather in Buffalo on October 1, 2010, follows:

```
www.wunderground.com/history/airport/KBUF/2010/10/1/DailyHistory
.html?req_city=NA&req_state=NA&req_statename=NA
```

If you remove everything after .html in the preceding URL, the same page still loads, so get rid of those. You don't care about those right now.

```
www.wunderground.com/history/airport/KBUF/2010/10/1/DailyHistory.html
```

The date is indicated in the URL with `/2010/10/1`. Using the drop-down menu, change the date to January 1, 2009, because you're going to scrape temperature for all of 2009. The URL is now this:

```
www.wunderground.com/history/airport/KBUF/2009/1/1/DailyHistory.html
```

Everything is the same as the URL for October 1, except the portion that indicates the date. It's `/2009/1/1` now. Interesting. Without using the drop-down menu, how can you load the page for January 2, 2009? Simply change the date parameter so that the URL looks like this:

```
www.wunderground.com/history/airport/KBUF/2009/1/2/DailyHistory.html
```

Load the preceding URL in your browser and you get the historical summary for January 2, 2009. So all you have to do to get the weather for a specific date is to modify the Weather Underground URL. Keep this in mind for later.

Now load a single page with Python, using the urllib2 library by importing it with the following line of code:

```
import urllib2
```

To load the January 1 page with Python, use the `urlopen` function.

```
page = urllib2.urlopen("www.wunderground.com/history/airport/
KBUF/2009/1/1/DailyHistory.html")
```

This loads all the HTML that the URL points to in the page variable. The next step is to extract the maximum temperature value you're interested

in from that HTML, and for that, Beautiful Soup makes your task much easier. After `urllib2`, import Beautiful Soup like so:

```
from BeautifulSoup import BeautifulSoup
```

At the end of your file, use Beautiful Soup to read (that is, parse) the page.

```
soup = BeautifulSoup(page)
```

Without getting into nitty-gritty details, this line of code reads the HTML, which is essentially one long string, and then stores elements of the page, such as the header or images, in a way that is easier to work with.

For example, if you want to find all the images in the page, you can use this:

```
images = soup.findAll('img')
```

This gives you a list of all the images on the Weather Underground page displayed with the `<img />` HTML tag. Want the first image on the page? Do this:

```
first_image = images[0]
```

Want the second image? Change the zero to a one. If you want the `src` value in the first `<img />` tag, you would use this:

```
src = first_image['src']
```

Okay, you don't want images. You just want that one value: maximum temperature on January 1, 2009, in Buffalo, New York. It was 26 degrees Fahrenheit. It's a little trickier finding that value in your soup than it was finding images, but you still use the same method. You just need to figure out what to put in `findAll()`, so look at the HTML source.

You can easily do this in all the major browsers. In Firefox, go to the View menu, and select Page Source. A window with the HTML for your current page appears, as shown in Figure 2-5.

Scroll down to where it shows Mean Temperature, or just search for it, which is faster. Spot the `26`. That's what you want to extract.

The row is enclosed by a `<span>` tag with a `nobr` class. That's your key. You can find all the elements in the page with the `nobr` class.

```
nobrs = soup.findAll(attrs={"class":"nobr"})
```

NOTE

Beautiful Soup provides good documentation and straightforward examples, so if any of this is confusing, I strongly encourage you to check those out on the same Beautiful Soup site you used to download the library.

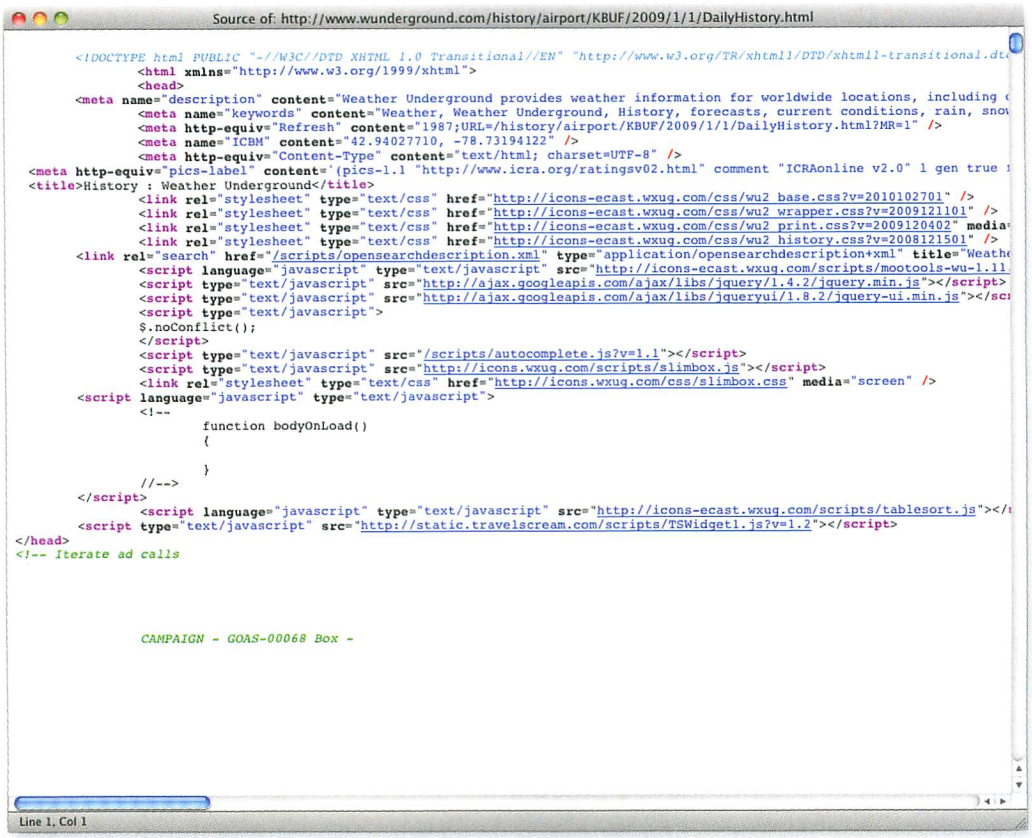


FIGURE 2-5 HTML source for a page on Weather Underground

As before, this gives you a list of all the occurrences of `nobr`. The one that you're interested in is the sixth occurrence, which you can find with the following:

```
print nobrs[5]
```

This gives you the whole element, but you just want the 26. Inside the `<span>` tag with the `nobr` class is another `<span>` tag and *then* the 26. So here's what you need to use:

```
dayTemp = nobrs[5].span.string
print dayTemp
```

Ta Da! You scraped your first value from an HTML web page. Next step: scrape all the pages for 2009. For that, return to the original URL.

`www.wunderground.com/history/airport/KBUF/2009/1/1/DailyHistory.html`

Remember that you changed the URL manually to get the weather data for the date you want. The preceding code is for January 1, 2009. If you want the page for January 2, 2009, simply change the date portion of the URL to match that. To get the data for every day of 2009, load every month (1 through 12) and then load every day of each month. Here's the script in full with comments. Save it to your `get-weather-data.py` file.

```
import urllib2
from BeautifulSoup import BeautifulSoup

# Create/open a file called wunder.txt (which will be a comma-delimited
file)
f = open('wunder-data.txt', 'w')

# Iterate through months and day
for m in range(1, 13):
    for d in range(1, 32):

      # Check if already gone through month
      if (m == 2 and d > 28):
        break
      elif (m in [4, 6, 9, 11] and d > 30):
        break

      # Open wunderground.com url
      timestamp = '2009' + str(m) + str(d)
      print "Getting data for " + timestamp
      url = "http://www.wunderground.com/history/airport/KBUF/2009/" +
str(m) + "/" + str(d) + "/DailyHistory.html"
      page = urllib2.urlopen(url)

      # Get temperature from page
      soup = BeautifulSoup(page)
      # dayTemp = soup.body.nobr.b.string
      dayTemp = soup.findAll(attrs={"class":"nobr"})[5].span.string

      # Format month for timestamp
      if len(str(m)) < 2:
        mStamp = '0' + str(m)
      else:
        mStamp = str(m)

      # Format day for timestamp
```

```
        if len(str(d)) < 2:
          dStamp = '0' + str(d)
        else:
          dStamp = str(d)

        # Build timestamp
        timestamp = '2009' + mStamp + dStamp

        # Write timestamp and temperature to file
        f.write(timestamp + ',' + dayTemp + '\n')

# Done getting data! Close file.
f.close()
```

You should recognize the first two lines of code to import the necessary libraries, urllib2 and BeautifulSoup.

```
import urllib2
from BeautifulSoup import BeautifulSoup
```

Next, start a text file called `wunder-data-txt` with write permissions, using the `open()` method. All the data that you scrape will be stored in this text file, in the same directory that you saved this script in.

```
# Create/open a file called wunder.txt (which will be a comma-delimited
file)
f = open('wunder-data.txt', 'w')
```

With the next line of code, use a `for` loop, which tells the computer to visit each month. The month number is stored in the `m` variable. The loop that follows then tells the computer to visit each day of each month. The day number is stored in the `d` variable.

```
# Iterate through months and day
for m in range(1, 13):
    for d in range(1, 32):
```

▶ See Python documentation for more on how loops and iteration work: `http://docs.python.org/reference/compound_stmts.html`

Notice that you used `range (1, 32)` to iterate through the days. This means you can iterate through the numbers 1 to 31. However, not every month of the year has 31 days. February has 28 days; April, June, September, and November have 30 days. There's no temperature value for April 31 because it doesn't exist. So check what month it is and act accordingly. If the current month is February and the day is greater than 28, break and

move on to the next month. If you want to scrape multiple years, you need to use an additional `if` statement to handle leap years.

Similarly, if it's not February, but instead April, June, September, or November, move on to the next month if the current day is greater than 30.

```
# Check if already gone through month
if (m == 2 and d > 28):
  break
elif (m in [4, 6, 9, 11] and d > 30):
  break
```

Again, the next few lines of code should look familiar. You used them to scrape a single page from Weather Underground. The difference is in the month and day variable in the URL. Change that for each day instead of leaving it static; the rest is the same. Load the page with the urllib2 library, parse the contents with Beautiful Soup, and then extract the maximum temperature, but look for the sixth appearance of the `nobr` class.

```
# Open wunderground.com url
url = "http://www.wunderground.com/history/airport/KBUF/2009/" +
str(m) + "/" + str(d) + "/DailyHistory.html"
page = urllib2.urlopen(url)

# Get temperature from page
soup = BeautifulSoup(page)
# dayTemp = soup.body.nobr.b.string
dayTemp = soup.findAll(attrs={"class":"nobr"})[5].span.string
```

The next to last chunk of code puts together a timestamp based on the year, month, and day. Timestamps are put into this format: yyyymmdd. You can construct any format here, but keep it simple for now.

```
# Format day for timestamp
if len(str(d)) < 2:
  dStamp = '0' + str(d)
else:
  dStamp = str(d)

# Build timestamp
timestamp = '2009' + mStamp + dStamp
```

Finally, the temperature and timestamp are written to `'wunder-data.txt'` using the `write()` method.

```
    # Write timestamp and temperature to file
    f.write(timestamp + ',' + dayTemp + '\n')
```

Then use `close()`when you finish with all the months and days.

```
# Done getting data! Close file.
f.close()
```

The only thing left to do is run the code, which you do in your terminal with the following:

```
$ python get-weather-data.py
```

It takes a little while to run, so be patient. In the process of running, your computer is essentially loading 365 pages, one for each day of 2009. You should have a file named `wunder-data.txt` in your working directory when the script is done running. Open it up, and there's your data, as a comma-separated file. The first column is for the timestamps, and the second column is temperatures. It should look similar to Figure 2-6.

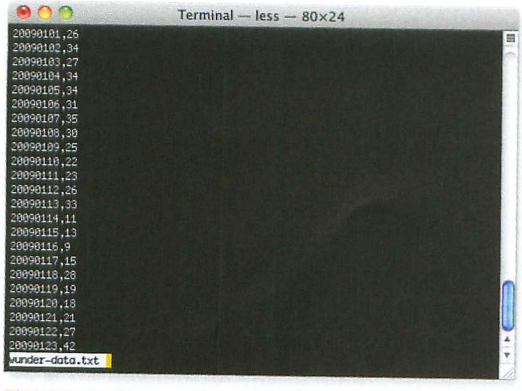


FIGURE 2-6 One year's worth of scraped temperature data

GENERALIZING THE EXAMPLE

Although you just scraped weather data from Weather Underground, you can generalize the process for use with other data sources. Data scraping typically involves three steps:

1. Identify the patterns.
2. Iterate.
3. Store the data.

In this example, you had to find two patterns. The first was in the URL, and the second was in the loaded web page to get the actual temperature value. To load the page for a different day in 2009, you changed the month and day portions of the URL. The temperature value was enclosed in the sixth occurrence of the `nobr` class in the HTML page. If there is no obvious pattern to the URL, try to figure out how you can get the URLs of all the pages you want to scrape. Maybe the site has a site map, or maybe you can go through the index via a search engine. In the end, you need to know all the URLs of the pages of data.

After you find the patterns, you iterate. That is, you visit all the pages programmatically, load them, and parse them. Here you did it with Beautiful Soup, which makes parsing XML and HTML easy in Python. There's probably a similar library if you choose a different programming language.

Lastly, you need to store it somewhere. The easiest solution is to store the data as a plain text file with comma-delimited values, but if you have a database set up, you can also store the values in there.

Things can get trickier as you run into web pages that use JavaScript to load all their data into view, but the process is still the same.

Formatting Data

Different visualization tools use different data formats, and the structure you use varies by the story you want to tell. So the more flexible you are with the structure of your data, the more possibilities you can gain. Make use of data formatting applications, and couple that with a little bit of programming know-how, and you can get your data in any format you want to fit your specific needs.

The easy way of course is to find a programmer who can format and parse all of your data, but you'll always be waiting on someone. This is especially evident during the early stages of any project where iteration and data exploration are key in designing a useful visualization. Honestly, if I were in a hiring position, I'd likely just get the person who knows how to work with data, over the one who needs help at the beginning of every project.

WHAT I LEARNED ABOUT FORMATTING

When I first learned statistics in high school, the data was always provided in a nice, rectangular format. All I had to do was plug some numbers into an Excel spreadsheet or my awesome graphing calculator (which was the best way to look like you were working in class, but actually playing Tetris). That's how it was all the way through my undergraduate education. Because I was learning about techniques and theorems for analyses, my teachers didn't spend any time on working with raw, preprocessed data. The data always seemed to be in just the right format.

This is perfectly understandable, given time constraints and such, but in graduate school, I realized that data in the real world never seems to be in the format that you need. There are missing values, inconsistent labels, typos, and values without any context. Often the data is spread across several tables, but you need everything in one, joined across a value, like a name or a unique id number.

This was also true when I started to work with visualization. It became increasingly important because I wanted to do more with the data I had. Nowadays, it's not out of the ordinary that I spend just as much time getting data in the format that I need as I do putting the visual part of a data graphic together. Sometimes I spend more time getting all my data in place. This might seem strange at first, but you'll find that the design of your data graphics comes much easier when you have your data neatly organized, just like it was back in that introductory statistics course in high school.

Various data formats, the tools available to deal with these formats, and finally, some programming, using the same logic you used to scrape data in the previous example are described next.

Data Formats

Most people are used to working with data in Excel. This is fine if you're going to do everything from analyses to visualization in the program, but if you want to step beyond that, you need to familiarize yourself with other data formats. The point of these formats is to make your data

machine-readable, or in other words, to structure your data in a way that a computer can understand. Which data format you use can change by visualization tool and purpose, but the three following formats can cover most of your bases: delimited text, JavaScript Object Notation, and Extensible Markup Language.

