Preface

Information is what we want, but data are what we've got. The techniques for transforming data into information go back hundreds of years. A good starting marker is 1592 with the publication of weekly "bills of mortality" in London. These bills were tabulations: a condensing of the data into a form more readily assimilated by the human reader. Constructing such tabulations was a manual operation.

As data became larger, machines were introduced to speed up the tabulations. A major step was Hollerith's development of punched cards and an electrical tabulating system for the US Census of 1890. This was so successful that Hollerith started a company, IBM, that came to play an important role in the development of today's electronic computers.

Also in the late 19th century, statistical methods began to develop rapidly. These methods have been tremendously important in interpreting data, but they were not intrinsically tied to mechanical data processing. Generations of students have learned to carry out statistical operations by hand on small sets of data, typically from a laboratory experiment.

Nowadays, it's common to have datasets that are so large they can be processed only by machine. In this era of "big data," often data are amassed by networks of instruments and computers. The settings for this are diverse: the genome, satellite observations of Earth, entries by web users, sales transactions, etc. With such data, there are new opportunities for finding and characterizing patterns using techniques such as data mining, machine learning, data visualization, and so on. As a result, everyday uses of data involve computer processing of data. This includes data cleaning, combining data from multiple sources, and transformation into a form suitable for input to data-condensing operations like visualization.

In writing this book I hope to help people gain the understanding and skills for data wrangling, a process of preparing data for visualization and other modern techniques of statistical interpretation. Doing so inevitably involves, at the center, using the computer in a sophisticated way.

Need sophisticated computing require extended study of computer programming? My view is that it does not. First, over the last half century, a coherent set of simple data operations have been developed that can be used as building blocks of sophisticated data wrangling processes. The trick is not mastering programming but learning to think in terms of these operations. Much of this book is intended to help you master such thinking. Second, it's possible to use recent developments in software to reduce vastly the amount of programming needed to use the data operations. I've drawn on such software — particularly R and the packages `dplyr` and `ggplot2` developed by Hadley Wickham and the many others who contribute to important open-source projects — to focus on a small subset of functions that accomplish data wrangling tasks in a concise and expressive way. The computer notation is clear enough that, once you have learned to read it, constructing new data wrangling programs can be done by modifying working examples. (Experienced R programmers will note the distinctive style of R statements in this book, including a consistent focus on a small set of key notation.)

I've tried to arrange this book to be accessible to a reader with no technical computing background. The book derives from notes for a course at Macalester College, *Data Computing Fundamentals*, that has no pre-requisites. To be effective, the book should be read along with practice using the computer. Some of this practice involves becoming familiar with the user-interface software. This includes the RStudio environment for computing with R, the use of a text editor, and RMarkdown, the notation that integrates the computer commands with graphical display and explanatory narrative. It's hard to write effective explanations of how to work with user-interfaces; doing it is the best way to learn, watching others do it is also helpful. If you're using this book in a classroom setting, your instructor can provide the demonstrations. If you are reading this book on your own, you can make use of the many videos that introduce using RStudio, its editor, and the RMarkdown document-writing system.

The contributions of many people are behind this book. At a start, this includes the work of the people in the R Core Team and the fantastic community of open-source R developers. Of particular importance to *Data Computing* are Hadley Wickham, Winston Chang, Joe Cheng, Garrett Grolemund, and JJ Allaire at RStudio. Professors Nicholas Horton, Randall Pruim, Elizabeth Shoop and Paul Overvoorde helped in defining the Data Computing Fundamentals course. Collaborators on Project MOSAIC helped encourage the *minimal-R* approach that let's you do a lot with a little bit of R. The many faculty participants in the Computing and Visualization Consortium of small liberal arts colleges provided important feedback to identify

gaps and shortcomings. The careful attention of Jeff Witmer, Craig Ching, Michael Schneider, and Dennis McGuire pointed out many deficiencies in the manuscript. Students in the Data Computing Fundamentals course at Macalester College willingly hiked the initial rough path of the course curriculum. Macalester students Barbara Borges Ribiero, Mengdie Wang, and Jingjing Yang were teaching assistants for the course and spent summer months developing case studies, visualizations, and interactive "apps" for displaying what data verbs do. Andrew Zieffler, Ethan Brown, and Erik Anderson from the University of Minnesota helped in sharpening assessment, developing case studies, and teaching an early version of the course. Prof. Jessen Havill at Denison University put together a Mellon Foundation workshop in 2010 to discuss how to bring computing into the science curriculum; the ideas of a short course and a focus on data emerged from that workshop.

Financial support for the Data Computing Fundamentals course and the Computation and Visualization Consortium was generously provided by the Howard Hughes Medical Institute. The National Science Foundation supported Project MOSAIC (NSF DUE-0920350). Both were instrumental to developing the Macalester course and this book.

My dear wife, Maya, and our beloved daughters Tamar, Liat, and Netta were encouraging (and patient) during many dinner-table discussions of case studies and the many hours I was away writing notes for the Data Computing course and teaching the Wednesday-evening class sessions.

Daniel Kaplan
St. Paul, Minnesota
August 2015

Contents

1
Tidy Data

Data can be as simple as a column of numbers in a spreadsheet file or as complex as the medical records collected by a hospital. A newcomer to working with data may expect each source of data to be organized in a unique way and to require unique techniques. The expert, however, has learned to operate with a small, standard set of tools. As you'll see, each of the standard tools performs a comparatively simple task. Combining those simple tasks in appropriate ways is the key to dealing with complex data.

One of the reason the individual tools can be simple is this: each tool gets applied to data arranged in a simple but precisely defined pattern called *tidy data*. At the heart of tidy data is the *data table*. To illustrate, Table 1.1 a handful of entries from a large US Social Security Administration tabulation of names given to babies. In particular, the table shows how many babies of each sex were given each name in each year.

TIDY DATA: A format for data that provides a systematic approach to computer processing. You will be using tidy data throughout this book and, likely, for the large majority of your professional work with data.

DATA TABLE: A rectangular array of data.

name	sex	count	year
Sherina	F	19	1970
Leketha	F	6	1973
Tahirih	F	6	1976
Radha	F	13	1979
Hatim	M	8	1981
Cissy	F	7	1984
Ahmet	M	6	1986
... and so on for 1,792,091 rows			

Table 1.1: A data table showing how many babies were given each name in each year in the US.

Table 1.1 shows there were 6 boys named Ahmet born in the US in 1986 and 19 girls named Sherina born in 1970. As a whole, the `BabyNames` data table covers the years 1880 through 2013 and includes a total of 333,417,770 individuals, somewhat larger than the current population of the US.

The data in Table 1.1 are "tidy" because they are organized according to two simple rules.

1. The rows, called *case*s, each refer to a specific, unique and similar sort of thing, e.g. girls named Sherina in 1970.
2. The columns, called *variable*s, each have the same sort of value recorded for each row. For instance, `count` gives the number of babies for each case; `sex` tells which gender the case refers to.

CASE: The individual people, objects, events, etc. from which variables are measured.

VARIABLE: The attributes of each case that are measured or observed.

When data are in tidy form, it's relatively straightforward to transform the data into arrangements that are more useful for answering interesting questions. For instance, you might wish to know which

were the most popular baby names over all the years. Even though Table 1.1 contains the popularity information implicitly, some re-arrangement is needed (for instance, adding up the counts for a name across all the years) before the information is made explicit, as in Table 1.2.

The process of transforming information that is implicit in a data table into another data table that gives the information explicitly is called *data wrangling*. The wrangling itself is accomplished by using *data verbs* that take a tidy data table and transform it into another tidy data table in a different form. In the following chapters, you will be introduced to the various *data verbs*.

sex	name	total_births
M	James	5091189
M	John	5073958
M	Robert	4789776
M	Michael	4293460
F	Mary	4112464
... and so on for 102,690 rows		

Table 1.2: The most popular baby names across all years.

