Preface

How to get the most out of this book

Contents:

0.1. Why we wrote this book

This textbook is a concise introduction to probability, statistics, and exploratory data analysis. It is intended for use in a one-quarter or one-semester course aimed at undergraduate students who are majoring in the liberal arts. As an instructor or a student, you may never have thought about the underlying reasons for writing a textbook. You may possibly have shared our cynical belief that many textbooks are overpriced and revised far too often, with the only underlying purpose being for the publisher to derive a profit. This book is different. It was not requested by a publisher, but rather sprang forth from our experience in developing and teaching a course at the Ohio State University called *Analyzing the sounds of languages*.

This course was designed with the goal of fostering numeric reasoning skills by presenting data analysis techniques using accessible real-world datasets to address research questions that can be understood by anyone who commands a spoken language. We designed this course because we think that it is at least as necessary to be ***numerate*** as it is to be literate, but many of the undergraduate students whom we have taught in our collective five decades of teaching are not numerate. This puts them at a disadvantage whenever they encounter probability-based models or statistical results cited as evidence for accepting some belief or taking some course of action. They will need to evaluate such evidence in many of the courses that they take in college, at the polls as they begin to vote in local and national elections, on the job market as they search for employment after graduating, and so on. Indeed, we cannot think of an area of modern life in which a facility with statistical reasoning is not important.

We think that this disadvantage is because in many public schools in North America, ***numeracy*** is shortchanged relative to literacy. We have encountered the attitude that, because numeric reasoning is inherently difficult and because some children do not have a natural talent for grasping mathematical concepts, numeracy is expendable for those children who struggle. We cannot agree with this attitude. That is, we acknowledge that numeric reasoning skills can be difficult to learn. Numbers and statistics are relatively recent inventions of our species. But writing systems, too, were invented recently enough that some children, perhaps as many as 17% of children in the U.S., have a neural condition that makes it difficult to learn to read (Shaywitz, 1998). Yet no one would conclude from this fact that learning how to read and write is optional. Numeracy, also, is too important to reserve for the talented few. Learning how to read patterns in numbers, and to reason from those patterns, is too important to be optional for young adults who are majoring in the arts or humanities. Students in these liberal arts need numeric reasoning ability as much as do students in the biological and physical sciences. Resources are needed to make numeric reasoning skills attainable for everyone.

What makes numeric reasoning skills attainable? We follow George Cobb (1991) in thinking that a key ingredient is “More data, less lecturing.” We also agree with him in thinking that

the quality of the datasets is critical. We ourselves have struggled through textbooks or tutorials that introduce statistical tools by applying them to timeworn data such as Frank Yates's (1935) plot yields for different combinations of oat seed strain and nitrogen. Or worse, they (mis-)apply statistical tools to made-up data, such as the number of pounds of cashews, brazil nuts, almonds, and peanuts in a hypothetical batch of mixed nuts, an example "from a popular introductory statistics textbook" that Robert Hayden cites to illustrate the kinds of errors in elementary statistics textbooks that "can go uncorrected for years" (Hayden, 2000, p. 2). Many students from the arts and humanities find these examples irrelevant, if not off-putting. We aim instead to use examples from a domain that should be accessible to a general audience, by presenting analyses of data about the sounds of languages. In section 0.3, we expand on this aim and illustrate the familiarity of such data with an example from one of our students in a general course on language structure.

0.2. Establishing a research question

In this book, we focus much more heavily on developing and evaluating research questions and hypotheses than on the mathematical operations underlying the data analysis tools presented in each chapter. Our justification for this falls along the same lines as our reason for using real and accessible data. Numerical analysis of data is very much about context and interpretation. It doesn't matter if you can calculate an average by hand if you do not know how to determine when it is appropriate to use an average to relate numbers to a specific question. If you don't understand where the numbers come from or what they mean in a broader context, then how can you analyze them appropriately?

Therefore, the first priority in each chapter (after this preface) is to familiarize you with the context for the research question. The first section will introduce a linguistic problem and a broad question or set of questions that help define the problem, such as (for Chapter 1) "How inconsistent is the English writing system?"

We will then ask more specific questions that can help us answer the larger question by using the techniques we will develop in each chapter. Broad questions are difficult to answer because they frequently have many components or are just too large in scope to tackle on their own. The smaller, more specific questions will narrow down the broader question into smaller, more quantifiable and testable chunks. The results will provide evidence for an answer to the broader question, but may not definitively "answer" it. For example, we may ask about spelling patterns using the relationships between a few letters and the sounds that they write in English words, and from this, we may infer that similar patterns could apply across the language. We can apply this inference to the broader question, as you will see in the last sections of the chapters.

Is is a fact about data analysis that no set of numbers or quantitative analyses can "answer" a question. Rather, these numerical analyses can provide evidence that supports or chal-

lenges a hypothesis. The hypothesis arises as a testable assumption about your data, from a well-defined research question. You cannot "prove" that the hypothesis is correct, but you can collect evidence that either supports or contradicts your hypothesis. Some evidence is more convincing, or more directly addresses the research question and hypothesis. Generally speaking, the more times a hypothesis is upheld by the evidence, the more likely that it is true. The hypotheses we present in this book are created with the intent of using specific numerical and analytical tools to provide the supporting evidence, and will be presented along with the relevant tool.

0.3. Why analyze language sounds?

A practical reason for using data about the sounds of languages is that we are linguists who specialize in studying sounds and sound patterns. So when we began to design this course, we had ready access to real datasets that we or our colleagues had gathered, in order to address questions about language that we ourselves find interesting. A more compelling reason for us to use these data is that we don't think we're alone in finding the questions interesting. Everyone who has English or some other spoken language as his or her native language can find ways to relate to the data being presented. Even before being introduced to linguistics, you may have wondered about some of the questions we can ask about language sounds.

For example, you may have wondered why some people have nicknames, and how they got these nicknames. Some nicknames, such as *Doc* or *Red*, are obvious references to events or personal characteristics, but many nicknames are just variations on the sound pattern of a person's given name or title, as in the nicknames *Sam*, *Jon, Cindy*, and *Bubba* for people called *Samantha (or Samuel)*, *Jonathan*, *Cynthia*, and *Brother*. One of our students, nicknamed *Lalo*, wondered whether his less typical-sounding nickname might be related to the more typical patterns that can result when other family members adopt the pronunciations of siblings who are in the early stages of language acquisition and aren't yet able to pronounce all of the sounds in the full name.

hɛrarðo

lalo

The student himself has the Spanish given name *Gerardo*, pronounced **hɛrarðo**, and his nickname came from his older sister, who was only 2 when he was born and called him *Lalo*, pronounced **lalo**. Something that Gerardo noticed about his nickname is that the consonant sounds in **lalo** are different from any of the consonant sounds in **hɛrarðo**. The changes seem a bit like the changes in the second consonant sounds in *Cynthia* and *Brother* when they are shortened to *Cindy* and *Bubba*. That is, the way that the two **r** sounds in **hɛrarðo** change to the **l** sounds in **lalo** seems a bit like the way that the **θ** sound that is spelled with 'th' in *Cynthia* changes to the "easier" **d** sound in *Cindy* and the way that the **ð** sound (also spelled 'th') in *Brother* changes to the **b** sound in Bubba. Gerardo wondered if there is any pattern in which sounds change to other sounds when nicknames are made from full names in this way. If so, does this pattern tell us something

about what sounds are easier for young children to pronounce?