DELIMITED TEXT

Most people are familiar with delimited text. You did after all just make a comma-delimited text file in your data scraping example. If you think of a dataset in the context of rows and columns, a delimited text file splits columns by a delimiter. The delimiter is a comma in a comma-delimited file. The delimiter might also be a tab. It can be spaces, semicolons, colons, slashes, or whatever you want; although a comma and tab are the most common.

Delimited text is widely used and can be read into most spreadsheet programs such as Excel or Google Documents. You can also export spreadsheets as delimited text. If multiple sheets are in your workbook, you usually have multiple delimited files, unless you specify otherwise.

This format is also good for sharing data with others because it doesn't depend on any particular program.

JAVASCRIPT OBJECT NOTATION (JSON)

This is a common format offered by web APIs. It's designed to be both machine- and human-readable; although, if you have a lot of it in front of you, it'll probably make you cross-eyed if you stare at it too long. It's based on JavaScript notation, but it's not dependent on the language. There are a lot of specifications for JSON, but you can get by for the most part with just the basics.

JSON works with keywords and values, and treats items like objects. If you were to convert JSON data to comma-separated values (CSV), each object might be a row.

As you can see later in this book, a number of applications, languages, and libraries accept JSON as input. If you plan to design data graphics for the web, you're likely to run into this format.

► Visit `http://json.org` for the full specification of JSON. You don't need to know every detail of the format, but it can be handy at times when you don't understand a JSON data source.

EXTENSIBLE MARKUP LANGUAGE (XML)

XML is another popular format on the web, often used to transfer data via APIs. There are lots of different types and specifications for XML, but at the most basic level, it is a text document with values enclosed by tags. For example, the Really Simple Syndication (RSS) feed that people use to subscribe to blogs, such as FlowingData, is actually an XML file, as shown in Figure 2-7.

The RSS lists recently published items enclosed in the `<item></item>` tag, and each item has a title, description, author, and publish date, along with some other attributes.

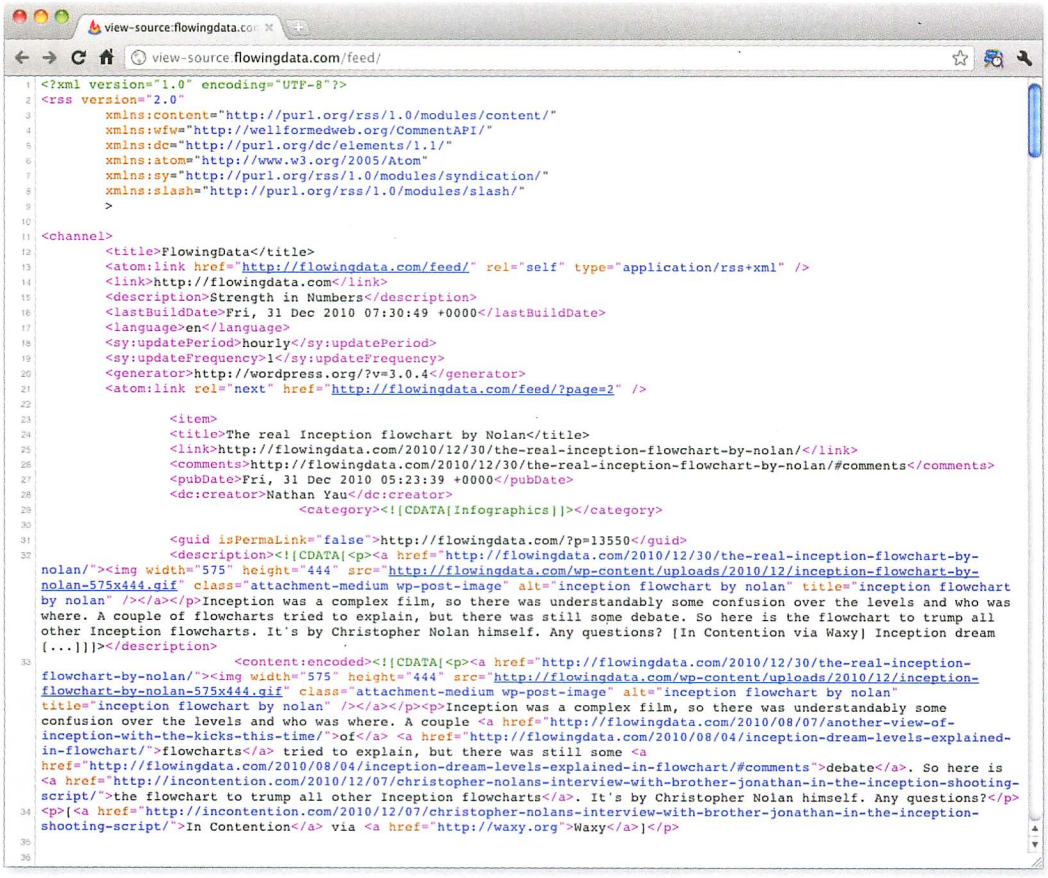


FIGURE 2-7 Snippet of FlowingData's RSS feed

XML is relatively easy to parse with libraries such as Beautiful Soup in Python. You can get a better feel for XML, along with CSV and JSON, in the sections that follow.

Formatting Tools

Just a couple of years ago, quick scripts were always written to handle and format data. After you've written a few scripts, you start to notice patterns in the logic, so it's not super hard to write new scripts for specific datasets, but it does take time. Luckily, with growing volumes of data, some tools have been developed to handle the boiler plate routines.

GOOGLE REFINE

Google Refine is the evolution of Freebase Gridworks. Gridworks was first developed as an in-house tool for an open data platform, Freebase; however, Freebase was acquired by Google, therefore the new name. Google Refine is essentially Gridworks 2.0 with an easier-to-use interface (Figure 2-8) with more features.

It runs on your desktop (but still through your browser), which is great, because you don't need to worry about uploading private data to Google's servers. All the processing happens on your computer. Refine is also open source, so if you feel ambitious, you can cater the tool to your own needs with extensions.

When you open Refine, you see a familiar spreadsheet interface with your rows and columns. You can easily sort by field and search for values. You can also find inconsistencies in your data and consolidate in a relatively easy way.

For example, say for some reason you have an inventory list for your kitchen. You can load the data in Refine and quickly find inconsistencies such as typos or differing classifications. Maybe a fork was misspelled as "frk," or you want to reclassify all the forks, spoons, and knives as utensils. You can easily find these things with Refine and make changes. If you don't like the changes you made or make a mistake, you can revert to the old dataset with a simple undo.

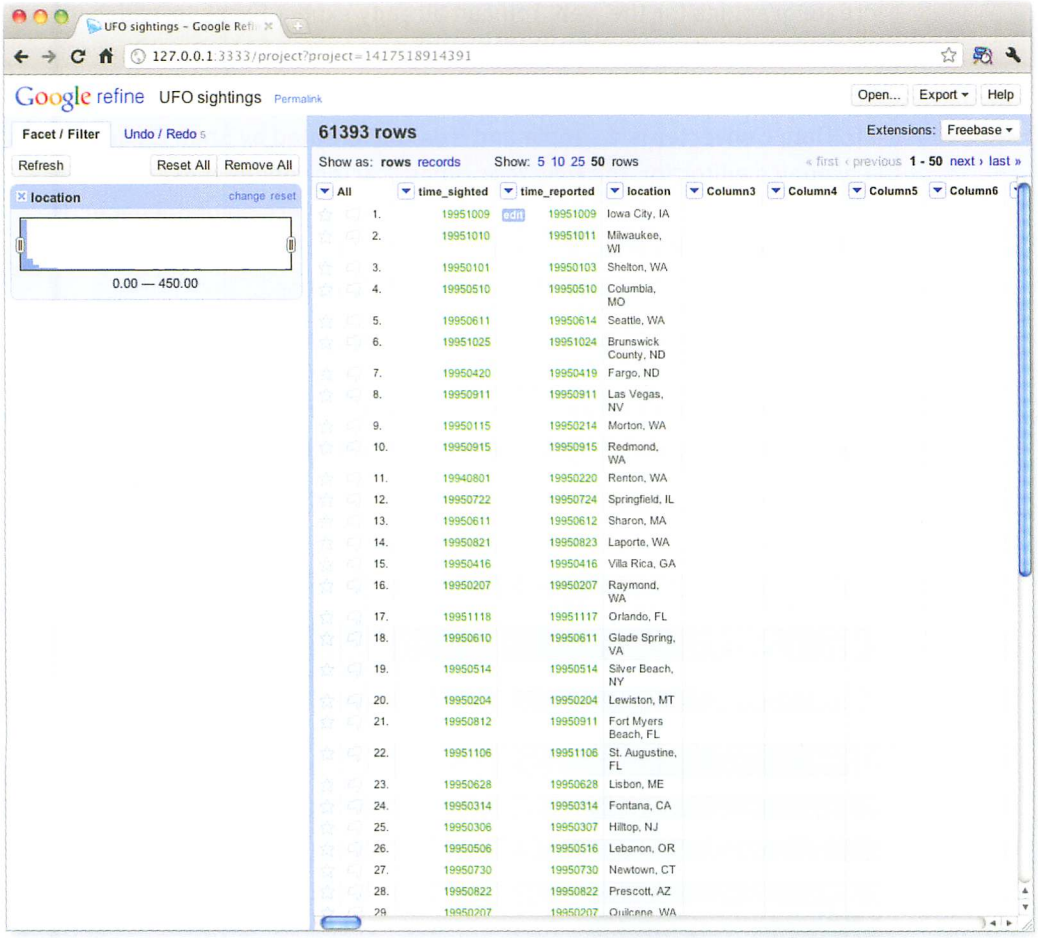


FIGURE 2-8 Google Refine user interface

Getting into the more advanced stuff, you can also incorporate data sources like your own with a dataset from Freebase to create a richer dataset.

If anything, Google Refine is a good tool to keep in your back pocket. It's powerful, and it's a free download, so I highly recommend you at least fiddle around with the tool.

▶ Download the open-source Google Refine and view tutorials on how to make the most out of the tool at `http://code.google.com/p/google-refine/`.

MR. DATA CONVERTER

Often, you might get all your data in Excel but then need to convert it to another format to fit your needs. This is almost always the case when you

create graphics for the web. You can already export Excel spreadsheets as CSV, but what if you need something other than that? Mr. Data Converter can help you.

Mr. Data Converter is a simple and free tool created by Shan Carter, who is a graphics editor for *The New York Times*. Carter spends most of his work time creating interactive graphics for the online version of the paper. He has to convert data often to fit the software that he uses, so it's not surprising he made a tool that streamlines the process.

It's easy to use, and Figure 2-9 shows that the interface is equally as simple. All you need to do is copy and paste data from Excel in the input section on the top and then select what output format you want in the bottom half of the screen. Choose from variants of XML, JSON, and a number of others.

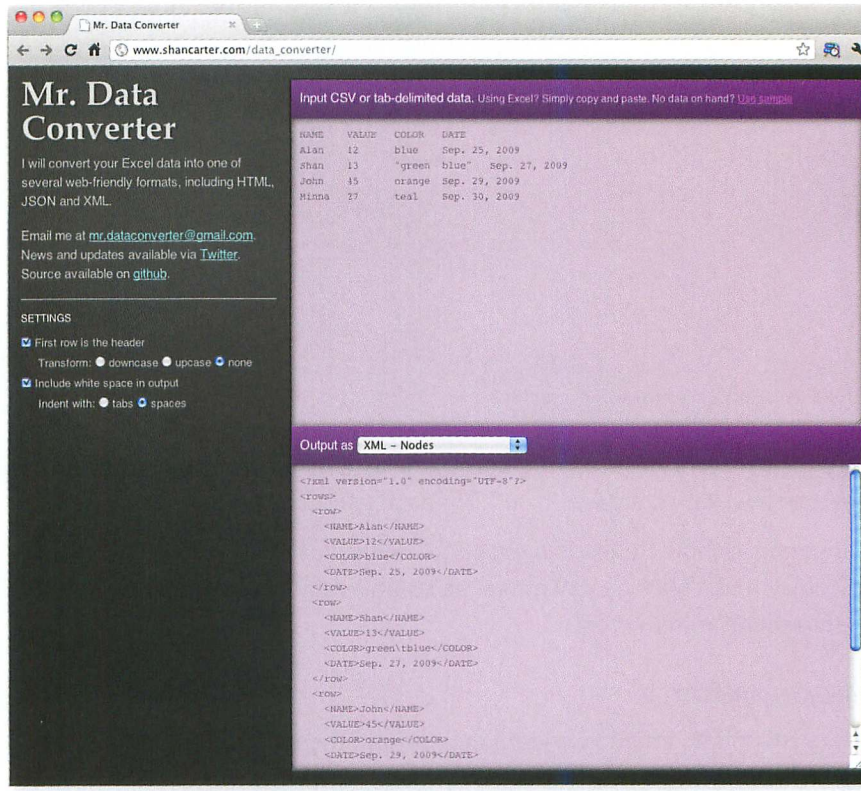


FIGURE 2-9 Mr. Data Converter makes switching between data formats easy.

The source code to Mr. Data Converter is also available if you want to make your own or extend.

MR. PEOPLE

Inspired by Carter's Mr. Data Converter, *The New York Times* graphics deputy director Matthew Ericson created Mr. People. Like Mr. Data Converter, Mr. People enables you to copy and paste data into a text field, and the tool parses and extracts for you. Mr. People, however, as you might guess, is specifically for parsing names.

Maybe you have a long list of names without a specific format, and you want to identify the first and last names, along with middle initial, prefix, and suffix. Maybe multiple people are listed on a single row. That's where Mr. People comes in. Copy and paste names, as shown in Figure 2-10, and you get a nice clean table that you can copy into your favorite spreadsheet software, as shown in Figure 2-11.

Like Mr. Data Converter, Mr. People is also available as open-source software on github.

SPREADSHEET SOFTWARE

Of course, if all you need is simple sorting, or you just need to make some small changes to individual data points, your favorite spreadsheet software is always available. Take this route if you're okay with manually editing data. Otherwise, try the preceding first (especially if you have a giganto dataset), or go with a custom coding solution.

▶ Try out Mr. Data Converter at `www.shancarter.com/data_converter/` or download the source on github at `https://github.com/shancarter/Mr-Data-Converter` to convert your Excel spreadsheets to a web-friendly format.

▶ Use Mr. People at `http://people.ericson.net/` or download the Ruby source on github to use the name parser in your own scripts: `http://github.com/mericson/people`.