	A	B	E	F	G	H	I	J
1			City of Minneapolis Statistics					
2			General Election November 5, 2013					
3	Ward	Precinct	Voters Registering by Absentee	Total Registrations	Voters at Polls	Absentee Voters	Total Ballots Cast	Total Turnout
4	City-Wide Total		708	6,634	75,145	4,954	80,099	33.38%
5								
6	1	1	3	28	492	27	519	27.23%
7	1	2	1	44	836	56	892	31.71%
8	1	3	0	40	905	19	924	38.87%
9	1	4	5	29	768	26	794	36.62%
10	1	5	0	31	683	31	714	37.46%
11	1	6	0	69	739	20	759	32.62%
12	1	7	0	47	291	8	299	15.79%
13	1	8	0	43	415	5	420	30.55%
14	1	9	0	42	596	25	621	25.42%
15	Ward 1 Subtotal		9	373	5,725	217	5,942	30.93%
16								
17	2	1	1	63	1,011	39	1,050	36.42%
18	2	2	5	44	679	37	716	50.39%
19	2	3	4	48	324	18	342	18.88%
20	2	4	0	53	117	3	120	7.34%
21	2	5	2	50	495	26	521	25.49%
22	2	6	1	36	433	19	452	39.10%
23	2	7	0	39	138	7	145	13.78%
24	2	8	1	50	1,206	36	1,242	47.90%
25	2	9	2	39	351	16	367	30.56%
26	2	10	0	87	196	5	201	6.91%
27	Ward 2 Subtotal		16	509	4,950	206	5,156	27.56%
28								
29	3	1	0	52	165	1	166	7.04%

Table 1.3: Ward and precinct votes cast in the 2013 Minneapolis mayoral election.

Table 1.3, in contrast to `BabyNames`, is not in tidy form. True, table 1.3 is attractive and neatly laid out. There are helpful labels and summaries that make it easy for a person to read and draw conclusions. (For instance, Ward 1 had a higher voter turnout than Ward 2, and both wards were lower than the city total.)

Being neat is not what makes data tidy. Table 1.3, however neat it is, violates the rules for tidy data.

- Rule 1: The rows, called *cases*, each must represent the same underlying attribute, that is, the same kind of thing.

  That's not true in Table 1.3. For most of the table, the rows represent a single precinct. But other rows give ward or city-wide totals. The first two rows are captions describing the data, not cases.

- Rule 2: Each column is a variable containing the same type of

value for each case.

That's mostly true in Table 1.3, but the tidy pattern is interrupted by labels that are not variables. For instance, the first two cells in row 15 are the label "Ward 1 Subtotal," different from the ward/precinct identifiers that are the values in most of the first column.

Conforming to the rules for tidy data simplifies summarizing and analyzing data. For instance, in the tidy baby names table, it's easy to find the total number of babies: just add up all the numbers in the `count` variable. It's similarly easy to find the number of cases: just count the rows. And if you want to know the total number of Ahmeds or Sherinas across the years, there's an easy way to do that.

In contrast, it would be more difficult in the Minneapolis election data to find, say, the total number of ballots cast. If you take the seemingly obvious approach and add up the numbers in column I of Table 1.3 (labelled "Total ballots cast"), the result will be *three times* the true number of ballots, because some of the rows contain summaries, not cases.

Indeed, if you wanted to do calculations based on the Minneapolis election data, you would be far better off to put it in a tidy form, like Table 1.4.

ward	precinct	registered	voters	absentee	total.turnout
1	1	28	492	27	0.27
1	4	29	768	26	0.37
1	7	47	291	8	0.16
2	1	63	1011	39	0.36
2	4	53	117	3	0.07
		... and so on for 117 rows			

Table 1.4: The Minneapolis election data in tidy form.

The tidy form is, admittedly, not as attractive as the form published by the Minneapolis government. But the tidy form is much easier to use for the purpose of generating summaries and analyses.

Once data are in a tidy form, you can present them in ways that can be more effective than a formatted spreadsheet. Figure 1.1 presents the turnout in each ward that makes it easy to see how much variation there is within and among precincts.

The tidy format also makes it easier to bring together data from different sources. For instance, to explain the variation in voter turnout, you might want to look at variables such as party affiliation, age, income, etc. Such data might be available on a ward-by-ward basis from other records, such as the (public) voter registration logs and census records. Tidy data can be *wrangle*d into forms that can be

WRANGLE: Data wrangling is the process of reforming, summarizing, and combining data to make it more suitable for a given purpose.

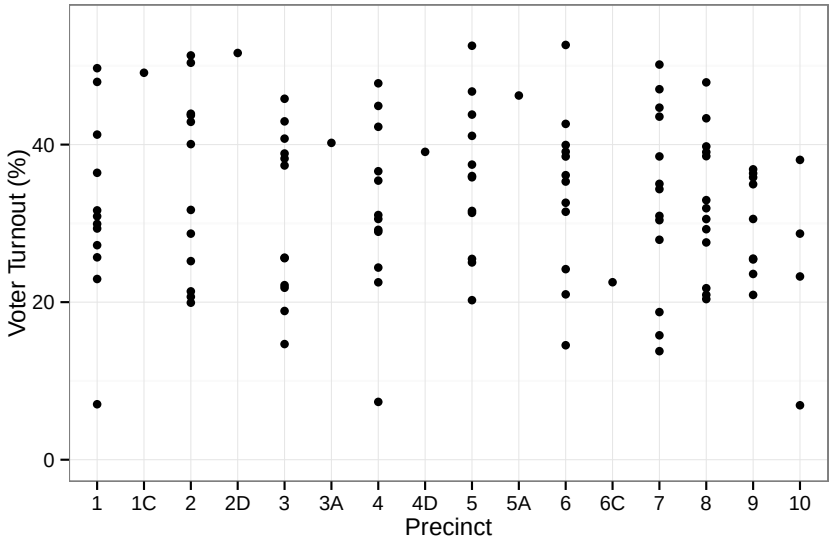


Figure 1.1: A graphical depiction of voter turnout in the different wards.

connected to one another. This would be difficult if you had to deal with an idiosyncratic format for each different source of data.

1.1 *Variables*

In data science, the word *variable* has a different meaning than in mathematics. In algebra, a variable is an unknown quantity. In data, a variable is known; it has been measured. "Variable" refers to a specific quantity or quality that can vary from case to case.

VARIABLE: A quantity or quality that varies from one case to another.

There are two major types of variables: *categorical* and *quantitative* A quantitative variable is just what it sounds like: a number.

A categorical variable tells which category or group a case falls into. For instance, in the baby names data table, `sex` is a categorical variable with two *levels* F and M, standing for female and male. Similarly the `name` variable is categorical. It happens that there are 92,600 different levels for `name`, ranging from Aaron, Ab, and Abbie to Zyhaire, Zylis, and Zymya.

CATEGORICAL: One of the two major kinds of variables. Categorical variables record type or category and often take the form of a word.

QUANTITATIVE: The other of the two major types of variables. A quantitative variable records a numerical attribute.

LEVELS: The individual possibilities for a categorical variable.

1.2 *Cases and what they represent*

As already stated, a row of a tidy data table refers to a case. To this point, you may have little reason to prefer the word *case* to *row*.

When working with a data table, it's important to keep in mind what a case stands for in the real world. Sometimes the meaning is obvious. For instance, Table 1.5 is a tidy data table showing the ballots in the Minneapolis mayoral election in 2013. Each case is an individual voter's ballot. (The voters were directed to mark their

ballot with their first choice, second choice and third choice among the candidates. This is part of a procedure called rank choice voting: `http://vote.minneapolismn.gov/rcv/`.)

Precinct	First	Second	Third	Ward
P-04	undervote	undervote	undervote	W-6
P-06	BOB FINE	MARK ANDREW	undervote	W-10
P-02D	NEAL BAXTER	BETSY HODGES	DON SAMUELS	W-7
P-01	DON SAMUELS	undervote	undervote	W-5
P-03	CAM WINTON	DON SAMUELS	OLE SAVIOR	W-1
... and so on for 80,101 rows				

Table 1.5: Individual ballots in the Minneapolis election. Each voter votes in one ward in one precinct. The ballot marks the voter's first three choices for mayor.

The case in Table 1.5 is a different sort of thing than the case in Table 1.4. In Table 1.4, a case is a ward in a precinct. But in Table 1.5, the case is an individual ballot. Similarly, in the baby names data (Table 1.1), a case is a name and sex and year while in Table 1.2 the case is a name and sex.

When thinking about cases, ask this question: What description would make every case unique? In the vote summary data, a precinct does not uniquely identify a case. Each individual precinct appears in several rows. But each precinct & ward combination appears once and only once. Similarly, in Table 1.1, name & sex do not specify a unique case. Rather, you need the combination of name & sex & year to identify a unique row.

