STATISTICS

(T H E   E A S I E R   W A Y)

WITH R

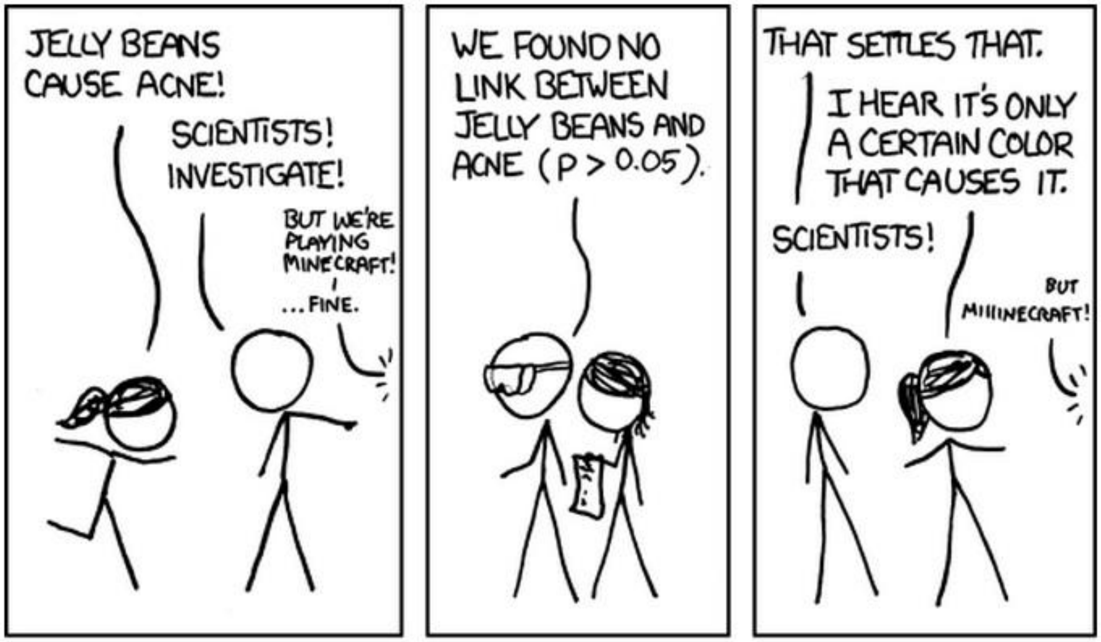
Nicole M. Radziwill

ISBN-13: 978-0-692-33942-8
ISBN-10: 0692339426

Cover Design: Morgan C. Benton
Indexing: Athena Benton
Publisher: Lapis Lucera, San Francisco, California.

For information on distribution, bulk sales, or classroom adoptions please contact the author directly via email (nicole.radziwill@gmail.com) with the subject line "251 BULK ORDER" or on Twitter at @nicoleradziwill

For a free PDF eBook, please email a copy of your receipt (and preferably, a picture of you reading your book in a beautiful location!) to the author. You will receive the PDF eBook by email AND your picture might even show up on the http://qualityandinnovation.com blog.


JELLY BEANS CAUSE ACNE!
SCIENTISTS! INVESTIGATE!
BUT WE'RE PLAYING MINECRAFT!
... FINE.
WE FOUND NO LINK BETWEEN JELLY BEANS AND ACNE (P > 0.05).
THAT SETTLES THAT.
I HEAR IT'S ONLY A CERTAIN COLOR THAT CAUSES IT.
SCIENTISTS!
BUT MIIIINECRAFT!

Part 1 of "Significant" - an XKCD cartoon by Randal Munroe (https://xkcd.com/882/)

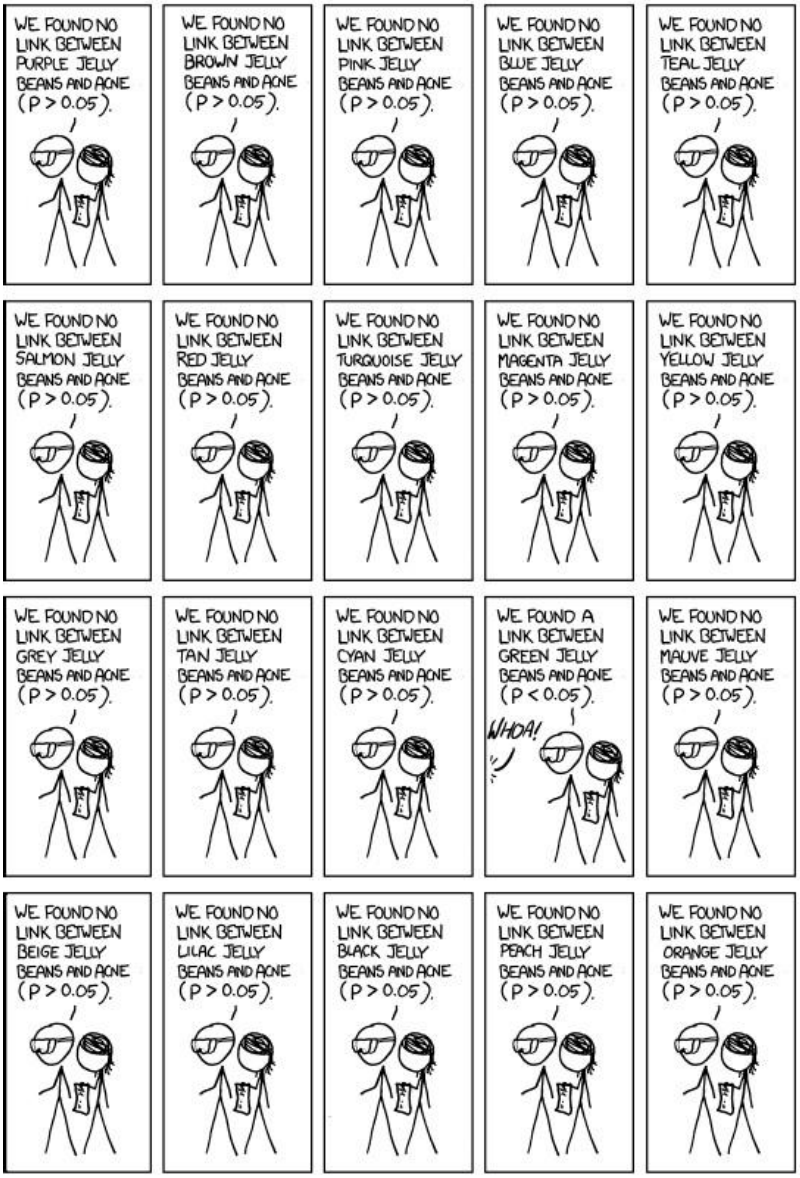
About the Author

Nicole M. Radziwill

As of Fall 2015, Nicole is an Associate Professor in the Department of Integrated Science and Technology (ISAT) at James Madison University (JMU) in Harrisonburg, Virginia, where she has worked since 2009. Prior to 2009, she spent nearly a decade hanging out with brilliant astronomers and engineers at the National Radio Astronomy Observatory (NRAO) working on software and systems to make giant radio telescopes work. Her teaching interests include quality consciousness, quality informatics, innovation, process improvement, predictive analytics, intelligent systems, industrial simulation, technology management, and applied statistics. She has been active in the American Society for Quality (ASQ) since the late 1990's, and in addition to serving as one of ASQ's official "Influential Voices" bloggers at http://qualityandinnovation.com, she was recognized by Quality Progress (ASQ's flagship publication) as one of the 40 New Voices of Quality in 2011.

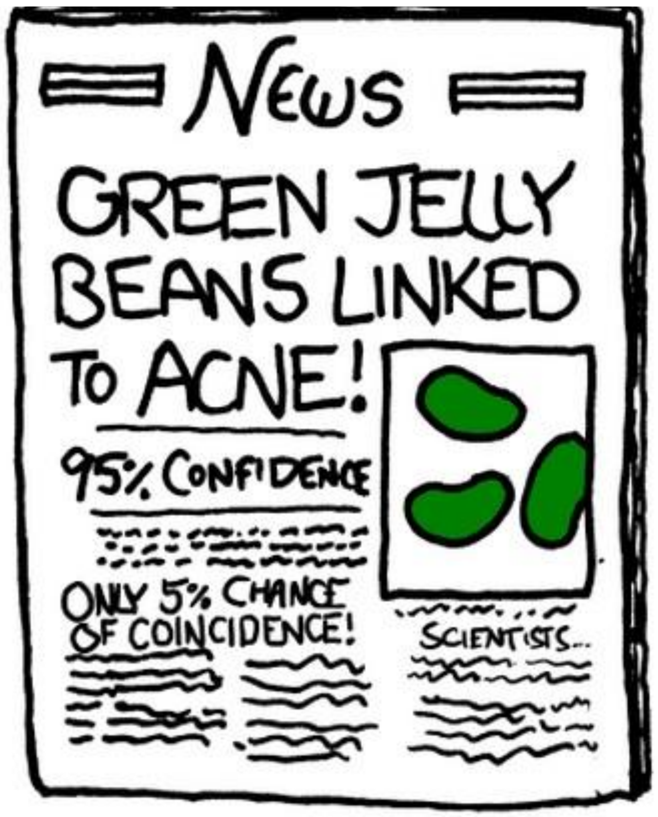
Nicole is certified by ASQ as Six Sigma Black Belt (CSSBB) #11952 and Manager of Quality and Organizational Excellence (CMQ/OE) #9583. She was Chair of the ASQ Software Division from 2009 to 2011, and served as a national examiner for the Malcolm Baldrige National Quality Award (MBNQA) in 2009 and 2010 appointed by the National Institute for Standards and Technology (NIST). She has a PhD in Technology Management and Quality Systems from Indiana State University, an MBA from Regis University in Denver, and a BS in Meteorology from Penn State.

Her research uses data science to explore new ways to think about quality systems and innovation, with a focus on emergent environments for living and learning, leveraging alternative economies and gift cultures such as Burning Man.


WE FOUND NO LINK BETWEEN PURPLE JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN BROWN JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN PINK JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN BLUE JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN TEAL JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN SALMON JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN RED JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN TURQUOISE JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN MAGENTA JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN YELLOW JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN GREY JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN TAN JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN CYAN JELLY BEANS AND ACNE (P > 0.05).
WE FOUND A LINK BETWEEN GREEN JELLY BEANS AND ACNE (P < 0.05).
WHOA!
WE FOUND NO LINK BETWEEN MAUVE JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN BEIGE JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN LILAC JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN BLACK JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN PEACH JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN ORANGE JELLY BEANS AND ACNE (P > 0.05).

Part 2 of "Significant" - an XKCD cartoon by Randal Munroe (https://xkcd.com/882/)

Brief Contents


NEWS
GREEN JELLY BEANS LINKED TO ACNE!
95% CONFIDENCE
ONLY 5% CHANCE OF COINCIDENCE!
SCIENTISTS...

Part 3 of "Significant" - an XKCD cartoon by Randal Munroe (https://xkcd.com/882/)

Table of Contents

Preface

I've become so tired of reading statistics textbooks that *feel like textbooks*. Why can't someone just write a book that makes me feel like I have a *real live person* sitting next to me, who *cares* about me, who *wants* me to understand this stuff? Why can't someone document some examples in R (my favorite statistical software package) that actually include *all of the steps I need to follow* instead of leaving out one or two critical steps that leave me tearfully troubleshooting for hours? Why can't someone write a cookbook that provides *just enough statistical theory and formulas* so I can understand how the analytical solutions match up with the solutions provided by the statistical software?

That's the kind of book I wanted to recommend for my students. But after years of searching, and a couple more years of trying out books that didn't quite fit, I came to realize that the ideal book I was looking for just didn't exist: if I wanted my students to have a friendly, accessible, affordable, non-threatening textbook... I would have to write it for them. Finally, I started hitting LaTeX and the word processor. *I'll do it*, I committed. It took about three years to get to this point, but late is better than never (I hope).

SO... the purpose of this book is to help you analyze real data – quickly and easily, no fuss no muss – using the R statistical software. Along the way, you'll learn some of the most important fundamentals of applied statistics, which will help you think more critically about the story your data is telling. **My style is to tell you only what I think you need to know to quickly become productive.** I'll provide you with some background, some examples, and an explanation of what each of the commands in R does (and the options you can provide to those commands). I want to give you just enough theory so you know exactly what's going on under the surface, and just enough practice so that you know how to gain insights from your own data.

As a result, this book is NOT intended to be a substitute for a full-length text or course in statistics! You won't learn how to solve homeworky-style problems. However, I've often found that it's easier to get interested in how and why these methods work - and under what conditions - after you've had some success using them, and maybe even working with your own data.

Most importantly, each chapter has been written so that you don't need to be a math or statistics or programming ninja to be able to complete the exercises! Rather than trying to impress you with my slick coding skills in R, **I have purposefully chosen less elegant but more instructive coding strategies**, and I've attempted to show you ALL the lines of code for EVERYTHING that is produced in this book. **I have also been repetitive, on PURPOSE**, so you don't have to remember every single detail from previous chapters to get the job done in a later chapter! I have piloted all the chapters in one or more of my classes, and students in majors ranging from anthropology to business to science to technology have completed the work and provided the feedback that's been used to clarify each chapter.

I wrote this book as the primary text for the undergraduate sophomores and juniors in my introductory applied statistics courses. I was strongly motivated to do this for two reasons: 1) as silent activism against textbook publishers who charge $200+ a pop for very beautiful, glossy statistics textbooks that all of my students are required to purchase, *yet none of my students actually read*, and 2) to give my students just the essential information I think they need to become confident and capable data analysts using the most essential statistics. (At this point, the statistics textbooks tend to become a lot more interesting to *everyone*.)

To be clear, I have nothing against traditional textbooks themselves – or the authors who have labored to create clear, engaging texts with tons of pretty pictures and armies of practice problems. Actually, I admire the authors who spent so much time constructing their vision of a clear, cogent presentation of a subject. (It's not hard for me to admire and appreciate these contributions, since I myself have actually been a part of these teams of authors who have written reasonably awesome, yet expensive, textbooks -- which publishers are now making unconscionable bank with.) I just think traditional textbooks are kind of outmoded and passé. I can find lots of pretty pictures and examples on the web, and I'm not going to flatter myself by thinking that any more than one or two students work practice problems beyond what I cover in class.

I thought it was a sin that my students should pay $200 for a paperweight with a lifetime of just a semester and a resale value of less than 50% of the initial investment. I decided the book that I would create would not be heavy with beautiful and elegant typesetting, or stock photos of cheerful people working on statistical problems, or discussions that took the standard tone of "I am the great textbook, here to confer to you my power and wisdom." Textbooks never admit that some concepts are difficult, or *why* they're difficult. Textbooks aren't conversational, and don't really provide moral support. Textbooks have to be politically correct, which can preclude talking about really intense examples that students usually remember... like whether smoking too much pot is related to lower test scores.

I figured that a textbook that cost only 15-20% of the "standard", and that actually had some useful recipes that students could leverage for semester projects in other classes, might have a little more lasting value and maybe even find a permanent home on my students' bookshelves. That's my dream, at least.

Who This Book Is For

Although originally designed for *my* undergraduate statistics students, I think this book will be useful to you if you are *any* of the following types of people:

- Undergraduate college students in the first semester of applied statistics who are looking for a textbook that does not spend too much time on fluffy explanations of concepts, but rather, presents information in a more direct and less mathematical-ese way

- Professors who want a relaxed, informal book for their intro stats classes that won't cost students $150-$250

- Graduate students who need to use statistics to plan and complete a thesis or dissertation, but don't have that much background in it, and need help NOW

- Early to mid-career data scientists who don't have a PhD in statistics and don't want one, but do want to get more familiar with R, with foundational concepts in statistics, and/or learning how to tell better and more compelling data-driven stories

- Smart, business-savvy people who want to do more data analysis and business analytics, but don't know where to start and don't want to invest hundreds or thousands of dollars on statistical software!

- People who are studying for their Six Sigma Black Belt exam. As a Black Belt myself, a lot of the understanding I've gained has come from the discipline of process improvement, so my treatment of topics naturally tends to reflect what's important for that exam.

- Middle school or high school students who want to collect some original data and do cool science fair projects.

- High school students who are taking regular or AP Statistics classes, and need a little extra help or information (or maybe you want to supplement the in-class instruction you're getting with some understanding of how to analyze real data).

I *do not* assume that you are a programmer. I also do not assume that you are super smart with either computers or statistics, only that you have the motivation to get some data analysis done with R. More about R in the next section. (I won't assume you know things like *how important it is to coerce an object to another form* if a certain algorithm won't work on the object you have. That would be cruel.)

I *do* assume that you already have R installed on your machine, and that you are impatient and just want to figure out how to do some useful stuff so you can start impressing your boss, your teachers, or yourself.

I *do* assume that you have a positive attitude, and that you'd like to learn some statistics and data analysis techniques that you can start using immediately. If you don't have a positive attitude, you should probably not be reading this book. In fact, it's likely that you shouldn't be doing statistics at all. You would be better served to go off and find some topic or some activity that *does* spin up a positive attitude in you. Why? Because you will have more fun. I wouldn't want you to do stats if it wasn't a little fun.

This Book Uses R

All of the examples in this book use the **R statistical software**. You need to download and install the software onto your own machine to be able to use it. Go to the R Project web site at http://www.r-project.org and select the link to "Download R". It will ask you to pick a "CRAN Mirror" so find a site that's geographically close to where you're sitting when you want to install R. For example, I live in Virginia so I might choose a Maryland or Ohio site rather than one in Argentina or Belgium. Data has to travel over geographic distances too, so I want all those mysterious bits and bytes to have to travel the shortest possible distance between where they live and MY laptop. That way, they will arrive as quickly as possible.

This book does not provide help installing R. Why? Simply because every time I have attempted to install R it has been really, really easy, and I don't consider myself an expert at installing anything on my computer. If you can't figure it out yourself, ask a friend (or if you're older, a friend's computer-savvy kid).

Or, you can try to decipher what's in the official R Installation and Administration manual, which can be found at http://cran.r-project.org/doc/manuals/R-admin.html.

How This Book Is Organized

It seems like every non-fiction book has a section in the front that talks about how the book is organized, so I thought I'd be a conformist and do the same. I mean, sure, you could *look* through the rest of the book to see how it's organized, but why do that when I can describe it right here so you don't have to flip through the pages?

- Section 1 talks about **basic concepts** (like variable types, and loading data)
- Section 2 explains how to create **charts, graphs, and plots** that can help you tell the story of your data through visualization
- Section 3 provides **the framework for how to do a research project** using basic applied statistics, including fundamental aspects of theory, and also shows how to structure the story of your data and your results (featuring "Dr. R's 12 Steps")

- Section 4 summarizes **how to create confidence intervals** for several common scenarios, and recommends best approaches to handle the most challenging cases
- Section 5 contains **step-by-step recipes for conducting statistical inference tests,** drawing conclusions, and interpreting your results
- Section 6 contains **recipes for regression** (which is just a fancy word for *finding relationships between variables*), and
- The Appendices provide some helpful reference information.