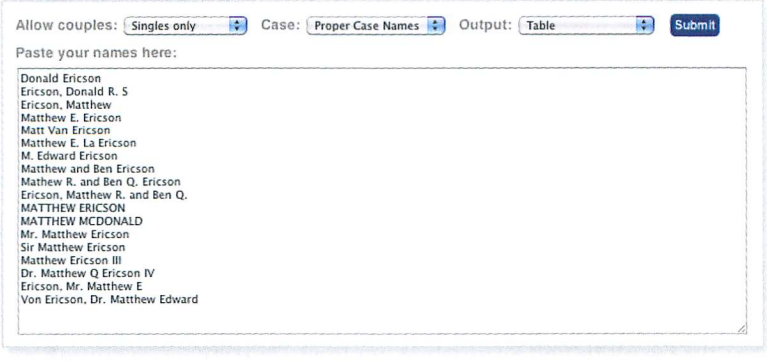
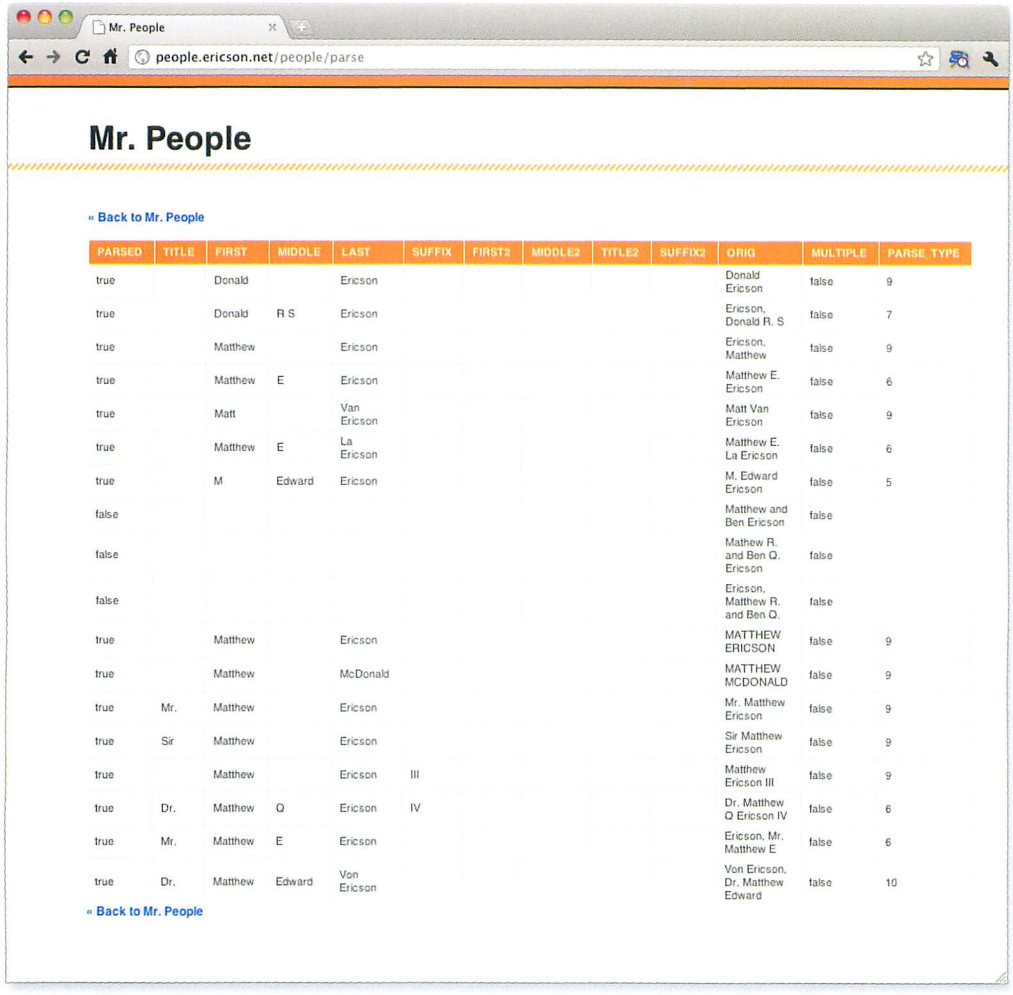


FIGURE 2-10 Input page for names on Mr. People



PARSED	TITLE	FIRST	MIDDLE	LAST	SUFFIX	FIRST2	MIDDLE2	TITLE2	SUFFIX2	ORIG	MULTIPLE	PARSE_TYPE
true		Donald		Ericson						Donald Ericson	false	9
true		Donald	R S	Ericson						Ericson, Donald R. S	false	7
true		Matthew		Ericson						Ericson, Matthew	false	9
true		Matthew	E	Ericson						Matthew E. Ericson	false	6
true		Matt		Van Ericson						Matt Van Ericson	false	9
true		Matthew	E	La Ericson						Matthew E. La Ericson	false	6
true		M	Edward	Ericson						M. Edward Ericson	false	5
false										Matthew and Ben Ericson	false	
false										Mathew R. and Ben Q. Ericson	false	
false										Ericson, Matthew R. and Ben Q.	false	
true		Matthew		Ericson						MATTHEW ERICSON	false	9
true		Matthew		McDonald						MATTHEW MCDONALD	false	9
true	Mr.	Matthew		Ericson						Mr. Matthew Ericson	false	9
true	Sir	Matthew		Ericson						Sir Matthew Ericson	false	9
true		Matthew		Ericson	III					Matthew Ericson III	false	9
true	Dr.	Matthew	Q	Ericson	IV					Dr. Matthew Q Ericson IV	false	6
true	Mr.	Matthew	E	Ericson						Ericson, Mr. Matthew E	false	6
true	Dr.	Matthew	Edward	Von Ericson						Von Ericson, Dr. Matthew Edward	false	10

FIGURE 2-11 Parsed names in table format with Mr. People

Formatting with Code

Although point-and-click software can be useful, sometimes the applications don't quite do what you want if you work with data long enough. Some software doesn't handle large data files well; they get slow or they crash.

What do you do at this point? You can throw your hands in the air and give up; although, that wouldn't be productive. Instead, you can write some

code to get the job done. With code you become much more flexible, and you can tailor your scripts specifically for your data.

Now jump right into an example on how to easily switch between data formats with just a few lines of code.

EXAMPLE: SWITCH BETWEEN DATA FORMATS

This example uses Python, but you can of course use any language you want. The logic is the same, but the syntax will be different. (I like to develop applications in Python, so managing raw data with Python fits into my workflow.)

Going back to the previous example on scraping data, use the resulting `wunder-data.txt` file, which has dates and temperatures in Buffalo, New York, for 2009. The first rows look like this:

```
20090101,26
20090102,34
20090103,27
20090104,34
20090105,34
20090106,31
20090107,35
20090108,30
20090109,25
...
```

This is a CSV file, but say you want the data as XML in the following format:

```
<weather_data>
    <observation>
        <date>20090101</date>
        <max_temperature>26</max_temperature>
    </observation>
    <observation>
        <date>20090102</date>
        <max_temperature>34</max_temperature>
    </observation>
    <observation>
        <date>20090103</date>
        <max_temperature>27</max_temperature>
    </observation>
    <observation>
```

```
        <date>20090104</date>
        <max_temperature>34</max_temperature>
    </observation>
    ...
</weather_data>
```

Each day's temperature is enclosed in <observation> tags with a <date> and the <max_temperature>.

To convert the CSV into the preceding XML format, you can use the following code snippet:

```
import csv
reader = csv.reader(open('wunder-data.txt', 'r'), delimiter=",")
print '<weather_data>'

for row in reader:
    print '<observation>'
    print '<date>' + row[0] + '</date>'
    print '<max_temperature>' + row[1] + '</max_temperature>'
    print '</observation>'

print '</weather_data>'
```

As before, you import the necessary modules. You need only the csv module in this case to read in wunder-data.txt.

```
import csv
```

The second line of code opens wunder-data.txt to read using open() and then reads it with the csv.reader() method.

```
reader = csv.reader(open('wunder-data.txt', 'r'), delimiter=",")
```

Notice the delimiter is specified as a comma. If the file were a tab-delimited file, you could specify the delimiter as '\t'.

Then you can print the opening line of the XML file in line 3.

```
print '<weather_data>'
```

In the main chunk of the code, you can loop through each row of data and print in the format that you need the XML to be in. In this example, each row in the CSV header is equivalent to each observation in the XML.

```
for row in reader:
    print '<observation>'
```

```
    print '<date>' + row[0] + '</date>'
    print '<max_temperature>' + row[1] + '</max_temperature>'
    print '</observation>'
```

Each row has two values: the date and the maximum temperature.

End the XML conversion with its closing tag.

```
print '</weather_data>'
```

Two main things are at play here. First, you read the data in, and then you iterate over the data, changing each row in some way. It's the same logic if you were to convert the resulting XML back to CSV. As shown in the following snippet, the difference is that you use a different module to parse the XML file.

```
from BeautifulSoup import BeautifulStoneSoup

f = open('wunder-data.xml', 'r')
xml = f.read()

soup = BeautifulStoneSoup(xml)
observations = soup.findAll('observation')
for o in observations:
    print o.date.string + "," + o.max_temperature.string
```

The code looks different, but you're basically doing the same thing. Instead of importing the `csv` module, you import BeautifulStoneSoup from BeautifulSoup. Remember you used BeautifulSoup to parse the HTML from Weather Underground. BeautifulStoneSoup parses the more general XML.

You can open the XML file for reading with `open()` and then load the contents in the `xml` variable. At this point, the contents are stored as a string. To parse, pass the `xml` string to `BeautifulStoneSoup` to iterate through each `<observation>` in the `XML` file. Use `findAll()` to fetch all the observations, and finally, like you did with the CSV to XML conversion, loop through each observation, printing the values in your desired format.

This takes you back to where you began:

```
20090101,26
20090102,34
20090103,27
20090104,34
...
```

To drive the point home, here's the code to convert your CSV to JSON format.

```
import csv
reader = csv.reader(open('wunder-data.txt', 'r'), delimiter=",")

print "{ observations: ["
rows_so_far = 0
for row in reader:

    rows_so_far += 1

    print '{'
    print '"date": ' + '"' + row[0] + '", '
    print '"temperature": ' + row[1]

    if rows_so_far < 365:
        print " },"
    else:
        print " }"

print "] }"
```

Go through the lines to figure out what's going on, but again, it's the same logic with different output. Here's what the JSON looks like if you run the preceding code.

```
{
    "observations": [
        {
            "date": "20090101",
            "temperature": 26
        },
        {
            "date": "20090102",
            "temperature": 34
        },
        ...
    ]
}
```

This is still the same data, with date and temperature but in a different format. Computers just love variety.

Put Logic in the Loop

If you look at the code to convert your CSV file to JSON, you should notice the *if-else* statement in the *for* loop, after the three print lines. This checks if the current iteration is the last row of data. If it isn't, don't put a comma at the end of the observation. Otherwise, you do. This is part of the JSON specification. You can do more here.

You can check if the max temperature is more than a certain amount and create a new field that is 1 if a day is more than the threshold, or 0 if it is not. You can create categories or flag days with missing values.

Actually, it doesn't have to be just a check for a threshold. You can calculate a moving average or the difference between the current day and the previous. There are lots of things you can do within the loop to augment the raw data. Everything isn't covered here because you can do anything from trivial changes to advanced analyses, but now look at a simple example.

Going back to your original CSV file, `wunder-data.txt`, create a third column that indicates whether a day's maximum temperature was at or below freezing. A 0 indicates above freezing, and 1 indicates at or below freezing.

```
import csv
reader = csv.reader(open('wunder-data.txt', 'r'), delimiter=",")
for row in reader:
    if int(row[1]) <= 32:
        is_freezing = '1'
    else:
        is_freezing = '0'

    print row[0] + "," + row[1] + "," + is_freezing
```

Like before, read the data from the CSV file into Python, and then iterate over each row. Check each day and flag accordingly.

This is of course a simple example, but it should be easy to see how you can expand on this logic to format or augment your data to your liking. Remember the three steps of load, loop, and process, and expand from there.

Wrapping Up

This chapter covered where you can find the data you need and how to manage it after you have it. This is an important step, if not the most important, in the visualization process. A data graphic is only as interesting as its underlying data. You can dress up a graphic all you want, but the data (or the results from your analysis of the data) is still the substance; and now that you know where and how to get your data, you're already a step ahead of the pack.

You also got your first taste of programming. You scraped data from a website and then formatted and rearranged that data, which will be a useful trick in later chapters. The main takeaway, however, is the logic in the code. You used Python, but you easily could have used Ruby, Perl, or PHP. The logic is the same across languages. When you learn one programming language (and if you're a programmer already, you can attest to this), it's much easier to learn other languages later.

You don't always have to turn to code. Sometimes there are click-and-drag applications that make your job a lot easier, and you should take advantage of that when you can. In the end, the more tools you have in your toolbox, the less likely you're going to get stuck somewhere in the process.

Okay, you have your data. Now it's time to get visual.

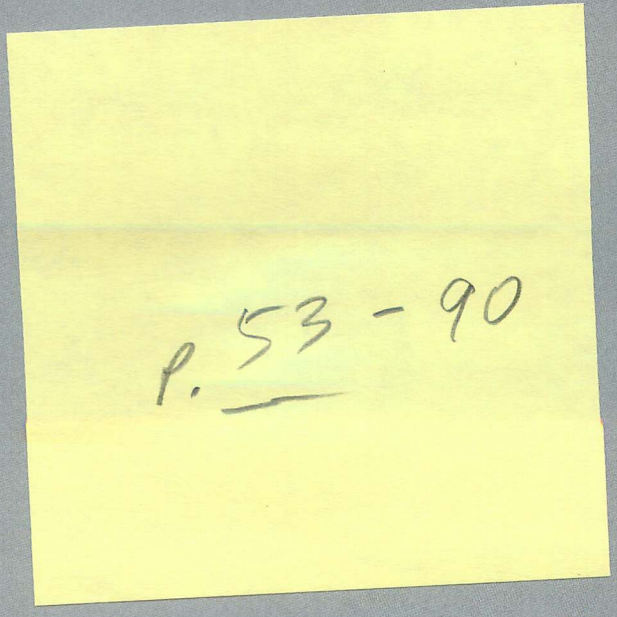
Choosing Tools to Visualize Data

3

In the last chapter, you learned where to find your data and how to get it in the format you need, so you're ready to start visualizing. One of the most common questions people ask me at this point is "What software should I use to visualize my data?"

Luckily, you have a lot of options. Some are out-of-the-box and click-and-drag. Others require a little bit of programming, whereas some tools weren't designed specifically for data graphics but are useful nevertheless. This chapter covers these options.

The more visualization tools you know how to use and take advantage of, the less likely you'll get stuck not knowing what to do with a dataset and the more likely you can make a graphic that matches your vision.



Out-of-the-Box Visualization

The out-of-the-box solutions are by far the easiest for beginners to pick up. Copy and paste some data or load a CSV file and you're set. Just click the graph type you want—maybe change some options here and there.

Options

The out-of-the-box tools available vary quite a bit, depending on the application they've been designed for. Some, such as Microsoft Excel or Google Documents, are meant for basic data management and graphs, whereas others were built for more thorough analyses and visual exploration.

MICROSOFT EXCEL

You know this one. You have the all-familiar spreadsheet where you put your data, such as in Figure 3-1.

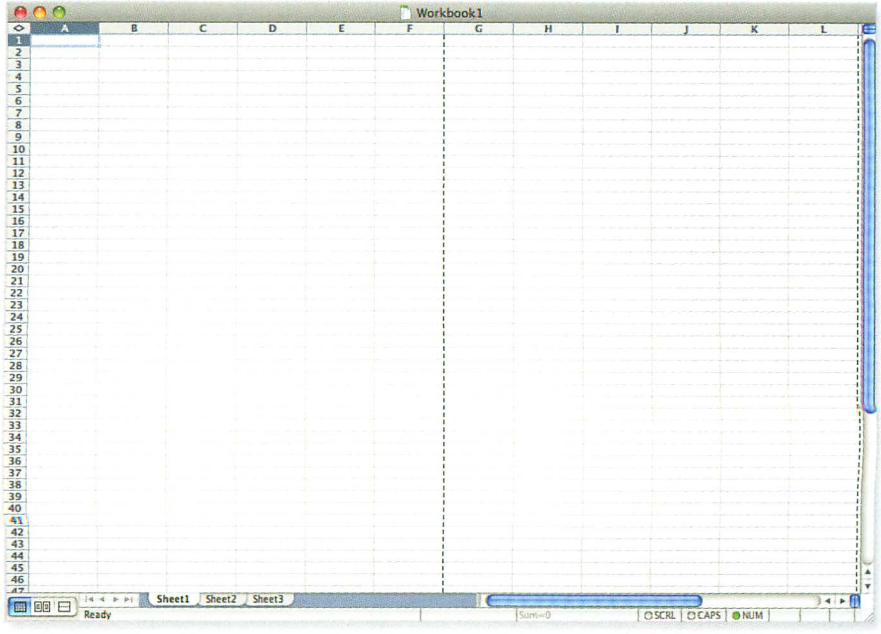


FIGURE 3-1 Microsoft Excel spreadsheet

Then you can click the button with the little bar graph on it to make the chart you want. You get all your standard chart types (Figure 3-2) such as the bar chart, line, pie, and scatterplot.