Example: Runners and Races Table 1.6 shows some of the results from a 10-mile running race held each year in Washington, D.C.

name.yob	sex	age	year	gun
jane polanek 1974	F	32	2006	114.50
jane poole 1948	F	55	2003	92.72
jane poole 1948	F	56	2004	87.28
jane poole 1948	F	57	2005	85.05
jane poole 1948	F	58	2006	80.75
jane poole 1948	F	59	2007	78.53
jane schultz 1964	F	35	1999	91.37
jane schultz 1964	F	37	2001	79.13
jane schultz 1964	F	38	2002	76.83
jane schultz 1964	F	39	2003	82.70
jane schultz 1964	F	40	2004	87.92
jane schultz 1964	F	41	2005	91.47
jane schultz 1964	F	42	2006	88.43
jane smith 1952	F	47	1999	90.60
jane smith 1952	F	49	2001	97.87
... and so on for 41,248 rows				

Table 1.6: An excerpt of runners' performance over the years in a 10-mile race.

What's the meaning of a case here? It's tempting to think that a case is a person. After all, it's people who run road races. But notice that individuals appear more than once: Jane Poole ran each year from 2003 to 2007. (Her times improved consistently as she got older!) Jane Smith ran in the races from 1999 to 2006, missing only the year 2000 race. This suggests that the case is a runner in one year's race.

CODEBOOK: A document separate from the data table detailing the meaning of each variable in the table, how levels of categorical variables are encoded, units for quantitative variables, etc.

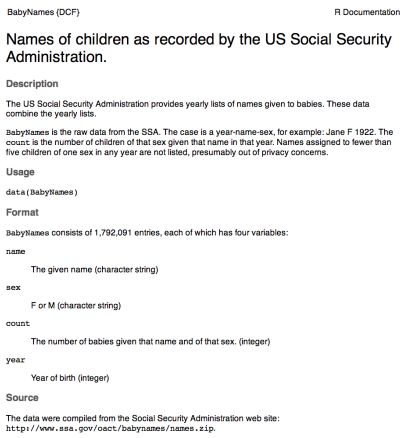
1.3 *The Codebook*

Data tables do not necessarily display all the variables needed to figure out what makes each row unique. For such information, you sometimes need to look at the documentation of how the data were collected and what the variables mean.

The *codebook* is a document — separate from the data table — that describes various aspects of how the data were collected, what the variables mean and what the different levels of categorical variables refer to. Figure 1.2 shows the codebook for the `BabyNames` data in Figure 1.1.

The word "codebook" comes from the days when data was encoded for the computer in ways that make it hard for a human to read. A codebook should also include information about how the data were collected and what constitutes a case.

For the runners data in Table 1.6, the codebook tells you that the meaning of the `gun` variable is the time from when the start gun went off to when the runner crosses the finish line and that the unit of measurement is "minutes." It should also state what might be obvious: that `age` is the person's age in years and `sex` has two levels, male and female, represented by M and F.



BabyNames (DCF) R Documentation

Names of children as recorded by the US Social Security Administration.

Description

The US Social Security Administration provides yearly lists of names given to babies. These data combine the yearly lists.

`BabyNames` is the raw data from the SSA. The case is a year-name-sex, for example: Jane F 1922. The `count` is the number of children of that sex given that name in that year. Names assigned to fewer than five children of one sex in any year are not listed, presumably out of privacy concerns.

Usage

```
data(BabyNames)
```

Format

`BabyNames` consists of 1,792,091 entries, each of which has four variables:

`name`
: The given name (character string)

`sex`
: F or M (character string)

`count`
: The number of babies given that name and of that sex. (integer)

`year`
: Year of birth (integer)

Source

The data were compiled from the Social Security Administration web site: `http://www.ssa.gov/oact/babynames/names.zip`.

Figure 1.2: The codebook for the `BabyNames` data table.

1.4 *Multiple Tables*

It's often the case that creating a meaningful display of data involves combining data from different sources and about different kinds of things. For instance, you might want your analysis of the runners' performance data in Table 1.6 to include temperature and precipitation data for each year's race. Such weather data is likely contained in a table of daily weather measurements.

In many circumstances, there will be multiple tidy tables, each of which contains information relative to your analysis but has a different kind of thing as a case. Chapter 10 is about techniques combining data from such different tables. For now, keep in mind that being tidy is not about shoving everything into one table.

Exercises

Problem 1.1
Here is an excerpt from the baby-name data table in `"DataComputing::BabyNames"`.

name	sex	count	year
Taffy	F	19	1970
Liliana	F	162	1973
Stan	M	55	1975
Nettie	F	45	1978
Kateria	F	8	1980
... and so on for 1,792,091 rows			

Consider these five entities, that appear in the table shown above (a) through (e):

a) `Taffy` b) `year` c) `sex` d) `name` e) `count`

For each, choose one of the following:

1. It's a categorical variable.
2. It's a quantitative variable.
3. It's the value of a variable for a particular case.

Problem 1.2
What's not tidy about this table?

`president`	`in office`	`number of states`
Lincoln, Abraham	1861-1865	it depends
George Washington	1791-1799	16
Martin Van Buren	1837 to 1841	26

Table 1.7: An untidy table

Problem 1.3
Re-write Table 1.7 in a tidy form. Take care to render the information about years and about the number of states as numbers.

Problem 1.4
Here are three different organizations (**A**, **B**, and **C**) of the same data:

Data Table A

Year	Algeria	Brazil	Columbia
2000	7	12	16
2001	9	14	18

Data Table B

Country	Y2000	Y2001
Algeria	7	9
Brazil	12	14
Columbia	16	18

Data Table C

Country	Year	Value
Algeria	2000	7
Algeria	2001	9
Brazil	2000	12
Columbia	2001	18
Columbia	2000	16
Brazil	2001	14

1. What are the variables in each table?
2. What is the meaning of a case for each table? Here are some possible choices.
   - A country
   - A country in a year
   - A year

Problem 1.5
The codebook for several data tables relating to airports, airlines, and airline flights in the US is published at `https://cran.r-project.org/web/packages/nycflights13/nycflights13.pdf`.

Within that document is the codebook for the data table `airports`.

1. How many variables are there?
2. What do the cases represent?
3. For each variable, make a reasonable guess about whether the values will be numerical or quantitative.

2
Computing with R

Data tables are often large, so it is not possible to undertake paper-and-pencil operations on them. The `BabyNames` data table 1.1 provides a case in point. `BabyNames` has 1,790,091 rows, far too many to carry out by hand even a simple sum or count. Data science relies on computers.

The human role in the process is managerial, to decide what forms of analysis are called for by the purpose at hand and to instruct the computer to carry out the operations needed. The process of instructing a computer is called *computer programming*. Programming is done using a language that is accessible both to humans and the computer; the human writes the instructions and the computer carries them out. These notes use a computer language called R.

COMPUTER PROGRAMMING: The process of writing instructions for a computer. Sometimes the unfortunate word "coding" is used. The point of programming is to make instructions clear, not to obscure them with code.

This chapter introduces concepts that underlie the use of R. The emphasis here will be on the *kinds of things* that R commands involve. Only a few R commands will be covered, but these will illustrate the most important patterns in writing with R. Chapter 3 will start to introduce the commands themselves that are used for working with data.

2.1 R and RStudio

The *R* language includes many useful operations for working with data, drawing graphics, and writing documents, among other things.

R: A language for working with data.

RStudio is a computer application that gives ready access to the resources of R. RStudio provides a powerful way for a person to interact with R. There are other ways to work with R, but for good reason RStudio has become the most popular. RStudio offers facilities both for newbies and for professionals; it's good for everybody.

RSTUDIO: An interface for organizing your work and interacting with R.

RStudio comes in two versions:

1. A *desktop* that runs on the user's own computer.
2. A *server* that runs on a computer in the cloud and is accessed by a

DESKTOP: An application that runs on your own computer.

SERVER: An application that is being run remotely, as a service to many users.

web browser.

You can install R and the desktop version of RStudio on all the most common operating systems: OS-X, Windows, Linux. There are now many text and video guides to installation. See `marin-stats-lectures` for an example.