Inside each section, there's lots of useful stuff, and lots of me rambling about it.

Conventions Used In This Book

Joy in Repetition

Although I have aimed to keep the text as simple and concise as possible, you may notice that there is substantial repetition between sections, especially throughout **Section 5: Statistical Inference**. That’s because I also plan to use this text as a reference for myself. When I am doing an inference test, or even a confidence interval, I don't want to have to flip back and forth between chapters to do things like *remember what all those assumptions are that I need to check.* Or how to load the data. When I am doing a Pareto chart, I want to quickly and easily reference all of the most important ways to change colors, line styles, and common options. When I create pie charts, I want to be able to easily and quickly reference all the most important options to the `pie` command. I don’t want to have to flip back and forth between sections just because something was covered earlier in the book. I don't want to pretend like I'm going to remember even the simplest syntaxes, because that's not easy (and gets less and less easy as your age and number of children increase).

My Fonts

Most of the text is written in Calibri 11-point font because I think it's pretty. Inline R code is in 10-point Courier New, whereas chunks of code are 9-point Courier New. Why all the variability? It's totally arbitrary. I just liked it that way, so I did it. Sometimes it looks good, and sometimes it looks less than good. I'm good with that. Sometimes I violate every design principle that exists by using 9-point font next to 8-point font with some 7-point font thrown in below it. But I have a good reason: I want you to be able to see all the output from R without it looking really horrible. You may think this is design heresy, but I call it practical and page-saving.

My R Code

Code that you can type directly into R is written in 9-point Courier New, and indented half an inch from the left margin, like this:

```
defect.counts <- c(12,29,18,3,34,4)

names(defect.counts) <- c("Weather","Overslept", "Alarm Failure",
"Time Change","Traffic","Other")

df.defects <- data.frame(defect.counts)
```

The code above *creates a vector of numbers* in the first line, *establishes names* for what each of those numbers means in the next two lines, and *creates a special object called a data frame* in the last line. I had to use a smaller font size for code that you can type into R so that most of my text and output would show up looking decent on the printed page. Design purists, I give you my apologies up front.

Code that I typed into R and the output that code produced is also recorded in 9-point Courier New*, but is not indented*. **This is important because it always has a leading caret** (that's the “>” at the beginning of each line). This caret is the R prompt, which you will see if you are using the R Console that comes with non-enterprise installations of the software. **DO NOT TRY TO TYPE THE CARET AS PART OF YOUR CODE OR YOU WILL GET ERROR MESSAGES.**

Here is an example:

```
> df.defects
              defect.counts
Weather                  12
Overslept                29
Alarm Failure            18
Time Change               3
Traffic                  34
Other                     4
```

This means that I typed in the command `df.defects` to see the contents of the data frame that I named "`df.defects`" in R. The rest is what R responded back to me. If I typed in "`> df.defects`" I would get an error message that looked like this:

```
      Error: unexpected '>' in ">"
```

(And for experienced R programmers: YES, I know that some of the R code in here is not optimized, and I know some of my simulations use loops instead of `apply`, and how could I ever possibly do something like that because that's not the best thing to do. You're right. I don't claim to provide elegant code in this book... just readable and/or explained code.)

My R Functions

I also keep many of the utility functions I use frequently on GitHub. You can scan a list of them at `https://github.com/NicoleRadziwill/R-Functions`. Even though the text of these functions is included in this book, you might just want to load the function into your R console without having to cut and paste. You can use the `source` command in R to load functions that are stored in separate files, even if those files are accessed over the web. However, if your function is on a secure site (whose address starts with https) the process is a little more complicated. Here's what you need to do to be able to load my functions into your R console from GitHub:

1. Check to see which directory is the default for your R installation using `getwd()`

2. Create a file called `sourceHttps.R` in THAT directory, and place THIS code in there, which comes from a guy named Tony Breyal and was originally posted on his blog at http://tonybreyal.wordpress.com/2011/11/24/source_https-sourcing-an-r-script-from-github/:

```
source_https <- function(u, unlink.tmp.certs = FALSE) {
  require(RCurl)
  if(!file.exists("cacert.pem"))
  download.file(url="http://curl.haxx.se/ca/cacert.pem", destfile =
       "cacert.pem")
  script <- getURL(u, followlocation = TRUE, cainfo = "cacert.pem")
  if(unlink.tmp.certs) unlink("cacert.pem")
       eval(parse(text = script), envir= .GlobalEnv)
}
```

3. Type `source("sourceHttps.R")` on your R command line, which will bring in that function for you to use. (If it doesn't work, check to make sure that you're using the right filename. A common error is to accidentally save the file as `sourceHttps.R.txt`, and if you do that, you'll have to type *that* in your `source` command.)

4. Check to make sure the function loaded properly by typing `source_https` - if all is OK, then you'll see the code for the function displayed on the screen.

5. Now check to make sure that you can actually use that function to bring in new functions from GitHub. First, do this:

```
source_https("https://raw.githubusercontent.com/NicoleRadziwill
/R-Functions/master/shadenorm.R")
```

6. If Step 5 didn't work, type `library(RCurl)` and then try the code in Step 5 again.

7. Finally, type `shadenorm` on the R command line, and press enter. If everything is set up properly, you'll see the code for the `shadenorm` function displayed. Every time you launch a new R session, you'll have to load `source_https` first using the method above before you can load functions directly from GitHub.

My Data

I keep all of the data I used in the examples in this book on GitHub. You can see a list of it all online at `https://github.com/NicoleRadziwill/Data-for-R-Examples`. Here is an example of how to load my data directly into R from the GitHub repository. To get a different file, be sure to change the part that says "`mnm-clean.csv`". You should be able to do this to get any of the data in any chapters! Replace `read.csv` with `read.table` if you are trying to upload data in a text (`*.txt`) file:

```
library(RCurl)
url <- "https://raw.githubusercontent.com/NicoleRadziwill/Data-"
url <- paste(url,"for-R-Examples/master/mnm-clean.csv",sep="")
x <- getURL(url,ssl.verifypeer=FALSE)
mnms <- read.csv(text = x)
```

How To Contact Me

If I have the time, I am more than happy to address comments and questions about this book, its examples, or problems that you are having working through your own data. I am more likely to have the time to respond to you in between semesters (which means most of December, and May through early August). If you don't hear back from me, don't despair: it just means that your message fell through the cracks, and I'm often either busy or distracted (or just plain scatterbrained), so I might have missed it. Try again. If I don't have the time to answer or explore your query, I'll let you know.

I also invite you to follow me on Twitter - my ID is **@nicoleradziwill** - but be advised, you will also be getting a lot of tweet spam about severe weather, tornadoes, solar flares, coronal mass ejections, and Burning Man. (You may like this.) Also, if there is ever an interesting planetary configuration or an asteroid that is about to impact Earth, you will be among the first to know.

Disclaimer

The purpose of this book is *not* to be mathematically elegant or comprehensive, but to give you enough of what you need to know to be productive - quickly - without leaving you statistically naive. I've included links to longer, more extensive proofs in many places if you want to know more about the mathiness underlying everything in this book. I am not a professional statistician or mathematical purist, but I am a realist who analyzes data almost daily, and I want you to be able to analyze your data too.

Acknowledgements

I would be remiss if I did not mention the people who influenced me to become a person who really likes to use, and teach, statistics. First of all, I am eternally indebted to all the students that test drove my explanations of concepts, and my early drafts of this book. They helped me recognize what can be confusing to learners, and helped me illuminate the things that were actually confusing to *me*. I'm also thankful to my teaching assistants Meghan Mooney, Aimee Cunningham, Cassidy Moellers, Paul Rose, and Chris Miller who helped me in the initial stages of piloting the new approaches and creating new course materials.

Dan Teague of the North Carolina School of Science and Math in Durham, NC was my first statistics teacher and deserves a lot of credit. He was brave enough to explain to the group of 14- to 17- year olds in my class nearly two decades ago that statistics is just a game of exploring variation in n-dimensional vector space. I was smitten. Bob Ryan of the 1990's statistics department at Penn State deserves some credit for *keeping* me hooked. He was one of the executives of Minitab, a company he founded with his wife at the time, totally down to earth, and told cheesy (yet "good" to me!) jokes in class. The jokes made me want to keep doing applied statistics. (Yes, I can be that easily influenced.) Plus, he made all of us buy Lyman Ott's applied statistics book, which I still have, and still use. Mike Hayden of Indiana State University also deserves a mention and much gratitude. His recipe for executing and documenting a statistical hypothesis test (which he calls "Hayden's 16 Steps") influenced me greatly, and certainly snuck into my own recipes and approach, even forming the basis for "Dr. R's 12 Steps" which are a theme within this book. I am also eternally grateful to him for helping me *really understand* what setting your alpha level is all about… balancing cost and risk. And ethical considerations... but I added that last part.

I am also thankful for H. M. Schey's book *Div, Grad, Curl, and All That: An Informal Text on Vector Calculus*. Had I not possessed that book as an undergrad, I never would have passed my vector calculus course, which was essential for my major (meteorology). The book you're reading now is in many ways a tribute to this author's style.. especially the subtitle, "an informal text on applied statistics."

I also appreciate the support and feedback from the Department of Integrated Science and Technology (ISAT) at James Madison University (JMU) in Harrisonburg, Virginia, especially my colleagues Anne Henriksen and Morgan C. Benton. Anne and I have had many discussions about how to teach introductory statistics in an integrated way (that embraces thinking about social context) for the past few years, and she helped me identify some really important typos in some of the early versions of my chapters. My partner and co-conspirator Morgan provided extensive technical, moral, and emotional support throughout the entire process of preparing this book, including extremely geeky conversations exploring things like the possible relationship between P-Values and the observer effect in quantum physics... and pointers to all of the XKCD cartoons. And Netflix. And many hours of fantastically blissful and enjoyable research, for which I have much gratitude.

SECTION 1: BASIC CONCEPTS

- Categorical and Quantitative Data
- Formatting Your Data for R
- Measures of Central Tendency and Variability
- Probability and Probability Distributions
- Using the Normal Model

1.1 Intro to R: Installation, Working Directories, & Packages

Objective

The purpose of this exercise is to get R installed on your computer, set your working directory, check out some basic concepts with a quick "Hello World!" exercise, and practice loading in a new package. By the end of this exercise, you'll have gained a rudimentary understanding of R and its potential.

Background

The R statistical software is a tool for analyzing and visualizing data. It was designed in 1993 as a prototype to see how a statistical computing platform might be built. Today, R has become more than just a simple testbed. It's a widely used, collaborative, open source environment for statistical modeling and data analysis.

The R programming language is based on the S language (a statistical language based around functions and the concept of objects). R is an *interpreted* language, which means that it interprets plain text and numerical values input by the user. For example, if you type in 2+2 at the caret prompt in the R console, the interactive R session will reply with 4. It's a calculator! The fact that R is an interpreted language makes coding at the command prompt very approachable; whatever you type in, you can see the results immediately when you press the `Enter` key. R handles all sorts of things, from classical statistical tests to data cleansing and advanced analysis. If you have some data and want to do something with it, R more than likely has a solution for you.

Part 1: Installing R

To start with R, first we have to download it from the web.

1. Go to the website http://www.r-project.org/. This is the home base for everything R, including manuals, FAQs, screenshots, and code repositories.

2. On the left-hand side of your screen, click on "CRAN". This will take you to a page titled "CRAN Mirrors".

3. On the page "CRAN Mirrors", scroll down until you see the heading “USA”. Underneath this heading you'll find lots of links. Each link allows you to download R, so no matter which one you press, you'll go to the R download page. Click on one that's close to you geographically: this will take you to a page called The Comprehensive R Archive Network (that's what CRAN stands for).

4. On that page, click the link that matches your operating system. This will take you to a page titled "R for (your operating system)".

5. There, do one of the following:
    - For Mac OS: click on "Download R for Mac OS". After that, select the link that ends in the file format ".pkg".
        - ❖ Note: If your browser informs you that this file may harm your computer, select "keep".
    - For Windows: click on "Download R for Windows". After that, click on the link titled "base". Then, click "Download R for Windows".
        - ❖ Note: If your browser informs you that this file may harm your computer, select "keep".

6. Once your package/file has downloaded, navigate to your operating system’s Downloads folder. From here, double click on the file/package and follow the on-screen instructions to install R. A shortcut for R should be created on your desktop if you are running Windows, or your applications folder if you are running Mac OS.

Part 2: Getting and Setting Your Working Directory

Now that you have R installed on your computer, you can set your *working directory.* This is the place R will look to find data, functions, and other resources on your computer. You can change your working directory to point R to *any location* on your local machine, depending on which local directory contains the data, functions, or other files you want to import into R.

1. Open up R by clicking on the Icon that was created for you in Part 1. An R coding environment should pop up. You'll see some introductory text and a caret (">") which is R's way of telling you it's ready to do your bidding.

2. The first thing we want to do is find out *where the working directory is already set*. You want to set it to a place that's convenient for you. To get your current directory, type `getwd()` and press Enter.

3. Your working directory will display below where you typed `getwd()`. I recommend creating a folder titled `R` in the default working directory (you'll do this outside of R, either in the "Finder" application on Macs, or through the "Computer" option that you see when you click the Windows start button). Navigate to it by typing

   ```
   setwd("/your/directory/name/goes/here/R")
   ```

   If you wish to change your working directory entirely, find a directory that you like and use `setwd` to specify the path to that directory.

4. If all is successful with the step above and you type `getwd()` once again, you should see the directory you just specified. Your working directory has now been set!

5. Double check by typing `dir()` -- after which you should see the contents of the directory you'd like R to point to.

Part 3: A Simple Hello World! Exercise

Now that your working directory is set, let’s do a simple "Hello World!" Exercise to become more familiar with R syntax.

1. In your R terminal, type the following and press `Enter`:

   ```
   x <- ("Hello")
   ```

2. Type `x` and you should see `[1] "Hello"`. In the step above, we assigned the word `Hello` to a variable `x` by using the assignment operator `<-` which looks like a left-pointing arrow. Note that any *string*, which is a series of characters containing non-numeric values, must be contained in quotes.

3. Once again in your terminal, type the following and press `Enter`:

   ```
   y <- ("World!")
   ```

4. Type `y` and you should see `[1] "World!"`. This time, we assigned the word `World!` to a variable y.

5. Now let's combine these two words and store them to a new variable that we'll call `z`. Type the following and press enter:

   ```
   z <- paste(x,y)
   ```

6. You'll see `[1] "Hello World!"` after typing `z` and `Enter`. The `paste` command combined the contents of the variables `x` and `y` that you defined previously.

Part 4: Installing Packages

One of the best things about R is that it is a highly versatile environment. It allows you to download packages to do all sorts of stuff like create awesome graphs, tables, and maps, and it can also help you mine and analyze data. Bringing new packages into your R environment is something you will do often. This is a three-step process. First, you have to figure out which R package will provide the functionality that you need. For example, if you want to use R to acquire data from Facebook, you might type "package to download

Facebook data in R" into Google. A search might produce many R packages that provide similar functionality, so you'll have to install the one that looks best, try it out to see if it's what you're looking for. If not, search for a different package and start over.

Let's say our search tells us that the `Rfacebook` package might provide us with what we need. The second step in the three-step process is to install this package (that is, download the code to our machine) from a CRAN mirror (one of the many repositories around the world that provide us with an up-to-date collection of all of the publicly downloadable R packages).

1. While in your R console, navigate to a dropdown menu titled `Packages` (Windows) or `Packages & Data` (Mac). Click on `Install Package(s)…` (Windows) or `Package Installer` (Mac). Alternatively, just type this on the R command line, replacing `package_name` with the actual name of the package you want to retrieve (being very careful to get the capitalization and spacing right):

   ```
   install.packages("package_name")
   ```

2. A window should appear that says something along the lines of "Please select a country". Select a location that's *geographically close* to you. That way, the bits and bytes won't have to flow all the way around the globe to get to your machine, and the installation process will probably be quicker. (Since I live in Virginia, I typically choose the Ohio or Pennsylvania sites.)

3. After selecting your country, you should see a long list of libraries. For Mac users, you may need to click on `Get List` to see the list (make sure `CRAN (binaries)` is selected from the dropdown menu above before clicking `Get List`).

4. Navigate down until you see `Rfacebook`. Highlight it, and click `OK` (Windows) or `Install Selected` (Mac) ***For Mac users, make sure to check the "Install Dependencies" box.* ***It will save you A LOT of hassle later.*** This will download your selected library. Note that you can download more than one library at once.

5. To load the library that you just downloaded into *active memory* so you can use it, type:

   ```
   library("Rfacebook")
   ```

   This will load in the selected library. As with downloading libraries, you can load multiple libraries as well (but we recommend that you just do one at a time). Now that you have the package installed on your local machine, you should never have to install it again. However, *every time you launch the R environment, you'll have to use the* `library` *command* for each package that you want to use during that session.

Now What?

Congratulations! You have just downloaded R and started working within the R environment. Over time, you'll see how versatile and cool R is, and how it can empower you as a programmer and data analyst. You're now ready for some more advanced exercises, or to begin exploring the useful examples at R Bloggers (http://www.r-bloggers.com).

You can also search Google to find different flavors of the "Hello World!" exercise that other new R learners have tried. See if you can expand your code to do something more interesting, like prompt you to enter your name from the keyboard using the `scan()` function and have R say hello to you by name.

1.2 Why I Love R

Objective

The purpose of this chapter is to give you a sense of why I use the R Statistical Software daily, wherever and whenever I can. On Valentine's Day in 2012, I decided that I would publicly declare my love for my favorite software package. Here is that declaration.

Preamble

My valentine is unique. It will not provide me with flowers, or chocolates, or a romantic dinner tonight, and will certainly not whisper sweet nothings into my good ear. *And yet – I will feel no less loved.* In contrast, my valentine will probably give me some routines for identifying control limits on control charts, and maybe a way to classify time series. I'm really looking forward to spending some quality time today with this great positive force in my life that saves me so much time and makes me so productive.

Today, on Valentine's Day, I am serenading one of the loves of my life – R. Technically, R is a statistical software package, but for me, it's the nirvana of data analysis. I am not a hardcore geek programmer, you see. I don't like to spend hours coding, admiring the elegance of the syntax and data structures, or finding more compact ways to get the job done. I just want to crack open my data and learn cool things about it, and the faster and more butter-like[1] the better.

Here are a Few of the Reasons Why I Love R

- **R did not play hard to get.** The first time I downloaded R from http://www.r-project.org, it only took about 3 minutes, I was able to start playing with it immediately, and it actually worked without a giant installation struggle.

[1] http://qualityandinnovation.com/2009/01/24/the-butter-test/

- **R is free.** I didn’t have to pay to download it. I don’t have to pay its living expenses in the form of license fees, upgrade fees, or rental charges (like I did when I used SPSS). If I need more from R, I can probably download a new package, and get that too for free.

- **R blended into my living situation rather nicely, and if I decide to move, I’m confident that R will be happy in my new place.** As a Windows user, I’m accustomed to having hellacious issues installing software, keeping it up to date, loading new packages, and so on. But R works well on Windows. And when I want to move to Linux, R works well there too. And on the days when I just want to get touchy feely with a Mac, R works well there too.