Some people scoff at Excel, but it's not all that bad for the right tasks. For example, I don't use Excel for any sort of deep analyses or graphics for a publication, but if I get a small dataset in an Excel file, as is often the case, and I want a quick feel for what is in front of me, then sure, I'll whip up a graph with a few clicks in everyone's favorite spreadsheet program.

GRAPHS REALLY *CAN* BE FUN

The first graph I made on a computer was in Microsoft Excel for my fifth grade science fair project. My project partner and I tried to find out which surface snails moved on the fastest. It was ground-breaking research, I assure you.

Even back then I remember enjoying the graph-making. It took me forever to learn (the computer was still new to me), but when I finally did, it was a nice treat. I entered numbers in a spreadsheet and then got a graph instantly that I could change to any color I wanted—blinding, bright yellow it is.

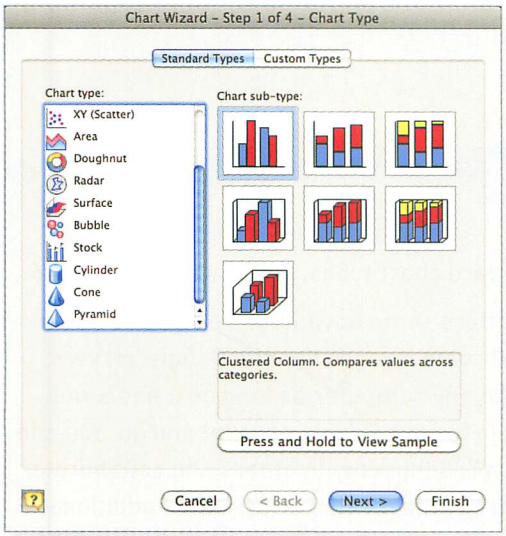


FIGURE 3-2 Microsoft Excel chart options

This ease of use is what makes Excel so appealing to the masses, and that's fine. If you want higher quality data graphics, don't stop here. Other tools are a better fit for that.

GOOGLE SPREADSHEETS

Google Spreadsheets is essentially the cloud version of Microsoft Excel with the familiar spreadsheet interface, obviously (Figure 3-3).

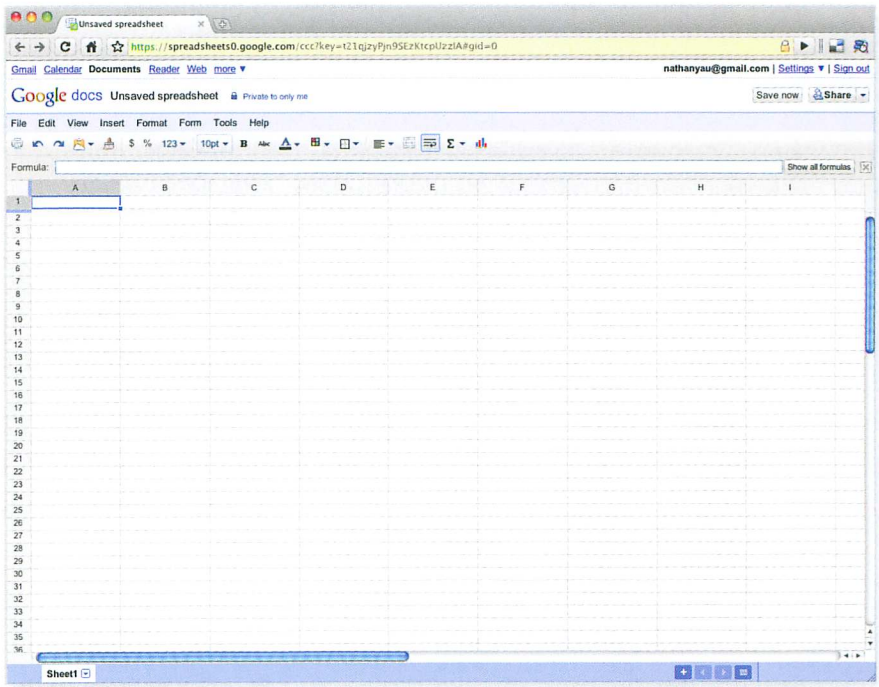


FIGURE 3-3 Google Spreadsheets

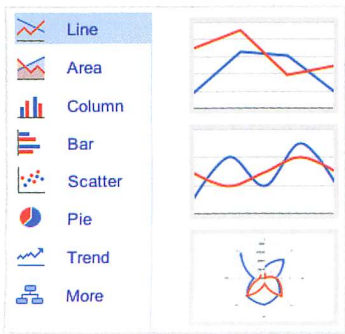


FIGURE 3-4 Google Spreadsheets charting options

It also offers your standard chart types, as shown in Figure 3-4.

Google Spreadsheets offers some advantages over Excel, however. First, because your data is stored on the Google servers, you can see your data on any computer as long as it has a web browser installed. Log in to your Google account and go. You can also easily share your spreadsheet with others and collaborate in real-time. Google Spreadsheets also offers some additional charting options via the Gadget option, as shown in Figure 3-5.

A lot of the gadgets are useless, but a few good ones are available. You can, for example, easily make a motion chart with your time series data (just like Hans Rosling). There's also an interactive time series chart that you might be familiar with if you've visited Google Finance, as shown in Figure 3-6.

▶ Visit Google Docs at `http://docs.google.com` to try spreadsheets.

FIGURE 3-5 Google gadgets

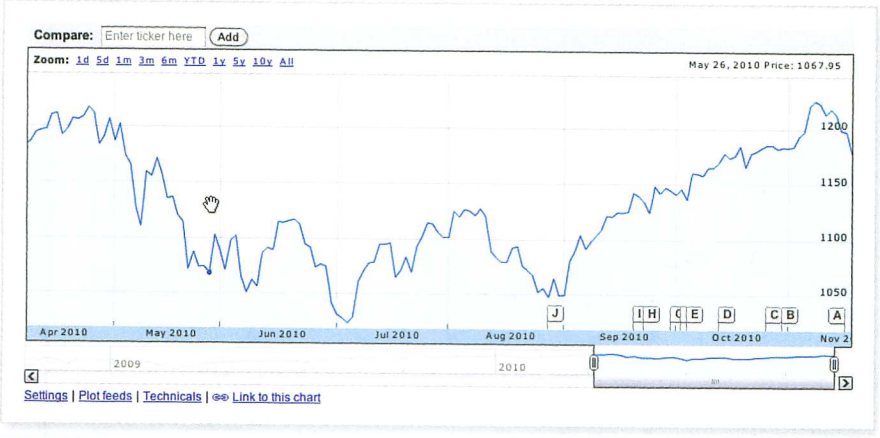


FIGURE 3-6 Google Finance

MANY EYES

Many Eyes is an ongoing research project by the IBM Visual Communication Lab. It's an online application that enables you to upload your data as a text-delimited file and explore through a set of interactive visualization tools. The original premise of Many Eyes was to see if people could explore large datasets as groups—therefore the name. If you have a lot of eyes on a large dataset, can a group find interesting points in the data quicker or more efficiently or find things in the data that you would not have found on your own?

Although social data analyses never caught on with Many Eyes, the tools can still be useful to the individual. Most traditional visualization types are available, such as the line graph (Figure 3-7) and the scatterplot (Figure 3-8).

One of the great things about all the visualizations on Many Eyes is that they are interactive and provide a number of customization options. The scatterplot, for example, enables you to scale dots by a third metric, and you can view individual values by rolling over a point of interest.

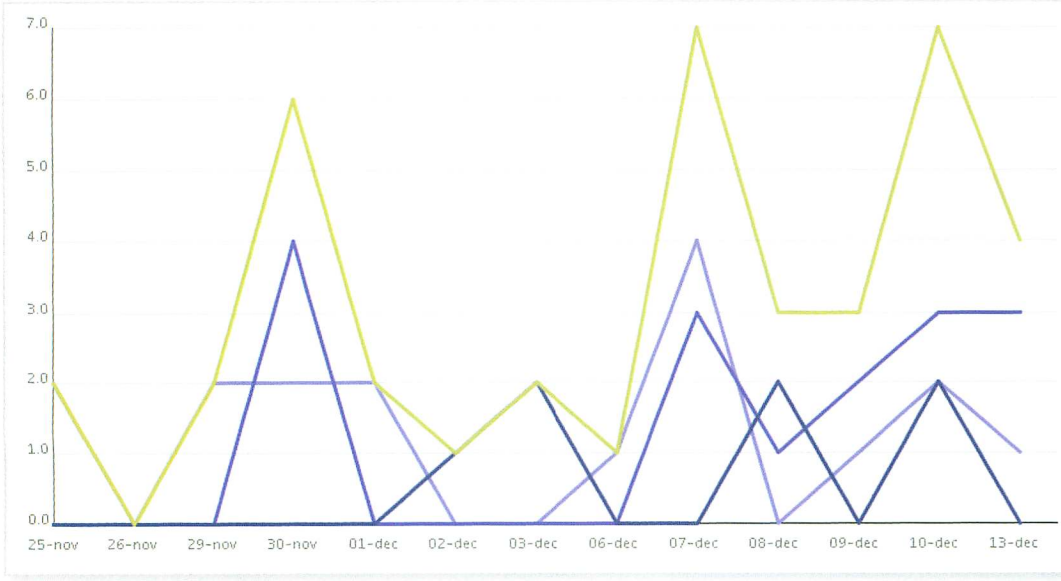


FIGURE 3-7 Line graph on Many Eyes

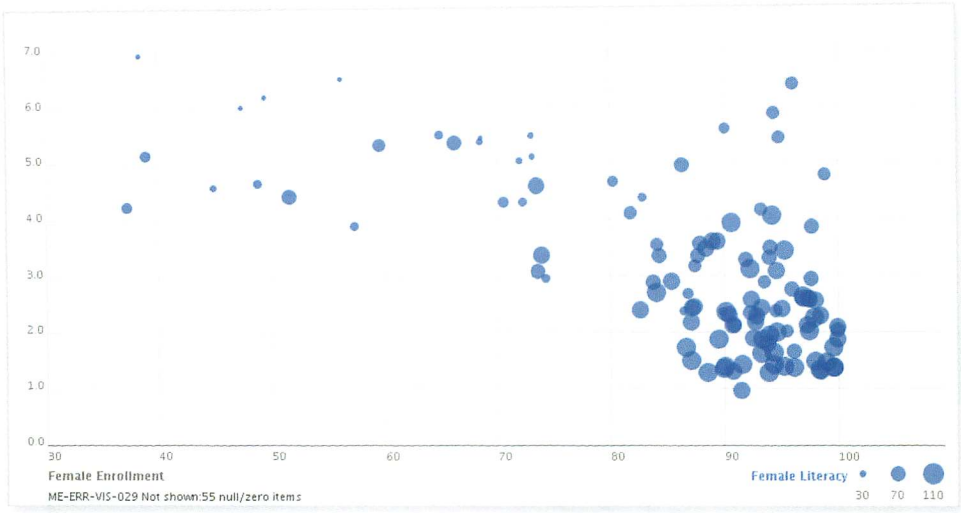


FIGURE 3-8 Scatterplot on Many Eyes

Many Eyes also provides a variety of more advanced and experimental visualizations, along with some basic mapping tools. A word tree helps you explore a full body of text, such as in a book or news article. You choose a word or a phrase, and you can see how your selection is used throughout the text by looking at what follows. Figure 3-9, for example, shows the results of a search for **right** in the United States Constitution.

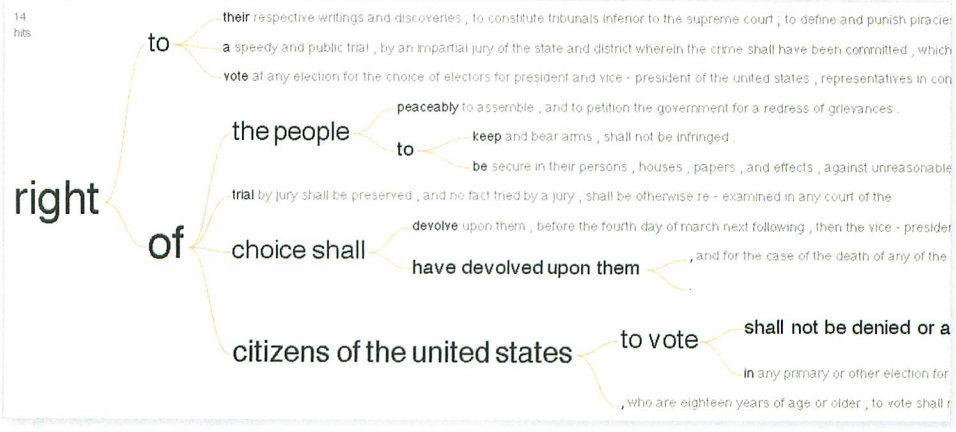


FIGURE 3-9 Word tree on Many Eyes showing parts of the United States Constitution

Alternatively, you can easily switch between tools, using the same data. Figure 3-10 shows the Constitution visualized with a stylized word cloud, known as a Wordle. Words used more often are sized larger.

FIGURE 3-10 Wordle of the United States Constitution

As you can see, Many Eyes has a lot of options to help you play with your data and is by far the most-extensive (and in my eyes, the best) free tool for data exploration; however, a couple of caveats exist. The first is that most of the tools are Java applets, so you can't do much if you don't have Java installed. (This isn't a big deal for most, but I know some people, for whatever reason, who are particular about what they put on their computer.)

▶ Try uploading and visualizing your own data `http://many-eyes.com`.

The other caveat, which can be a deal breaker, is that all the data you upload to the site is in the public domain. So you can't use Many Eyes, for example, to dig into customer information or sales made by your business.

TABLEAU SOFTWARE

Tableau Software, which is Windows-only software, is relatively new but has been growing in popularity for the past couple of years. It's designed mainly to explore and analyze data visually. It's clear that careful thought has been given to aesthetics and design, which is why so many people like it.

Tableau Software offers lots of interactive visualization tools and does a good job with data management, too. You can import data from Excel, text files, and database servers. Standard time series charts, bar graphs, pie

charts, basic mapping, and so on are available. You can mix and match these displays, hook in a dynamic data source for a custom view, or a dashboard, for a snapshot of what's going on in your data.

Most recently, Tableau released Tableau Public, which is free and offers a subset of the functionality in the desktop editions. You can upload your data to Tableau's servers, build an interactive display, and easily publish it to your website or blog. Any data you upload to the servers though, like with Many Eyes, does become publicly available, so keep that in mind.

If you want to use Tableau and keep your data private, you need to go with the desktop editions. At the time of this writing, the desktop software is on the pricier side at $999 and $1,999 for the Personal and Professional editions, respectively.

▶ Visit Tableau Software at `http://tableausoftware.com`. It has a full-functioning free trial.

YOUR.FLOWINGDATA

My interest in personal data collection inspired my own application, your.flowingdata (YFD). It's an online application that enables you to collect data via Twitter and then explore patterns and relationships with a set of interactive visualization tools. Some people track their eating habits or when they go to sleep and wake up. Others have logged the habits of their newborn as sort of a baby scrapbook, with a data twist.