If you are to use the server version of RStudio, someone has set up a server for you and provided a login ID. You access the server through an ordinary web browser — Chrome, Firefox, Safari, etc. — that may be running on a laptop, a tablet, or even a smartphone.

Setting up an RStudio server? This requires some information technology (IT) knowledge and knowledge of the UNIX operating system. If you're comfortable with such things, there are many guides on the web.

The two RStudio versions, desktop or server, are almost identical from a user's point of view. The desktop version is often faster but can only be used on one computer; the server version requires almost no set-up and is available from *any* computer browser, but it can be slower.

The server version slows down when there are many simultaneous users, as might happen in a classroom.

2.2 *Components of the RStudio application*

PANES: Regions of the RStudio application that are displayed side-by-side.

TABS: Subdivisions of a pane, only one of which can be viewed at one time.

Just as a window in a house has several individual panes of glass, so the RStudio application appears in an graphical interface window divided up into several *panes*. All panes can be viewed at the same time. Each of the panes can contain several *tabs*.

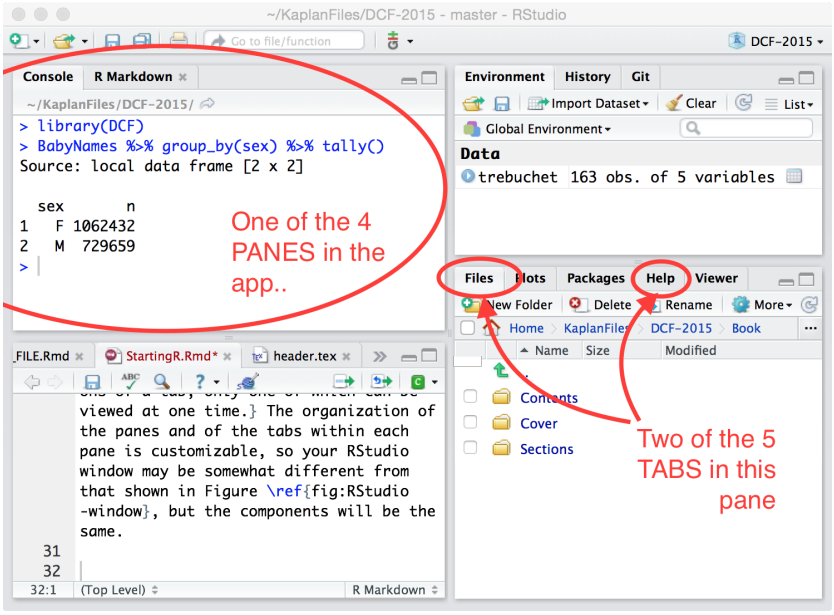


Figure 2.1: The RStudio window is divided into four panes. Each panes can contain multiple tabs.

The organization of the panes and of the tabs within each pane is customizable, so your RStudio window may be somewhat different from that shown in Figure 2.1, but the components will be the same.

You will often be using the Console tab and tabs for editing documents. Each document-editing tab has the name of the document it contains. In Figure 2.1, three such editing tabs are shown, one of which is labeled `StartingR`. Other important tabs: Packages for installing new R software and Help for displaying documentation.

2.3 *Commands and the Console*

This is a good time to start up your version of RStudio. Once RStudio is running:

- Enter R-language commands into the *console*. Find the console tab in your RStudio app and the prompt, >.
- Move the cursor to the console tab so that anything you type shows up there. Type the simple command shown in Figure 2.2.
- After you press *enter*, ↩, R will *execute* your command and print out a response.

Notice that after the response, there is another prompt. You're ready to go again.

PRACTICE. Enter each of the following arithmetic commands at the command prompt, one at a time. Confirm that the response is the right arithmetic answer.

```
16 * 9
sqrt(2)
20 / 5
18.5 - 7.21
```

Figure 2.2: The console tab showing the command prompt > (top), a command composed but not yet entered (middle), and after entering the command. (bottom)

2.4 *Sessions*

Your work in R occurs in *sessions*. A session is a kind of ongoing dialog with the R system.

A new session is begun every time you start RStudio. The session is terminated only when you close or *quit* the RStudio program. If you are using RStudio server (via your browser), the R session will be maintained for days or weeks or months, and will be retained even when you login to the server from a different computer.

When a session is first started, it is in an empty environment. RStudio displays this in the *Environment* tab. (Figure 2.3)

3
R Command Patterns

Almost everyone who writes computer commands starts by copying and modifying existing commands. To do this, you need to be able to **read** command expressions. Once you can read, you will know enough to identify the patterns you need for any given task and consider what needs to be modified to suit your particular purpose.

As you read this chapter, you will likely get an inkling of what the commands used in examples are intended to do, but that's not what's important now. Instead, focus on

- Distinguishing between *functions* and arguments.
- Distinguishing between *data tables* and the *variables* that are contained in them.
- Recognizing when a function is being used.
- Identifying the arguments to a function.
- Observing how assignment allows values to be stored and referred to by name.
- Discerning when the output of one function becomes the input to another.

You are **not** expected at this point to be able to *write* R expressions. You'll have plenty of opportunity to do that once you've learned to *read* and recognize the several different patterns used in R data wrangling and visualization commands.

3.1 Language Patterns and Syntax

In a human language like English or Chinese, *syntax* is the arrangement of words and phrases to create well-formed sentences. For example, "Four horses pulled the king's carriage," combines noun phrases ("Four horses", "the king's carriage") with a verb.

Consider this pair of English language sentence patterns, a statement and a question:

1. I did go to school .
2. Did I go to school ?

The content of each of the boxes can be replaced by an equivalent object.

- I can be replaced with "you", "we", "he", "Janice", "the President", and so on.
- go to can be replaced with "stay in", "attend", "drive to", "see", ...
- school can be replaced with "the lake", "the movie", "Fred's parents' house", ...

Such replacements produce sentences like these:

- Did we stay in Fred's parents' house?
- Janice did drive to the lake.
- Did the President see the movie?
- Did he attend school?

Each sentence expresses something different, but they all follow the same patterns.

R HAS A FEW PATTERNS THAT SUFFICE for many data wrangling and visualization tasks. Consider these patterns:

- `object_name` <- `function_name` (`arguments`)
  This is called *function application*. The output of the function will be stored under the name to the left of <-.
- `object_name` <-
  `Data_Table` %>%
  `function_name` (`arguments`)
  This is called *chaining syntax*.
- `object_name` <-
  `Data_Table` %>%
  `function_name` (`arguments`) %>%
  `function_name` (`arguments`)
  This is an extended form of chaining syntax. Such chains can be extended indefinitely.

4
Files and Documents

In your work with data, you will be using and creating computer files of various sorts. Of particular importance are three basic roles for files:

1. storing data tables
2. listing R instructions
3. writing reports with narrative, graphics and conclusions

Different file types are used for each of these roles. These types can be referred to by the filename extension of the files.

Table 4.1: Common file types and their uses

File Role	Common file types	Software
Data storage	`.csv`, `.xlsx`, etc.	Spreadsheets
R instructions	`.Rmd`	Text editor, compiler
End reports	`.html`, `.pdf`, `.docx`, etc.	web browser, word processor

Filename Extensions

The *filename extension* is one or more letters following the last period in the name: `.xlsx`, `.docx`, `.html`, `.mpeg`. (Other widely used extensions are `.pdf`, `.zip`, `.png`.) These letters indicate the format of the file and often set which program will be used to open the file: a spreadsheet, a word processor, a browser, or a music player in the examples. Or, in plainer language, the filename extension tells you the *kind* of file.

FILENAME EXTENSION: The letters following the last period in a file name. This suffix indicates the file type, that is, what software to use to interpret the file.

If you are looking at files through RStudio, the filename extension will always be displayed. If you are using your your own computer's file browser, your system may have been set up to hide the extension.

When referring to files within R statements or an Rmd file, you must **always** include the filename extension.

Paths

A filename is analogous to a person's first name. Just as first names are unique within a nuclear family, so filenames must be unique within a folder or directory.

You are probably used to seeing folders and the files they contain organized on your computer as in Figure 4.1. The *file path* is the set of successive folders that bring you to the file.

There is a standard format for file paths. An example:

```
/Users/kaplan/Downloads/0021_001.pdf
```

Here the filename is `0021_001`, the filename extension is `.pdf`, and the file itself is in the `Downloads` folder contained in the `kaplan` folder, which is in turn contained in the `Users` folder. The starting `/` means "on this computer".