- **R gets a lot of exercise, so it’s always in pretty good shape.** There is an enthusiastic global community of R users who number in the tens of thousands (and maybe more), and report issues to the people who develop and maintain the individual packages. It’s rare to run into an error with R, especially when you’re using a package that is very popular.

- **R is very social; in fact, it’s on Facebook.** And if you friend “R Bloggers” you’ll get updates about great things you can do with the software (some basic techniques, but some really advanced ones too). Most updates from R Bloggers come with working code.

- **Instead of just having ONE nice package, R has HUNDREDS of nice packages.** And each performs a different and unique function, from graphics, to network analysis, to machine learning, to bioinformatics, to super hot-off-the-press algorithms that someone just developed and published. (I even learned how to use the “`dtw`” package over the weekend, which provides algorithms for time series clustering and classification using a technique called Dynamic Time Warping. Sounds cool, huh!) If you aren’t happy with one package, you can probably find a comparable package that someone else wrote that implements your desired functions in a different way.

- (And if you aren’t satisfied by those packages, **there’s always someone out there coding a new one**.)

- **R helps me meditate.** OK, so we can't go to tai chi class together, but I do find it very easy to get into the flow (a la Mihaly Csikzentmihalyi) when I'm using R.
- **R doesn't argue with me for no reason.** Most of the error messages actually make sense *and* mean something.
- **R always has time to spend with me.** All I have to do is turn it on by double-clicking that nice R icon on my desktop. I don't ever have to compete with other users or feel jealous of them. R never turns me down or says it's got other stuff to do. R always makes me feel important and special, because it helps me accomplish great things that I would not be able to do on my own. R supports my personal and professional goals.
- **R has its own journal (http://journal.r-project.org).** Wow. Not only is it utilitarian and fun to be around, but it's also got a great reputation and is recognized and honored as a solid citizen of the software community.
- **R always remembers me**. I can save the image of my entire session with it and pick it up at a later time.
- **R will never leave me**. (Well, I hope. It was really crushing that one time *Java* left me. It was in the mid-90s, and I had spent about a year with Java 1.0 and was really starting to get productive, at a time when the language was fresh and new and amazing. But Sun deprecated all of my favorite classes and methods in Java 1.1, and it hit my productivity and my heart so deeply, I've just never gone back to Java... even though the language is much better behaved now, and has a lot more people holding it to standards.)

The most important reason I like R is that I *just like spending time with it*, learning more about it, and feeling our relationship deepen as it gently helps me analyze all my new data. (This is seriously geeky – yeah, I know. At least I won't be disappointed by the object of *MY* affection.)

1.3 Variables and the Case Format

Objective

Before you analyze your data, prepare charts or graphs, or do statistical tests, it's important to gain a sense of what that data is all about. What's the *story* behind the data? How *good* is it? Why do I *care* about what research questions I'm asking? The purpose of this section is to introduce you to some conceptual approaches for understanding your data before you load it into a software package like R for further analysis. You will:

- Learn the difference between categorical and quantitative variables
- Learn how to make categorical variables out of quantitative raw data
- Find out the difference between independent and dependent variables
- See how to characterize your data in terms of the 5 W's (and 1 H)
- Be introduced to the concept of a *case*, and why it's important to prepare your raw data in case format so that a statistical software package can understand it

When I was in school, going through my first few classes on statistics and data analysis, I was really frustrated by all of these classifications of variables. I mean, why do you have to know this stuff? It just seemed like worthless memorization. What I discovered later was that *the type of data you have available* ***dictates*** *what you can do with it*.

Would you like to prepare a scatterplot and maybe perform a linear regression to generate a predictive model? Well, OK, as long as you have two quantitative variables (which are preferably both continuous). Want to do a one-way Analysis of Variance (ANOVA)? No problem, as long as you have multiple collections of quantitative variables, and you can split them up into groups using one of the categorical variables you've collected. Want to construct a bar plot? OK, but you'll need categorical data or else you should be preparing a histogram instead. Want to calculate your average grade? No problem, if your grades have been measured quantitatively, but you'll have a hard time computing your average hair color (a categorical variable). Knowing the types of variables you're working with is not only essential to prevent you from going down unproductive dead ends, but also, it will help you decide what data structures to use if you need to do more advanced programming to facilitate your data analysis.

Categorical and Quantitative Variables

A categorical variable places an observation in one (and only one) category chosen from two or more possible categories. An observation can't be in more than one category at the same time! If there is no ordering that can be done between the categories, the variable is *nominal*, whereas if there is some intrinsic order that can be assigned to the categories, those variables are *ordinal*. Here are some examples of categorical variables:

- Your **gender** (Male, Female, or Other)
- Your **class** in school (Freshman, Sophomore, Junior, Senior, Graduate)
- Your **performance status** (Probation, Regular, Honors)
- Your **political party** affiliation (Democrat, Republican, Independent)
- The **color** of some object (red, orange, yellow, green, blue, purple)
- What **type of degree program** someone is in (BS, BA)
- Your **hair color** (blonde, brown, red, black, white, other)
- What type of **pet** someone has (cat, dog, ferret, rabbit, other)
- The result of someone's **performance in a game** (win, lose)
- **Race** (Hispanic, Asian, African American, Caucasian)
- **Machine settings** (Low, Medium, High)
- **Method of payment** (Cash, Credit)

(Only two of the above examples are ordinal, and the rest are nominal. Can you pick out the ordinal variables?)

Quantitative variables, in contrast, are measured as numbers. Here are some examples of quantitative variables:

- Your **age**
- The number of **siblings** you have

- The number of **spouses** you've had
- The number of **children** you have
- Your weekly, monthly, or annual **salary**
- Your monthly **rent or mortgage payment**
- The **mass** or **weight** of an object
- The **number of speeding tickets** you've received
- The **speed** you were going when you got each ticket
- Your cumulative or most recent **GPA** (measured on a continuous scale from 0.0 to 4.0 or 5.0, depending on your school)

Just because a value is sampled as a number doesn't mean it's automatically a quantitative variable! For example, here are some numbers which are actually categorical variables in disguise:

- Your **social security number** (yeah, can't add or subtract those... logically)
- The **outcome of a game** where you have won (1) or lost (0)
- The **boarding class** on your airplane ticket (1, 2, 3, or 4)
- Which **group you were assigned to** for a team project in one of your classes (1,2, 3, 4, or 5)
- Your **level of agreement** with a particular statement (measured on a *Likert scale* where 1=disagree, 2=slightly disagree, 3=neutral, 4= slightly agree, and 5=agree)

Sometimes researchers will treat values measured on a Likert scale as quantitative, and create scatterplots or do inference tests that require quantitative variables. Purists, like mathematicians, think this is a terrible practice (although researchers who use this approach typically argue that the mean of values measured on a Likert scale has meaning to them). I tend to side with the mathematicians on this one.

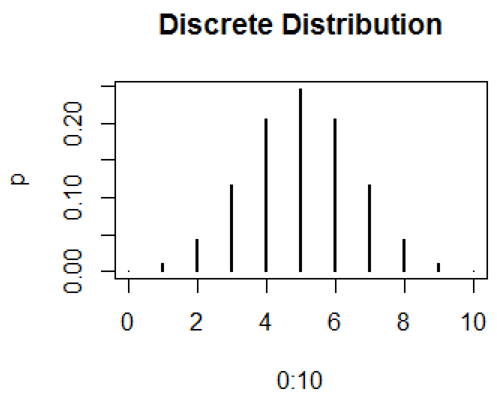
There are Different Kinds of Quantitative Variables

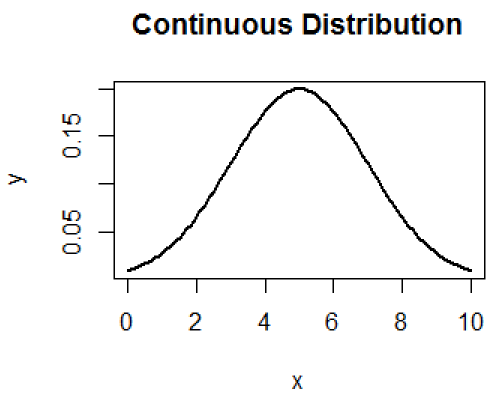
Quantitative variables can be classified further as being on the *interval* or *ratio* scales of measurement. With interval data, you can perform logical operations between them, and you can add and subtract them, but you can't multiply or divide them. Temperature measurements are on the *interval* scale, because although you can tell which of two temperatures is higher or lower, and you can say that 80 degrees is 10 degrees warmer than 70 degrees, you can't say that 80 degrees is exactly twice as warm as 40 degrees. Length and time are quantitative variables on the *ratio* scale of measurement: a distance of two miles is exactly twice as long as a distance of one mile.

Fortunately, these distinctions are not so important if you're just trying to figure out what statistical tests you can use, given that you know you have quantitative data.

Discrete and Continuous Data

Values that variables take on can also be classified as discrete or continuous. For example when you roll a six-sided die, it can only take on one of six values (1 through 6). Categorical data is, by its nature, discrete. Continuous variables can take on values anywhere within an interval of possibilities. *Distributions* can also be discrete or continuous. It's easy to tell whether a distribution is discrete or continuous by **checking to see whether it's smooth or spiky**. Note that in the discrete distribution below, variables can only take on values that are numbers between 0 and 10. In the continuous distribution, variables can take on values *anywhere* within the interval from 0 to 10.





Here is the R code that produced the distributions you see here:

```
par(mfrow=c(1,2)) # Set up plot area with one row and two columns
p <- dbinom(0:10,size=10,prob=0.5)
plot(0:10,p,type="h",lwd=2,main="Discrete Distribution")
x <- seq(0,10,length=100)
y <- dnorm(x,mean=5,sd=2)
plot(x,y,type="l",lwd=2,main="Continuous Distribution")
```

Independent and Dependent Variables

This characterization of variables describes *how you plan to use* your categorical and quantitative variables. A *dependent* variable is one that you've decided to predict from other (*independent*) variables. As a result, independent variables are sometimes called *predictors*, and dependent variables are sometimes called *response* variables. If you have only one predictor and one response variable (that is, one independent variable that you use to predict one dependent variable), and both are quantitative, the dependent variable will be the one plotted on the vertical (y) axis.

Because this label describes how you're *using* your data, you can generate multiple predictive models from one dataset and change what variables you're treating as independent and dependent. Here are some examples of what you can do:

- Use **simple linear regression** to predict one quantitative dependent variable based on the values of one quantitative independent variable
- Use **multiple linear regression** to predict one quantitative dependent variable based on the values of two or more quantitative independent variables
- Use **logistic regression** to predict one categorical dependent variable based on the values of one or more quantitative (or categorical) independent variables

Recoding Quantitative Variables into Categorical Variables

Sometimes it is useful to generate categorical variables from quantitative values you have collected. This process is called *recoding*. For example, say you want to do a study to see whether top performing students smoke fewer cigarettes than students who don't do as well. How do you determine which students are the high performers, and which students are not? Perhaps you might ask them if they're in an honors program or not, but the criteria for getting into an honors program is different from school to school. Also, there may be lots of high performing students who have chosen not to enroll in an honors program due to schedule constraints, program requirements, or other unrelated factors. One way to get around this challenge is to gather information about student performance in terms of a *quantitative variable* and then recode that variable to a categorical variable after the data is collected.

For example, you may decide that cumulative GPA is a reasonable measure for student assessment. You go out and ask 200 students to tell you their cumulative GPA (so far). You end up with a wide variety of values! A substantial fraction of students in your sample didn't do so well, and they are reporting cumulative GPAs between 0.5 and 1.8. However, there were also quite a few students whose GPAs were greater than 3.7 (something you didn't expect). How do you decide which students are the high performers? Since you have so many, you may decide that a cumulative GPA of 3.7 is a good threshold to set.

But what would happen if, in your sample of 200 students, you didn't have very many *at all* who achieved a cumulative GPA greater than 3.7? You might look through your data and say well, it might be more reasonable (for this group of students) to set the threshold at a cumulative GPA of 3.5. Or maybe you will decide to set the bar a little lower, at 3.2. It all depends on what kind of data you collect.

The nice thing about collecting quantitative data wherever you can is that *you always have the flexibility to decide on the boundaries for your categories later!* What would have happened if you decided, before the fact, that a "high performer" was someone who had a cumulative GPA greater than 3.8? Then, you went out and collected data, and *only* asked students if their GPA was greater than 3.8 (thus, you collected that data as a categorical variable). When you reviewed the data you had collected, you found out that *no students reported a cumulative GPA greater than 3.8!* In this case, your research study would be dead in the water. SOL, and not in the educational-standards sense. You would have to start over... and you probably wouldn't be happy about it.

You *lose information* when you recode quantitative variables into categorical variables. As a rule, it's always best to collect data as quantitative when you can, and reserve the privilege of creating categorical variables from your quantitative values later.

To recode a quantitative variable into a categorical value, just 1) decide on the boundaries for your categories, and 2) assign each quantitative variable into a category. You can revise your recoding as often as you need to. It's a great superpower to have.

A Simple Characterization of Metadata Using the 5 W's (and 1 H)

Metadata is data that tells you more about data. What this means, in real people terms, is that it can be useful to talk about *how* you collected your data and *why* you collected your data so that people in the future will know how (and whether) to use the beautiful data resources that you have painstakingly prepared. Think of metadata as your way to ensure your legacy in the realm of personal time travel: some data analyst in the future, who finds the data that you collected, will try to determine whether he or she can use that data for *their* own unique purposes. And you can help them. Some of the questions you will want to answer for them (what I call the "5W/1H questions") are:

- **Who**: *There are two parts to answering this question!* First, who are your data about? The data can be about a person, a group of people, an object or class of objects (e.g. computers, machines, or a particular type of computers or machines), a type of animal, or maybe even an inanimate object like a lake or pond (where you might be collecting information about water quality or other aspects of the environment). Second, who is collecting the data? A future researcher will treat data collected by an elementary school student much differently than he or she would treat data collected by a scientist or a professional in industry. Be sure to establish your qualifications for collecting the data, and in what context you are applying your skills. You may even want to include contact information in case anyone has a question about your data.

- **What**: What is your data about? What types of data are you collecting about whatever subject you identified when you answered the "who" question? I like to provide a subjective overview about each of my variables here, sometimes articulating which are categorical and which are quantitative, the enumerations of each of the possible categories, and whether I recoded some quantitative data to get some of the categorical variables.
- **When**: When were your data collected? This is particularly important if you wish to look at changes in the values of a variable that occur over time (for example, measurements of CO2 in the atmosphere if you are studying global warming). Also, values collected earlier in time will be subject to the limitations of the less advanced sensors or technologies that are used to collect the measurements.
- **Where**: In what location(s) did you collect your data? Sometimes, this can be as easy as collecting a geographical marker (for example, a latitude and longitude) for each item in your sample. Or, you might just describe the location that is associated with all of your data, for example, the city, country, or organization within which your data was acquired.
- **Why**: For what original purpose did you collect your data? What were the objectives of your study? This will help a future researcher understand the assumptions and limitations that you were aware of at the time the data were collected.
- **How**: How were the data collected? You might want to explore what instruments you used to measure the values for each variable, or what sampling strategies you used to ensure a random sample where the observations are independent. If you obtained your data from an archive (or from multiple archives), be sure to include the locations of the archives, so that someone in the future will potentially be able to retrieve the source data directly from the same location you did.

Why is it important to describe these things? First and foremost, because if you don't do it at the point of data collection... *you'll probably forget*. If you're young, or if this data is important to you, you might be thinking "no way... I'm not going to forget." But you will. I've collected so much data over the past two decades that when I go back and look at an old dataset, I have to remind myself what I was thinking about the first time I worked with it. So metadata provides a way for my *past self* to communicate with my *future self* and help her be more productive.

Second, while you're analyzing your data, you may need to enlist the help of friends, colleagues, or other smart people on the internet who can help you troubleshoot the issues you're having analyzing your data in a software system like R. People won't be able to help you if you can't provide them with useful information about your data, how you got it, and what you've done with it already. So be ready to exchange metadata and a step-by-step history of what you've done with (and to) your data with any person (or bot; yes, we're almost there) who may be able to provide help.

Third, *reproducibility* is becoming a more and more significant issue in the sciences. What this means is that if you've done an experiment once, you should be able to collect the data and analyze it again, and draw pretty much the same conclusions. However, this is much easier said than done. There's a huge volume of published research in the academic literature, but not all of it can be reproduced - even by the researchers who designed and executed the initial studies! The Wikipedia page on reproducibility (http://en.wikipedia.org/wiki/Reproducibility) includes a section on "noteworthy irreproducible results" which, though sparse, includes the famous "cold fusion" case from the late 1980's. Although this would have represented a huge breakthrough in our ability to produce energy, the experiment was not reproducible and could not be verified. Also, because the results could not be reliably reproduced, further studies to explore the technological potential of cold fusion were impossible.

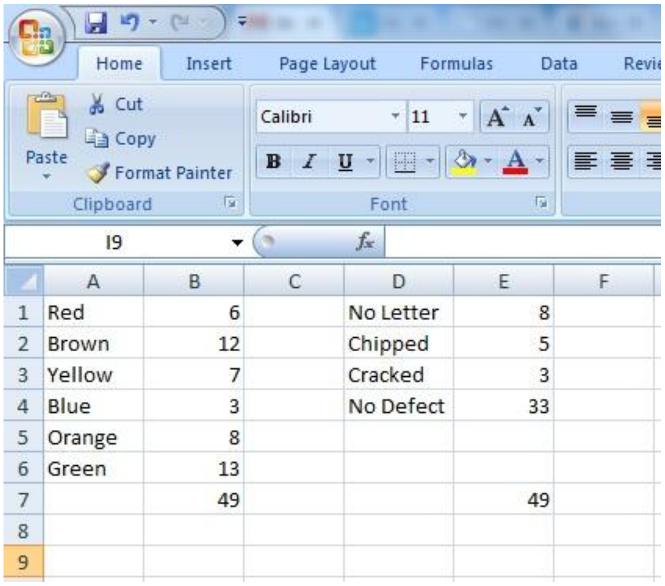
Finally, when you encounter data that's *new to you* that someone else has collected, it is always useful to ask yourself the 5W/1H questions to get a better sense of what that data is all about. Also, thinking about the data will help you understand how to structure the data for analysis in R or another statistical software system using cases (described below).

Case Format

I've found that the most challenging aspect of gathering data is making sure that you format your results so that a statistical software package will actually be able to do something with it. Here are some heuristics (rules of thumb) to help you format your data. In your mind, picture a spreadsheet with rows, columns, and cells where you will enter each element of your data.