YFD was originally designed with personal data in mind, but many have found the application useful for more general types of data collection, such as web activity or train arrivals and departures.

▶ Try personal data collection via Twitter at `http://your.flowingdata.com`.

Trade-Offs

Although these tools are easy to use, there are some drawbacks. In exchange for click-and-drag, you give up some flexibility in what you can do. You can usually change colors, fonts, and titles, but you're restricted to what the software offers. If there is no button for the chart you want, you're out of luck.

On the flip side, some software might have a lot of functions, but in turn have a ton of buttons that you need to learn. For example, there was one program (not listed here) that I took a weekend crash course for, and it was obvious that it could do a lot if I put in the time. The processes to get things done though were so counterintuitive that it made me not want to

learn anymore. It was also hard to repeat my work for different datasets, because I had to remember everything I clicked. In contrast, when you write code to handle your data, it's often easy to reuse code and plug in a different dataset.

Don't get me wrong. I'm not saying to avoid out-of-the-box software completely. They can help you explore your data quickly and easily. But as you work with more datasets, there will be times when the software doesn't fit, and when that time comes you can turn to programming.

Programming

This can't be stressed enough: Gain just a little bit of programming skills, and you can do so much more with data than if you were to stick only with out-of-the-box software. Programming skills give you the ability to be more flexible and more able to adapt to different types of data.

If you've ever been impressed by a data graphic that looked custom-made, most likely it was coded or designed in illustrative software. A lot of the time it's both. The latter is covered a little later.

Code can look cryptic to beginners—I've been there. But think of it as a new language because that's what it is. Each line of code tells the computer to do something. Your computer doesn't understand the way you talk to your friends, so you have to talk to the computer in its own language or syntax.

Like any language, you can't immediately start a conversation. Start with the basics first and then work your way up. Before you know it, you'll be coding. The cool thing about programming is that after you learn one language, it's much easier to learn others because the logic is similar.

Options

So you decide to get your hands dirty with code—good for you. A lot of options are freely available. Some languages are better at performing certain tasks better than others. Some solutions can handle large amounts of data, whereas others are not as robust in that department but can produce much better visuals or provide interaction. Which language you

use largely depends on what your goals are for a specific data graphic and what you're most comfortable with.

Some people stick with one language and get to know it well. This is fine, and if you're new to programming, I highly recommend this strategy. Familiarize yourself with the basics and important concepts of code.

Use the language that best suits your needs. However, it's fun to learn new languages and new ways to play with data; so you should develop a good bit of programming experience before you decide on your favorite solution.

PYTHON

The previous chapter discussed how Python can handle data. Python is good at that and can handle large amounts of data without crashing. This makes the language especially useful for analyses and heavy computation.

Python also has a clean and easy-to-read syntax that programmers like, and you can work off of a lot of modules to create data graphics, such as the graph in Figure 3-11.

From an aesthetic point of view, it's not great. You probably don't want to take a graphic from Python direct to publication. The output usually looks kind of rough around the edges. Nevertheless, it can be a good starting point in the data exploration stages. You might also export images and then touch them up or add information using graphic editing software.

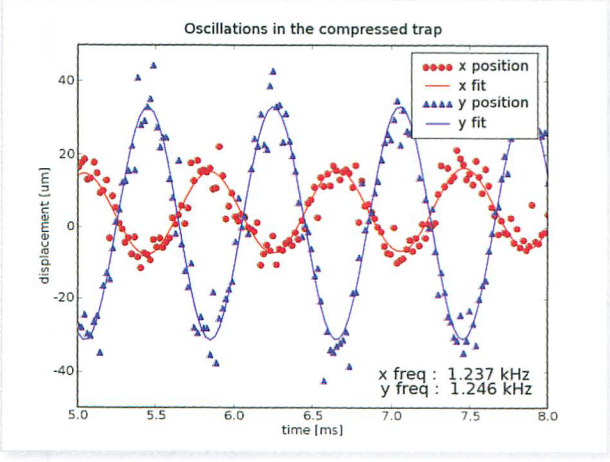


FIGURE 3-11 Graph produced in Python

USEFUL PYTHON RESOURCES

- Official Python website (`http://python.org`)
- NumPy and SciPy (`http://numpy.scipy.org/`)—Scientific computing

PHP

PHP was the first language I learned when I started programming for the web. Some people say it's messy, which it can be, but you can just as easily keep it organized. It's usually an easy setup because most web servers already have it installed, so it's easy to jump right in.

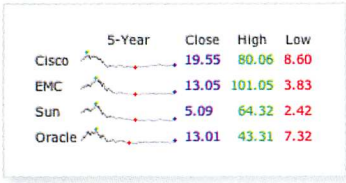


FIGURE 3-12 Sparklines using a PHP graphing library

There's a flexible PHP graphics library called GD that's also usually included in standard installs. The library enables you to create images from scratch or manipulate existing ones. Also a number of PHP graphing libraries exist that enable you to create basic charts and graphs. The most popular is the Sparklines Graphing Library, which enables you to embed small word-size graphs in text or add a visual component to a numeric table, as shown in Figure 3-12.

Most of the time PHP is coupled with a database such as MySQL, instead of working with a lot of CSV files, to maximize usage and to work with hefty datasets.

USEFUL PHP RESOURCES

- Official PHP website (`http://php.net`)
- Sparkline PHP Graphing Library (`http://sparkline.org`)

PROCESSING

Processing is an open-source programming language geared toward designers and data artists. It started as a coding sketchbook in which you could produce graphics quickly; however, it developed a lot since its early days, and many high-quality projects have been created in Processing. For example, *We Feel Fine*, mentioned in Chapter 1, "Telling Stories with Data," was created in Processing.

The great thing about Processing is that you can quickly get up and running. The programming environment is lightweight, and with just a few lines of code, you can create an animated and interactive graphic. It would of course be basic, but because it was designed with the creation of visuals in mind, you can easily learn how to create more advanced pieces.

Although the audience was originally for designers and artists, the community around Processing has grown to be a diverse group. Many libraries can help you do more with the language.

One of the drawbacks is that you do end up with a Java applet, which can be slow to load on some people's computers, and not everyone has Java installed. (Although most people do.) There's a solution for that, though. There's a JavaScript version of Processing recently out of development and ready to use.

Nevertheless, this is a great place to start for beginners. Even those who don't have any programming experience can make something useful.

USEFUL PROCESSING RESOURCE

- Processing (`http://processing.org`)—Official site for Processing

FLASH AND ACTIONSCRIPT

Most interactive and animated data graphics on the web, especially on major news sites such as *The New York Times*, are built in Flash and ActionScript. You can design graphics in just Flash, which is a click-and-drag interface, but with ActionScript you have more control over interactions. Many applications are written completely in ActionScript, without the use of the Flash environment. However, the code compiles as a Flash application.

For example, an interactive map that animates the growth of Walmart, as shown in Figure 3-13, was written in ActionScript. The Modest Maps library was used, which is a display and interaction library for tile-based maps. It's BSD-licensed, meaning it's free, and you can use it for whatever you want.

NOTE

Although there are many free and open-source ActionScript libraries, Flash and Flash builders can be pricey, which you should consider in your choice of software.

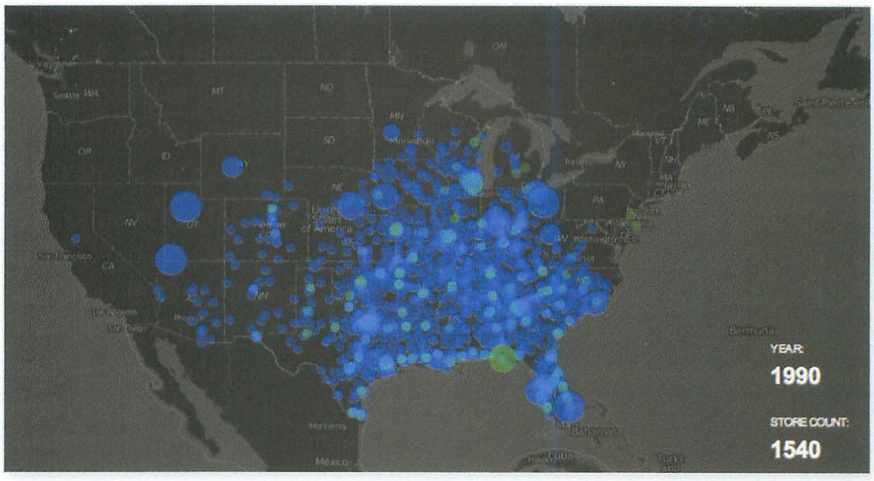


FIGURE 3-13 Map animating the growth of Walmart, written in ActionScript

The interactive stacked area chart in Figure 3-14 was also written in ActionScript. It enables you to search for spending categories over the years. The Flare ActionScript library by the UC Berkeley Visualization Lab was used to do most of the heavy lifting.

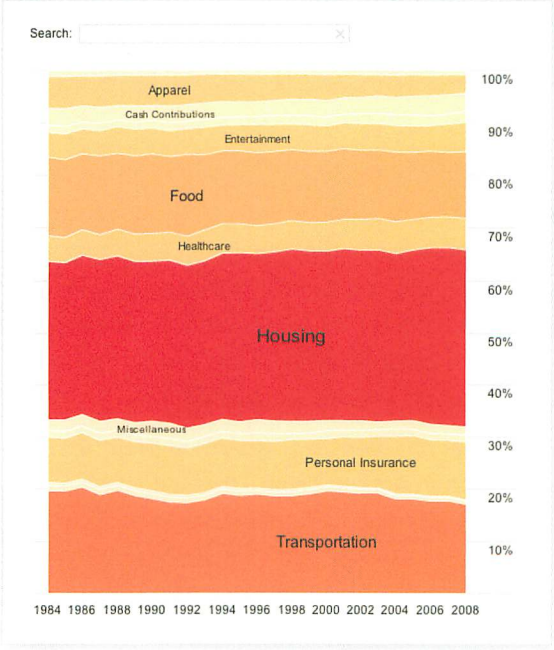


FIGURE 3-14 Interactive stacked area chart showing consumer spending breakdowns, written in ActionScript

If you want to get into interactive graphics for the web, Flash and ActionScript is an excellent option. Flash applications are relatively quick to load, and most people already have Flash installed on their computers.

It's not the easiest language to pick up; the syntax isn't that complicated, but the setup and code organization can overwhelm beginners. You're not going to have an application running with just a few lines of code like you would with Processing. Later chapters take you through the basic steps, and you can find a number of useful tutorials online because Flash is so widely used.

Also, as web browsers improve in speed and efficiency, you have a growing number of alternatives.

USEFUL FLASH AND ACTIONSCRIPT RESOURCES

- Adobe Support `www.adobe.com/products/flash/whatisflash/`)—Official documentation for Flash and ActionScript (and other Adobe products)
- Flare Visualization Toolkit (`http://flare.prefuse.org`)
- Modest Maps (`http://modestmaps.com`)

HTML, JAVASCRIPT, AND CSS

Web browsers continue to get faster and improve in functionality. A lot of people spend more time using their browsers than any other application on their computers. More recently, there has been a shift toward visualization that runs native in your browser via HTML, JavaScript, and CSS. Data graphics used to be primarily built in Flash and ActionScript if there were an interactive component or saved as a static image. This is still often the case, but it used to be that these were the only options.

Now there are several robust packages and libraries that can help you quickly build interactive and static visualizations. They also provide a lot of options so that you can customize the tools for your data needs.

For example, Protovis, maintained by the Stanford Visualization Group, is a free and open-source visualization library that enables you to create web-native visualizations. Protovis provides a number of out-of-the-box visualizations, but you're not at all limited by what you can make,

geometrically speaking. Figure 3-15 shows a stacked area chart, which can be interactive.

This chart type is built into the Protovis, but you can also go with a less traditional streamgraph, as shown in Figure 3-16.

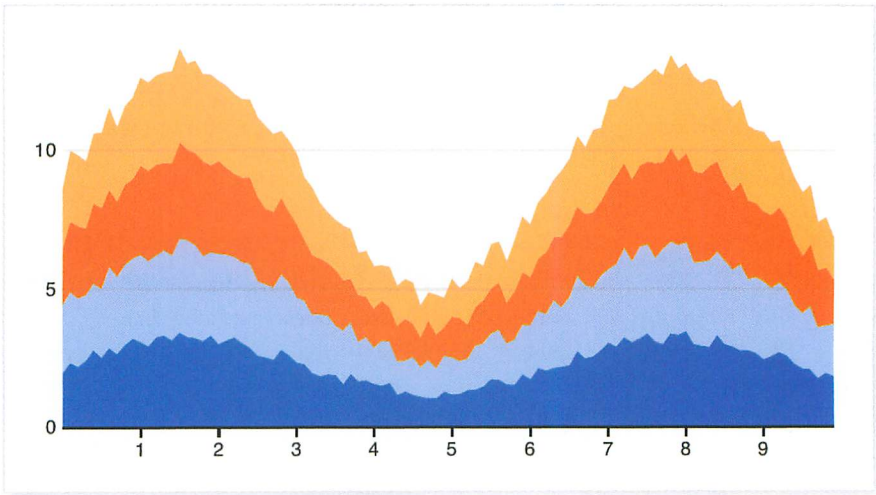


FIGURE 3-15 Stacked area chart with Protovis

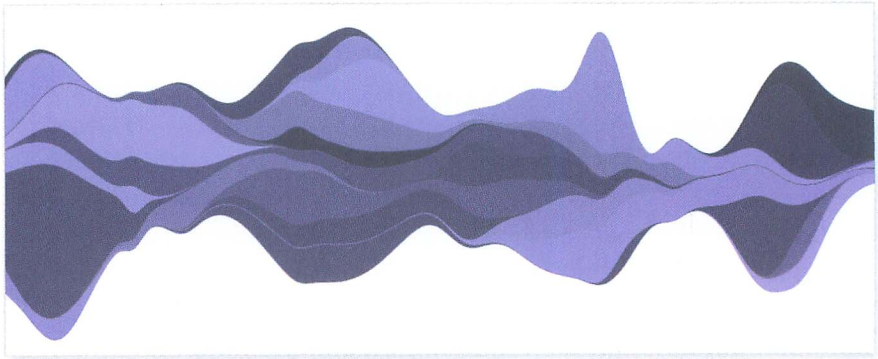


FIGURE 3-16 Custom-made streamgraph with Protovis

You can also easily use multiple libraries for increased functionality. This is possible in Flash, but JavaScript can be a lot less heavy code-wise. JavaScript is also a lot easier to read and use with libraries such

as jQuery and MooTools. These are not visualization-specific but are useful. They provide a lot of basic functionality with only a few lines of code. Without the libraries, you'd have to write a lot more, and your code can get messy in a hurry.

Plugins for the libraries can also help you with some of your basic graphics. For example, you can use a Sparkline plugin for jQuery to make small charts (see Figure 3-17).

FIGURE 3-17 Sparklines with jQuery Sparklines plugin

You can also do this with PHP, but this method has a couple of advantages. First, the graphic is generated in a user's browser instead of the server. This relieves stress off your own machines, which can be an issue if you have a website with a lot of traffic.

The other advantage is that you don't need to set up your server with the PHP graphics library. A lot of servers are set up with graphics installed, but sometimes they are not. Installation can be tedious if you're unfamiliar with the system.

You might not want to use a plugin at all. You can also design a custom visualization with standard web programming. Figure 3-18, for example, is an interactive calendar that doubles as a heatmap in your.flowingdata.

There are, however, a couple of caveats. Because the software and technology are relatively new, your designs might look different in different browsers. Some of the previously mentioned tools won't work correctly in an old browser such as Internet Explorer 6. This is becoming less of a problem though, because most people use modern browsers such as Firefox or Google Chrome. In the end it depends on your audience. Less than 5 percent of visitors to FlowingData use old versions of Internet Explorer, so compatibility isn't much of an issue.