The R `file.choose()` — which should be used **only in the console**, not in an Rmd file — brings up an interactive file browser. You can select a file with the browser. The returned value will be a quoted character string with the path name.

```
file.choose() # then select a file
```

```
[1] "/Users/kaplan/Downloads/0021_001.pdf"
```

In RStudio, the *Files* tab will display the path near the top. In Figure 4.2, the ten files lised are all in the same folder, whose path ends with `\ DCF-2014/ReOrganizedJune10/CourseNotes/DataOrganization`.

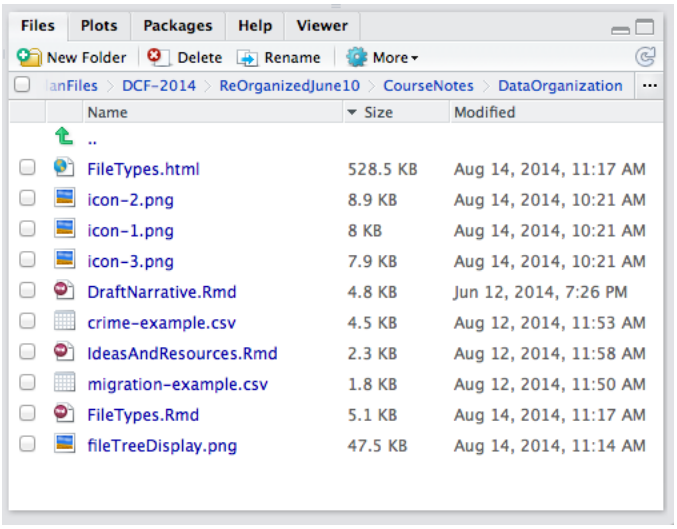


Figure 4.2: Filenames and their file path shown in the Files tab in RStudio. Only part of the path is shown: the folders closest to the files.

To run on with the family metaphor ... You are identified within a household by your first name. The path would tell you which specific nuclear family you belong to, perhaps in the form of your address, like this: *USA/Saint Paul/55105/703 Lincoln Avenue*.

FILE PATH: Information that specifies the location of a file in your file system.

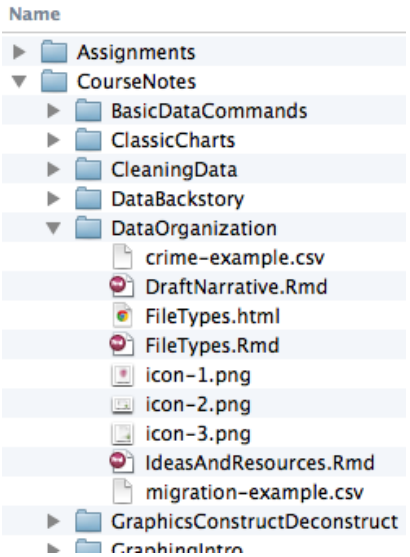


Figure 4.1: Folders contained within folders, as shown by a file browser on Apple OS-X.

QUOTES OR NOT FOR A FILE PATH? When you are referring to a file path in R, it will always be in quotation marks; it's a character

string. Other software such as web browsers don't use quotes when specifying a path.

URLs

You have probably noticed URLs in the locator window near the top of your browser. In Figure 4.3, the URL is:
`http://tiny.cc/dcf/index.html`.

A URL includes in its path name the location of the server on which the file is stored (e.g. `tiny.cc`) followed by the path to the file on that surver. Here, the path is `/dcf` and the file is `index.html`.

You will sometimes need to copy URLs into your work in R, to access a dataset, to make a link to some reference, etc. Remember to copy the entire URL, including the `http://` part if it is there.

Some common filename extensions for the sort of web resources you will be using:

- `.png` for pictures
- `.jpg` or `.jpeg` for photographs
- `.csv` or `.Rdata` for data files
- `.Rmd` for the human editable text of a document
- `.html` for web pages themselves

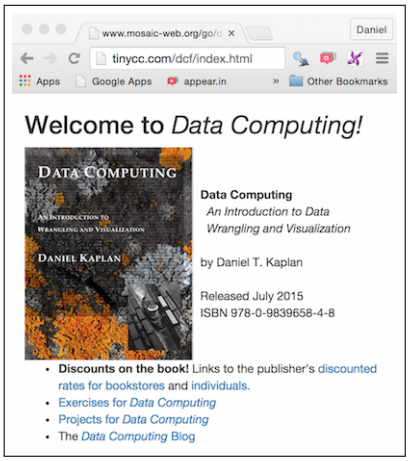


Figure 4.3: A browser directed to the URL `http://tiny.cc/dcf/index.html` .

Data Files

Data tables are often stored individually as files. There are many formats for data files. Among the most common is the `.csv` spreadsheet format, popular because reading it is a standard feature of many data analysis packages (including R).

If you use R extensively, you will also encounter data in `.Rda` (or `.Rdata`), an efficient format for storing data and other information specifically for R. When you get data from an R package, like this:

```
data(CPS85, package="mosaicData")
```

you are in fact reading in an `.Rda` file associated with the package.

There are many other formats for files containing data tables or the information needed to put the contents in data-table format. These are discussed in Chapter 15. Increasingly, data are accessed through *database* systems. In such a case, rather than reading the database as a whole, you make "queries" to access specific data you need.

DATABASE: A computer system used to store, update, and access data. Relational data bases consist of data tables.

Files containing data tables are often distributed via the web. These files are not any different than files on your own computer.

You access such files through a Uniform Resource Locator, better known as a *URL*. For instance, on the *Data Computing* web site, there are a number of `.csv` files. You can access them from R with commands like this:

URL: A name for a specific resource, such as a file, on the Internet.

```
Engines <- read.file("http://tiny.cc/mosaic/engines.csv")
Engines
```

Engine	mass	ncylinder	strokes	displacement	bore	stroke	BHP	RPM
Webra Speedy	0.14	1	2	1.80	13.50	12.50	0.45	22000
Motori Cipolla	0.15	1	2	2.50	15.00	14.00	1.00	26000
Webra Speed 20	0.25	1	2	3.40	16.50	16.00	0.78	22000
			... and so on for 39 rows					

So far as accessing data is concerned, there's nothing fundamentally different in reading a file from a URL than reading a file on your own computer.

Documenting your work with Rmd files

The purpose of data wrangling and visualization is communication: condensing and presenting data in a form that conveys information. An important part of communication is documentation and reporting.

Writing is not a linear process. Ideas are presented, revised or abandoned, corrected, re-focussed, and re-arranged. Data-oriented technical reports tie together narrative, graphics and summary tables and are based on potentially complex computer commands. There is an interplay between the computer commands and the narrative. Results from the computer may drive reconsideration of the narrative. Gaps in the narrative may point to shortcomings or omissions in the computer commands. And, always, there is the possibility of errors in writing commands and the need to document commands so that they can be checked and corrected. As well, data are commonly updated, corrected or extended.

The familiar practice of cutting and pasting from the computer console into a word processor does not address these features of technical reports. Cutting and pasting makes it hard to revise or update a report; you've got to cut out the old and paste in the new, figuring out for yourself which is which. This introduces the likelihood of error. And, there's nothing to document the linkages between the computer commands and the word-processed document.

An important concept in data-driven reporting is "reproducibility." The idea is to be able to reproduce your entire document without any manual intervention, and, more important, to be easily able to

generate a new report in response to changes in data or revisions in computer commands. In other words, reproducible reports contain all the information needed to generate a new report. Common document formats such as `.pdf`, `.docx`, or `.html` do not offer support for reproducibility.

In R, reproducible reporting is provided by the `.Rmd` file format and related software. An `.Rmd` file integrates computer commands into the narrative so that, for instance, graphics are produced by the commands rather than being inserted from another source.

YOU CREATE AND MODIFY REPRODUCIBLE REPORTS within RStudio's text *editor*. They contain ordinary text and punctuation: no formatting, color, images, etc. Instead, you use the `.Rmd` file to describe, using ordinary characters, both what you want the eventual format to look like and what R commands you want the computer to carry out in generating the report.