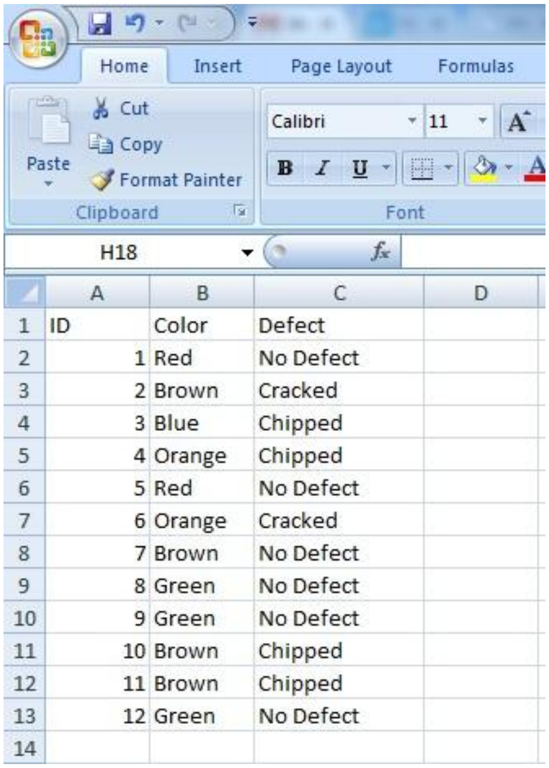
- Each of the columns should contain ONE AND ONLY ONE variable
- Each of the rows should contain ONE AND ONLY ONE "who" from your sample
- The total number of rows should equal n, the number of items in your sample

For example, if you are collecting information about the distributions of colors and defects in a bag of M&Ms, it's very easy to produce a spreadsheet that looks like this:



	A	B	C	D	E	F
1	Red	6		No Letter	8	
2	Brown	12		Chipped	5	
3	Yellow	7		Cracked	3	
4	Blue	3		No Defect	33	
5	Orange	8				
6	Green	13				
7		49			49	
8						
9						

However, ***<u>this is bad</u>*****!** If you try to upload data in this format into a statistical software package, it won't know what to do. It's not in *case format*. After you answer the 5W/1H questions, you'll know that that answer to "Who are your data about?" is "the M&Ms." Each case (or row) in your properly formatted data should be about one (and only one) "who" - which, in this case, is one M&M. Each row should contain data about one (and only one) of the M&Ms in your sample. In contrast, **here's what data properly formatted as cases looks like:**



	A	B	C	D
1	ID	Color	Defect	
2	1	Red	No Defect	
3	2	Brown	Cracked	
4	3	Blue	Chipped	
5	4	Orange	Chipped	
6	5	Red	No Defect	
7	6	Orange	Cracked	
8	7	Brown	No Defect	
9	8	Green	No Defect	
10	9	Green	No Defect	
11	10	Brown	Chipped	
12	11	Brown	Chipped	
13	12	Green	No Defect	
14				

Notice how each row contains data collected from one M&M. Also, we didn't include any summary information in our spreadsheet, for example, the total number of items in our sample (49) that's been tallied in the top example. Our statistical software can very easily produce any of this summary information, so we don't need to add it up ourselves. We have been consistent in labeling the values from our categories so that our statistical software will be easily able to compile the values (that is, we didn't say "Red" in one place and "RED" in another - R will treat those as two completely different values).

Other Resources

- Here is a web-based game that lets you guess whether a variable is categorical or quantitative: http://mathnstats.com/applets/Categorical-Quantitative.html
- Here's a useful article on continuous quantitative vs. categorical data: https://eagereyes.org/basics/data-continuous-vs-categorical
- Jeff Good of UC Berkeley has written "A Gentle Introduction to Metadata" at http://www.language-archives.org/documents/gentle-intro.html
- A catalog of R resources for reproducible research is available at http://cran.r-project.org/web/views/ReproducibleResearch.html
- There is even a Coursera class where you can learn all about reproducible research! Find it here: https://www.coursera.org/course/repdata

1.4 Central Tendency and Variability

Objective

A probability distribution (or "distribution" for short) describes the possible outcomes for an event, and how likely each of those outcomes are. When characterizing a distribution of categorical or quantitative values, the things you typically want to know are: 1) where's the center of it, 2) how fat or thin is it, and 3) does it have any unusual characteristics (e.g. is it skewed to the right or left, or does it have more than one hump, or is it asymmetric). **"Central tendency" is a fancy sounding phrase that just means: *if you have a whole bunch of values, what's in the middle (and what does "middle" even mean)?*** Measures of variability (e.g. variance and standard deviation) tell you how fat or thin your distribution is. This chapter covers these two basic topics, including how to compute them analytically and how to compute them in R. Distribution shapes are covered in the chapter on histograms.

Who cares about such a simple topic? Well, just knowing the values for the mean, median, and mode, coupled with the variability information provided by the variance and standard deviation, you can get a sense of the shape of your distribution. You can also sometimes get a sense for what proportion of your observations fall within various bounds. Also, many of the statistical inference tests you perform are related to *figuring out if you really know where the center of a particular distribution is located*. If you read fancy proofs, you'll hear the term "Expected Value" quite a lot. What they mean is: ***the value we think is in the middle of all the possibilities.*** That's all. This chapter covers the first two of these basic topics, including how to compute them analytically and how to compute them in R. Distribution shapes are covered in the chapter on histograms.

Mean (Arithmetic)

The arithmetic mean identifies the midpoint between all the values in a dataset of numbers, even if there isn't an observed value *at* the midpoint. It is calculated by adding up all of the values, then dividing by the total number of observations. Finding a mean in R is simple, because you can pass values directly to `mean` *or* give it a vector or data frame selection:

```
> mean(c(1,2,3,4,5,6,7,8))
[1] 4.5
> x <- c(1,2,3,4,5,6,7,8)
> mean(x)
[1] 4.5
```

The arithmetic mean is not the *only* variety of mean that can be used to express central tendency! The geometric mean and harmonic mean are also potential measures if your quantitative data is on the *ratio* level of measurement (and, for the geometric mean, if your values are positive). The geometric mean characterizes the *average growth rate* between values, and is often used in financial applications. The harmonic mean characterizes an *average rate*. I've never used the geometric or harmonic means for data analysis personally, though, so I can't provide any recommendations beyond noting that you *can* potentially use these.

Median

The median identifies *the observation* that sits at the midpoint between all the observations in a dataset. If there are an even number of observations, the median is computed as the arithmetic mean of those two observations that sit at the midpoint. Like `mean`, you can pass values directly to `median` *or* give it a vector or data frame selection. Here are some examples in R:

```
> median(c(1,2,3,4,5,6,7,8,9))
[1] 5
> median(c(1,2,3,4,5,6,7,8,9,10,11,12))
[1] 6.5
```

As long as the values in a dataset can be *ordered*, you can find the one (or two) observations that sit at the midpoint, and thus find the median. You can compute a median for all kinds of variables... except categorical variables that are nominal. As an example, you can't compute the median of several M&M colors (because that's a categorical variable at the nominal level of measurement) but you can compute the median of several scrabble tile values (because those letters can be ordered). Because of its versatility, there are many inference tests that can be done to compare the medians of datasets.

Mode

The mode identifies the *most frequently observed value* in a dataset. Unlike the median, calculation of the mode is not precluded by the level of measurement. For example, it's possible to determine:

- The mode of M&M colors in a bag (that is, the most frequently observed color)
- The mode of Scrabble tiles still left in the bag (that is, the letter or letters that have the most tiles available)
- The mode of customer satisfaction responses (that is, which item between "Strongly Dissatisfied" and "Strongly Satisfied" was most frequently selected)
- The mode for daily high temperatures in summer (that is, which high temperature occurs most frequently)

These illustrate what the mode looks and feels like for nominal, ordinal, interval, and ratio level data, respectively. The only one you have to be careful with are modes for ratio level quantitative variables, because sometimes the mode is just not meaningful. As an example, think about what your data might look like if you're measuring how much people weigh. If you're collecting that data in pounds to the second decimal place, it's likely that your data looks something like this, with no repeated values:

145.59   220.31   197.63   105.25   118.80   145.14   170.28   166.37

It would be really unlikely for the mode to be meaningful in this case: we might not have any numbers that are observed more than once, making the "most frequently appearing observation" an absurd notion. **Bottom line: if you have ratio level data, check to make sure the mode is meaningful before you report it.** It's actually pretty easy with R to determine if a mode is "meaningful" or not. Here's an example where the mode is clearly meaningful. First, we pull 100 **r**andom numbers from a **unif**orm distribution between 1 and 10. (This is just like rolling dice, only we're working with a 10-sided die where each of the numbers 1 through 10 is equally likely to appear. That's the purpose of the `runif` function.)

```
> x <- round(runif(100,1,10))
> x
  [1]  7  8  1  3  5  4  3  3  4  3  3  7  2  4  1  7  2  7  8  9  6
 [22]  1  7  2  1  5  2  1 10  8 10  3  3  5  3  7  4  9  6  1  2  4
 [43]  5 10  8  1  5  3  2  3  1  8  7  4  4  6  7  7  9  3  6  2  9
 [64]  5  2  3  4  1  8  5  5 10 10  8  7  6  5  7  8 10  9 10  4  7
 [85]  2  7  6  9  8  9  5 10  9  2  1  6  8  3  6  5
```

Now, let's add up how many 1's, 2's, 3's and so forth we got using the random number generator by invoking the `table` command. The top row represents the numbers from 1 to 10, and the bottom row includes the counts of how many of those random numbers were generated by `runif`. Using `sort` arranges the numbers so that the lowest frequencies appear first, and the higher frequencies appear later.

```
> y <- sort(table(x))
> y
x
 6  9 10  4  1  2  8  5  3  7
 8  8  8  9 10 10 10 11 13 13
```

Here are the commands we can use to pull out which elements of x appear most frequently (`mode.names`) and how many times they appear (`max(y)`). As long as you store your sorted table in a variable called `y`, you will be able to use these commands to identify the mode (or modes, because you can have more than one) and the observation count that corresponds to that maximum value.

```
> mode.value <- max(y)
> mode.value
[1] 13
> mode.names <- names(y[y==max(y)])
> mode.names
[1] "3" "7"
```

This also works if your values are categorical (at the nominal level of measurement). Imagine that you own a restaurant specializing in New Mexican cuisine. You want to find out how your customers prefer the chile on their burritos: red, green, or Christmas. Here's a way to simulate that data from 100 customers:

```
> x <- sample(c("RED","GREEN","CHRISTMAS"), 100, replace=TRUE)
> x
  [1] "GREEN"     "RED"       "RED"       "RED"       "CHRISTMAS"
  [6] "RED"       "GREEN"     "CHRISTMAS" "CHRISTMAS" "GREEN"
 [11] "GREEN"     "RED"       "CHRISTMAS" "CHRISTMAS" "CHRISTMAS"
```

The mode can be determined the same way as in the previous example:

```
> y <- sort(table(x))
> names(y[y==max(y)]) # these are the modes
[1] "RED"
> max(y) # we just have one mode... how many customers preferred red?
[1] 35
```

What about the case where the mode is meaningless? Well, that's what would have happened if we didn't round our randomly sampled numbers between 1 and 10. Try this, and you'll see a mode that's not meaningful at all:

```
> x <- runif(100,1,10)
> y <- sort(table(x))
> max(y)
[1] 1
> names(y[y==max(y)])
  [1] "1.08012562571093" "1.12060043844394" "1.17615319066681"
  [4] "1.18749733804725" "1.39209767919965" "1.41802837909199"
  [7] "1.4363781651482"  "1.62966463225894" "1.63671832019463"...
```

What's the frequency of our mode? One. Just one observation. That means EVERY SINGLE ONE of the values we generated is the mode. A hundred values, a hundred modes. If *everyone* is an important mode, then *no one* is an important mode. The mode, in this case, is meaningless as an indicator of central tendency.

Relationships Between Mean, Median, & Mode

If you have a collection of quantitative values, and you know the mean, median, and mode, there are a lot of things you can figure out about the shape of your distribution. As a result, these measures of central tendency are like *clues* that you can use to draw a picture of the distribution in your head:

- The mode is always at the highest point of the distribution... the peak.
- If the distribution is skewed to the left, meaning that it has a tail stretching out along the left side of the x-axis... the median is pulled to the left.
- If the distribution is skewed to the left, the mean is also pulled to the left. But all it takes is one or two outliers to really really pull that mean even farther to the left. The mean is much more sensitive to outliers than the median.
- If the distribution is skewed to the right, meaning that it has a tail stretching out along the right side of the x-axis... the median is pulled to the right.
- If the distribution is skewed to the right, the mean is also pulled to the right. But all it takes is one or two outliers to really really pull that mean even farther to the right. The mean is much more sensitive to outliers than the median.
- In a symmetric distribution like the normal, the mean, median, and mode are all lined up together at the peak of the distribution. The most frequently observed value (mode) is the same as the average value (mean). Exactly 50% of the observations are below the mean, and 50% of the observations are above the mean, putting the median at the same spot as the mean.

In summary, here are the measures of central tendency that can be applied to the various variable types and levels of measurement:

Variable Type	Level of Measurement	Ways to Represent Central Tendency
Categorical	Nominal	Mode
	Ordinal	Median
Quantitative	Interval	Arithmetic Mean, Median, Mode
	Ratio	Arithmetic Mean, Median, Mode, Geometric Mean, Harmonic Mean

Variance

The variance represents how spread apart the values in your distribution are, and as a result, characterizes *how fat or thin* the distribution will be when plotted. To get the variance, you need to know all of the values in your dataset. You use them to calculate variance like this:

1. First, **find the arithmetic mean** of all the values.
2. **Find the squared deviations**: Take each value one at a time, and subtract the mean (which gives you negative numbers for all values that are *below* the mean, and positive numbers for all values that are *above* the mean) and then square whatever you get (that is, multiply it by itself). This makes all the values positive.
3. Now **divide the total of all those squared deviations you just figured out by one less than the number of observations**. (Why don't we just use the actual number of observations, and take the average? Because that would be *biased*... but more on that later when we talk about Bessel's correction.)

The equation that represents variance looks like this. Variance is represented by σ^2, and *n* is the number of observations. The sum is over all elements of your dataset (from the first one through the *n*th one), and you're adding up the squared differences between each value and the overall mean:

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

Variance is also very easy to figure out in R, as long as you have your data arranged in a vector (or can extract a vector out of a data frame). For simplicity, let's just use the values that are stored in *x* - the ones we got from sampling 100 random numbers between 1 and 10:

```
> var(x)
[1] 6.556027
```

Standard Deviation

Take the square root of the variance, and you'll get the standard deviation:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

This is also dead easy to do in R using the `sd` command. Of course, if you have the variance handy, you can just take the square root of that to get the standard deviation and it all works out the same:

```
> v <- var(x)
> sqrt(v)
[1] 2.560474
> sd(x)
[1] 2.560474
```

Bessel's Correction: Why We Divide by (n-1)

If you Google around to find equations for calculating the variance and standard deviation, sometimes they tell you to divide by the total number of observations n, and other times they tell you to divide by (*n-1*). Oh no!! Which one should you believe? What do you do? And which one does R do?

- **Which one should you believe?** Believe the (n-1) version. It's more accurate, especially when the sample size is small.
- **What do you do?** If you calculate variance and standard deviation yourself, be sure to use the (n-1) version.
- **And which one does R do?** Fortunately, R uses the (n-1) version for both the variance and the standard deviation... so you can trust R.

What's the Rationale for Bessel's Correction?

The smaller your sample, the less likely you are to *realistically* capture the magnitude of the variance in a distribution. It takes a lot of observations to get a good sense of the true spread-outed-ness of values within a distribution!

But to determine the variance and standard deviation, you have to know the mean of the *entire population* (or at least have a good idea what it is). You're only guaranteed to estimate this well if you have a really big sample. So with an ordinary sized sample, you're not going to capture *all* of the variance that's really there. You need to bump up your estimate to account for the variability that you can't see. And since this is always true for sample sizes that are 1 or more...

$$\frac{1}{n-1} > \frac{1}{n}$$

... that means if you use the (*n-1*) version, you'll be making your estimates of the variance and standard deviation *just a little bigger*. The bigger the sample size *n* gets, the closer the *biased* estimator (where you divide by *n*) will be to the *unbiased* estimator (where you divide by *n-1*). **Unbiased is better**. This adjustment improves your variance estimate tremendously, and your standard deviation estimate *almost* as tremendously.

Other Resources

- Did you know there are guidelines on how to report central tendency and variability in APA style research papers? Details are provided here: http://statistics-help-for-students.com/How_do_I_report_central_tendency_and_dispersion_data_in_APA_style.htm#.VQtgtI7F-o0
- Find out more about Bessel's correction (n-1 in the calculation of the variance and standard deviation) at http://en.wikipedia.org/wiki/Bessel%27s_correction
- A really nice, intuitive explanation for Bessel's correction with graphs is here: http://www.physics.ohio-state.edu/~durkin/phys416/Fall2011/LectureExtras/besselfactor.pdf

1.5 Descriptive Statistics

Objective

Sometimes, you want an easy way to quantitatively summarize the characteristics of a collection of data, whether that information comes from an experiment, an archive, a survey, or some other kind of source. *Descriptive statistics* provide simple summaries about the sample you have collected, and complement charts and graphs that provide an additional representation of the data. With descriptive statistics, you are not attempting to draw any conclusions about the data: you are just presenting what the data shows, *prior to any analysis*.

Descriptive statistics are often supplemented by measures such as skewness and kurtosis, which describe the basic state of the distribution, or basic charts and graphs, such as histograms, boxplots, scatterplots, and contingency tables, which are covered in **Section 2: Charts, Graphs, and Plots**.

Data Acquisition

First, let's load some data from the National Oceanic and Atmospheric Administration (NOAA) severe weather data inventory at http://www.ncdc.noaa.gov/swdi. We're going to choose our data, download it to our local machine, unzip it, and then make sure R is looking in the right directory. Finally, we can examine *descriptive statistics* on our data.