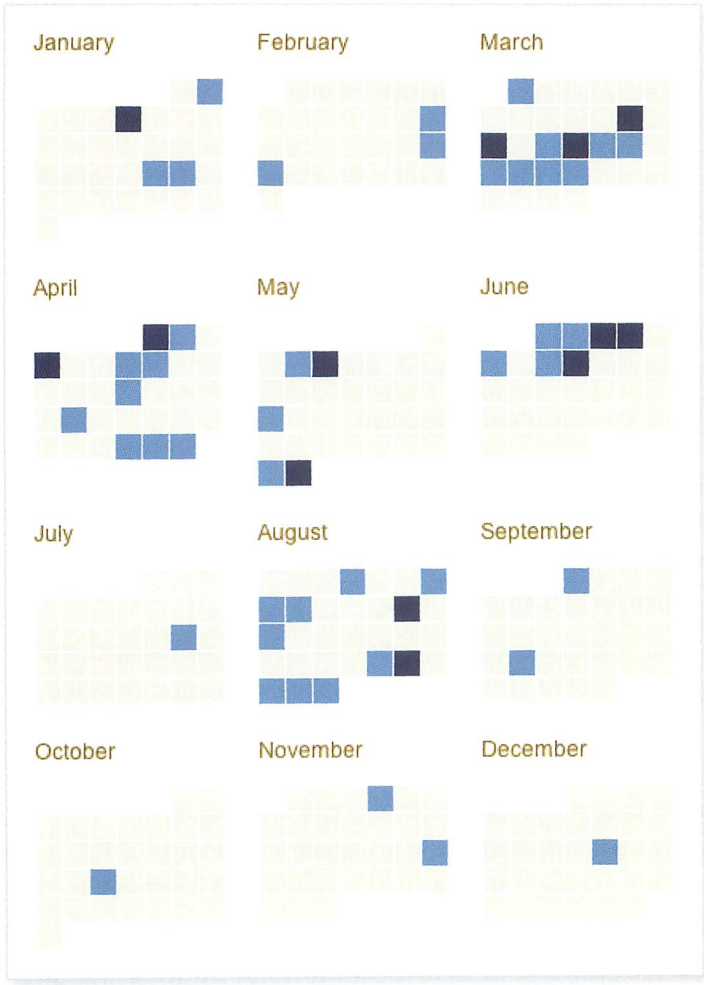


FIGURE 3-18 Interactive calendar that also serves as a heatmap in your.flowingdata

Also related to the age of the technology, there aren't as many libraries available for visualization in JavaScript as there are in Flash and ActionScript. This is why many major news organizations still use a lot of Flash, but this will change as development continues.

USEFUL HTML, JAVASCRIPT, AND CSS RESOURCES

- jQuery (`http://jquery.com/`)—A JavaScript library that makes coding in the language much more efficient and makes your finished product easier to read.
- jQuery Sparklines (`http://omnipotent.net/jquery.sparkline/`)—Make static and animated sparklines in JavaScript.
- Protovis (`http://vis.stanford.edu/protovis/`)—A visualization-specific JavaScript library designed to learn by example.
- JavaScript InfoVis Toolkit (`http://datafl.ws/15f`)—Another visualization library, although not quite as developed as Protovis.
- Google Charts API (`http://code.google.com/apis/chart/`)—Build traditional charts on-the-fly, simply by modifying a URL.

R

If you read FlowingData, you probably know that my favorite software for data graphics is R. It's free and open-source statistical computing software, which also has good statistical graphics functionality. It is also most statisticians' analysis software of choice. There are paid alternatives such as S-plus and SAS, but it's hard to beat the price of free and an active development community.

One of the advantages that R has over the previously mentioned software is that it was specifically designed to analyze data. HTML was designed to make web pages, and Flash is used for tons of other things, such as video and animated advertisements. R, on the other hand, was built and is maintained by statisticians for statisticians, which can be good and bad depending on what angle you're looking from.

There are lots of R packages that enable you to make data graphics with just a few lines of code. Load your data into R, and you can have a graphic with even just one line of code. For example, you can quickly make a treemap using the Portfolio package, as shown in Figure 3-19.

Just as easily, you can build a heatmap, as shown in Figure 3-20.

And of course, you can also make more traditional statistical graphics, such as scatterplots and time series charts, which are discussed in Chapter 4, "Visualizing Patterns over Time."

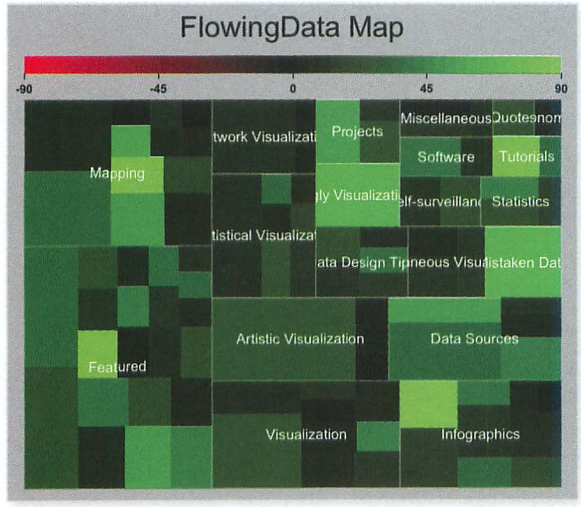


FIGURE 3-19 Treemap generated in R with the Portfolio package

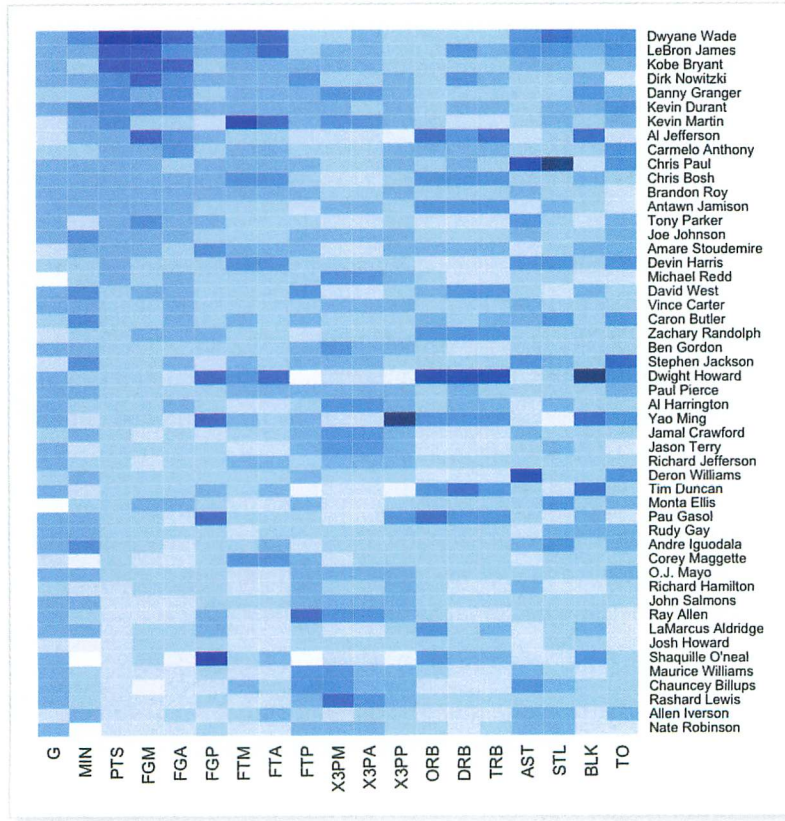


FIGURE 3-20 Heatmap generated in R

To be completely honest though, the R site looks horribly out-dated (Figure 3-21), and the software itself isn't very helpful in guiding new users. You need to remember though that R is a programming language, and you're going to get that with any language you use. The few bad things that I've read about R are usually written by people who are used to buttons and clicking and dragging. So when you come to R don't expect a clicky interface, or you will of course find the interface unfriendly.

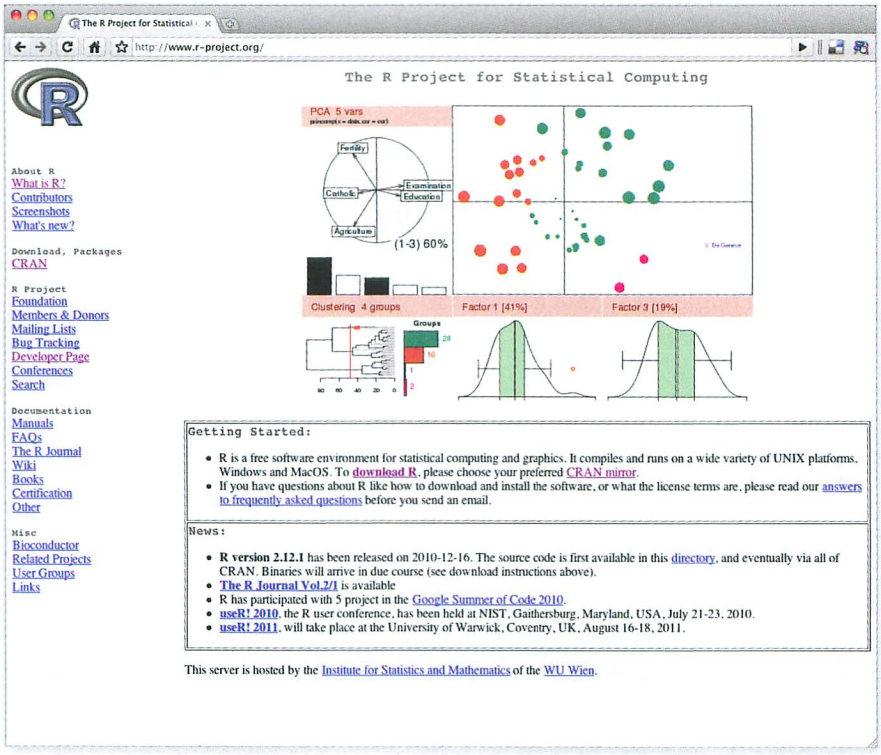


FIGURE 3-21 R homepage, `www.r-project.org`

But get past that, and there's a lot you can do. You can make publication-quality graphics (or at least the beginnings of them), and you can learn to embrace R's flexibility. If you like, you can write your own functions and packages to make graphics the way you want, or you can use the ones that others have made available in the R library.

TIP

When you search for something about R on the web via search engines, the basic name can sometimes throw off your results. Instead, try searching for **r-project** instead of just **R**, along with what you're looking for. You'll usually find more relevant search results.

R provides base drawing functions that basically enable you to draw what you want. You can draw lines, shapes, and axes within a plotting framework, so again, like the other programming solutions, you're limited only by your imagination. Then again, practically every chart type is available via some R package.

Why would you use anything besides R? Why not just do everything in R? Following are a few reasons. R works on your desktop, so it's not a good fit for the dynamic web. Saving graphics and images and putting them on a web page isn't a problem, but it's not going to happen automatically. You can generate graphics on-the-fly via the web, but so far, the solutions aren't particularly robust when you compare them to the web-native stuff such as JavaScript.

R is also not good with interactive graphics and animation. Again, you can do this in R, but there are better, more elegant ways to accomplish this using, for example, Flash or Processing.

Finally, you might have noticed that the graphics in Figures 3-19 and 3-20 lack a certain amount of polish. You probably won't see graphics like that in a newspaper any time soon. You can tighten up the design in R by messing with different options or writing additional code, but my strategy is usually to make the base graphic in R and then edit and refine in design software such as Adobe Illustrator, which is discussed soon. For analyses, the raw output from R does just fine, but for presentation and storytelling, it's best to adjust aesthetics.

USEFUL R RESOURCE

- R Project for Statistical Computing (`www.r-project.org`)

Trade-Offs

Learning programming is learning a new language. It's your computer's language of bits and logic. When you work with Excel or Tableau for example, you essentially work with a translator. The buttons and menus are in your language, and when you click items, the software translates your interaction

and then sends the translation to your computer. The computer then does something for you, such as makes a graph or processes some data.

So time is definitely a major hurdle. It takes time for you to learn a new language. For a lot of people, this hurdle is too high which I can relate to. You need to get work done now because you have a load of data sitting in front of you, and people waiting on results. If that's the case, in which you have only this single data-related task with nothing in the future, it might be better to go with the out-of-the-box visualization tools.

However, if you want to tackle your data and will most likely have (or want) lots of data-related projects in the future, the time spent learning how to program now could end up as saved time on other projects, with more impressive results. You'll get better at programming on each project you go through, and it'll start to come much easier. Just like any foreign language, you don't start writing books in that language; you start with the essentials and then branch out.

Here's another way to look at it. Hypothetically speaking, say you're tossed into a foreign country, and you don't speak the language. Instead, you have a translator. (Stay with me on this one. I have a point.) To talk to a local, you speak, and then your translator forwards the message. What if the translator doesn't know the meaning or the right word for something you just said? He could leave the word out, or if he's resourceful, he can look it up in a translation dictionary.

For out-of-the-box visualization tools, the software is the translator. If it doesn't know how to do something, you're stuck or have to try an alternative method. Unlike the speaking translator, software usually doesn't instantly learn new words, or in this case, graph types or data handling features. New functions come in the form of software updates, which you have to wait for. So what if you learn the language yourself?

Again, I'm not saying to avoid out-of-the-box tools. I use them all the time. They make a lot of tedious tasks quick and easy, which is great. Just don't let the software restrict you.

As you see in later chapters, programming can help you get a lot done with much less effort than if you were to do it all by hand. That said, there are also things better done by hand, especially when you're telling stories with data. That brings you to the next section on illustration: the opposite end of the visualization spectrum.

Illustration

Now you're in graphic designers' comfort zone. If you're an analyst or in a more technical field, this is probably unfamiliar territory. You can do a lot with a combination of code and out-of-the-box visualization tools, but the resulting data graphics almost always have that look of something that was automatically generated. Maybe labels are out of place or a legend feels cluttered. For analyses, this is usually fine—you know what you're looking at.

However, when you make graphics for a presentation, a report, or a publication, more polished data graphics are usually appropriate so that people can clearly see the story you're telling.

For example, Figure 3-19 is the raw output from R. It shows views and comments on FlowingData for 100 popular posts. Posts are separated by category such as Mapping. The brighter the green, the more comments on that post, and the larger the rectangle, the more views. You wouldn't know that from the original, but when I was looking at the numbers, I knew what I was looking at, because I'm the one who wrote the code in R.

Figure 3-22 is a revised version. The labels have been adjusted so that they're all readable; lead-in copy has been added on the top so that readers know what they're looking at; and the red portion of the color legend was removed because there is no such thing as a post having a negative number of comments. I also changed the background to white from gray just because I think it looks better.

I could have edited the code to fit my specific needs, but it was a lot easier to click-and-drag in Adobe Illustrator. You can either make graphics completely with illustration software, or you can import graphics that you've made in, for example, R, and edit it to your liking. For the former, your visualization choices are limited because visualization is not the primary purpose of the software. For anything more complex than a bar chart, your best bet is to go with the latter. Otherwise, you will have to do a lot of things by hand, which is prone to mistakes.

The great thing about using illustration software is that you have more control over individual elements, and you can do everything by clicking and dragging. Change the color of bars or a single bar, modify axes width, or annotate important features with a few mouse clicks.