In section 0.4, we will explain the symbols that we have used to represent the pronunciations of Gerardo's name and nickname and to describe the changes to the pronunciations of the second consonants in making the nicknames for *Cynthia* and *Brother*, but first let's talk about an even more obvious change that we can describe without any such technical tools. Gerardo's name is three syllables, but his family's nickname for him is only two syllables. Two-year olds don't use very many three-syllable words, so it might have been easier for his sister to say his name by shortening it in this way. In the following chapters, we will introduce ways to begin exploring these questions quantitatively, using data from the real world. The numbers that we will give you only make sense if you can relate them to the question that you are trying to address. So we will start each chapter first by developing a research question. While the aim of this textbook is to give you resources to help you get comfortable with numerical analyses and statistical reasoning skills, the numbers have to be about something for the analyses to be meaningful, which is why we use language data.

0.4. An aside on letters versus sounds

Both in posing the broader question in each chapter, and in narrowing it down into more tractable smaller questions, we will try, as much as possible, to avoid using technical jargon from linguistics. In some courses at some universities, this textbook has been used in conjunction with Peter Ladefoged's book *Vowels and Consonants*, as part of a structured major in linguistics. But we want this textbook to be generally accessible, so we will try to make the questions and datasets accessible without any previous specialist knowledge of linguistics as a scientific discipline.

At the same time, there are a few things that are much easier to talk about if you have access to some specialist tools. The most important of these is the conventions for distinguishing between letters and sounds. As you probably noticed in the nickname examples in section 0.3, the pronounced form of a word often is not the same as its spelled form. For example, both *Brother* and *Cynthia* have a 'th' consonant in them, which is pronounced as **ð** in *Brother*, and **θ** in *Cynthia*. In fact, you may not have realized before that these are two different consonant sounds because they are spelled the same. We will will use different ways of writing to differentiate pronounced forms from spelled forms, in order to be clear.

Specifically, we will adopt the following conventions from Peter Ladefoged's book. We will write examples of spelled forms of words in the Roman alphabet, setting them off from the surrounding text by using italics, as in *Gerardo, Samantha,* and *Cindy*. We will write letters for parts of spelled forms using ordinary type but enclosing them in single quotes, as in the letters 'r', 'd', 'a', and 'o', in the name *Gerardo*, or the letter sequence 'th' that spells the third consonant in *Cynthia*. We will write examples of spoken forms of words in the International Phonetic Alphabet (IPA), setting them off by using boldface,

as in **hɛɾaɾðo** for the spoken form of the name *Gerardo*, **sɪnθiə** for the spoken form of *Cynthia*, and **bəbə** for the spoken form of *Bubba*. We will also write individual vowel and consonant sounds using boldface IPA, as in the vowel sounds **ɑ**, and **o**, and the consonant sounds **ɾ** and **ð** in the spoken form **hɛɾaɾðo** for the written name *Gerardo*, or the **θ** sound that the 'th' represents in *Cynthia*. (Sometimes, though, we will have to substitute other symbols, when the IPA font is not available in the plotting program that we are using to make charts and graphs. In those cases, we will explain the symbols that we are substituting for the IPA, as we do in the caption of Figure 1.1 in Chapter 1.)

Please don't feel bogged down by this way of distinguishing between letters and sounds. In some chapters, aspects of the research question focus on the analysis of certain sounds. In that case, we will tell you about these sounds in detail when we introduce their IPA symbols. In many other chapters, we will use the IPA to give examples that illustrate how words are pronounced or what differences there can be in how words are pronounced. When we give such examples, we will not introduce every sound and IPA symbol in detail, but we will always provide you with enough of the necessary information for you to understand the example.

If you are interested in learning more about the individual sounds that we write in IPA symbols, you can look up the symbols in the figures and tables in this section of the preface. On the next two pages, we give you the full set of symbols in the consonant and vowel charts that are provided by the International Phonetic Association. These charts are in Figures 0.1 and 0.2. The shaded box below Figure 0.1 gives you a rough idea of how to pronounce each of the consonant sounds. (In later chapters, too, we'll use such shaded boxes to set off other tidbits about technical jargon or interesting tangential information about the data sets and so on.)

Directly after each of the figures, tables indicate which symbols and symbol sequences we will be using to represent all of the consonant sounds (Tables 0.1) and all of the vowel sounds (Table 0.2) in the dialects of American English that are relevant in this book. For each symbol in these two tables, we also provide one or more sample English words containing that consonant sound or that vowel sound.

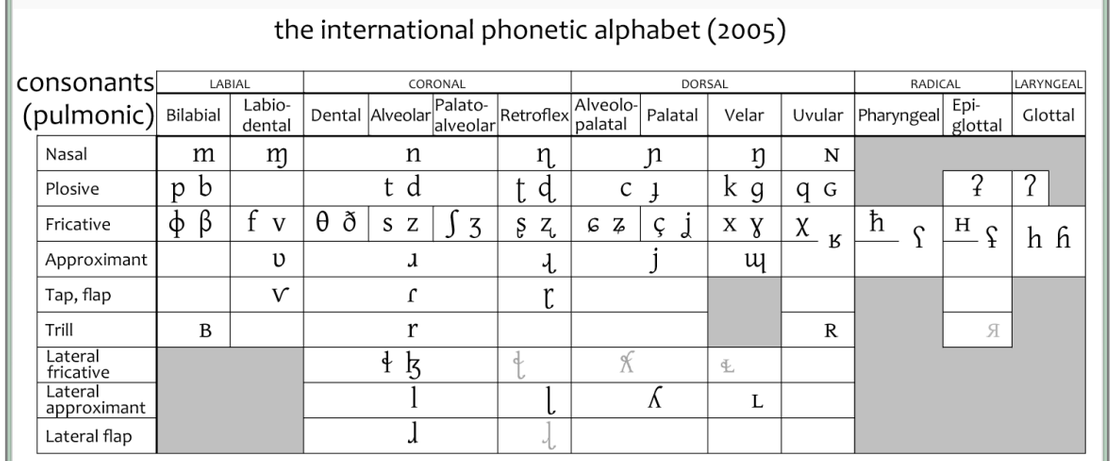
The information in the column and row headings of these two figures provides a very bare-bones description, using technical terms from the discipline of phonetics. The tables will be useful for interpreting the descriptions if you are a native speaker of American English. If you are not a native speaker, you may need to consult with a friend who is a native speaker, to help you interpret the sounds from the friend's pronunciation of the words in the tables. Alternatively, you can consult an introductory phonetics textbook, such as Peter Ladefoged's *Vowels and Consonants*. (Even if you are native speaker, if you are interested in learning more about the IPA and the classificatory features that organize the charts of consonants and of vowels, you might want to read *Vowels and Consonants*. It is a concise and well-written introduction to phonetics that has been used even in high school classes.) Additionally (at the time of

this writing), a website associated with his book provides clickable recordings to illustrate each of the sounds in the IPA, located at http://www.phonetics.ucla.edu/course/chapter1/chapter1.html .