After going to the main SWDI page, I clicked on the option for "Bulk Download" and then "HTTP". This took me to `http://www1.ncdc.noaa.gov/pub/data/swdi/` which is a directory listing for several CSV files. (I notice that all the files have a `.gz` extension, which means that I need to *unzip* the file I download before I can use it. To be able to handle `.gz` files, I have to download a utility program that unzips this particular kind of zipped file. Since I use a Windows machine, I first downloaded and installed a program called 7-Zip from `http://download.cnet.com`). Then, I navigated back to the SWDI repository and downloaded `tvs-201407.csv.gz` to my local machine, unzipped it, and saved the resulting `tvs-201407.csv` file to my `C:/Temp` directory. This is a document containing all the Tornado Vortex Signatures (TVS) in the United States during the month of July 2014. Now I can read the data into R:

```
        setwd("C:/Temp")
        tvs <- read.csv("tvs-201407.csv",header=T,skip=2)
```

Because the first two lines of the data file contain comments (they start with a #, and so are ignored by R), the `skip=2` argument tells R to read in my CSV data, *but only after skipping those first two extraneous lines*. Now, I can take a look at what's in the file:

```
> head(tvs)
      X.ZTIME       LON      LAT WSR_ID CELL_ID CELL_TYPE RANGE AZIMUTH
1 2.01407e+13 -91.53766 41.34569   KDVN      M7       TVS    46     250
2 2.01407e+13 -94.51676 40.24530   KMCI      J1       TVS    46      13
3 2.01407e+13 -88.06882 41.52247   KORD      W7       TVS    19     210
4 2.01407e+13 -94.79196 40.04702   KMCI      C1       TVS    33     356
5 2.01407e+13 -87.99919 41.58894   KORD      W7       TVS    14     207
6 2.01407e+13 -91.13138 41.24971   KDVN      S7       TVS    33     229
  AVGDV LLDV MXDV MXDV_HEIGHT DEPTH BASE TOP MAX_SHEAR MAX_SHEAR_HEIGHT
1    34   83   83           4    26    4  30        29                4
2    32   52   52           3    17    3  20        18                3
3    37   50   53           2     7    1   8        46                2
4    32   31   70          14    18    2  20        33               14
5    41   56   56           1     6    1   6        63                1
6    33   53   53           3     6    3   9        26                3
```

What is this data about? It looks like the first column contains a timestamp, and the second two contain the longitude and latitude of the observed TVS (a place where Doppler radar detected a likely tornado). `WSR_ID` is the name of the radar station that reported the observation. `CELL_ID` provides a way for us to distinguish between tornadoes associated with different thunderstorms. `CELL_TYPE` indicates that we are looking at TVS observations, so this value should be the same for all rows. `RANGE` and `AZIMUTH` tell us where the TVS was observed, relative to the radar, and the remaining values tell us physical parameters about the tornado signature. The information we will be interested in gathering descriptive statistics for are the `DEPTH` of the tornado signature (in thousands of feet), the `TOP` of the tornado signature (also in thousands of feet), and the `MAX_SHEAR` (or maximum wind shear, in meters per second). Since we are only interested in a subset of the data, let's construct a smaller data frame containing only those values that are of interest to us. This creates a new data frame, `sub.tvs`, which consists of all rows from `tvs`, and just these three named columns:

```
sub.tvs <- tvs[,c("DEPTH","TOP","MAX_SHEAR")]
```

Descriptive Statistics with `summary`

Once we have our data in the proper format, we can generate descriptive statistics using the `summary` command:

```
> summary(sub.tvs)
     DEPTH            TOP           MAX_SHEAR
 Min.   : 5.00   Min.   : 5.00   Min.   : 10.00
 1st Qu.: 7.00   1st Qu.: 9.00   1st Qu.: 24.00
 Median :10.00   Median :13.00   Median : 30.00
 Mean   :12.34   Mean   :15.57   Mean   : 38.03
 3rd Qu.:17.00   3rd Qu.:21.00   3rd Qu.: 42.00
 Max.   :63.00   Max.   :69.00   Max.   :283.00
```

The descriptive statistics that are displayed are:

- The **minimum** value observed for this variable in the dataset
- The **first quantile** (also called Q1); 25% of the observations are BELOW this value and 75% of the observations are ABOVE this value
- The **median**; an actual observation within the dataset indicating that 50% of the observations are BELOW this value and 50% are ABOVE this value
- The **mean**; a number characterizing the center of the distribution that is obtained by finding the average value across all the observations
- The **third quantile** (also called Q3); 75% of the observations are BELOW this value and 25% of the observations are ABOVE this value
- The **maximum** value observed for this variable in the dataset

Descriptive statistics can give us a quick indication of some of the characteristics of the distribution: for example, if the mean is far greater than the median (to the right of it on the probability distribution), we know that the distribution must be skewed to the right. Similarly, if the mean is far less than the median (to the left of it on the probability distribution), we know that the distribution must be skewed to the left.

One limitation of the `summary` command in the base R package is that it doesn't give you really useful and critical descriptive statistics like the standard deviation, variance, and mode (the most frequently observed value in the set of observations). Fortunately, you can compute those separately, but you have to do each variable one at a time:

```
> sd(sub.tvs$DEPTH)
[1] 7.19
> var(sub.tvs$DEPTH)
[1] 51.7
```

To obtain the mode, first cut and paste the following code to load in the `mode` function:

```
mode <- function(x) {
	uniq.vals <- unique(x)
	uniq.vals[which.max(tabulate(match(x, uniq.vals)))]
}
```

Now you can use that function on your data:

```
> mode(sub.tvs$TOP)
[1] 7
```

Some of these limitations can be overcome by using an alternative approach: the `stat.desc` function which is part of the `pastecs` package.

Descriptive Statistics with `stat.desc` from the `pastecs` Package

If you don't have it already, first `install.packages("pastecs")` and then when it's available on your machine, call it into memory using `library(pastecs)`. Only then will the following commands work.

```
> options(scipen=100)
> options(digits=3)
> stat.desc(sub.tvs)
                 DEPTH       TOP MAX_SHEAR
nbr.val       1532.000  1532.000  1532.000
nbr.null         0.000     0.000     0.000
nbr.na           0.000     0.000     0.000
min              5.000     5.000    10.000
max             63.000    69.000   283.000
range           58.000    64.000   273.000
sum          18905.000 23850.000 58260.000
median          10.000    13.000    30.000
mean            12.340    15.568    38.029
SE.mean          0.184     0.205     0.631
CI.mean.0.95     0.360     0.401     1.238
var             51.724    64.179   610.432
std.dev          7.192     8.011    24.707
coef.var         0.583     0.515     0.650
```

Since `stat.desc` likes to use scientific notation, the first two commands allow you to set the number of significant digits you want to see. In addition to the basic descriptive statistics provided by summary, `stat.desc` shows you:

- The **number of observations** as `nbr.val`
- The **number of null observations** as `nbr.null`
- The **number of "not available" (NA) observations** as `nbr.na`
- The **range** of values (that is, how far it is to get from the minimum observed value to the maximum observed value) as `range`
- The **sum** of all values as `sum`

- The **standard error** of the mean (`SE.mean`) indicates how precisely you can know the true mean of the population, given only what you have in your sample. It accounts for the sample size and the scatteredness (or dispersion) of the dataset.
- The **width** of the 95% confidence interval as `CI.mean.0.95` (meaning that the mean minus this value indicates the lower bound of your confidence interval, and the mean plus this value indicates the upper bound of the confidence interval)
- The **variance** of all the values, which gives you an indication of how scattered or dispersed they are, as `var`
- The **standard deviation**, which is the square root of the variance and indicates pretty much the same thing as the variance, as `std.dev`
- The **coefficient of variance**, which is the standard deviation divided by the mean, and also gives a sense of how scattered or dispersed the observations are, as `coef.var`

Here is a summary of the arguments that you can use to generate descriptive statistics:

Argument to `stat.desc`	**What it does**
`basic=TRUE`	Include number of observations, number of nulls, number of NAs, min, max, range and sum
`desc=TRUE`	Include median, mean, standard error of the mean, width of the confidence interval, variance, standard deviation, and coefficient of variation
`norm=TRUE`	Include measures for skewness and kurtosis
`p=0.99`	Specify the width of the desired confidence interval (e.g. 99%)
`options(digits=3)`	Specify the number of significant digits to display (e.g. 3)
`options(scipen=999)`	Specify a "penalty" to use when determining whether scientific notation (e.g. 1E+03 or 1000); higher values incur greater penalty, and a 999 will prevent use of scientific notation almost entirely

Other Resources

Here are some other resources to help you understand the concepts in this chapter and explore the severe weather datasets that were obtained:

- This article discusses the difference between *descriptive statistics* and *inferential statistics*: https://statistics.laerd.com/statistical-guides/descriptive-inferential-statistics.php
- This article and online quiz will help you get a better handle on definitions: http://study.com/academy/lesson/what-is-descriptive-statistics-examples-lesson-quiz.html
- This applet can be used to explore descriptive statistics on simple datasets: http://www.rossmanchance.com/applets/Dotplot.html
- Find out more about Tornado Vortex Signatures (TVS) at http://en.wikipedia.org/wiki/Tornado_vortex_signature
- More information about the data in the TVS datasets can be found here: http://www.ncdc.noaa.gov/swdiws/csv/nx3tvs
- There is an R package to access data directly from the SWDI. Find out more at http://cran.r-project.org/web/packages/rnoaa/vignettes/swdi_vignette.html
- The complete NEXRAD radar data archive can be accessed at http://www.ncdc.noaa.gov/nexradinv/
- Comprehensive documentation for the pastecs package, which does a lot more than just descriptive statistics, can be accessed from this location: http://cran.r-project.org/web/packages/pastecs/pastecs.pdf

1.6 More Ways to Acquire and Inspect Data

Objective

Before you can manipulate and analyze data, first you have to go get it and load it into memory in R. The purpose of this section is to introduce you to some techniques for acquiring data and loading it into data structures in R for further analysis. You will:

- Learn how to open a file on your local machine
- Learn how to retrieve data that is accessible on the web using a URL
- Learn how to retrieve data that is stored in Google Docs (using RCurl)
- Understand why it's important to be aware of data types and data structures as you're acquiring data from different kinds of sources
- Be introduced to helpful utilities like `aggregate`, `paste`, and the practice of casting between data types
- Learn where and how to acquire various kinds of government, political, sports, weather, and news data

Files on Your Local Machine

One of the most common scenarios you will encounter is that you have your data stored on your local machine, and you need to load it into R so you can manipulate and analyze it. The most common commands you will use to navigate around your machine, find files, and load them are `getwd(), setwd(), dir(), read.csv(), read.table(), head()` and `str()`.

Central to these commands is the concept of the *working directory*. Basically, R can only be pointing to one directory on your machine at any given time. If I want to find out which directory R is looking at right now and then examine the contents, I can "**get** my **w**orking **d**irectory" by typing `getwd()` and then find out what's in that directory by typing `dir()`:

```
> getwd()
[1] "C:/Users/Nicole/Documents"
> dir()
 [1] "344-labs-grading-fall2012.doc"            "3dtco_v3 (Autosaved).csv"
 [3] "Adobe"                                          "AlienFX"
 [5] "Alienware TactX"                                "ALLMYEDGES.txt"
 [7] "ALLMYNODES.txt"                                 "Apr10_KayakoInvoice.pdf"
 [9] "Avery_Extenci.pdf"                              "berente-notes.doc"
[11] "Bluetooth Exchange Folder"                      "Bluetooth Folder"
[13] "CSSBB.pe"                                       "CyberLink"
[15] "desktop.ini"                                    "Downloads"
[17] "EasleyKleinberg_NetworksBook.pdf"               "EastPrussiaWar41-45.pdf"
[19] "GitHub"                                        "Grimshaw_Presentation.pdf"
```

Without changing my working directory, I can access the contents of any one of these files. I notice that there is one CSV file in the top row, so I'll try opening that to see what's inside. I use the `read.csv()` command because I know it's a CSV file. I'm also going to store my entire file in a variable that I will call `temp` (short for "temporary" because I'm probably not going to use this variable after I poke around the file a little bit). After I read the file, I'll take a look at the first six lines of the file using `head` and then ask R what data structures it has assigned to each variable using `str`:

```
> temp <- read.csv("3dtco.csv")
> head(temp)
  group subsystem product priority reltco satis customiz
1     1  Database  Oracle  Highest 0.9985 0.978    0.000
2     1  Database  Oracle Moderate 0.9997 0.933    0.000
3     1  Database  Oracle   Lowest 1.0000 0.927    0.000
4     2  Database IBM DB2  Highest 0.5739 0.782    0.196
5     2  Database IBM DB2 Moderate 0.5752 0.760    0.173
6     2  Database IBM DB2   Lowest 0.5754 0.768    0.159
> str(temp)
'data.frame':   39 obs. of  7 variables:
 $ group    : int  1 1 1 2 2 2 3 3 3 4 ...
 $ subsystem: Factor w/ 4 levels "AM/OSM","CMES",..: 3 3 3 3 3 3 2 2 2 2 ...
 $ product  : Factor w/ 13 levels "IBM DB2","IBM Maximo",..: 10 10 10 1 1 1
3 3 3 11 ...
 $ priority : Factor w/ 3 levels "Highest","Lowest",..: 1 3 2 1 3 2 1 3 2 1
 $ reltco   : num  0.999 1 1 0.574 0.575 ...
 $ satis    : num  0.978 0.933 0.927 0.782 0.76 0.768 0.9 0.816 0.754 0.7
 $ customiz : num  0 0 0 0.196 0.173 0.159 0 0.084 0.146 0.2 ...
```

The head and str commands tell me complementary information. For example, I know from str that there are 7 variables, and I know from head that the names of these variables are group, subsystem, product, priority, reltco, satis, and customiz. I can also find out the variable names from str, but str gives me even *more* information - it tells me the data type for each of the variables. Let's say we want to find the average reltco for each subsystem. (This stands for "relative total cost of ownership." It's the total cost of ownership of a product relative to other comparable choices, which allows for more accurate within-group comparisons.) The subsystem variable is a factor with 4 levels, each of which refers to a software subsystem at a major organization. AM/OSM refers to an asset management system, where information is stored to track equipment and computers. CMES refers to an internal message bus, which is basically an application that helps other software packages communicate with one another. DPMS is a data processing management system, and Database is a database. Fortunately, reltco is a number, and we can compute averages of numbers. But how do we split up our data set in terms of which subsystem each row is associated with? In R, there are *many* different ways to do things, so don't think that this is the *only* way to do it... but I like using aggregate like this. I underlined each of the three arguments that are being passed to the aggregate command. (When you type this in, please don't try to type in the underlines or the code won't work!)

```
> aggregate(reltco~subsystem, data=temp, FUN=mean)
  subsystem    reltco
1    AM/OSM 0.5476667
2      CMES 0.6475000
3  Database 0.7871167
4      DPMS 0.6330000
```

How you say this in English is: "I want to aggregate the reltco variable by subsystem, from the data set named temp, using the mean function." The first one tells R to split up the reltco variable into groups according to the value of subsystem. The second argument specifies that these values will come from the data set called temp, and the third argument says to apply the function mean to the groups.

You can use all sorts of different functions in aggregate. Here are a few examples. Notice how some of the functions operate on the *complete list of reltco values*, whereas other functions *operate on the aggregations themselves.* The first example below simply lists the contents of the aggregated data set. The second example computes the length of each list that is returned (meaning, it will tell you *how many* data points are in each aggregation). The third example adds up each of the lists, using the `sum` function. The fourth example tells us whether each value is missing (or not) using `is.na` (which stands for "is not available"). This would be a good way to figure out whether there is some pattern in your missing data.

```
> aggregate(reltco~subsystem, data=temp, FUN=list)
  subsystem reltco
1    AM/OSM   1.0000, 1.0000, 1.0000, 0.5746, 0.5746, 0.5746, 0.0684, 0.0684, 0.0684
2      CMES   0.90, 0.96, 1.00, 0.67, 0.72, 0.80, 0.48, 0.55, 0.60, 0.32, 0.37, 0.40
3  Database                           0.9985, 0.9997, 1.0000, 0.5739, 0.5752, 0.5754
4      DPMS 1.000, 1.000, 1.000, 0.760, 0.776, 0.800, 0.520, 0.540, 0.600, 0.200,
0.200, 0.200
> aggregate(reltco~subsystem, data=temp, FUN=length)
  subsystem reltco
1    AM/OSM      9
2      CMES     12
3  Database      6
4      DPMS     12
> aggregate(reltco~subsystem, data=temp, FUN=sum)
  subsystem reltco
1    AM/OSM 4.9290
2      CMES 7.7700
3  Database 4.7227
4      DPMS 7.5960
> aggregate(reltco~subsystem, data=temp, FUN=is.na)
  subsystem reltco
1    AM/OSM  FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE
2    CMES FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,
FALSE
3  Database  FALSE, FALSE, FALSE, FALSE, FALSE, FALSE
4  DPMS FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,
FALSE
```

CSV Files on the Web

There are many data files stored on the web, and fortunately these are easily accessible from within the R environment if you know the URL where the data is stored. In Section 10.2.5 of "The Art of R Programming" by Matloff, there is a quick example that shows how to access some data that is in the machine learning repository at the University of California

Irvine (UCI). Before we access the data within R, just cut and paste this URL into the address window of your web browser to browse the contents:

```
http://archive.ics.uci.edu/ml/machine-learning-
databases/echocardiogram/echocardiogram.data
```

It looks a lot like a CSV file. Here are the first 10 rows:

```
        11,0,71,0,0.260,9,4.600,14,1,1,name,1,0
        19,0,72,0,0.380,6,4.100,14,1.700,0.588,name,1,0
        16,0,55,0,0.260,4,3.420,14,1,1,name,1,0
        57,0,60,0,0.253,12.062,4.603,16,1.450,0.788,name,1,0
        19,1,57,0,0.160,22,5.750,18,2.250,0.571,name,1,0
        26,0,68,0,0.260,5,4.310,12,1,0.857,name,1,0
        13,0,62,0,0.230,31,5.430,22.5,1.875,0.857,name,1,0
        50,0,60,0,0.330,8,5.250,14,1,1,name,1,0
        19,0,46,0,0.340,0,5.090,16,1.140,1.003,name,1,0
        25,0,54,0,0.140,13,4.490,15.5,1.190,0.930,name,1,0
```

We can read the contents of the file into R as if it were a CSV file, following the code in Matloff. But then, we will also use the `str` command to examine the structure of the data, and then the head command to look at some of the values of the variables:

```
> uci <- "http://archive.ics.uci.edu/ml/machine-learning-databases/"
> uci <- paste(uci,"echocardiogram/echocardiogram.data",sep="")
> ecc <- read.csv(uci)
> str(ecc)
'data.frame':   132 obs. of  13 variables:
 $ X11   : Factor w/ 57 levels "",".03",".25",..: 18 16 54 18 27 14 50 18...
 $ X0    : Factor w/ 4 levels "","?","0","1": 3 3 3 4 3 3 3 3 3 4 ...
 $ X71    : Factor w/ 40 levels "","?","35","46",..: 30 12 17 14 26 19 17 4
...
 $ X0.1  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ X0.260: Factor w/ 74 levels "","?","0.010",..: 65 50 47 26 50 42 59 60
...
 $ X9     : Factor w/ 93 levels "","?","0","10",..: 69 57 13 46 62 56 79 3
...
 $ X4.600: Factor w/ 106 levels "","?","2.32",..: 25 6 54 92 38 85 76 70 ...
 $ X14    : Factor w/ 48 levels "","?","10","10.5",..: 16 16 21 27 8 36 16
...
```

```
 $ X1     : Factor w/ 67 levels "","?","1","1.04",..: 48 3 37 60 3 52 3 11
...
 $ X1.1   : Factor w/ 32 levels "","?","0.140",..: 14 30 25 13 27 27 30 31
...
 $ name  : Factor w/ 3 levels "","?","name": 3 3 3 3 3 3 3 3 3 3 ...
 $ X1.2  : Factor w/ 5 levels "","?","1","2",..: 3 3 3 3 3 3 3 3 3 3 ...
 $ X0.2  : Factor w/ 5 levels "","?","0","1",..: 3 3 3 3 3 3 3 3 3 4 ...
> head(ecc)
  X11 X0 X71 X0.1 X0.260     X9 X4.600  X14    X1  X1.1 name X1.2 X0.2
1  19  0  72    0  0.380      6  4.100   14 1.700 0.588 name    1    0
2  16  0  55    0  0.260      4  3.420   14     1     1 name    1    0
3  57  0  60    0  0.253 12.062  4.603   16 1.450 0.788 name    1    0
4  19  1  57    0  0.160     22  5.750   18 2.250 0.571 name    1    0
5  26  0  68    0  0.260      5  4.310   12     1 0.857 name    1    0
6  13  0  62    0  0.230     31  5.430 22.5 1.875 0.857 name    1    0
```