EDITOR: A program for constructing pure-text documents, such as R instructions and Rmd files

Figure 4.4 shows a simple `.Rmd` document, something you might write. The figure also shows the `.html` file that is the result of compiling the `.Rmd`.

```
# My First Document

The text here will be **compiled** into a prettily formatted
document.  This process allows you to:

* Explain and record your work
* Construct and store the R instructions needed for your task.
* Produce formatting appropriate for and end report.

Most important, the compilation process will execute your R
instructions and include the results in the document.

## Example calculation

```{r}
library(DataComputing)
BabyNames %>%
  group_by(year) %>%
  summarise(total_births=sum(count)) %>%
  ggplot( aes(x=year, y=total_births)) +
  geom_line()
```
```

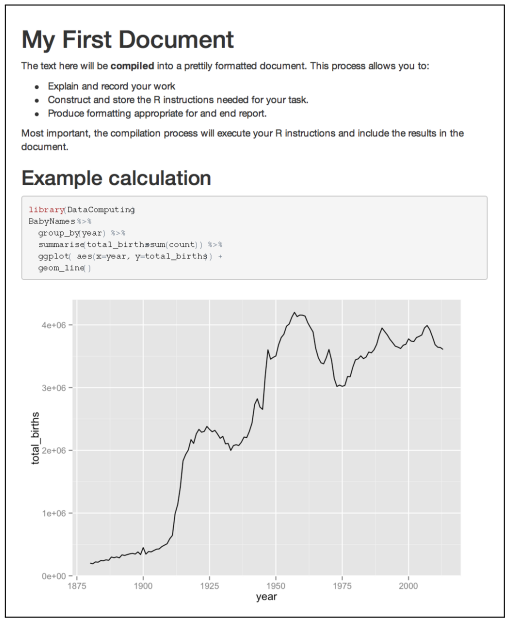


Figure 4.4: The `.Rmd` file on the left consists of text. It is automatically compiled to the formatted `.html` document, with the results of calculations, shown on the right.

COMPILE: The process of a computer translating a file from one format to another.

The Write/Compile Cycle

After you edit an `.Rmd` file, you *compile* the report into a reader-friendly document format such as `.pdf`, `.docx`, or `.html` that can easily be printed or displayed on the report-reader's computer. You

never edit the reader-friendly document; it's created automatically from your `.Rmd` instructions. When you want to update or modify or correct the end report, you edit the `.Rmd` file.

It's a good strategy to compile your `.Rmd` frequently. Start with a small, simple document. Then add a bit more to it: one paragraph or one "chunk" (see below). Compile again. If something goes wrong, you will have a good idea of where the problem lies. Go back and fix things. Make small changes, compile, see if it worked. Repeat. Figure 4.5 shows several steps in such an editing cycle.

It's impossible to avoid errors; even professionals make them. Instead, adopt a process that let's you identify errors quickly so that you can fix them before moving on. The shorter the write/compile cycle, the easier it will be to know when you have erred.

Interactive commands such as `file.choose()` or `scatterGraphHelper()` cannot be used in an an `.Rmd` file; there's no opportunity for interaction in the compilation process. Instead, use the interactive command in the console, get the result, and paste that result into your `.Rmd` file.

Command "Chunks"

The R commands in an `.Rmd` file go into *chunk*s, a range of lines in the documents that are delimited in a special way so that they will be executed as part of the `.Rmd` → `.html` compilation process.[1] The opening delimiter is ```` ```{r} ````. The closing delimiter is simply ```` ``` ````. For example:

CHUNK: A region in an `.Rmd` file that contains R statements to be executed when the `.Rmd` is compiled.

[1] Exactly the same applies when compiling to `.pdf` of `.docx`.

````
```{r}
Saltpeter <- read.file("http://tiny.cc/DCF/saltpeter.csv")
```
````

Both the opening and closing delimiters for a chunk are "back-quotes," a quote character that goes from upper-left to bottom-right. On many keyboards, back-quote is on the same key as tilde (~), like this:

MOST .RMD FILES WILL DRAW ON A LIBRARY that needs to be loaded into the R session. When you compile `.Rmd` → `.html`, R starts a brand new session that is, initially, empty and with no libraries loaded. When the compilation is complete, that session evaporates, leaving as its only residue the `.html` result.

Often, the first chunk in your document will be an instruction to load one or more libraries. Since this will be in just about every `.Rmd` document, it can be called the *boilerplate* chunk. It looks like this: A boilerplate chunk goes at the start of the document. It loads the libraries that the following commands will use.

BOILERPLATE: A standardized piece of text intended for re-use in many documents.

````
```{r include=FALSE}
library(DataComputing)
# and any others that you need, e.g.
library(mosaic)
```
````

The `include=FALSE` chunk argument helps to prettify the document: it is an instruction not to show the contents of the chunk in the output file.

5
Introduction to Data Graphics

Data graphics provide one of the most accessible, compelling, and expressive modes to investigate and depict patterns in data. This chapter presents examples of standard kinds of data graphics: what they are used for and how to read them. To start, you'll make simple examples of graphics using an interactive tool. Later, in Chapter 6, you'll see a unifying framework — a grammar — for describing and specifying graphics, so that you can create custom graphics types that support displaying data in a purposeful way.

5.1 Common Kinds of Graphs

There are many different genres of data graphics, and many different variations on each genre. Here are some commonly encountered kinds.

- **Scatterplots** showing relationships between two or more variables.
- **Displays of distribution**, such as histograms.
- **Bar charts**, comparing values of a single variable across groups.
- **Maps**, showing how a variable relates to geography.
- **Network diagrams**, showing how entities are connected to one another.

Scatter plots

The main purpose of a scatter plot is to show the relationship between two variables across several or many cases. Most often, there is a Cartesian coordinate system in which the x-axis represents one variable and the y-axis the value of a second variable.

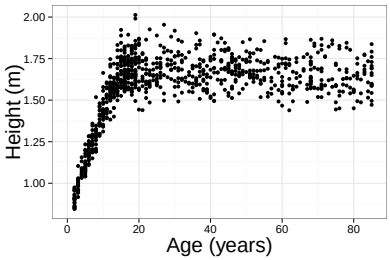


Figure 5.1: A scatter plot.

EXAMPLE: GROWING UP. The NCHS data table gives medical and morphometric measurements of individual people. The scatter plot in Figure 5.1 shows the relationship between two of the variables, height and age. Each dot is one case. The position of that dot signifies the value of the two variables for that case.

Scatterplots are useful for visualizing a simple relationship between two variables. For instance, you can see in the 5.1 the familiar pattern of growth in height from birth to the late teens.

Displays of Distribution

A histogram shows how many cases fall into given ranges of the variable. For instance, Figure 5.2 is a histogram of heights from NCHS. The most common height is about 1.65 m — that's the location of the tallest bar. Only a handful are taller than 2.0 m.

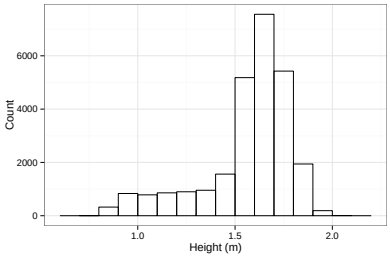


Figure 5.2: A histogram.

A simple alternative to a histogram is a *frequency polygon*. Frequency polygons let you break things up by other variables. Figure 5.3 shows the distribution of height for each sex, separately.

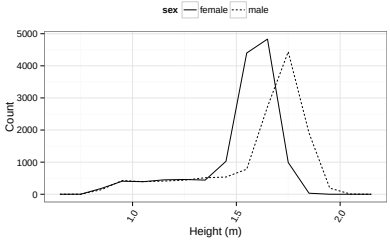


Figure 5.3: A frequency polygon

Bar Charts

The familiar bar chart is effective when the objective is to compare a few different quantities.

EXAMPLE: SMOKING AND DEATH. Based on the NCHS data, how likely is a person to have died during the follow-up period, based on their age and whether they smoke? It's easy to compare bars to their neighbors. From Figure 5.4, for instance, you can see that at each age, non-smokers were more likely to survive.