Reading the IPA consonant chart

You can figure out roughly how a consonant sound is pronounced by the labels at the top and on the sides. The place of articulation means what parts of the vocal tract are involved in making the sound. In consonants, this means where the articulators are closest together. So, a labio-dental sound, such as **f**, is one that involves the lips (labial or labio-) and teeth (dental) coming close together or touching. An alveolar sound, such as **t**, is one where the tongue approaches or touches the alveolar ridge (which is just behind the upper teeth). The labels on the left refer to the manner of articulation. A fricative, such as **f**, is produced with a close constriction but not a complete one, so that the air passing through makes a hissing or buzzing noise. A plosive (also called a stop), such as **t**, is produced by stopping the air entirely at the place of articulation. Many consonants in the chart appear in pairs. In this case, the right member of the pair is voiced, which means the vocal folds are vibrating during the consonant sound, and the left member is voiceless, which means that the vocal folds are not vibrating during the consonant sound.

Figure Preface.1 Consonants of the IPA



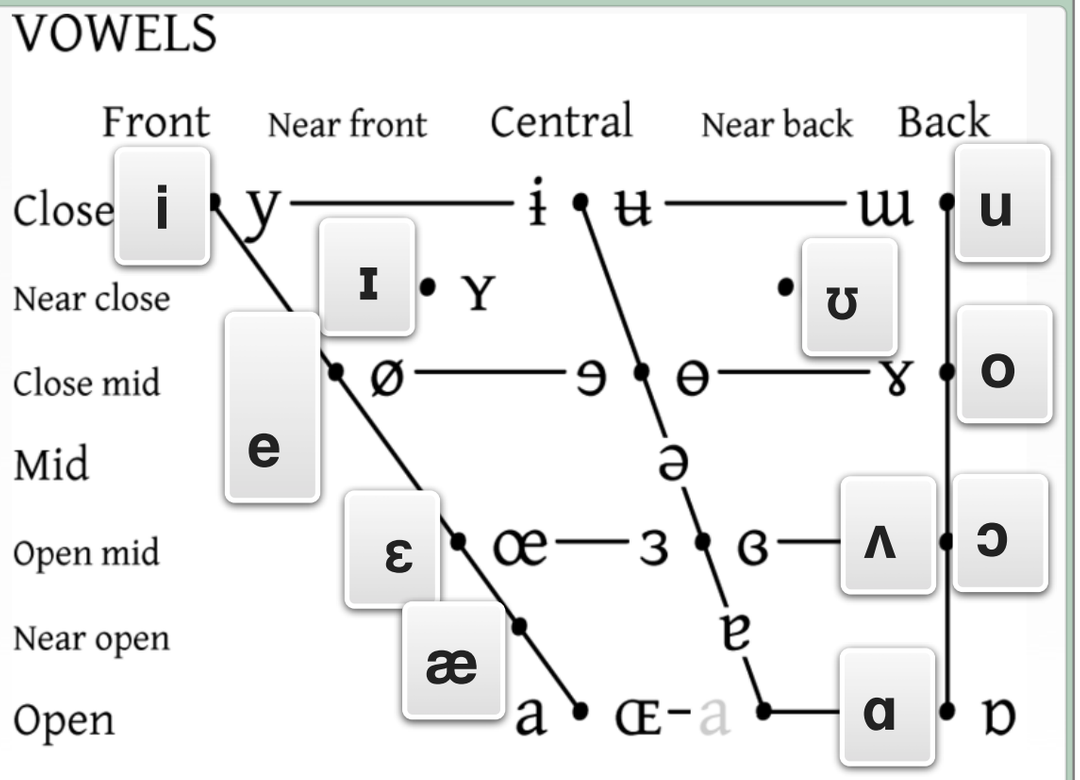
Click on upper left corner to enlarge. Tap outside image to close, or hit escape.

Table 0.1. American English consonants, illustrated where possible in initial and final position.

p *pea, ape*		**t** *tea, ate*	**ʧ** *chew, itch*	**k** *key, ache*
b *bee, Abe*		**d** *day, ad*	**ʤ** *jaw, edge*	**g** *go, rug*
m *me, aim*		**n** *no, on*		**ŋ** *rung*
f *fee, off*	**θ** *thigh, oath*	**s** *sow, ace*	**ʃ** *show, ash*	
v *vow, eve*	**ð** *the, seethe*	**z** *zoo, ooze*	**ʒ** *rouge*	
		ɹ **row, oar*	**j** *you*	**w** *we*
		l *low, awl*		

*We've put the IPA symbol for the American English sound, ɹ, here in the table, but in the text we often substitute the IPA symbol **r**, which stands for a different sound. The IPA symbol **r** represents an alveolar trill. This is a sound that is written with the letter sequence 'rr' in Spanish words such as *perro* (meaning 'dog'). There are many English dialects (e.g., Irish English, Indian English) that have an alveolar trill (i.e., **r**) as the sound that is written with the letter 'r' in English words such as *row*. Because the difference in pronunciation doesn't change the meaning of the word, we feel that the ease of reading the more familiar form justifies this substitution.

Figure Preface.2 IPA symbols for vowels.



Reading the IPA vowel chart

Vowels are pronounced with the least amount of constriction of the vocal tract, which is one characteristic that sets them apart from consonants. You can figure out roughly how a vowel sound is pronounced by the labels at the top and on the sides of the chart, which indicate how the tongue is positioned and the corresponding shape of the mouth. The labels on the left refer to the height of the tongue and thus the degree of closure of the mouth. A close vowel (also called a high vowel) such as **i**, is produced with the tongue high in the mouth, which draws the jaw more closed, which is why photographers may tell you to say *cheese*, so you don't have your mouth hanging open in the photograph. Likewise, a doctor who wants to examine your throat may instruct you to say **ɑ** so that your mouth is maximally open and your tongue is low and not obstructing their view. Labels along the top refer to whether the tongue is closer to the front of the mouth, as in **i**, or to the back of the mouth, as in **u**.

Table 0.2. American English vowels

i	*heed, beat, key*	**diphthongs**	
ɪ	*hid, bit, kiss*	**aɪ**	*hide, bite, Kaiser*
e	*hay, bait, case*	**aʊ**	*how, bout, cow*
ɛ	*head, bet, Casey*	**oɪ**	*hoy, boy, coy*
æ	*had, bat, Cass*		
ɚ	*heard, Bert, curse*		
ɔ	*haught, bought, cause*		
ɑ	*ha, bot, cost*		
ʌ	*Hud, butt, custard*		
o	*hoe, boat, coast*		

0.5. Data analysis techniques and data sources

After introducing the research question, in the middle sections of each chapter, we will introduce some data analysis techniques that we can use to address the research question and to provide the evidence that will lead us to some kind of answer to the question. We will be using real data collected from experiments and corpora used by researchers in the relevant sciences. We will give full references for the datasets that we use in the body of the text and the exercises that follow. This is a standard procedure that holds researchers accountable for their measurements and analyses, so that others can replicate an analysis presented in a study to ensure its ***reliability***. That is, it is critical to say exactly where the numbers in a study came from to see if other studies using numbers gathered in the same way can get the same results. We also give a brief description of each dataset so that you can decide for yourself if we are analyzing the data in a way that is appropriate to the data and the questions we have about the data — a concept which is known as ***validity***. Getting critical skills in evaluating the reliability and validity of a study is far more important than remembering the technical details about how to do a particular statistical test. This is why we will focus so much on figuring out what kind of data we are looking at, and on formulating and answering appropriate research questions.