This is a data frame with 132 rows (observations), each containing 13 variables. Most of the variables are *factors*, but one is an *integer* (int). However, we know that the third column (labeled X71) is the age of each of the people who has received an echocardiogram in this sample. What if we want to figure out the *mean age* of people in this sample?

```
> mean(ecc$X71)
[1] NA
Warning message:
In mean.default(ecc$X71) : argument is not numeric or logical:
returning NA
```

We can't do it. Why? Because we are trying to compute an average value for a *factor*, which is essentially a categorical variable. It's almost like trying to calculate the average color of M&Ms in one bag. But we know that there are ages stored in that column, and we know that we *should* be able to compute an average. What we can do is *cast* these factors to numerical values to compute the mean:

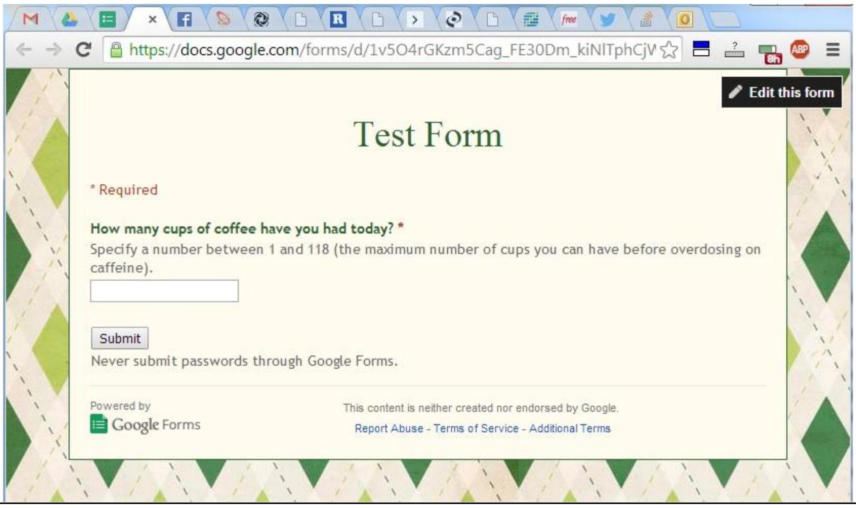
```
> mean(as.numeric(ecc$X71))
[1] 19.23485
```

We're doing exactly the same thing as we did in the last statement, but now we're asking R to treat the values for the variable `X71` in the `ecc` data set as numbers. This is read as "give me the mean of the values inside the variable `X71`, from the data set `ecc`, and treat them as numbers."

Collect Your Own Data: Google Forms and Google Spreadsheets

Many people are now storing data in the cloud, for example, in a Google Spreadsheet. This is a useful system to be familiar with because multiple people can edit the same document, and see each others' changes in real time. So if you are part of a research or analysis team, everyone can contribute to filling in the data, and everyone can access it in their R analysis environment. Furthermore, you can use Google Forms to *automatically populate* a Google Spreadsheet to analyze!

Creating a Google Form is ridiculously easy, however, you must have a Google account before you connect to Google Drive at http://drive.google.com and begin. Once you're in Drive, click on the red "Create" button in the top left corner of your screen, and select "Form" in the dropdown list that appears. You will be prompted to create different types of questions. When you are done, click "Done". Here is the quick test form I just created:


https://docs.google.com/forms/d/1v5O4rGKzm5Cag_FE30Dm_kiNlTphCjV
Edit this form

Test Form

* Required

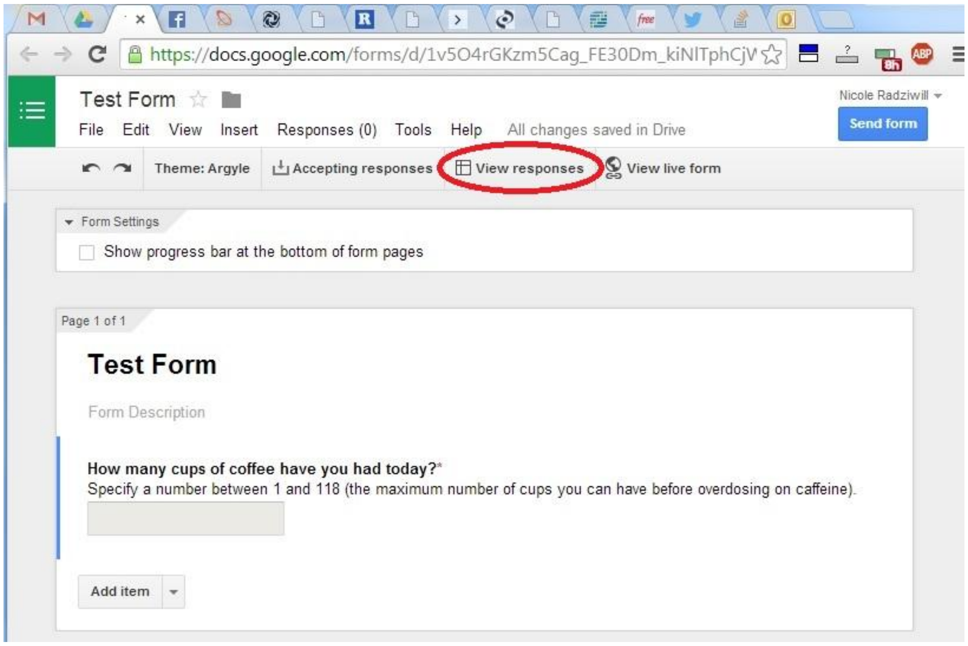
How many cups of coffee have you had today? *
Specify a number between 1 and 118 (the maximum number of cups you can have before overdosing on caffeine).

Submit
Never submit passwords through Google Forms.

Powered by Google Forms

This content is neither created nor endorsed by Google.
Report Abuse - Terms of Service - Additional Terms

When you click "Send Form" you will see a text box containing a URL under "Link to Share" -- that's going to be the link you send to your respondents to *collect data*. When you click "View Responses" in the top menu bar (circled in Figure X.2), it will bring you to the Google Spreadsheet where your responses will automatically show up. *This is the URL you should copy and paste so you can access your data in R using RCurl* (we'll step through that process next!)


https://docs.google.com/forms/d/1v5O4rGKzm5Cag_FE30Dm_kiNlTphCjV
Test Form
Nicole Radziwill
File Edit View Insert Responses (0) Tools Help All changes saved in Drive
Send form
Theme: Argyle
Accepting responses
View responses
View live form
Form Settings
Page 1 of 1
Add item

Show progress bar at the bottom of form pages

Test Form

Form Description

How many cups of coffee have you had today?*
Specify a number between 1 and 118 (the maximum number of cups you can have before overdosing on caffeine).

Analyze Your Own Data: Google Spreadsheets and RCurl

Buried within the list of CRAN packages available to download and utilize in R is a package called `RCurl`. One of the functions `RCurl` allows you to do is request the contents of a URL, process the contents in a way that lets you look at it, and ultimately stores the contents into a variable in R. To demonstrate this, let’s walk through the example.

First, open up an R environment and navigate to the dropdown menu titled `Packages` (Windows) or `Packages & Data` (Mac). Click on `Install Package(s)…` (Windows) or `Package Installer` (Mac). In the list of libraries, look for a library titled `RCurl`. Select it and download it. (If you are a Mac user, remember to install `RCurl`'s dependencies by checking the `Install Dependencies` box.) Now that `RCurl` is downloaded on your computer, load it into R like this:

```
        library("RCurl")
```

If you get an error, check to make sure your quotation marks are in the right place (and closed), and also be sure you are capitalizing both the R and C in `RCurl`.

For this example, we will use another Google Spreadsheet that already has data loaded into it. This is the points spreadsheet for a class I taught in a previous semester. This is a really long URL so the first thing we will do is piece it together using the paste command. (There is no value in doing this OTHER than being able to type the URL in using smaller pieces on multiple lines. Sometimes this is easier than trying to type a huge, long URL into R and not being able to see the whole thing.)

```
url <- "https://docs.google.com/spreadsheet/pub?key="
url <- paste(url," 0AoVN55HxlNvKdHh1S2sxQzRkSHlzLW1kVjREVFh4Z2c", sep="")
url <- paste(url," &single=true&gid=0&output=csv",sep="")
```

Once you have the URL set up as a variable, you can use it to open the Google Spreadsheet and make sure that the data is in the format you desire. We load the contents of our spreadsheet into a variable that we decide to call `spreadsheet.data` using the `getURL` command, and add in the argument `ssl.verifypeer=FALSE` just in case there are permission problems that come from using the secure HTTP protocol, https.

```
        > spreadsheet.data <- getURL(url,ssl.verifypeer=FALSE)
        > str(spreadsheet.data)
         chr "who,what,when,points\nesquildf,RNG
        Exponential,1/27/2014,2\nesquildf,RNG
        Weibull,1/27/2014,1\nesquildf,RNG Triangular,1/27/2014"|
        __truncated__
```

But here's the problem: *all of our data came in as ONE character string, so we will find it impossible to work with unless we create a data frame, somehow*. Notice that it looks kind of like a CSV file, and the first line is indeed a header containing our variable names: `who`, `what`, `when`, `points`. But then you'll notice there are lots of `"\n"` in there too, which is the code for "line feed" (indicating the end of a line in your data file). We need to 1) split that looooong character string into individual lines, separated by the `\n` characters that we see, and 2) then split those lines into individual values that correspond to each of our variables.

We can accomplish the first part using `textConnection`, and the second part using `read.csv` - and lucky for us, this automatically gives us a data frame that we can use:

```
> points <- read.csv(textConnection(spreadsheet.data))
> str(points)
'data.frame':   422 obs. of  4 variables:
 $ who    : Factor w/ 47 levels "ander3ad","ayersjc",..: 12 12 12 12
37 37 ...
 $ what   : Factor w/ 91 levels "ABM Exploration",..: 78 81 80 72 78
81 80 ...
 $ when   : Factor w/ 30 levels "1/27/2014","2/1/2014",..: 1 1 1 1 1 1
1 1 ...
 $ points: int  2 1 1 2 2 1 1 2 2 1 ...
> head(points)
       who            what      when points
1 esquildf RNG Exponential 1/27/2014      2
2 esquildf     RNG Weibull 1/27/2014      1
3 esquildf  RNG Triangular 1/27/2014      1
4 esquildf Output Analysis 1/27/2014      2
5   roseph RNG Exponential 1/27/2014      2
6   roseph     RNG Weibull 1/27/2014      1
```

How would you figure out how many points `roseph` acquired over the course of the semester?

Accessing Real Repositories: APIs, JSON, and rjson

Often, data is not *stored* on the web but is *accessible* from the web. You just have to know how to get it, and how to deal with the data format that you are served. It is not uncommon for organizations to offer access to their databases *if you know a little programming*. They will publish Application Programming Interfaces (or APIs) that provide you with objects and methods to get to their data. Two common data formats are eXtensible Markup Language (XML) and JavaScript Object Notation (JSON). This section covers the JSON format and how to obtain data in JSON format through the R console, using the `rjson` package. (You can explore XML on your own, if you like.) Be sure to install `rjson` first!

This example uses the Weather Underground API which is documented online at http://www.wunderground.com/weather/api/. To use it, you first have to register to gain access and get what's called an "API Key". Services like this often want to make sure that *you are a real human*, and not just a bot who will overload their system with requests for information. Where it says **KEY** is where I used my personal API key:

```
> json.file <-
"http://api.wunderground.com/api/KEY/conditions/q/CA/San_Francisco.json"
> json.data <- fromJSON(paste(readLines(json.file), collapse=""))
> str(json.data)
List of 2
 $ response           :List of 3
  ..$ version       : chr "0.1"
  ..$ termsofService: chr
"http://www.wunderground.com/weather/api/d/terms.html"
  ..$ features      :List of 1
  .. ..$ conditions: num 1
 $ current_observation:List of 56
  ..$ image                  :List of 3
  .. ..$ url  : chr "http://icons.wxug.com/graphics/wu2/logo_130x80.png"
  .. ..$ title: chr "Weather Underground"
  .. ..$ link : chr "http://www.wunderground.com"
  ..$ display_location       :List of 12
  .. ..$ full           : chr "San Francisco, CA"
  .. ..$ city           : chr "San Francisco"
  .. ..$ state          : chr "CA"
  .. ..$ state_name     : chr "California"
.. ..$ country        : chr "US"
.. ..$ country_iso3166: chr "US"
.. ..$ zip            : chr "94101"
.. ..$ magic          : chr "1"
.. ..$ wmo            : chr "99999"
.. ..$ latitude       : chr "37.77500916"
.. ..$ longitude      : chr "-122.41825867"
.. ..$ elevation      : chr "47.00000000"
..$ observation_location   :List of 8
.. ..$ full           : chr "NEMA, San Francisco, California"
.. ..$ city           : chr "NEMA, San Francisco"
.. ..$ state          : chr "California"
```

The `fromJSON` command reads input in JSON format and automatically converts it into a data structure that is accessible from within R. This is a complicated data structure that consists of lists embedded within other lists! Here are some examples of 1) how to access the variables within this data structure created from the JSON data that we acquired, and 2) how to determine the data type for each variable:

```
> json.data$current_observation$display_location$latitude
[1] "37.77500916"
> is.numeric(json.data$current_observation$display_location$latitude)
[1] FALSE
> is.character(json.data$current_observation$display_location$latitude)
[1] TRUE
> json.data$current_observation$observation_time
[1] "Last Updated on August 16, 12:35 PM PDT"
> is.character(json.data$current_observation$observation_time)
[1] TRUE
> json.data$current_observation$temperature_string
[1] "65.8 F (18.8 C)"
> is.character(json.data$current_observation$temperature_string)
[1] TRUE
> json.data$current_observation$temp_f
[1] 65.8
> is.character(json.data$current_observation$temp_f)
[1] FALSE
> is.numeric(json.data$current_observation$temp_f)
[1] TRUE
```

Another very useful capability is to compress individual pieces of the JSON-formatted data into smaller data structures that you can manipulate more easily. For example, if we want to capture *only* the 8 elements of

```
       json.data$current_observation$observation_location
```

into a list, we can do this. Remember, don't type the carets or the plus signs if you are trying to reproduce this in your own R session.

```
> obs.location <-
rep(NA,length(json.data$current_observation$observation_location))
> for (n in 1:length(json.data$current_observation$observation_location)) {
+ obs.location[n] <- json.data$current_observation$observation_location[[n]]
+ }
> obs.location
[1] "NEMA, San Francisco, California" "NEMA, San Francisco"
[3] "California"                      "US"
[5] "US"                              "37.776077"
[7] "-122.417542"                     "102 ft"
```

Now you can refer to individual elements of obs.location - for example, the station elevation can be directly retrieved by calling the last element of the list:

```
> obs.location[8]
[1] "102 ft"
```

Note that you can also cut and paste the entire API call into the address box in your browser. As long as you've typed it in correctly and used a valid API key, you should see the text of the entire JSON object that's been extracted from the Weather Underground database. This is a great way to test and see whether your API call is working, if your R code using `fromJSON` is giving you an error.

Lots of data repositories have APIs that allow you to retrieve data in JSON format, and using the `rjson` package and the `fromJSON` command, you should be able to bring any of this data into R for further analysis. Here are some repositories that might have useful information for you to try acquiring.

- Votes, Membership Lists, Bills, and Schedules from the US Congress: http://developer.nytimes.com/docs/read/congress_api/congress_api_examples
- Opensecrets.org (Center for Responsive Politics): https://www.opensecrets.org/resources/create/api_doc.php
- Federal Communications Commission, TV Station Profiles and Public Inspection Files: https://stations.fcc.gov/developer/
- New York Times API: http://developer.nytimes.com/docs
- There are more examples of government, political, social, weather, sports, and news APIS at http://blog.visual.ly/data-sources/

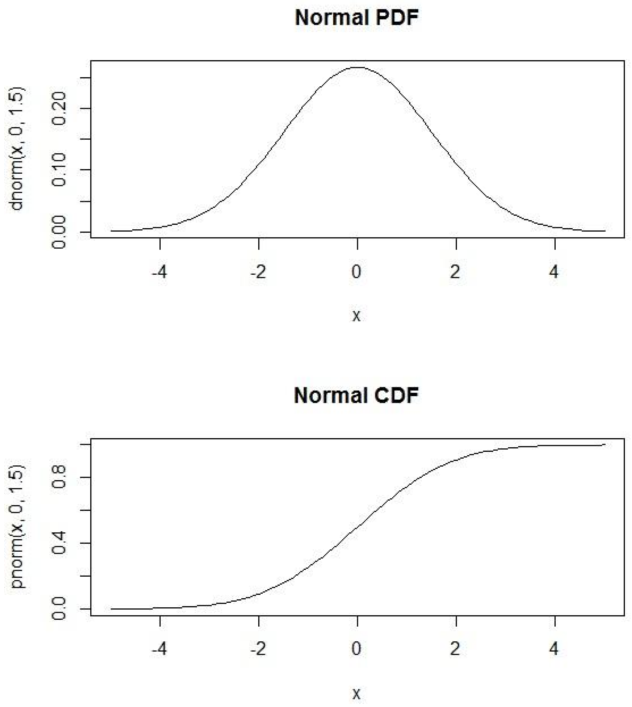
Now What?