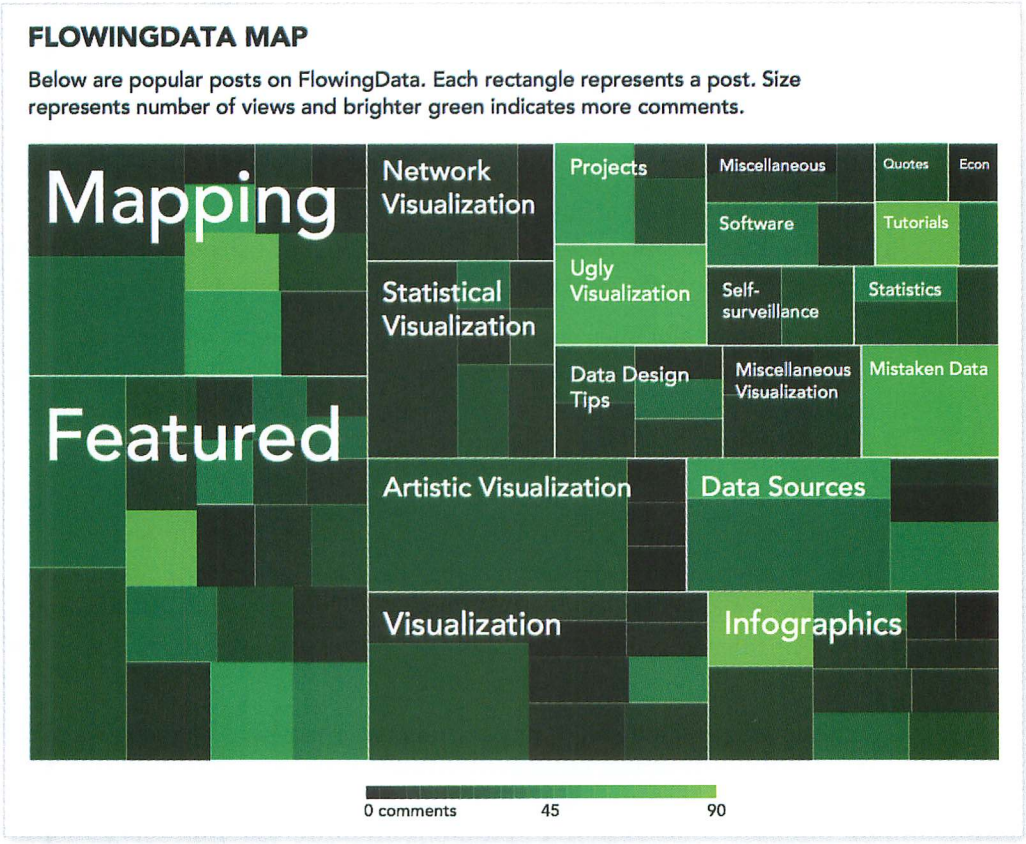


FIGURE 3-22 Treemap created in R, and edited in Adobe Illustrator

Options

A lot of illustration programs are available but only a few that most people use—and one that almost everyone uses. Cost will most likely be your deciding factor. Prices range from free (and open-source) to several hundred dollars.

ADOBE ILLUSTRATOR

Any static data graphic that looks custom-made or is in a major news publication most likely passed through Adobe Illustrator at some point. Adobe Illustrator is the industry standard. Every graphic that goes to print at *The New York Times* either was created or edited in Illustrator.

Illustrator is so popular for print because you work with vectors instead of pixels. This means you can make the graphics big without decreasing the quality of your image. In contrast, if you were to blow up a low-resolution photograph, which is a set number of pixels, you would end up with a pixelated image.

The software was originally designed for font development and later became popular among designers for illustrations such as logos and more art-focused graphics. And that's still what Illustrator is primarily used for.

However, Illustrator does offer some basic graphing functionality via its Graph tool. You can make the more basic graph types such as bar graphs, pie charts, and time series plots. You can paste your data into a small spreadsheet, but that's about the extent of the data management capabilities.

The best part about using Illustrator, in terms of data graphics, is the flexibility that it provides and its ease of use, with a lot of buttons and functions. It can be kind of confusing at first because there are so many, but it's easy to pick up, as you'll see in Chapter 4, "Visualizing Patterns over Time." It's this flexibility that enables the best data designers to create the most clear and concise graphics.

Illustrator is available for Windows and Mac. The downside though is that it's expensive when you compare it to doing everything with code, which is free, assuming you already have the machine to install things on. However, compared to some of the out-of-the-box solutions, Illustrator might not seem so pricey.

As of this writing, the most recent version of Illustrator is priced at $599 on the Adobe site, but you should find substantial discounts elsewhere (or go for an older version). Adobe also provides large discounts to students and those in academia, so be sure to check those out. (It's the most expensive software I've ever purchased, but I use it almost every day.)

USEFUL ADOBE ILLUSTRATOR RESOURCES

- Adobe Illustrator Product Page (`www.adobe.com/products/illustrator/`)
- VectorTuts (`http://vectortuts.com`)—Thorough and straightforward tutorials on how to use Illustrator

INKSCAPE

Inkscape is the free and open-source alternative to Adobe Illustrator. So if you want to avoid the hefty price tag, Inkscape is your best bet. I always use Illustrator because when I started to learn the finer points of data graphics on the job, Illustrator was what everyone used, so it just made sense. I have heard good things about Inkscape though, and because it's free, there's no harm in trying it. Just don't expect as many resources on how to use the software.

USEFUL INKSCAPE RESOURCES

- Inkscape (`http://inkscape.org`)
- Inkscape Tutorials (`http://inkscapetutorials.wordpress.com/`)

TIP

Parts of this book use Adobe Illustrator to refine your data graphics; however, it shouldn't be too hard to figure out how to do the same thing in Inkscape. Many of the tools and functions are similarly named.

OTHERS

Illustrator and Inkscape are certainly not your only options to create and polish your data graphics. They just happen to be the programs that most people use. You might be comfortable with something else. Some people are fond of Corel Draw, which is Windows-only software and approximately the same price as Illustrator. It might be slightly cheaper, depending on where you look.

There are also programs such as Raven by Aviary and Lineform, that offer a smaller toolset. Remember that Illustrator and Inkscape are general tools for graphic designers, so they provide a lot of functionality. But if you just want to make a few edits to existing graphics, you might opt for the simpler (lower-priced) software.

Trade-Offs

Illustration software is for just that—illustration. It's not made specifically for data graphics. It's meant for graphic design, so many people do not use a lot of functions offered by Illustrator or Inkscape. The software is also not good for handling a lot of data, compared to when you program or use visualization-specific tools. Because of that, you can't explore your data in these programs.

That said, these programs are a must if you want to make publication-level data graphics. They don't just help with aesthetics, but also readability and clarity that's often hard to achieve with automatically generated output.

Mapping

Some overlap exists between the covered visualization tools and the ones that you use to map geographic data. However, the amount of geographic data has increased significantly in the past years as has the number of ways you can map. With mobile location services on the rise, there will be more data with latitude and longitude coordinates attached to it. Maps are also an incredibly intuitive way to visualize data, and this deserves a closer look.

Mapping in the early days of the web wasn't easy; it wasn't elegant either. Remember the days you would go to MapQuest, look up directions, and get this small static map? Yahoo had the same thing for a while.

It wasn't until a couple of years later until Google provided a *slippy map* implementation (Figure 3-23). The technology was around for a while, but it wasn't useful until most people's Internet speed was fast enough to handle the continuous updating. Slippy maps are what we're used to nowadays. We can pan and zoom maps with ease, and in some cases, maps aren't just for directions; they're the main interface to browse a dataset.

NOTE

Slippy maps are the map implementation that is now practically universal. Large maps, that would normally not fit on your screen, are split into smaller images, or tiles. Only the tiles that fit in your window display, and the rest are hidden from view. As you drag the map, other tiles display, making it seem as if you're moving around a single large map. You might have also seen this done with high-resolution photographs.

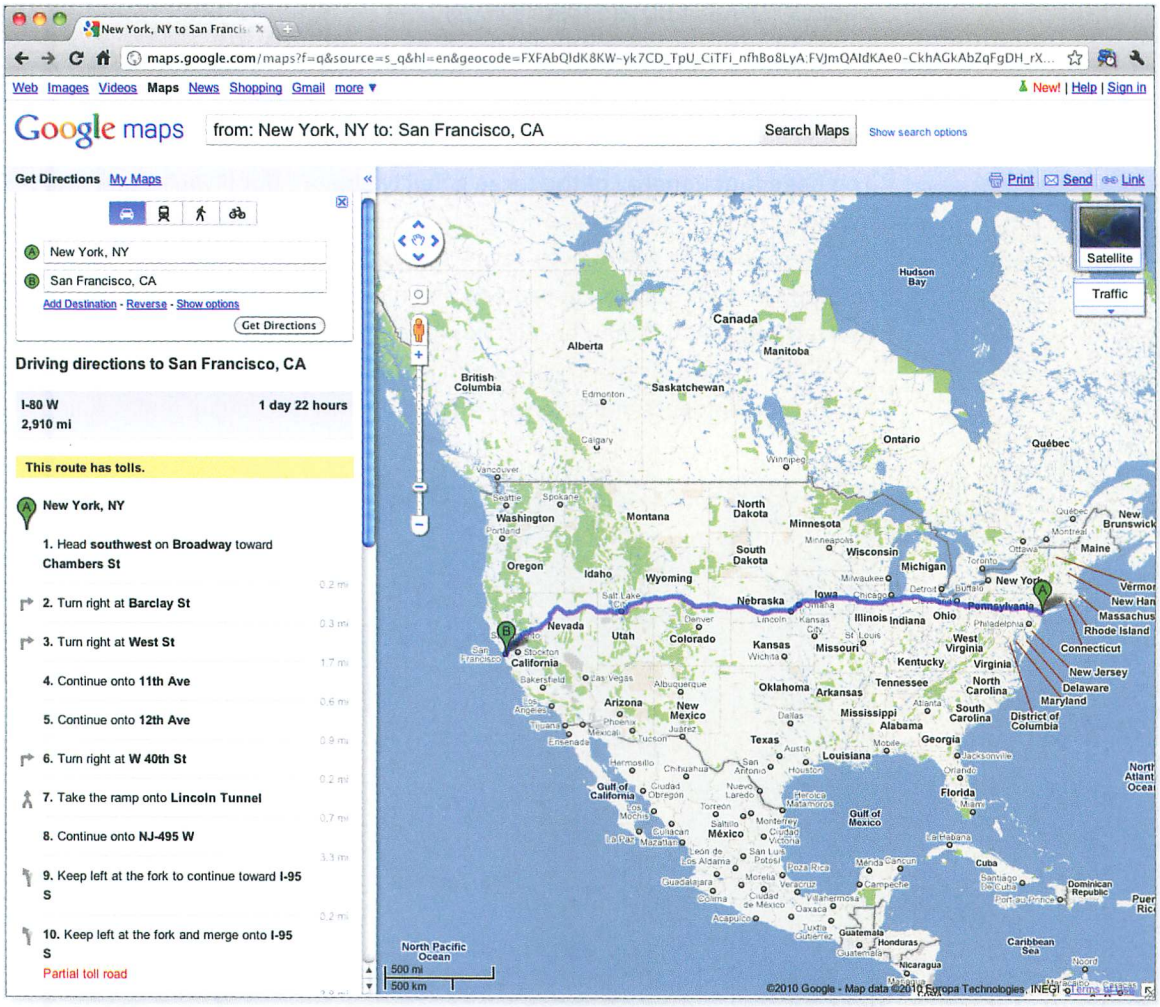


FIGURE 3-23 Google Maps to look up directions

Options

Along with all the geographic data making its way into the public domain, a variety of tools to map that data have also sprung up. Some require only a tiny bit of programming to get something up and running whereas others need a little more work. There are also a few other solutions that don't require programming.

GOOGLE, YAHOO, AND MICROSOFT MAPS

This is your easiest online solution; although, it does require a little bit of programming. The better you can code, the more you can do with the mapping APIs offered by Google, Yahoo, and Microsoft.

The base functionality of the three is fairly similar, but if you're just starting out, I recommend you go with Google. It seems to be the most reliable. They have a Maps API in both JavaScript and Flash, along with other geo-related services such as geocoding and directions. Go through the Getting Started tutorial and then branch out to other items such as placing markers (Figure 3-24), drawing paths, and adding overlays. The comprehensive set of code snippets and tutorials should quickly get you up and running .

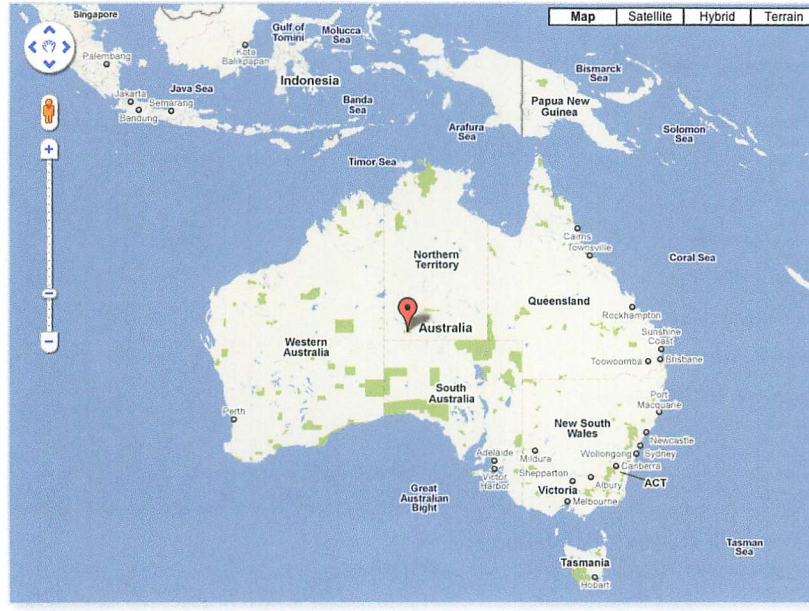


FIGURE 3-24 Marker placement on Google Maps

Yahoo also has JavaScript and Flash APIs for mapping, plus some geoservices, but I'm not sure how long it'll be around given the current state of the company. As of this writing, Yahoo has shifted focus from applications and development to content provider. Microsoft also provides a JavaScript API (under the Bing name) and one in Silverlight, which was its answer to Flash.

USEFUL MAPPING API RESOURCES

- Google Maps API Family (`http://code.google.com/apis/maps/`)
- Yahoo! Maps Web Services (`http://code.google.com/apis/maps/index.html`)
- Bing Maps API (`http://www.microsoft.com/maps/developers/web.aspx`)

ARCGIS

The previously mentioned online mapping services are fairly basic in what they can do at the core. If you want more advanced mapping, you'll most likely need to implement the functionality yourself. ArcGIS, built for desktop mapping, is the opposite. It's a massive program that enables you to map lots of data and do lots of stuff with it, such as smoothing and processing. You can do all this through a user interface, so there's no code required.

Any graphics department with mapping specialists most likely uses ArcGIS. Professional cartographers use ArcGIS. Some people love it. So if you're interested in producing detailed maps, it's worth checking out ArcGIS.

I have used ArcGIS only for a few projects because I tend to take the programming route when I can, and I just didn't need all that functionality. The downside of such a rich feature set is that there are so many buttons and menus to go through. Online and server solutions are also available, but they feel kind of clunky compared to other implementations.

USEFUL ARCGIS RESOURCE

- ArcGIS Product Page (`www.esri.com/software/arcgis/`)

MODEST MAPS

I mentioned Modest Maps earlier, with an example in Figure 3-13. It shows the growth of Walmart. Modest Maps is a Flash and ActionScript library

for tile-based maps, and there is support for Python. It's maintained by a group of people who know their online mapping and do great work for both clients and for fun, which should tell you a little something about the quality of the library.