At the same time that we introduce the data, we will introduce various useful methods and tests for analyzing the data. Some chapters will focus on numeric measurements used to describe the distribution of data, and some later chapters will focus on statistical tests that are commonly used to describe how well the data fit some model or hypothesis. Not only do we describe all of these methods, we will also apply each of them in turn to some data set, to help us answer the research questions posed in the beginning of that chapter.

We will also spend a fair amount of time in the first few chapters discussing different ways of visualizing data. A good part of numeracy is being able to display data in an honest and intelligent way. This means using the display method that is appropriate for the data and research question at hand. It is important to understand how to design and describe the appropriate graphical representation for each type of data, because pictures are often easier to grasp than words. It is also important to be able to understand and critically analyze graphical representations, because you will often see data and numerical evidence summarized in this way. Developing the skills to be able to read a graph in an instruction manual or to identify when a graph in a news article is misleading could even save your life.

0.6. Summary sections to expect

Toward the end of each chapter, we will always have a section called “Answering the research question.” In this section, we will remind you of the research questions, sum up the evidence we have uncovered using the data and data analysis techniques introduced in the chapter, and then conclude with

a summation of the answers we have inferred from our application of data analysis techniques.

We also will give a summary of the key terms that were introduced in the chapter to help you recall what you've learned. It will be in a box set off from the rest of the text, as shown below.

This iBook version also gives you the option of finding the glossary term by clicking on the ***dark blue italicized word*** as it appears for the first time in the text.

Summary of key terms

Numeracy: The fundamental understanding and ability to conceptualize how numbers work, including relationships between numbers, magnitude of numbers, and how operations can be applied to numbers. Also known as numeric literacy.

Numerate: Possessing numeric literacy, or "numeracy". That is, the ability to think about, conceptualize, and understand reasoning, numbers, numeric systems, etc.

Reliability: Also known as replicability, that is, whether performing the same experiment multiple times or performing the same analysis on multiple samples will yield the same results. If your analysis is reliable, others should be able to duplicate your results.

Validity: Your results are not valid unless you use the appropriate analysis for the type of data you have and the analysis answers the questions you have set out to research.

0.7. Data analysis software

Note that while we want you to develop facility in the methods that we introduce, we don't want you to get bogged down in the calculations and technical details at the expense of a good understanding that can be gained by practicing the skills. So we will encourage you to learn to use data analysis and graphics programs that can do the more tedious parts of the calculation of numbers and data plotting for you.

There are a variety of software options available for graphing, statistics, and other forms of data analysis. In this book, we have chosen to use the open source data analysis package called R. R has many advantages over other programs that we have used. It is available for free download, it can be used with any of the three most widely used operating systems (Windows, Mac, Linux), it is highly stable, and it is at least as powerful as any of the expensive high-end commercial statistical packages that we used before R was developed. Unlike commercial software, open source software like R is created and updated by people who use it. New packages and updates to the base program are created when the community of R users wants them enough to make them, not when some company wants to boost profits.

As with any new software package, or even a new calculator, you will need to spend some time familiarizing yourself with how it works. When we introduce a particular graphical technique or a statistical test in the middle sections of a chapter, we often will show some relevant R code in a box called an "R Note" like the box below.

R Note 0.1 – Doing arithmetic in R

The English word *computer* was first used to mean a person who performs arithmetic and other mathematical computations for a living. For example, the six women who set up the original programs for the ENIAC machine in the 1950s were drafted from the many "computers" who had been hired to work for the Manhattan Project. Today, of course, *computer* instead means a machine that does arithmetic and many other types of computation electronically. Although we will be using R for much more complicated calculations, it's good to first learn how to use it for arithmetic. You can perform addition, subtraction, multiplication, and so on just by typing the relevant formula. For example, to subtract the number of letters in the spelled form *Cindy* from the number of letters in *Cynthia*, you can type:

```
7 - 5
```

This returns the following value:

```
2
```

That is, if you type the formula in the R console window, followed by a carriage return to execute the formula, the result of performing that operation will appear as the next line of text in the window. See the R code in section 0.9 for examples of more elaborate formulas, as well as for information about how to download R, and so on.

0.8. R code

After the summary sections at the end of each chapter, there will be a section called "R code" to help you learn how to use this program to do the types of data analysis covered in that chapter. The course website, http://kb.osu.edu/dspace/handle/1811/77848, includes all the files that are associated with this textbook, and each file will have a name that is associated with the relevant Chapter number. For example, the first data file you will need is called `Ch00.Textfile1.txt`. All of the "R code" sections introduce and explain useful bits of R code, which will be included in the associated files.

In each of the later chapters, the code that is introduced is similar to the code that we used to make some of the tables and figures in that chapter. The R code section in this preface chapter is a bit different. It's intended to be a more general introduction and reference section. That is, we will first explain what you need to do in order to download and install R and to take best advantage of the R code sections. Then we will give a quick overview of some of the R code data structures and operators that you will be using in every chapter.

Note that the textbook web page also has at least one ***R script*** file for every chapter that gives the code in a format that lets you open the file within R, to work through the code interactively, as you read the R code section. A copy of this R note is saved inside a script that is called `Chapter00.R` and can be found in the list on the course website, at

http://hdl.handle.net/1811/77848, or directly at http://kb.osu.edu/dspace/bitstream/handle/1811/77848/Chapter00.R. Once you have opened the script from within R, you can read each comment (a sequence of lines that is marked off by "#" at the beginning) and then run the line(s) of code that the comment describes. Note that while you can type the lines directly into the console, it is easier (and less error-prone) to open the script editing window, type the command there, and then run the code directly from the script editing window. You can run a section of code from the script editing window by highlighting the command(s) you want to run, and then typing (together) the <command>+<return> keys (on a Mac) or the <ctrl>+<r> (on a PC running Windows). We encourage you to download and install **R** and use these R code sections of the textbook and the associated script files to teach yourself how to use R. (We also encourage you to open a new script file whenever you start working on a particular concept, so that you can try out commands by writing them in the script editing window and making notes for yourself in comment lines about what a command was intended to do, whether it worked or not, how you modified it to get it to work if it didn't do what you wanted it to do the first time, and so on. That is, get in the habit of using **R** to help you keep track of your developing understanding of the different types of data and graphs and statistical techniques that you will be learning about in this course.)

Part 1. Downloading and installing R

1) Get on the internet and point your browser to http://cran.r-project.org/
2) Under the first heading, "Download and Install R", click on the appropriate platform for your system (i.e., Mac, Windows, or Linux). You will be given the choice of downloading from a list of mirror sites. The download may be faster if you choose one that is near to you.

The rest of the steps are specific to the platform that you are using.

For the Windows platform:

3) Choose the "base" installation. We do not need any of the other packages for material covered in this book.
4) Click on Download R *X.X.X* for Windows, where *X.X.X* stands for whatever happens to be the most recent version (as of this writing 3.2.3 was the most recent version).
5) Install by double clicking on the .exe file, and choosing a directory to install R to. It is easiest if you install it directly to your C drive. Then, just follow the instructions in the install wizard.
6) If you need more help, click on the link labeled 'Installation and other instructions' located on the same page as the download.