You now have the background and experience to try playing with any of the data files at http://archive.ics.uci.edu/ml/machine-learning-databases/, or to explore using conditional formatting of Google Spreadsheets... where you can do cool things like turn boxes different colors based on the data in the cells, or selectively execute mathematical operations on the data. There are tons of opportunities here. Get started with these examples:

- http://thejournal.com/articles/2014/03/19/extending-conditional-formatting-in-google-sheets-using-dynamic-date-calls.aspx
- See the most recent YouTube videos on conditional formatting by going to https://www.youtube.com/results?search_query=conditional+formatting+google+docs&filters=year&search_sort=video_avg_rating

1.7 PDFs and CDFs

When an outcome is random, how do you mathematically describe the scope of all possibilities? You can do this by characterizing the distribution of all those possibilities as a probability density function (PDF). The PDF illustrates: *how densely are your observations packed around a particular value?* The cumulative distribution function (or CDF) complements the PDF, and provides additional information. It shows what proportion of all observations will be less than or equal to a certain value. When you think of the normal distribution as a bell curve, you're thinking about the PDF.



```
> x <- seq(-5,5,0.1) # Generate a sequence of x-values from -5 to +5
> par(mfrow=c(2,1)) # Plot them as 2 rows in 1 column
> plot(x,dnorm(x,0,1.5),type="l",main="Normal PDF")
> plot(x,pnorm(x,0,1.5),type="l",main="Normal CDF")
```

A probability density just tells you, for any given x-value, the long-run relative frequency of seeing a particular outcome occur. The CDF is the integral of the PDF:

$$\int_{-\infty}^{\infty} PDF = CDF$$

Because integration means to *add up all the areas* underneath a function, we can think about it this way. Pretend you are a tiny person standing in front of the normal PDF. The normal PDF is on a stage, and there's a stage curtain hanging on rings along the top edge of the graph. We're going to start at x=-4 and walk along the x-axis towards the right, all the way to x=+4. As we do, we're going to *hold the curtain and pull it more and more closed* as we walk. How much of the area under the normal PDF are we covering as we walk by? Let's *visually integrate* to construct the CDF:

- When you just start walking, and you go from x=-4 to x=-2, you're only capturing a tiny bit of the area under the normal PDF. The y-values on the CDF start at 0, because you have not covered any of the area under the normal PDF yet. The y-value at x=-2 will be about 0.16, since that's how much area we have covered by the time we have walked to x=-2.
- As you walk from x=-2 to x=0, you're capturing the area at a *faster and faster rate*. By the time you get to x=0, you have captured 50% of the area under the normal PDF. The slope on the normal CDF thus increases to reflect that we're capturing the area faster. The y-value of the normal CDF at x=0 is 0.5, to reflect that we have captured half the area.
- As we keep walking to the right, from x=0 to x=+2, we are capturing area at a slower rate. The slope of the CDF begins to decrease. When we get to x=+2, the y-value on the CDF will be 0.84, reflecting that we have covered 84% of the area under the normal PDF by this point.
- Finally, as we walk from x=+2 to x=+4, we are adding area at a much, much slower rate. The slope on the normal CDF decreases to reflect that. By the time we've

walked all the way across the stage, we have covered 100% of the area under the normal PDF, and the y-value on the normal CDF levels off at 1.0.

The probability density function, then, represents the *rate of change* of the cumulative distribution. It is the *derivative* of the CDF. If you know the equation of either the PDF or the CDF, you can determine the equation for the other function by integrating or differentiating, respectively.

You can *visually differentiate* a CDF by pretending you're a tiny person walking from the leftmost position on the x-axis to the rightmost position. All you have to do to create a PDF is to eyeball the slope of the CDF, and plot that value on the y-axis of the PDF you are creating. The area under a PDF is, by definition, 1. **The range of values on the y-axis of the CDF is always 0.0 to 1.0. This is a very convenient property** that enables us to create *inverse transform equations* to generate random numbers from any target distribution we want. Fortunately there are several functions built into R which make it super easy to work with PDFs and CDFs. All you need to know is the value of the parameters that uniquely specify any given distribution. The normal distribution is fully specified by its mean (which shows where the center of the distribution is at) and standard deviation (which shows how spread out the values are).

R Command	**What it does**
`rnorm(n,mean,sd)`	Generates n **random variates** selected from a normal distribution with specified mean and standard deviation
`dnorm(x,mean,sd)`	Plots the **PDF** from a collection of values x (which represent various points along the x-axis) centered at the specified mean, and with the specified standard deviation
`pnorm(x,mean,sd)`	Plots the **CDF** from a collection of values x (which represent various points along the x-axis) centered at the specified mean, and with the specified standard deviation
`qnorm(area,mean,sd)`	Finds the quantiles, or **Inverse CDF**. If you know an area under the normal PDF to the left of a particular x-value, this command helps you find the number of standard deviations

	above or below the mean where the x-value that forms that boundary sits. (In conjunction with a random number generator, this function can you generate random numbers that were pulled from a distribution with characteristics that you specify.)

Common Distributions

There are PDF and CDF functions for *many* common distributions in R. Each of them works just like `rnorm`, `dnorm`, `pnorm`, and `qnorm`, only you need to know the parameters that uniquely specify that other type of distribution. For example, the exponential distribution is uniquely specified by just one parameter: the *mean* of the distribution. Here are several (but not all!) of the distributions built into R that you can use:

Distribution	**R Functions**
`Beta`	pbeta, qbeta, dbeta, rbeta
`Binomial`	pbinom, qbinom, dbinom, rbinom
`Cauchy`	pcauchy, qcauchy, dcauchy, rcauchy
`Chi-Square`	pchisq, qchisq, dchisq, rchisq
`Exponential`	pexp, qexp, dexp, rexp
`F`	pf, qf, df, rf
`Gamma`	pgamma, qgamma, dgamma, rgamma
`Geometric`	pgeom, qgeom, dgeom, rgeom
`Hypergeometric`	phyper, qhyper, dhyper, rhyper
`Logistic`	plogis, qlogis, dlogis, rlogis
`Log Normal`	plnorm, qlnorm, dlnorm, rlnorm
`Negative Binomial`	pnbinom, qnbinom, dnbinom, rnbinom
`Normal`	pnorm, qnorm, dnorm, rnorm
`Student's t`	pt, qt, dt, rt
`Uniform`	punif, qunif, dunif, runif
`Weibull`	pweibull, qweibull, dweibull, rweibull

Other Resources

- There is an amazing list of many probability distributions on Wikipedia at http://en.wikipedia.org/wiki/List_of_probability_distributions. In addition, each statistical distribution (e.g. normal, uniform, Weibull) has its own Wikipedia page that includes PDFs, CDFs, and a wealth of other information about that distribution. These are fantastic pages and I encourage you to get to know them.
- There is a fantastic interactive PDF/CDF explorer that you NEED to play with at http://www.che.utah.edu/~tony/course/material/Statistics/18_rv_pdf_cdf.php
- A comparison of `qnorm()` and `pnormGC()` is located here: http://cran.r-project.org/web/packages/tigerstats/vignettes/qnorm.html

1.8 Z-Score Problems with the Normal Model

Objective

Lots of data in the world is naturally distributed *normally*, with most of the values falling around the mean, but with some values less than (and other values greater than) the mean. A lot *more* data is distributed normallyWhen you have data that is distributed normally, you can use the normal model to answer questions about the characteristics of the entire population. That's what we'll do in this chapter. You will learn about:

- The N notation for describing normal models
- What z-scores mean
- The 68-95-99.7 rule for approximating areas under the normal curve
- How to convert each element of your data set into z-scores
- How to answer questions about the characteristics of the entire population

The Normal Model and Z-Scores

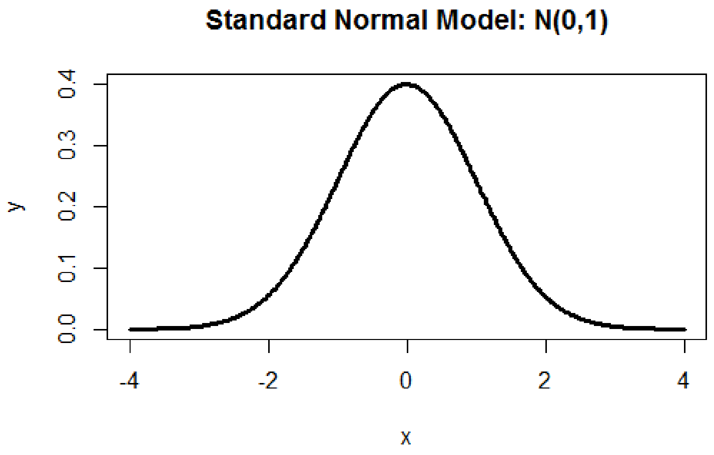
The normal model provides a way to characterize how *frequently* different values will show up in a population of lots of values. You can describe a normal model like this:

$$N(\mu, \sigma)$$

Here's what you SAY when you see this: "The normal model with a mean of μ and a standard deviation of σ." There is no way for you to mathematically break this statement down into something else. It's just a shorthand notation that tells us *we're dealing with a normal model here, here are the two values that uniquely characterize the shape and position of that bell curve.* To produce that bell curve requires an equation (called the *probability density function* or pdf):

$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

This may look complicated at first, but it's not. The left hand side says that the normal model is a function (f) of three variables: x, μ, and σ. Which makes sense: we have to plot some value on the vertical (y) axis based on lots of x-values that we plug into our equation, and the shape of our bell curve is going to depend on the mean of the distribution μ (which tells us how far to the right or left on the number line we should slide our bell curve) and the standard deviation σ (which tells us how fat or skinny the bell will be... bigger standard deviation = more dispersion in the distribution = fatter bell curve). When the mean is 0 and the standard deviation is 1, this is referred to as the *standard normal model*. It looks like this, and was produced by the code below.



```
x <- seq(-4,4,length=500)
y <- dnorm(x,mean=0,sd=1)
plot(x,y,type="l",lwd=3,main="Standard Normal Model: N(0,1)")
```

The first line just produces 500 x values for us to work with. The second line creates 500 y values from those x values, produced by the dnorm command (which stands for "`d`ensity of the `norm`al model"). Because `dnorm` contains the equation of the normal model, we don't actually have to write out the whole equation. Now we have 500 (x,y) pairs which we can use to plot the standard normal model, using a type of "`l`" to make it a line, and a line width (using `lwd=3`) to make it a little thicker (and thus easier to see) than if we used a line width of only one pixel.

The z-score tells us *how many standard deviations above or below the mean* a particular x-value is. You can calculate the z-score for any one of your x-values like this:

$$z = \frac{x - \mu}{\sigma}$$

The z-score describes what the difference is between your data point (x) and the mean of the distribution (μ), scaled by how skinny or fat the bell curve is (σ). The z-score of the *mean* of your distribution, then, will be zero - because if x equals the mean, x - μ will be zero and the z-score will be zero. So, ALWAYS:

- Positive z-scores are associated with data points that are ABOVE the mean
- Negative z-scores are associated with data points that are BELOW the mean

Consider an example where we're thinking about the distribution of several certification exam scores: the ASQ Certified Six Sigma Black Belt (CSSBB) exam from December 2014. Let's say, hypothetically, that we know the population of all scores for this exam can be described by the normal model with a mean of 78 and a standard deviation of 5:

$$N(78, 5)$$

There are a LOT of things we know about the test scores simply by knowing what model represents the data. For example:

- The test score that is one standard deviation *below* the mean is 73 (which we get by taking the mean, 78, and *subtracting* one standard deviation of 5). This test score of x=73 corresponds to a z-score of -1.
- The test score that is one standard deviation *above* the mean is 83 (which we get by taking the mean, 78, and *adding* one standard deviation of 5). This test score of x=83 corresponds to a z-score of +1.

- The test score that is two standard deviations ***below*** the mean is 68 (which we get by taking the mean, 78, and *subtracting* two times the standard deviation of 5, which is 10). This test score of x=68 corresponds to a z-score of -2.
- The test score that is two standard deviations ***above*** the mean is 88 (which we get by taking the mean, 78, and *adding* two times the standard deviation of 5, which is 10). This test score of x=88 corresponds to a z-score of +2.
- The test score that is three standard deviations ***below*** the mean is 63 (which we get by taking the mean, 78, and *subtracting* three times the standard deviation of 5, which is 15). This test score of x=63 corresponds to a z-score of -3.
- The test score that is three standard deviations ***above*** the mean is 93 (which we get by taking the mean, 78, and *adding* three times the standard deviation of 5, which is 15). This test score of x=93 corresponds to a z-score of +3.

Let's say YOU scored an 85. (There's no way to actually know this, because the certification administrators don't reveal any information about the CSSBB exam beyond whether you passed it or not.) What's your z-score? It's easy to calculate:

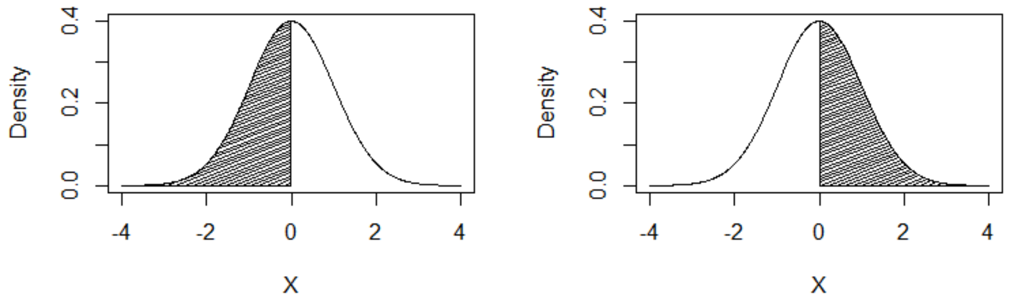
$$z = \frac{x - \mu}{\sigma} = \frac{85 - 78}{5} = \frac{7}{5} = 1.4$$

A z-score of +1.4 means that your test score was 1.4 standard deviations *above* the mean of 78. There is also other information that we can find out by knowing what normal model represents the scores of all test-takers.

For example, we know that a very tiny portion of the test-takers (in fact, only 0.3%) scored either above a 93, or below a 63. We can also show that your score of 85% was better than 91.9% of all test-takers. But how??

The 68-95-99.7 Rule

The *area* under the normal curve reflects the *probability* that an observation will fall within a particular interval. Area = Probability! There are a couple simple things that you can memorize about the normal model that will help you double-check any problem solving you do with it. Called the *empirical rule*, this will help you remember how much of the area under the bell curve falls between different z-scores. First, think about how the normal model is *symmetric*... if you fold it in half (from left to right) at the mean, the curve is a mirror image of itself. The right half of the bell is exactly the same shape and size as the left half. (The code to produce these charts is below the images.)

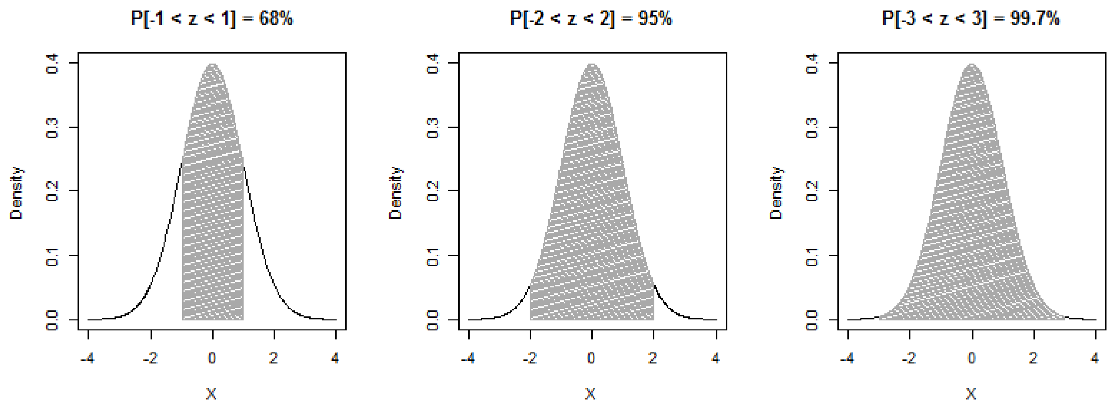


```
url <- "https://raw.githubusercontent.com/NicoleRadziwill/"
url <- paste(url, "R-Functions/master/shadenorm.R", sep="")
# Note: You need to download sourceHttps.R before the next two
# lines will work. Find out how in the Appendix on sourceHttps!
source("sourceHttps.R")
source_https(url)
par(mfrow=c(1,2))
shadenorm(between=c(-4,0),color="black")
shadenorm(between=c(0,4),color="black")
```

Because the total area under the normal curve is 100%, this also means that 50% of the area under the curve is to the *left* of the mean, and the remaining 50% of the area under the curve is to the *right* of the mean. **The 68-95-99.7 Empirical Rule provides even more information:**

- 68% of your observations will fall between one standard deviation below the mean (where z = -1) and one standard deviation above the mean (where z = +1)
- 95% of your observations will fall between two standard deviations below the mean (where z = -2) and two standard deviations above the mean (where z = +2)
- 99.7% (or pretty much ALL!) of your observations will fall between three standard deviations below the mean (where z = -3) and three standard deviations above the mean (where z = +3)

Here's what those areas look like. You read "P[-1 < z < 1]" as "the probability that the z-score will fall between -1 and +1".



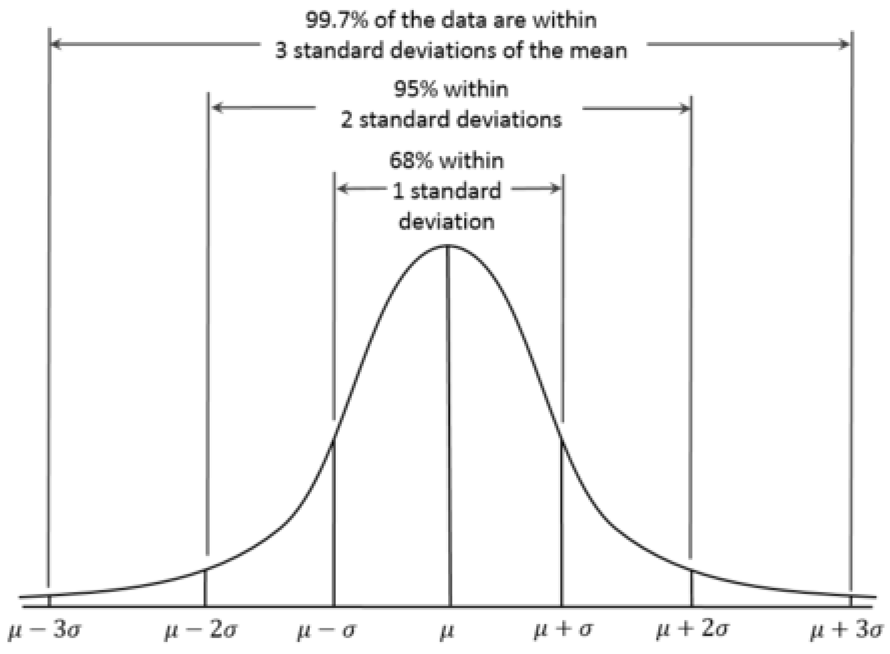
```
par(mfrow=c(1,3)) # set up the plot area with 1 row, 3 columns
shadenorm(between=c(-1,+1),color="darkgray")
title("P[-1 < z < 1] = 68%")
shadenorm(between=c(-2,+2),color="darkgray")
title("P[-2 < z < 2] = 95%")
shadenorm(between=c(-3,+3),color="darkgray")
title("P[-3 < z < 3] = 99.7%")
```

These graphs show that:

- There is a *probability of 68%* that an observation will fall between one standard deviation below the mean (where z = -1) and one standard deviation above the mean (where z = +1).
- There is a *probability of 95%* that an observation will fall between two standard deviations below the mean (where z = -2) and two standard deviations above the mean (where z = +2)
- There is a *probability of 99.7%* that an observation will fall between three standard deviations below the mean (where z = -3) and three standard deviations above the mean (where z = +3)

When data are distributed normally, there is only a VERY TINY (0.3%!) chance that an observation will be smaller than whatever value is three standard deviations below the mean, or larger than three standard deviations above the mean! **Nearly all values will be within three standard deviations of the mean.** That's one of the reasons why you can use the z-score for a particular data point to figure out just how common or uncommon that value is.