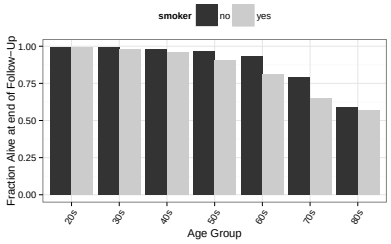


Figure 5.4: A bar chart

Maps

Using a map to display data geographically helps both to identify particular cases and to show spatial patterns and discrepancy. The map in Figure 5.5 shows oil production in each country. That is, the shading of each country represents the variable `oilProd` from `DataComputing::CountryData`. This sort of map, where the fill color of each region reflects the value of a variable, is sometimes called a *choropleth map*.

oil-map-1.pdf

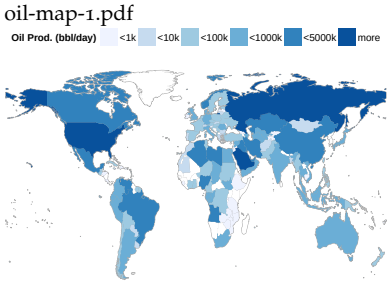


Figure 5.5: A choropleth map

Networks

A *network* is a set of connections, called *edges*, between elements, called *vertices*. A vertex corresponds to a case. The network describes which vertices are connected to other vertices.

6
Frames, Glyphs, and other Components of Graphics

Data graphics are built from parts. Chapter 5 showed the parts assembled together. This chapter looks at the parts individually.

Of course, a data table provides the basis for drawing a data graphic. The relationship between a data table and a graphic is simple: Each case in the data table becomes a mark in the graph. The designer of the graphic — you — chooses which variables the graphic will display and how each variable is to be represented graphically: position, size, color, and so on. The marks themselves are called *glyphs*. A data graphic has one glyph for each case in the data table.

Key Graphics Vocabulary

FRAME: The relationship between position and the data being plotted.

GLYPH: The basic graphical "unit" that represents one case. Other terms used include "mark" and "symbol." Variables set graphical attributes of the shape: size, color, shape, and so on. The location of the glyph — location is an important graphical attribute! — is set by the two variables defining the frame.

AESTHETIC: Any graphical attribute of a glyph: size, location, shape, color, etc.

SCALE: The relationship between the value of a variable and the graphical attribute to be displayed for that value.

GUIDE: An indication for the human viewer of the scale, that is, graphics how a variable encodes into its graphical attribute. Common guides are x- and y-axis tick marks and color keys.

6.1 *The Frame*

The frame of a graphic provides the space for drawing glyphs. But there is more to a frame than a blank canvas or piece of paper. The frame defines what position means. Most often, the frame is a rectangular region and position is described in terms of the familiar (x, y) Cartesian coordinate system. In creating a frame, you must decide which variable in your data will correspond to the x coordinate, and which to the y coordinate.

For instance, consider a dataset relevant to economic productivity. Table 6.1 gives per capita GDP for each country as well as some of the explanatory candidates: average educational level in the population, length of roadways per unit area, Internet use as a fraction of the population.

country	gdp	educ	roadways	net_users
Australia	44353.87	5.60	0.11	>60%
Ghana	3509.96	8.10	0.46	>0%
Mozambique	1140.04	5.00	0.04	>0%
	... and so on for 256 rows			

Table 6.1: Data relevant to economic performance. The complete table is available at `http://tiny.cc/dcf/table-6-2.csv`

You define a frame by selecting two variables from the glyph-

ready data table. For instance, Figure 6.1 shows a frame based on GDP and length of roadways. The frame provides the meaning to location in space.

6.2 *Glyphs*

The frame itself doesn't display any of the cases. Instead, the glyphs positioned in the frame represent the cases. There will be one glyph for each case in the data table.

The basic shape used in scatter plots is a simple glyph: a dot, a square, a triangle, an x, and so on. The following graph uses small dots. Since each case is a country, each dot represents one country.

In Figure 6.2 the glyphs are simple. Only position in the frame distinguishes one glyphs from another. The shape, size, etc. of all of the glyphs are identical. There's nothing about the glyph itself which identifies the country. It's possible to use a glyph with several attributes. Figure 6.3 location and label, mapping country name to the label.

But glyphs can have several properties. The aspects of each glyph that we can perceive are called *aesthetics*, or equivalently *graphical attributes*. The word *aesthetics* applied in the context of glyphs is not used in the modern sense. Nowadays, most people associate aesthetics with notions of beauty and artistic taste. The earlier meaning of the word, *properties relating to perception by the senses*, is the one intended when it comes to glyphs.

Location in the frame are the (x, y) aesthetics for a glyph, but other aesthetics can display variables in the data table. For instance, color could be used to show Internet use (as a fraction of the population), as in Figure 6.4. Another aesthetic is size. The size is fixed in 6.4; the same for every country. Figure 6.5 maps the average years of eduction onto the size aesthetic.

6.3 *Scales and Guides*

There are four aesthetics in Figure 6.5. Each of the four aethetics is set in correspondence with a variable; we say the variable is *mapped* to the aesthetic. Length of roadways is being mapped to horizontal position, GDP to vertical position, Internet connectivity to color, and educational attainment to size.

A *scale* is the relationship between a variable and the aesthetic to which it is mapped. For `roadways`, the scale says what value of the variable will correspond to position at the bottom of the frame, what value will correspond to the top of the frame, and where things fall inbetween.

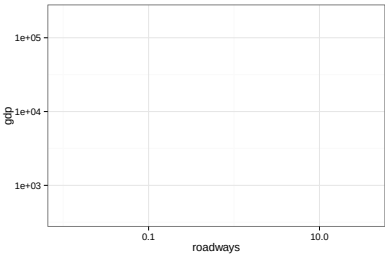


Figure 6.1: A graphics frame set by the `GDP` and `roadway` variables. No glyphs have been set in this frame.

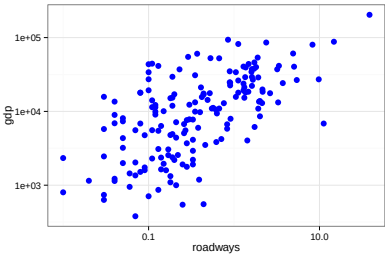


Figure 6.2: Using only position as the aesthetic for glyphs.

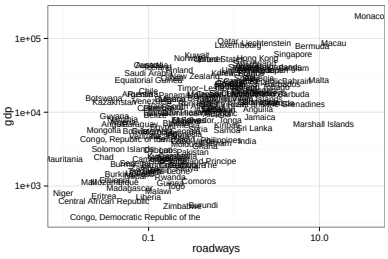


Figure 6.3: Using both location and label as aethetics.

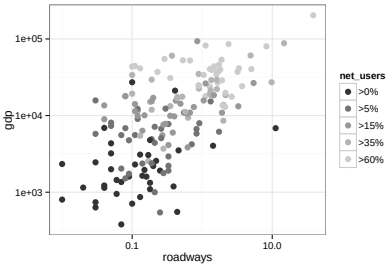


Figure 6.4: `net_users` mapped to color.

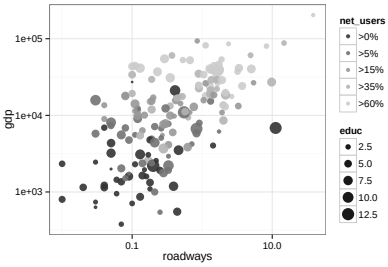


Figure 6.5: `net_users` mapped to color, `educ` mapped to size. Compare this graphic to Figure effig:facet-internet, which shows the same data using facets.

7
Wrangling and Data Verbs

When data are in glyph-ready form it is straightforward to construct data graphics using the concepts and techniques in Chapters 5. First, you choose an appropriate sort of glyph: dots, stars, bars, countries, etc. Next, select which variables are to be mapped to the various aesthetics for that glyph. Then let the computer do the drawing.

On occasion, data will arrive in glyph-ready form. More typically, you have to *wrangle* the data into the glyph-ready form appropriate for your own purpose.

WRANGLE: Transforming data from one or more data tables into a glyph-read format. The literal meaning of "wrangle" is to herd or round-up animals for some purpose such as bringing them to market. "Data wrangling" is a metaphor for putting data into a form suitable for the data visualization or analysis that's desired.

7.1 Examples of Wrangling

EXAMPLE: COUNTING THE BALLOTS. Table 7.1 records the choice of each of 80101 individual voters in the 2013 mayoral election in Minneapolis, MN, USA.