Mac (OSX 10.5 and later)

3) Click on the R-*X.X.X*.pkg (where *X.X.X* stands for the latest version, which was 3.2.3 as of this writing)

4) Either save and install by double clicking on the zipped package, or allow your archive utility or Stuffit to unzip it as it downloads (that is, choose save or open). Follow the instructions in the install wizard and make sure the R program is saved under Applications.

Mac (earlier versions than 10.5)

3) Go to the following page to find a version that is compatible with your legacy OS:

http://cran.r-project.org/bin/macosx/old/index-old.html

4) Click on the R-*X.X.X*.pkg suitable for your OS.

5) Either save and install by double clicking on the zipped package, or allow your archive utility or Stuffit to unzip it as it downloads (that is, choose save or open). Follow the instructions in the install wizard and make sure the R program is saved under Applications.

Linux

3) Click on the folder for the particular flavor of Linux that you have installed and then follow the instructions there.

Welcome to the wonderful world of R!

Movie Preface.1 Opening a new scripting window in R
external link: http://streaming.osu.edu/player/knowledgebank/?f=LING/LING_kbdirect_2051_Ch00_Movie1_m4v.html

Part 2. Data files and script files

One of the biggest advantages of using R is that it lets you cleanly separate the act of recording the measurements and other observations that constitute your data from the act of developing the formulas and numeric analyses that you will be learning to apply to the data. That is, the R program is a programming environment that lets you read in data from one file (such as text file of words or numbers that you created by copying them from a browser screen) and also to read in a different file of formulas and other code that you can apply to the data.

We'll call the first kind of file a ***data file*** and we'll call the second kind of file an ***R script***. After you apply the analyses to

the data, you can save the results in a different data file, without overriding the original data. You also can edit the script file and save the edited version to a different file. This lets you experiment easily with different ways of manipulating the numbers that are your data without overwriting any of the numbers or any of the code.

As we said above, we'll be providing data files for many of the research questions that we introduce in the later chapters and we'll be providing at least one R script for the R code section of every chapter. We'll also be providing a section of exercises for each chapter, to help you practice the concepts that are introduced in that chapter. We recommend that you practice your understanding of the concepts by doing as many of the exercises as you can. To take best advantage of R, we recommend that you also work through the R code in the R script for the chapter, line by line, to make sure that you understand what the code is doing, and then practice your understanding by using and adapting the R code in the exercises where it is appropriate.

Part 3. Getting started

In the rest of this section, we will cover some R basics. The code that we will introduce is saved inside an R script that you can use interactively. All of the commands in parts 4-12 are in a script called `Chapter00.R`. You can follow along more easily if you download that script now, and try opening it. Here's how.

When opening an R script, if you are running Windows or Linux, you should first start the R program. Then from within the R program, choose the File menu, then select "open script" and browse to where you have saved the R script, for example a copy of this R note is saved inside a script that is called `Chapter00.R` and can be found in the list on the course website, at http://hdl.handle.net/1811/77848 or individually at http://kb.osu.edu/dspace/bitstream/handle/1811/77848/Chapter00.R.

If you open the script by double-clicking on it on a PC, it will open in a text editor (such as WordPad), so you will not be able to use it directly as an R script.

On a Mac, you can open a script the same way from within R, or you can double click directly on the script, and it should open in R.

Once you have opened the script, you should have two windows running R. The window that contains the text and code from this section is called by the name of the R script. The main window, where all of the output is listed, is called the ***R console***. The difference is that you can write and erase as much as you want inside the script, and it doesn't do anything until you transfer a line of code to the console. Inside the console window, you cannot change a line of code once it is entered; you instead have to re-enter the changed line of code. To run any line of code, do any of the following:

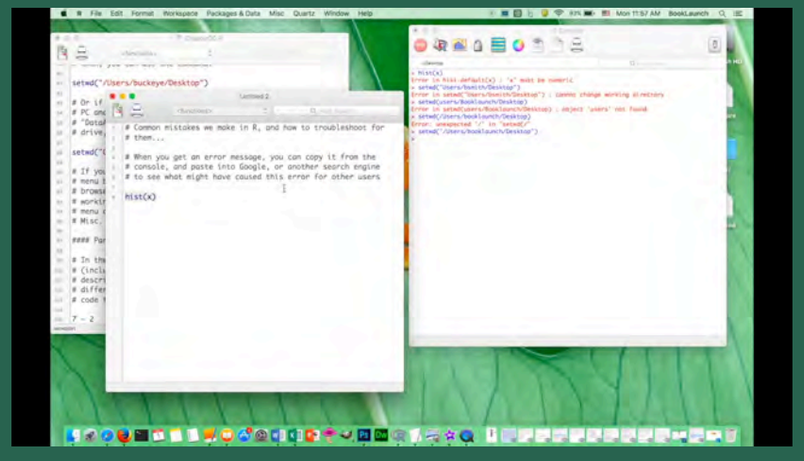
1) Type the line directly into the R console window followed by a carriage return.

2) Use the keyboard or mouse to copy the line (including the carriage return at the end) from the script and paste it into the R console window.

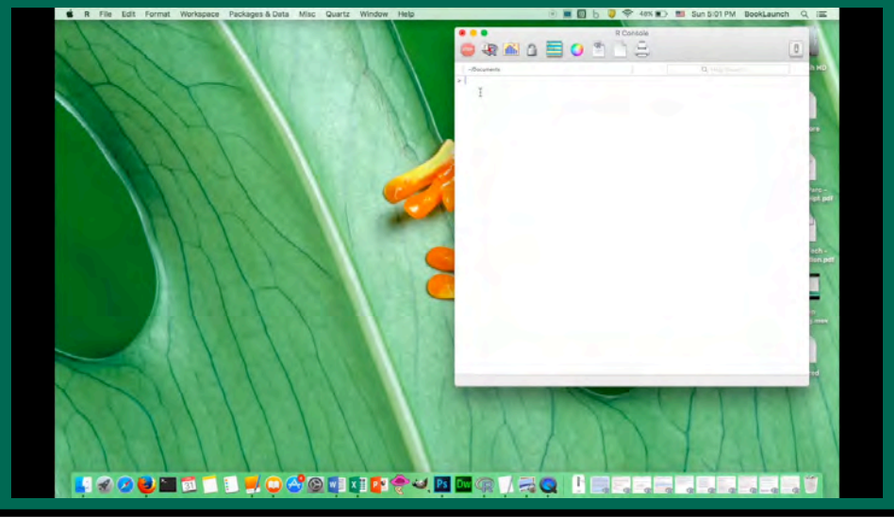
3) Highlight the part of the code you want to use from the script by clicking and dragging your mouse over it. Then, for a Mac, hold down the `<command>` key (the swirlygig button, formerly the apple button) on the left of the keyboard, and then also hit `<return>` (as you are continuing to hold the `<command>` key). If you use R on a PC running Windows (or Linux), you will highlight the code and press `<ctrl>` and the letter "`r`".