The fun thing about Modest Maps is that it's more of a framework than a mapping API like the one offered by Google. It provides the bare minimum of what it takes to create an online map and then gets out of the way to let you implement what you want. You can use tiles from different providers, and you can customize the maps to fit with your application. For example, Figure 3-13 has a black-and-blue theme, but you can just as easily change that to white and red, as shown in Figure 3-25.

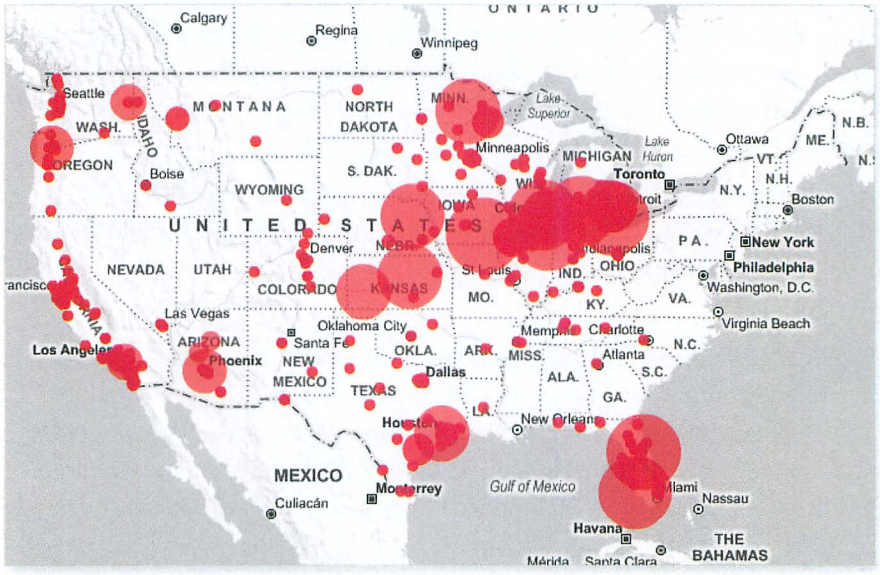


FIGURE 3-25 White-and-red themed map using Modest Maps

It's BSD-licensed, so you can do just about anything you want with it at no cost. You do have to know the ropes around Flash and ActionScript, but the basics are covered in Chapter 8, "Visualizing Spatial Relationships."

POLYMAPS

Polymaps is kind of like the JavaScript version of Modest Maps. It was developed and is maintained by some of the same people and provides the

same functionality—and then some. Modest Maps provides only the basics of mapping, but Polymaps has some built-in features such as choropleths (Figure 3-26) and bubbles.

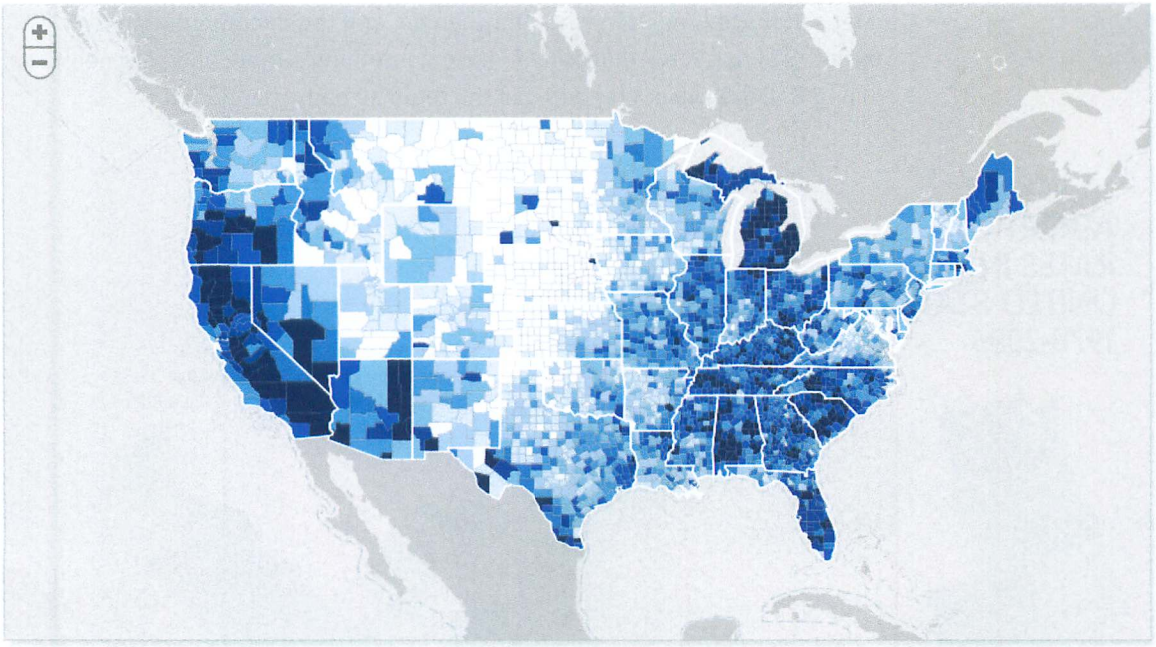


FIGURE 3-26 Choropleth map showing unemployment, implemented in Polymaps

Because it's JavaScript, it does feel more lightweight (because it requires less code), and it works in modern browsers. Polymaps uses Scalable Vector Graphics (SVG) to display data, so it doesn't work in the old versions of Internet Explorer, but most people are up-to-date. As a reference, only about 5 percent of FlowingData visitors use a browser that's too old, and I suspect that percentage will approach zero soon.

My favorite plus of a mapping library in JavaScript is that all the code runs native in the browser. You don't have to do any compiling or Flash exports, which makes it easier to get things running and to make updates later.

USEFUL POLYMAPS RESOURCE

- Polymaps (`http://polymaps.org/`)

R

R doesn't provide mapping functionality in the base distribution, but there are a few packages that let you do so. Figure 3-27 is a map that I made in R. The annotation was added after the fact in Adobe Illustrator.

Maps in R are limited in what they can do, and the documentation isn't great. So I use R for mapping if I have something simple and I happen to be using R. Otherwise, I tend to use the tools already mentioned.

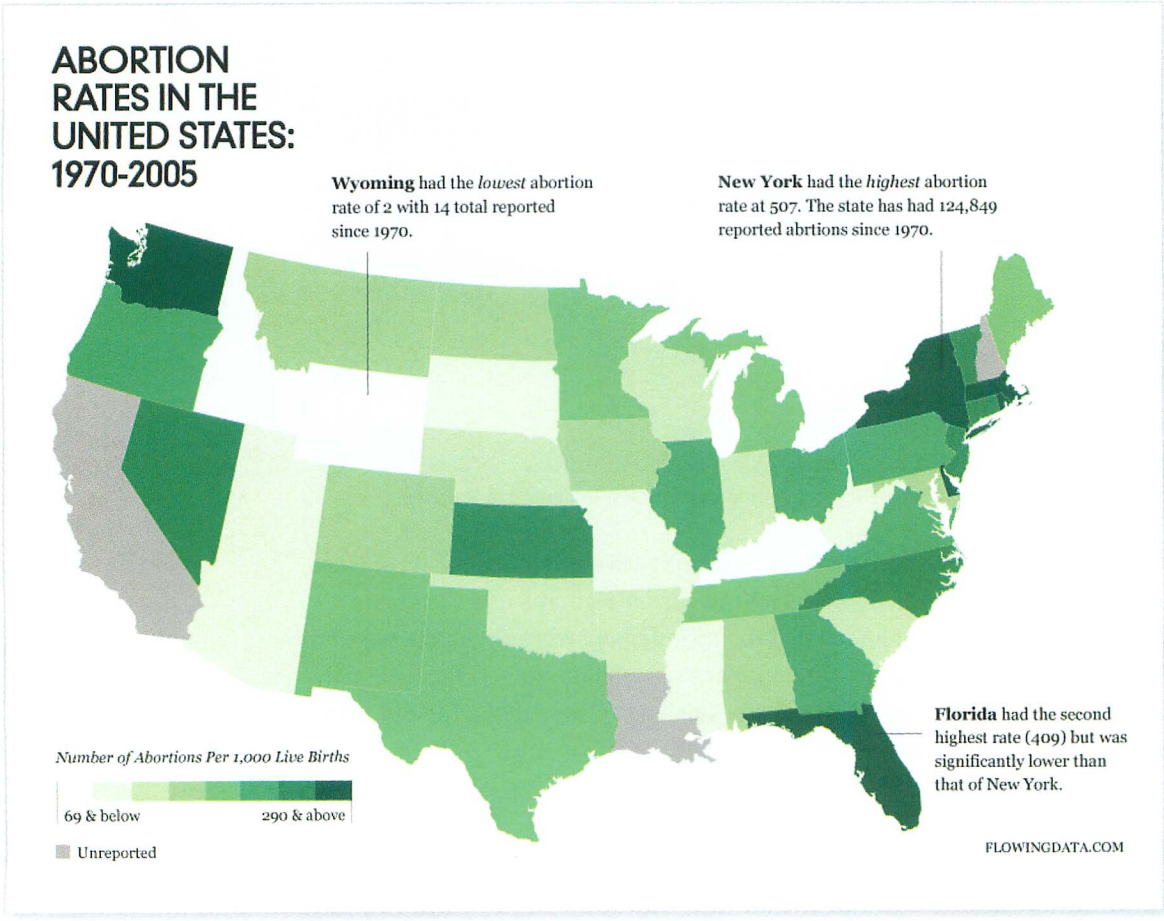


FIGURE 3-27 United States map created in R

USEFUL R MAPPING RESOURCES

- Analysis of Spatial Data (`http://cran.r-project.org/web/views/Spatial.html`)—Comprehensive list of packages in R for spatial analysis
- A Practical Guide to Geostatistical Mapping (`http://spatial-analyst.net/book/download`)—Free book download on how to use R and other tools for spatial data

ONLINE-BASED SOLUTIONS

A few online mapping solutions make it easy to visualize your geographic data. For the most part, they've taken the map types that people use the most and then stripped away the other stuff—kind of like a simplified ArcGIS. Many Eyes and GeoCommons are two free ones. The former, discussed previously, has only basic functionality for data by country or by state in the United States. GeoCommons, however, has more features and richer interaction. It also handles common geospatial file formats such as shapefiles and KML.

A number of paid solutions exist, but Indiemapper and SpatialKey are the most helpful. SpatialKey is geared more toward business and decision making whereas Indiemapper is geared toward cartographers and designers. Figure 3-28 shows an example I whipped up in just a few minutes in Indiemapper.

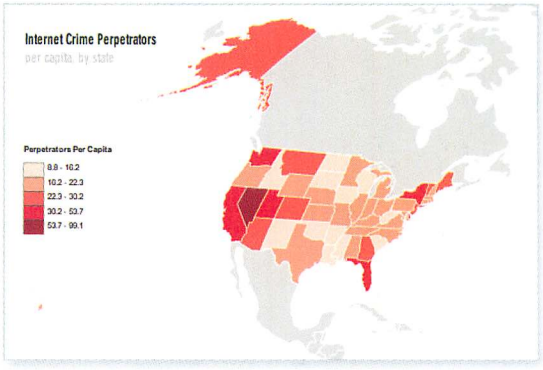


FIGURE 3-28 Choropleth map created in Indiemapper

Trade-Offs

Mapping software comes in all shapes and sizes suited to fit lots of different needs. It'd be great if you could learn one program and be able design every kind of map imaginable. Unfortunately, it doesn't work that way.

For example, ArcGIS has a lot of functions, but it might not be worth the time to learn or the money to purchase if you only want to create simple maps. On the other hand, R, which has basic mapping functionality and is free, could be too simple for what you want. If online and interactive maps

are your goal, you can go open-source with Modest Maps or Polymaps, but that requires more programming skills. You'll learn more about how to use what's available in Chapter 8.

Survey Your Options

This isn't a comprehensive list of what you can use to visualize data, but it should be enough to get you started. There's a lot to consider and play with here. The tools you end up using largely depend on what you want to accomplish, and there are always multiple ways to accomplish a single task, even within the same software. Want to design static data graphics? Maybe try R or Illustrator. Do you want to build an interactive tool for a web application? Try JavaScript or Flash.

On FlowingData, I ran a poll that asked people what they mainly used to analyze and visualize data. A little more than 1,000 people responded. The results are shown in Figure 3-29.

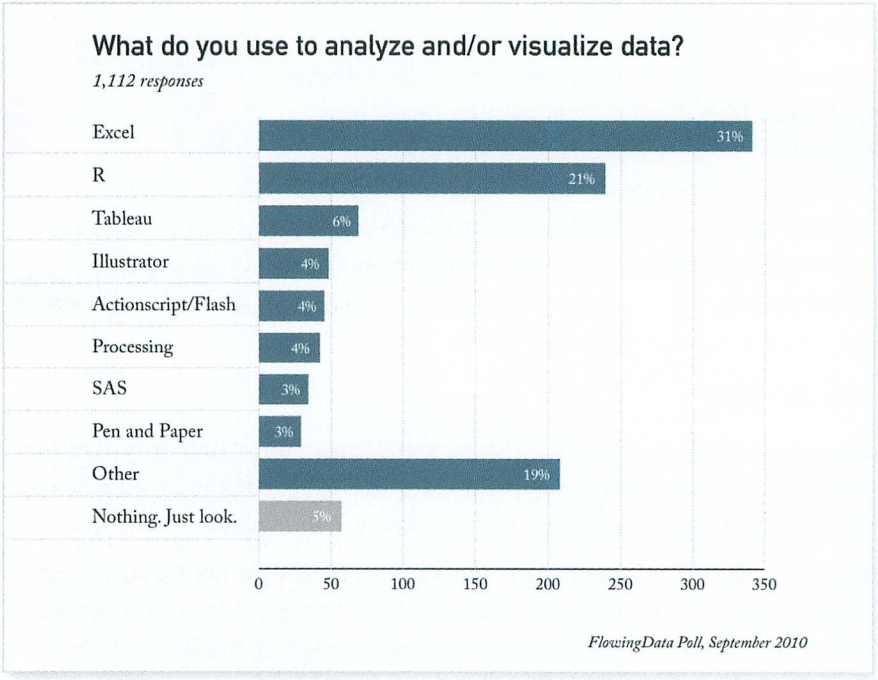


FIGURE 3-29 What FlowingData readers use to analyze and visualize data

There are some obvious leaders, given the topic of FlowingData. Excel was first, and R followed in second. But after that, there was a variety of software picks. More than 200 people chose the Other category. In the comments, many people stated that they use a combination of tools to fill different needs, which is usually the best route for the long term.

Combining Them

A lot of people like to stick to one program—it's comfortable and easy. They don't have to learn anything new. If that works, then by all means they should keep at it. But there comes a point after you've worked with data long enough when you hit the software's threshold. You know what you want to do with your data or how to visualize it, but the software doesn't let you do it or makes the process harder than it has to be.

You can either accept that, or you can use different software, which could take time to learn but helps you design what you envision—I say go with the latter. Learning a variety of tools ensures that you won't get stuck on a dataset, and you can be versatile enough to accomplish a variety of visualization tasks to get actual results.

Wrapping Up

Remember that none of these tools are a cure-all. In the end, the analyses and data design is still up to you. The tools are just that—they're tools. Just because you have a hammer doesn't mean you can build a house. Likewise, you can have great software and a super computer, but if you don't know how to use your tools, they might as well not exist. You decide what questions to ask, what data to use, and what facets to highlight, and this all becomes easier with practice.

But hey, you're in luck. That's what the rest of this book is for. The following chapters cover important data design concepts and teach you how to put the abstract into practice, using a combination of the tools that were just covered. You can learn what to look for in your data and how to visualize it.


90