The primary use for data like Table 7.1 is to determine how many votes were given to each candidate Counting the votes for each candidate is a simple wrangling task that transforms the ballot data (Table 7.1) into a glyph-ready form (Table 7.2 right) that makes the answer obvious. The count information is still latent in the raw ballot data; it is not yet in a form where it is easily seen.

Precinct	First	Ward
P-10	BETSY HODGES	W-7
P-06	BOB FINE	W-10
P-09	KURTIS W. HANNA	W-10
P-05	BETSY HODGES	W-13
P-01	DON SAMUELS	W-5
... and so on for 80,101 rows		

Table 7.1: The choices of each voter in the mayoral election. See `Minneapolis2013` in the `DataComputing` package.

The instructions for carrying out the wrangling are simple to give in English: Count the number of ballots that each candidate received and sort from highest to lowest. To carry out the process on a computer, you need to express this idea in a form the computer can carry out.

candidate	count
BETSY HODGES	28935
MARK ANDREW	19584
DON SAMUELS	8335
CAM WINTON	7511
JACKIE CHERRYHOMES	3524
... and so on for 38 rows	

Table 7.2: The number of votes for each candidate. This has been wrangled from Table 7.1 into a glyph-ready form, that is, a form in which the sought after information is readily seen.

7.2 Planning your wrangling

Before you start wrangling data, it's crucial to envision what the goal is to be. In general, the purpose of wrangling is to take the data you have at hand and put it into glyph-ready form. But glyph-ready for

what? This is something you, the data analyst, need to determine based on your own objectives.

age	sex	smoker
2	female	no
77	male	no
10	female	no
1	male	no
49	male	yes
... and so on for 29,375 rows		

Table 7.3: The NCHS data from the DataComputing package. In NCHS, the case is an individual person.

EXAMPLE: DEMOGRAPHIC PATTERNS OF SMOKING Table 7.3 has variables indicating each person's age, sex, and whether he or she smokes..

Suppose you want to use NCHS to explore the links between smoking and age. Often you will have some kind of presentation graphic in mind. Making an informal sketch like Figure 7.1 can be a helpful way to chart a path for data wrangling. The sketch can be based on your imagination of what the data might show. Even so, the sketch contains important information. For instance, from the sketch you can determine what a single glyph represents. As always, each glyph will be one case in the glyph-ready data. In this sketch, the glyph describes a *group* of people — people of one sex in one age group. This differs from the cases in NCHS: individual people.

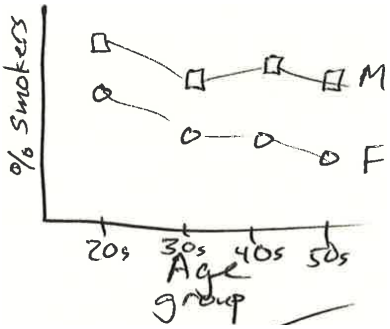


Figure 7.1: A made-up depiction of trends in the fraction of people of different ages who smoke.

To make your goal for data wrangling even more explicit, rough out the form of the glyph-ready data table, as in Table 7.4. Don't worry about calculating precise data values; the computer will do that after you have implemented your data wrangling plan.

The glyph-ready data for that graph will has a form like Table 7.4. That's glyph-ready form is your target.

Age group	sex	smokers
20s	F	31%
30s	F	15%
40s	M	20%

Table 7.4: A sketch of the form of glyph-ready data corresponding to Figure 7.1. For each aesthetic in Figure 7.1 there is one variable in the glyph-ready table.

ONCE YOU HAVE A TARGET GLYPH-READY FORMAT in mind, use plain English to plan the individual data wrangling steps. For instance, a plan for wrangling Table 7.3 into Table 7.4 might look like this:

1. Turn the age in years into a new variable, the age group (in decades).
2. Drop the people under 20-years old.
3. Divide up the table into separate groups for each age decade and sex.
4. For each of those groups, count the number of people and the number of people who smoke. Divide one by the other to get the fraction who smoke.

Each of these step would be easy enough to do by hand (if you had the time and patience to work through 31126 cases in NCHS). To get the computer to do the work for you, you have to be able to describe each process to the computer. The next sections present a framework for describing wrangling operations that can works for both the human describing the wrangling and the computer that will carry out the calculations.

Project: Bicycle Sharing

Figure A.14: Washington, D.C.

Capital BikeShare is a bicycle-sharing system in Washington, D.C. At any of about 400 stations, a registered user can unlock and check out a bicycle. After use, the bike can be returned to the same station or any of the other stations.

Such sharing systems require careful management. There need to be enough bikes at each station to satisfy the demand, and enough empty docks at the destination station so that the bikes can be returned. At each station, bikes are checked out and are returned over the course of the day. An imbalance between bikes checked out and bikes returned calls for the system administration to truck bikes from one station to another. This is expensive.

In order to manage the system, and to support a smart-phone app so that users can find out when bikes are available, Capital BikeShare collects real-time data on when and where each bike is checked out or returned, how many bikes and empty docks there are at each station. Capital BikeShare publishes the station-level information in real time. The organization also publishes, at the end of each quarter of the year, the historical record of each bike rental in that time.

You can access the data from the Capital Bikeshare web site. Doing this requires some translation and cleaning, skills that are introduced in Chapter (15). For this project, however, already translated and cleaned data are provided in the form of two data tables:

- `Stations` giving information about location of each of the stations in the system.

  ```
  Stations <- mosaic::read.file("http://tiny.cc/dcf/DC-Stations.csv")
  ```

- `Trips` giving the rental history over the last quarter of 2014.

  ```
  data_site <- "http://tiny.cc/dcf/2014-Q4-Trips-History-Data-Small.rds"
  Trips <- readRDS(gzcon(url(data_site)))
  ```

  The `Trips` data table is a random subset of 10,000 trips from the full quarterly data. Start with this small data table to develop your analysis commands. When you have this working well, you can

access the full data set of more than 600,000 events by removing `-Small` from the name of the `data_site`.

In this activity, you'll work with just a few variables:

- From `Stations`:
  - the latitude and longitude of the station (`lat` and `long`, respectively)
  - `name`: the station's name
- From `Trips`:
  - `sstation`: the name of the station where the bicycle was checked out.
  - `estation`: the name of the station to which the bicycle was returned.
  - `client`: indicates whether the customer is a `"regular"` user who has paid a yearly membership fee, or a `"casual"` user who has paid a fee for five-day membership.
  - `sdate`: the time and date of check-out
  - `edata`: the time and date of return

Notice that the location and time variable start with an "s" or an "e" to indicate whether the variable is about the start of a trip or the end of a trip.

Time/dates are stored as a special kind of number: a *POSIX date*. You can use `sdate` and `edate` in the same way that you would use a number. For instance, Figure A.15 shows the distribution of times that bikes were checked out.

POSIX DATE: A representation of date and time of day that facilitates using dates in the same way as numbers, e.g. to find the time elapsed between two dates.

```
Trips %>%
  ggplot(aes(x=sdate)) +
  geom_density(fill="gray", color=NA)
```

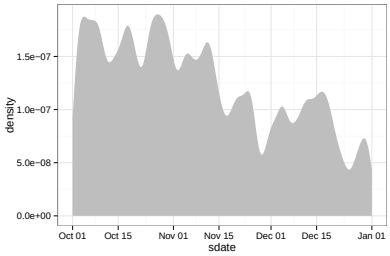


Figure A.15: Use of shared bicycles over the three months in Q3.

A.1 How long?

Your Turn: Make a box-and-whisker plot, like Figure A.16 showing the distribution of the duration of rental events, broken down by the `client` type. The duration of the rental can be calculated as `as.numeric(edate - sdate)`. The units will be in either hours, minutes, or seconds. You figure it out.

When you make your plot, you will likely find that the axis range is being set by a few outliers. These may be bikes that were forgotten. Arrange your scale to ignore these outliers, or filter them out.

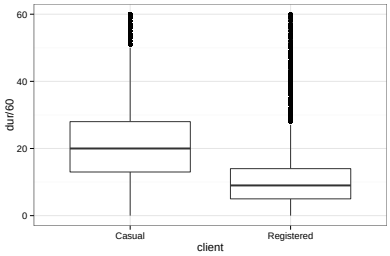


Figure A.16: The distribution of bike-rental durations as a box-and-whisker plot.