The last option is the fastest way of getting a line of code to work because it transfers the code to the console and executes the command all at once. Even if you are writing the commands or functions yourself, you should write them in a script

Movie Preface.2 Setting the working directory, and troubleshooting common errors.
external link: http://streaming.osu.edu/player/knowledgebank/?f=LING/LING_kbdirect_2051_Ch00_Movie2_m4v.html



Movie Preface.3 Opening a script, changing the working directory, and command versus comment lines
external link: http://streaming.osu.edu/player/knowledgebank/?f=LING/LING_kbdirect_2051_Ch00_Movie3_m4v.html



window first and then execute them from the script. That way, it is easier to go back and make changes to your code.

This is important because you will probably have to make changes to your code fairly often at first, as you learn about the different operators and functions that you will be using. It is also important because one of the more effective ways to learn about these operators and functions is to try out small variations on a line of code and see what happens when you change some part.

When you change the working directory, the path might be something like the following (note that forward slashes are used in the path name):

```
setwd("C:/My Documents/data_analysis")
```

If you are using a Mac, you can also go to `Misc` in the top menu bar, and choose “Change Working Directory” and then

browse until you find the directory where you want to be working. If you are working on a PC running Windows, this menu choice will be under the `File` heading, rather than `Misc`. Choose "`Change dir...`", and then browse for the directory you want.

R Note 0.2

At this point, you should have downloaded the R program and opened both the console and the `Chapter00.R` script that you downloaded from http://kb.osu.edu/dspace/bitstream/handle/1811/77848/Chapter00.R. We will list the same information on the following pages as is contained in the R script, but you should use the instructions in the script to be working through the R script line-by-line instead of reading it in the book. The videos will give you an idea of the **R** experi-

0.8 R code

R code (parts 4-12) and exercises

Part 4. Set the working directory

Next, it is a good habit to always set the working directory to where you have any data files located that you will be using. For example, to execute the command to read in the data file of names and nicknames that we will be working with in the exercises, you should have downloaded the nicknames.txt file to some directory such as the place where you are storing your notes from class for this course. For example, if you were an Ohio State University student and you were working in the computer lab where we teach the course in Columbus, you would download the file to the desktop. To set the working directory to the desktop then, you can use the command:

```
setwd("/Users/buckeye/Desktop")
```

Or if you are working at home, and your home computer is a PC and you're saving your notes in a directory called "data_analysis" in the "My Documents" directory on your hard drive, the path might be something like the following (note that forward slashes are used in the path name):

```
setwd("C:/My Documents/data_analysis")
```

If you are using a Mac, you can also go to `Misc` in the top menu bar, and choose "`Change Working Directory`" and then browse until you find the directory where you want to be working. The directory you want is the folder on your computer that contains the data set that you want to work with. If you are working on a PC running Windows, go to `File` instead, choose "`Change dir...`", and then browse for the directory you want.

Part 5. Command lines versus comment lines

In the R Note box above and in every R code section (including this one), we will be distinguishing our descriptions of R commands from the code proper by using different fonts. For example, the following line is the R code for subtracting 2 from 7:

```
7 - 2
```

If you are using the script `Chapter00parts3-5.R`, you will probably have noticed by now (but it doesn't hurt to point out) that strings of text preceded by the hash mark # are notes inside the R script. R ignores anything that is preceded by a hash mark #, so that's why we use these to "hide" notes in a

script. For example, the following is the same line of R code, with a note about what it does.

```
7 - 2    # Subtracts 2 from 7
```

You can write additional notes for yourself inside the script and save it to look over later. We encourage you to make notes for yourself directly inside your scripts, so that you can remember what you learned about some code right in the file where you wrote and ran the code.

Try it. Write a note to yourself in the R script file that you have opened, starting the line with the "#" symbol, and then on the next line write a line of code such as a subtraction problem. Then, highlight both lines together and run them. See what happens on the R console. Then edit the first line to remove the "#" and highlight and run the lines again. Compare what happens on the R console now with what you saw when you ran the two lines before.

Part 6. Arithmetic operators

The easiest thing to do in R is basic arithmetic operations. Basic operator signs are as follows:

```
459 + 51 # Adds 51 to 459

459 - 51 # Subtracts 51 from 459

459 * 51 # Multiplies 459 by 51

459 / 51 # Divides 459 by 51

51 ^ 3   # Calculates 51 to the power of 3
-- i.e., the same as:

51 * 51 * 51
```

Remember, to solve each of these equations, you could 1) type it into the console then hit return, 2) copy and paste it into the console and press return, or 3) highlight the relevant portion of the R script and press <command>+<return> (or <ctrl>+<R> for a PC running Windows or Linux). Try executing the commands in each of the three ways. Note that you do not want to write an equal sign after the equation because R interprets “=” as an assignment operator, as described in part 8.

If it has been a while since you’ve done a lot of this kind of arithmetic, it might be good to review some other basics, such as the difference between the following two commands.

```
(459 + 51) / 3 # Adds 51 to 459 and then di-
vides the result by 3.

459 + (51 / 3) # Adds 459 to the result of
dividing 51 by 3.
```

Notice that when you run any of the commands above, the next line in the R console window is the result of performing the operation that you specified by the command. The symbols “+” and “*” and so on are called ***operators***, because they are what cue the operation.

Part 7. Functions

Addition also can be performed using the R function `sum()`. A ***function*** is a named command that can stand for any sequence of operations. That is, it is like a shortcut for more complicated mathematical functions or processes including operating the graphical device, that have been pre-programmed into R. A function will be followed by parentheses where you will specify further bits of information that R needs in order to perform the selected function. Each bit of information is an ***argument***. If there are several arguments, they are separated from each other by commas. For example, the `sum()` function adds its arguments together, so the following two commands return the same result:

```
459 + 51 + 327      # This adds together
459, 51, and 327.

sum(459, 51, 327)  # This also adds to-
gether 459, 51, and 327.
```

Part 8. The R assignment operator

Another very important special symbol is "=", the assignment operator. You can also write the assignment operator as "<-". This operator tells R to assign the value to its right to the thing on its left. In the simplest case, this is just like giving a name to the value. So for example:

```
x1 = 459 + 51 # Adds 51 to 459 and assigns
the result the name "x1".

x2 = sum(459, 51, 327) # Stores the sum of
these 3 numbers in "x2".

x3 = 7 - 2 # Takes 2 from 7 and stores the
result in "x3".

x4 = sum(7, -2) # Stores the sum of 7 and
-2 in "x4" (same as x3).
```

Once you have stored the result of an operation in this way, you can retrieve the computed value just by typing the "name" that you've given to the value. So, for example, if you type `x1` or `x2` or `x3` or `x4` in the R console window after running the above four commands, the next line on the R console window is the same value that you would have got by running the original command again. This is especially convenient if you want to store more complicated values, such as a vector of numbers instead of a single number.

Part 9. The R vector function

First of all, a ***vector*** is a single row of items. In R, you can specify that a set of values is a vector by typing them, separated by commas, as arguments to the `c()` function, like this:

```
c(459, 51, 327)
```

The `c()` part of this command is a function that tells R to "concatenate" the arguments, which means "to group these items together in order," which is a simple definition of a vector. Again, a function is like a shortcut for more complicated mathematical functions or processes. The name of the function will be followed by parentheses where you will specify further information that R needs in order to perform the selected function. In this case, the `()` parentheses enclose the items you want to be grouped together, and the items are separated by commas, to show where one item ends and the next begins. If you assign the vector a name, like this:

```
x5 = c(459, 51, 327)
```

you have a way of referring back to it, so that you don't have to keep typing the same numbers in over and over again. So, after you run the above command, the following two lines of code return the same value.

```
sum(459, 51, 327)

sum(x5)
```

The `length()` function lets you count the number of items in a vector. So the value that is returned by the following command is the number of items in the vector `x5` that you created earlier.

```
length(x5)
```

You can refer to a value at any position in a vector by following the vector with the position number enclosed in square brackets. So after you have defined `x5` as above, the following three commands all return the same result.

```
51
# Just type the number and R will echo it.

c(459, 51, 327)[2]
# Specify the second item in the vector.

x5[2]
# Specifies the second item in the vector
too.
```

The following three commands are also equivalent ways of adding 459 and 51.

```
459 + 51

x5[1] + x5[2]

sum(x5[1], x5[2])
```

This equivalence may seem a bit boring and trivial now, but wait until you see what this buys you when you're dealing with longer vectors or more complicated items.

Part 10. Numbers versus character strings

The values that we've been talking about so far have all been numbers, such as 459 or the result of summing up the numbers 459, 51, and 327 that were assigned to the vector `x5`. But R lets you also make observations about character strings,

such as the string of letters that constitute the spelled forms of the names *Cynthia* and *Elizabeth*. You distinguish character strings from number strings by enclosing them in quotation marks, like this:

```
name1 = "Robert"

name2 = "Jonathan"

name3 = "Cynthia"

name4 = "Elizabeth"
```

These four lines of code assign the character strings "Robert", "Jonathan", "Cynthia", and "Elizabeth" to the variables name1, name2, name3, and name4. You can check to see whether these are numbers or character strings using the is.numeric() and is.character() functions, like this:

```
is.numeric("Robert")

is.character("Robert")

is.numeric(name1)

is.character(name1)
```

Compare the results of the above four commands to the results of running the following:

```
is.numeric(6)

is.character(6)

is.numeric("6")

is.character("6")
```

You can use the assignment operator and the vector function with character strings, just as you can do with numbers. For example, try executing the following two commands. Then type the names of the variables (somenames and somenicknames) all by themselves.

```
somenames = c("Robert", "Jonathan",
  "Cynthia", "Elizabeth")

somenicknames = c("Rob", "Jon", "Cindy",
  "Lisa")
```

The first of these two lines makes a vector of the character strings that are the spelled forms of these four names and assigns it to the variable "somenames". The second line does a similar thing for the associated nicknames. If you have assigned each of the four names to variables, as shown above, then the first line is equivalent to the following:

```
somenames = c(name1, name2, name3, name4)
```

Also, once you have defined these variables, you can tell R to count the number of names or nicknames, like this:

```
length(somenames)

length(somenicknames)
```

You can also check to see if the vector contains numbers or character strings, like this:

```
is.numeric(somenames)

is.character(somenames)
```

If you have `x5` defined from Part 9, try applying these functions to that vector of numbers, too.

Part 11. Counting the number of characters

You can use the `nchar()` function to count the number of characters in a character string, like this:

```
nchar("Cynthia")

nchar("Cindy")
```

Of course, if you assigned the character string "`Cynthia`" to the variable `name3`, the first of these lines is the same as:

```
nchar(name3)
```

Also, once you have defined the two vectors of names and nicknames above, you can refer to the items by the their positions in the list, so the following two commands will give you the same result – specifically, the number of letters in the name *Cynthia*.

```
nchar("Cynthia")

nchar(somenames[3])
```

You can confirm that the result of using the `nchar()` function is a number by embedding it in the `is.numeric()` command, like this:

```
is.numeric(nchar("Cynthia"))

is.numeric(nchar("Cindy"))
```

Since the result of applying the `nchar()` function is a number, you can use it in arithmetic operations. So the following three commands will all return the same result, too – specifically, the difference in number of letters between the name *Cynthia* and the nickname *Cindy*:

```
7 - 5

nchar("Cynthia") - nchar("Cindy")

nchar(somenames[3]) - nchar(somenick
  names[3])
```

The `nchar()` function can also take a vector of character strings as its argument, in which case it returns a vector of numbers. Try typing the following commands to see this.

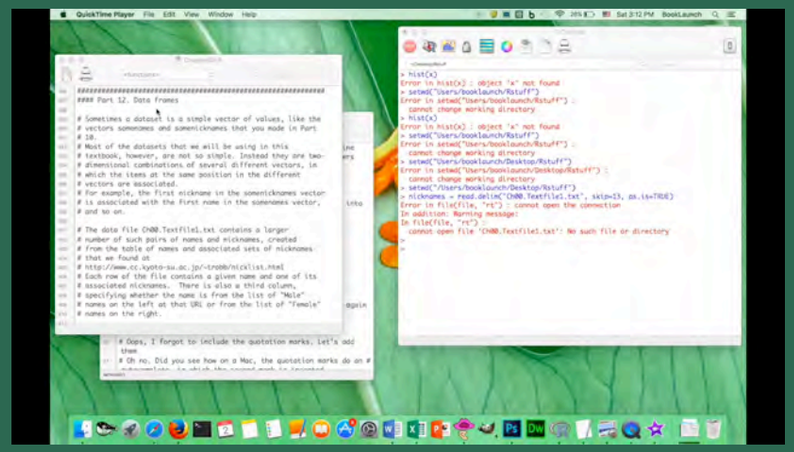
```
nchar(somenames)

nchar(somenicknames)
```

You can also combine commands in more complicated ways. For example, try typing the following command.

```
nchar(somenames) - nchar(somenicknames)
```

This should return a vector of four items, which are number values for the difference in length for each of the four pairs of full name and nickname.

Movie Preface.4
Using Part 12 of Chapter00.R to read in a data file file
external link: http://streaming.osu.edu/player/knowledgebank/?f=LING/LING_kbdirect_2051_Ch00_Movie4_m4v.html"



Part 12. Data frames

Sometimes a dataset is a simple vector of values, like the vector `somenames` or the vector `somenicknames` here. Most of the datasets that we will be using in this textbook, however, are not so simple. Instead, they are two-dimensional combinations of several different vectors, in which the items at the same position in the different vectors are associated. For example, the first nickname in `somenicknames` is associated with the first name in `somenames`, and so on.

The data file `Ch00.Textfile1.txt` (found on the textbook website, at http://hdl.handle.net/1811/77848, or individually at http://kb.osu.edu/dspace/bitstream/handle/1811/77848/Chapter00.Textfile1.txt) contains a larger number of such pairs of nicknames and names, created from the table of names and associated sets of nicknames that we found at http://www.cc.kyoto-su.ac.jp/~trobb/nicklist.html. Each row of the table in the file contains a given name (in the first column) and one of its associated nicknames (in the second column). The third column specifies whether the name is from the list of "`male`" names (on the left at that URL) or from the list of "`female`" names (on the right at that URL). There is also a fourth column, specifying the number of syllables that we judged the name to have, and a fifth column, specifying the number of syllables that we judged the nickname to have. That is, there are exactly five columns, each representing a vector. The first and second columns are the vectors of character string values for the spelled forms of a full name and the associated nickname. The third column is the vector specifying the gender of the name-nickname pair. The fourth and fifth columns are the two vectors of numbers specifying the syllable counts for the spoken forms of the name and the associated nickname.

In R, such a two dimensional combination is called a ***data-frame***. If you have set the working directory to the directory

(i.e. the folder) where you stored the `Ch00.Textfile1.txt` file, you can use the function `read.delim()` to read the data into this kind of two-dimensional tabular information, like this:

```
nicknames = read.delim("Ch00.Textfile.txt",
  skip=13, as.is=TRUE)
```

This reads in the content of the file and assigns it to the variable `nicknames`. The first argument is the name of the file that you want to read in. Note that the file name is a character string and needs to be enclosed in quotation marks. The second argument (i.e., the part `skip=13`) tells R to skip the first 13 rows of the file. (These 13 rows are an extended comment identifying what the file is.) The last argument (i.e., the part `as.is=TRUE`) tells R to read in the data "as is" without converting to a different format for vectors of character strings. That is, here we want the first column to be a simple vector of character strings and we also want this to be true for the second column. (The default treatment is more appropriate for vectors such as the "`gender`" vector, where each value is just the name of one of two or a small handful of category types. We will explain this in Chapter 1, where we will use a version of the `read.delim()` function that doesn't tell R to keep things "as.is" as the data are read in.)

You can look at the content of the dataframe in various ways. For example, the following command returns the first six rows of the dataframe.

```
head(nicknames)
```

And the following command lets you open the dataframe in a text editor, in a format that will be familiar if you've used programs such as Excel.

```
edit(nicknames)
```

Part 13. Retrieving values at specific positions in a data frame

Finally, you can look at values at particular positions in the dataframe by specifying their row numbers or their column numbers or both. You specify these positions by enclosing the numbers in square brackets, just as you did for the positions in a vector. But since a dataframe has two dimensions, now there should be two reference numbers, separated by a comma, for the row position and the column position. For example, the following command returns the value that is in the third row of the dataframe under the second column (i.e., the third value in the vector of nicknames that makes up the second column):

```
nicknames[3,2]
```

You can leave out the column number to return all of the values in a particular row. For example, the following command returns the values from all five of the columns for the third row of the dataframe:

```
nicknames[3,]
```

Since any row in the dataframe is a vector, with one value from each of the columns of the dataframe, you can ask R to tell you how many columns there are using the `length()` function, like this:

```
length(nicknames[3,])
```

Similarly, you can leave out the row number to return all of the values in a particular column. For example, the following command returns all of the values from the second column (the nicknames column) for all rows of the dataframe:

```
nicknames[,2]
```

Since this column is a vector, too, you can ask R to tell you the total number of rows in the dataframe, like this:

```
length(nicknames[,2])
```

There is also a function that returns a vector with the number of rows followed by the number of columns. That's the `dim()` function, which takes the name of the dataframe as its argument, like this:

```
dim(nicknames)
```

Also, now that you know how to return a row vector or a column vector, you can use the `nchar()` function to tell R to count the number of letters in the spelled form of each of the nicknames, like this:

```
nchar(nicknames[,2])
```

Try running that command, and then editing it to tell R to count the number of letters in the spelled form of each of the full names in the first column.

Once you've figured out how to do that, you can also then calculate the difference in number of letters using the same command that you used for the shorter vectors that you created in Part 10, like this:

```
nchar(nicknames[,2]) - nchar(nicknames[,1])
```

This gives you the difference in length for all 91 pairs of full names and nicknames.

0.9. Exercises

Following the R code, we will present you with a variety of exercises to help reinforce the topics covered in the chapter. To illustrate, here we give you three exercises that involve comparing the lengths of nicknames and their associated full names.

1. Download the file `Ch00.Textfile1.txt` from the web site for this course (or recreate the file using the information at http://www.cc.kyoto-su.ac.jp/~trobb/nicklist.html). Open R and set the working directory to where you put the `Ch00.Textfile.txt` file. Use the R commands in Part 13 of section 0.8 (the R code section) to read in the file and as-

sign it to the variable `nicknames`, and then to assign names to the columns.

Then use the `nchar()` function, as described in Part 11 of Section 0.8, to make two vectors, one for the length of the nickname in number of letters and one for the length of the name in number of letters.

Use these two vectors of numbers to address the following question: In this sample of nicknames and associated names, how often is the spelled form of the nickname shorter than the spelled form of the full name that it is "short for"?

2. Now adapt the R code in Part 13 of Section 0.8 to make vectors for the values in the last two columns of the data frame and use these two vectors to address the following question: In this sample of nicknames and associated names, how often does the spoken form of the nickname have fewer syllables than the spoken form of the full name that it is "short for"?

3. Download the file `Ch00.Textfile2.txt` from the course website, or directly at http://kb.osu.edu/dspace/bitstream/handle/1811/77848/Ch00.Textfile2.txt. Or point your browser to http://deron.meranda.us/data/nicknames.txt and download that file, and rename it `Ch00.Textfile2.txt`. Read the file into a variable called `x`. You will need to specify a different number of rows to skip, so the command is `x=read.delim("Ch00.Textfile2.txt",skip=12)`. Then adapt the code that you used in exercise 1 to address the same question for this sample.

0.10. References

The very last numbered section of each chapter will list the references from each chapter and the sources for any data sets used. For example, here are the references to publications which we cited in this chapter.

George Cobb (1992). Teaching statistics: More data, less lecturing. *UME Trends*, 3(4): 2–3.

Robert W. Hayden (2000). Advice to mathematics teachers on evaluating introductory statistics textbooks. In Thomas L. Moore, editor, *Teaching statistics: Resources for undergraduate instructors*. Washington, DC: Mathematical Association of America.

Peter Ladefoged (2005). *Vowels and Consonants: An introduction to the sounds of languages*. 2nd Edition. Malden, MA: Blackwell Publishing.Sally A. Shaywitz (1998). Dyslexia. *The New England Journal of Medicine*, 338(5): 307-312.

F. Yates (1935). Complex experiments. *Supplement to the Journal of the Royal Statistical Society*, 2: 181-247.

The `Ch00.Textfile1.txt` data file that you can read in to do exercises 1 and 2 was created from the sample of names

and associated nicknames at the following URL, which we accessed to make this file on September 29, 2011:

http://www.cc.kyoto-su.ac.jp/~trobb/nicklist.html

The `Ch00.Textfile2.txt` data file that you can use to do exercise 3 is another, even longer list that you can download from Deron Meranda's "Deron's Data Pages: Collections of interesting data, facts, and wordlists" at the following URL, which we accessed on October 9, 2011:

http://deron.meranda.us/data/nicknames.txt

The header section of this data file is a good model for how to explain the data and cite your sources when you build "mash-up" data sets of your own from other sources.

Peter Ladefoged's website to accompany Vowels and Consonants is still being hosted at http://www.phonetics.ucla.edu/course/chapter1/chapter1.html by the University of California at Los Angeles. If you browse the table of contents, you can find many useful interactive elements that can help you appreciate some of the phonetic concepts.