The chart for the 68-95-99.7 rule as presented on Wikipedia is shown on the next page (it's from http://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule). From the 68-95-99.7 rule, we can estimate what proportion of the population will have scored below our certification score of 85, compared to the normal model with the mean of 78 and the standard deviation of 5, or N(78,5).



The 68-95-99.7 Rule is Great, But Prove It To Me

When you *integrate* a function, you are computing the area under the curve. So if we integrate the equation for the normal model between z=-1 and z=+1, we should get an area of 68%. Let's do that. First we take the equation of the normal probability distribution function:

$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

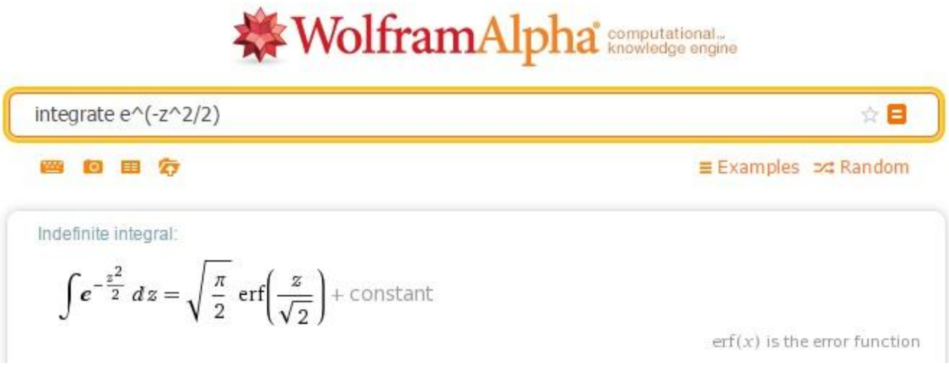
Then simplify it using the *standard normal model* of N(0,1) which is centered at a mean (μ) of 0, with a standard deviation (σ) of 1. Meaning, plug in 0 for μ and 1 for σ. You get:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

Now, let's integrate it from a z-score of -1 to a z-score of +1 to find the area between those left and right boundaries. We can pull the first fraction outside the integral since it's a constant:

$$\frac{1}{\sqrt{2\pi}} \int_{-1}^{1} e^{-x^2/2} dx$$

How do we integrate this expression? My solution (since I'm not a mathematician) is to look at a table of integrals, or use the Wolfram Alpha computational engine at http://www.wolframalpha.com. All we need to do is figure out how to evaluate the stuff on the right side of the integral, then multiply it by one over the square root of 2π. I'll show you what I typed into Wolfram to make it determine the integral for me:



integrate e^(-z^2/2)

Indefinite integral:

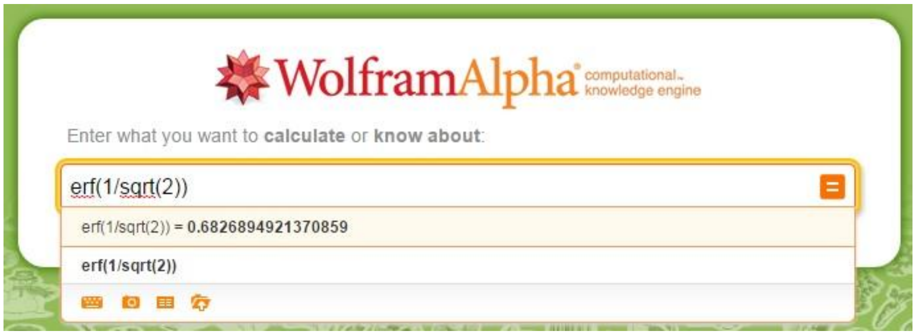
$$\int e^{-\frac{z^2}{2}} dz = \sqrt{\frac{\pi}{2}} \operatorname{erf}\left(\frac{z}{\sqrt{2}}\right) + \text{constant}$$

erf(x) is the error function

The evaluated integral contains something called `erf`, the "error function". This is a special function that (fortunately) Wolfram knows how to evaluate as well. Let's plug the result from evaluating this integral back into our most recent expression. That vertical bar on the right hand side means "evaluate the error function of x over the square root of 2 using x=1, then subtract off whatever you get when you evaluate the error function of x over the square root of 2 using x=-1".

$$= \frac{1}{\sqrt{2\pi}} \sqrt{\frac{\pi}{2}}\, erf \frac{x}{\sqrt{2}} \Bigg|_{-1}^{1}$$

We can simplify all the stuff on the left hand side of `erf` because they are all constants... it reduces to a very nice and clean 1/2. So we just need to take the difference between evaluating the error function at x=1, and evaluating the error function at x=-1, and then chop it in half to get our answer. Wolfram will help:


WolframAlpha computational knowledge engine
Enter what you want to calculate or know about:

```
erf(1/sqrt(2))
erf(1/sqrt(2)) = 0.6826894921370859
erf(1/sqrt(2))
```

All we had to do was type in `erf(1/sqrt(2))` and Wolfram evaluates the right hand side of our expression at x=1, giving us approximately 0.683. If we do this again using x=-1, we'll get a value of -0.683. Now let's plug it all in together:

$$= \frac{1}{2}(0.683 - (-0.683)) = 0.683$$

The area under the standard normal curve between -1 and +1 is 0.683, or 68.3%... nearly the same value that we get from our "rule of thumb" 68-95-99.7% rule! You can try this same process to determine the area under the normal between -2 and +2, or between -3 and +3, to further confirm the empirical 68-95-99.7% rule for yourself.

Calculating All of the Z-Scores for a Data Set

There may come a time where you would like to easily compute the z-scores for each element in a data set that's normally (or nearly normally) distributed. You *could* take each value individually and use this equation to compute the z-scores one by one:

$$z = \frac{x - \mu}{\sigma}$$

Or you could just enter your data set into R:

```
      scores <- c(81, 91, 78.5, 73.5, 66, 83.5, 76, 81, 68.5, 83.5)
```

And then have it compute all the z-scores for you at once, using the `scale` command:

```
> scale(scores)
             [,1]
 [1,]  0.36689321
 [2,]  1.70105036
 [3,]  0.03335393
 [4,] -0.63372464
 [5,] -1.63434250
 [6,]  0.70043250
 [7,] -0.30018536
 [8,]  0.36689321
 [9,] -1.30080321
[10,]  0.70043250
```

Do these values make sense? Let's check. The mean of our test scores is around 78, so all the scores above 78 should have positive z-scores, and all the scores below 78 should have negative z-scores. We see by examining the original data that scores 1, 2, 3, 6, 8, and 10 are all above the mean, and so should have z-scores that are positive. The output from scale confirms this expectation. We can also see that the third value of 78.5 is just slightly above the mean, so its z-score should be very tiny and positive. It is, at 0.0333.

Using the Normal Model to Answer Questions About a Population

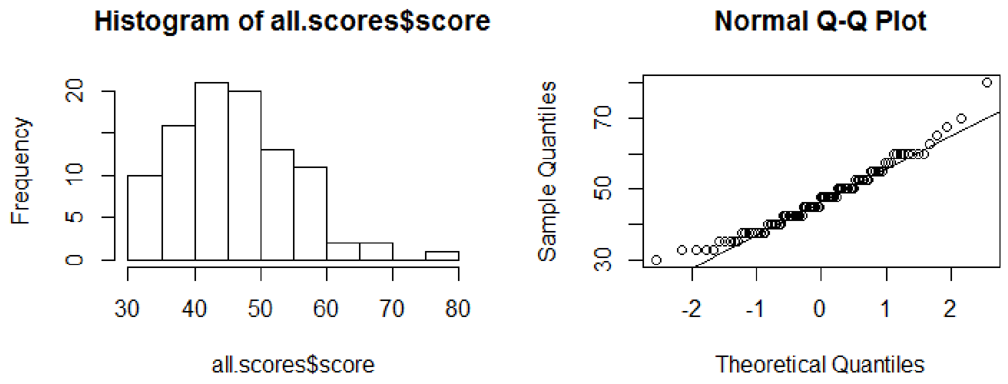
For this collection of examples, we'll use real exam scores from a test I administered last year. You can get my data directly from GitHub as long as you have the `RCurl` package installed. Here's what you do with it:

```
library(RCurl)
url <- "https://raw.githubusercontent.com/NicoleRadziwill"
url <- paste(url,"/Data/master/compare-scores.csv", sep="")
data <- getURL(url,ssl.verifypeer=FALSE)
all.scores <- read.csv(textConnection(data))
```

If the code above has successfully found and retrieved the data, you should be able to see the semester when the students took the test (in the `when` variable) and the raw scores (stored in the `score` variable) when you use `head`. There are 96 observations in this dataset.

```
> head(all.scores)
  when score
1 FA14  45.0
2 FA14  55.0
3 FA14  42.5
4 FA14  37.5
5 FA14  30.0
6 FA14  47.5
```

First, we should check and see whether the scores are approximately normally distributed. We can do this by plotting a histogram, and by doing a QQ plot which (if are scores *are* nearly normal) should show all of our data points nearly along the diagonal. QQ plots and tests for normality are covered more extensively in Chapter 2.8.

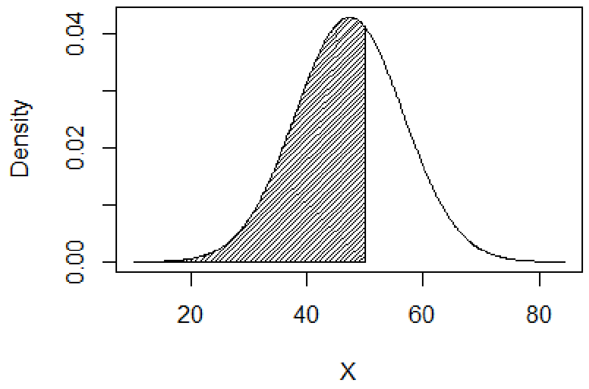


```
par(mfrow=c(1,2)) # set up the plot area with 1 row, 2 columns
hist(all.scores$score)
qqnorm(all.scores$score)
qqline(all.scores$score)
```

The histogram is skewed a little to the right, but it's nearly normal, so we can proceed. To figure out what normal model can be used to represent the data, we need to know the mean and standard deviation of the scores:

```
> mean(all.scores$score)
[1] 47.29167
> sd(all.scores$score)
[1] 9.309493
```

Rounding a bit, we should be able to use N(47.3,9.3) (or "the normal model with a mean of 47.3 and a standard deviation of 9.3") to represent the distribution of all our scores. Using this model, we can answer a lot of questions about what the population of test-takers looks like. Looking at the histogram, we can see that a score of 50 is about in the middle. **What proportion of students got below a 50?** We can answer this question by determining the area under N(47.3,9.3) to the LEFT of x=50. It looks like this:



```
shadenorm(below=50,justbelow=TRUE,color="black",mu=47.3,sig=9.3)
```

Since the mean is 47.3, we know that a test score of 50 is TO THE RIGHT OF THE MEAN. The z-score associated with 50 is going to be *positive*. How positive will it be? Well, since the standard deviation is 9.3, we know that the test score which is one standard deviation above the mean will be 47.3 + 9.3 = 56.6. Our test score of 50 is just a little bit above the mean, so we can estimate our z-score at +0.3 or +0.4. That means the area under the normal to the left of x=50 will be greater than 50%, but not much greater than 50%. Even before we do the problem, we can estimate that our answer should be between 55% and 65%.

To definitively determine the area below the curve to the left of x=50, we use the `pnorm` function in R. The `pnorm` function ALWAYS tells us the area under the normal curve to the LEFT of a particular x value (remember this!!) So we can ask it to tell us the area to the left of x=50, given a normal model of N(47.3,9.3):

```
> pnorm(50,mean=47.3,sd=9.3)
[1] 0.6142153
```

We can predict that 61.4% of the test-takers *in the population* received a score greater than 50%. This means even though our data set only includes students from a couple of semesters of *my* class, we've found a way to use this sample to determine what the scores from the *entire population* of students who took this test must be! As long as my students are representative of the larger population, this should be a pretty good bet.

(But what if you don't have R? Well, don't worry, you can still use "Z Score Tables" or Z Score Calculators to figure out the area underneath the normal curve. Z Score Tables are available in the back of most statistics textbooks, and tables and calculators are also available online. Let's do the same problem we just did, AGAIN, using tables and calculators.)

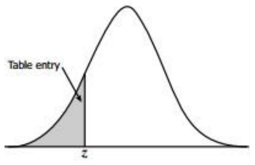
Let's say we had to do this problem with a Z Score Table. First Rule of Thumb: **ALWAYS PICK A Z SCORE TABLE THAT HAS A PICTURE.**

- The table at http://www.stat.ufl.edu/~athienit/Tables/Ztable.pdf HAS a picture. Use this kind of table!
- The table at http://www.utdallas.edu/dept/abp/zscoretable.pdf DOES NOT HAVE a picture. DO NOT USE these kind of tables.

It's best to use Z Score Tables that have pictures so you can *match the picture representing the area under the curve you're trying to find* with the picture. To find the area under the curve, you need a z-score. The z-score that corresponds with a test score of x=50 is

$$z = \frac{x - \mu}{\sigma} = \frac{50 - 47.3}{9.3} = 0.29$$

When we look at the picture we drew, we notice that the shaded portion is bigger than 50% of the total area under the curve. When we look at the picture at the Z Score Table from http://www.stat.ufl.edu/~athienit/Tables/Ztable.pdf, we notice that it does NOT look like what we drew:



This particular Z Score Table ONLY contains areas within the tails. The trick to using a Z Score Table like this is to recognize that because the normal distribution is symmetric, the area to the LEFT of z=+0.29 can be found by taking 100% of the area, and *subtracting* the area to the LEFT of the z-score at z=-0.29 (what's in the area of the tail). Using the Z Score Table from http://www.stat.ufl.edu/~athienit/Tables/Ztable.pdf, we look in the row containing z=-0.2, and the column containing .09, because these add up to our computed z-score of 0.29. We get an area of 0.3859. But we're looking for an area greater than 50% (which we know because we drew a PICTURE!), so we take 1 - 0.3859 to get 0.6141, or **61.4%.**

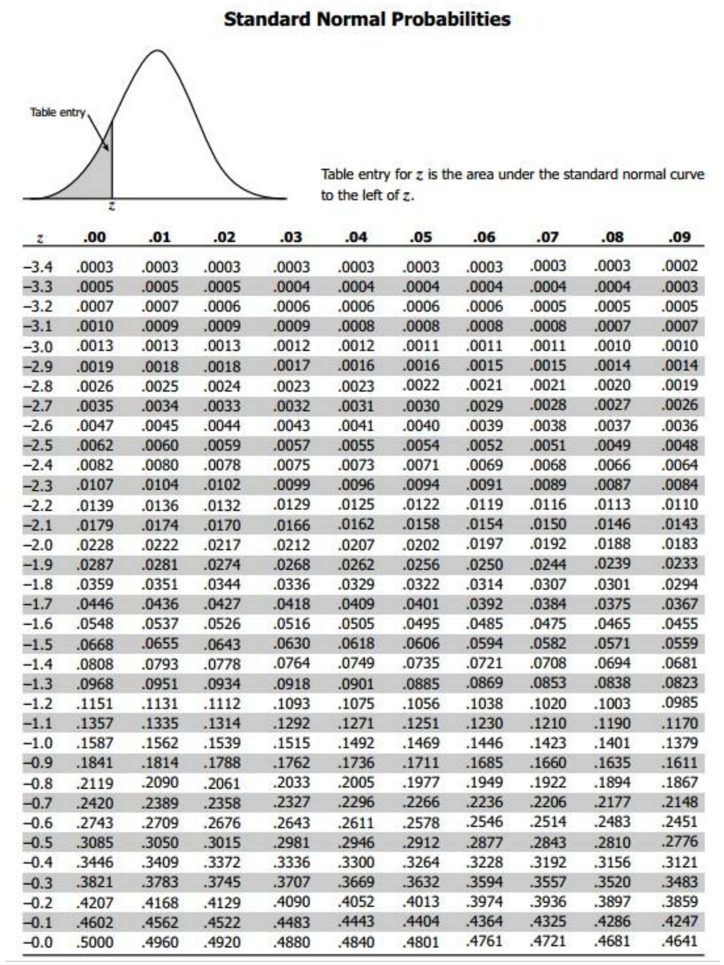


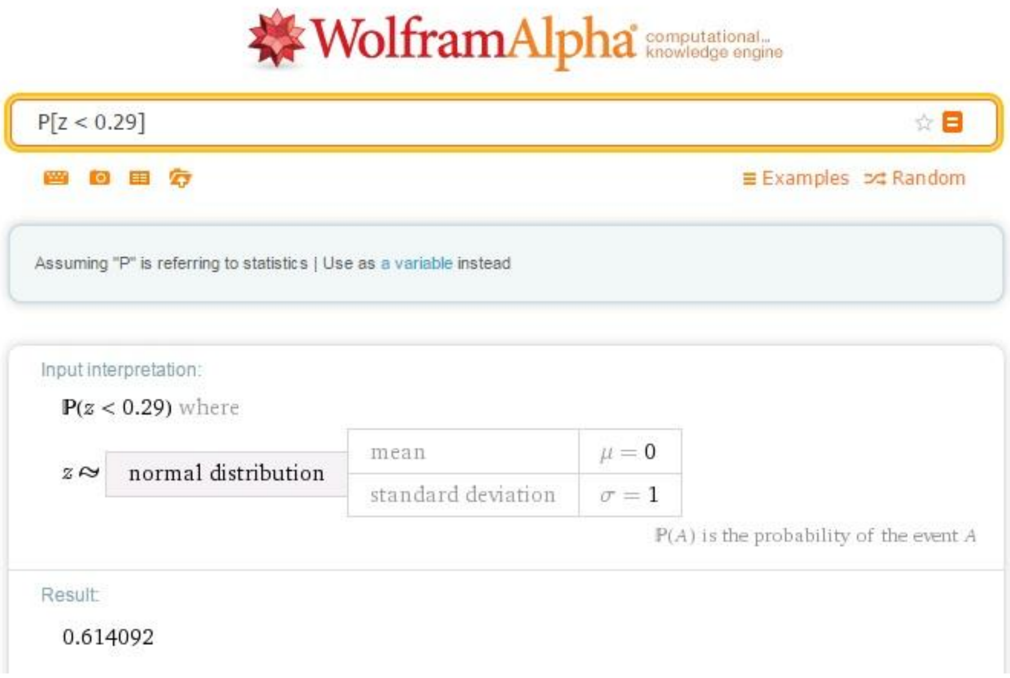
Table entry
z

Standard Normal Probabilities

Table entry for z is the area under the standard normal curve to the left of z.

z	.00	.01	.02	.03	.04	.05	.06	.07	.08	.09
−3.4	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0002
−3.3	.0005	.0005	.0005	.0004	.0004	.0004	.0004	.0004	.0004	.0003
−3.2	.0007	.0007	.0006	.0006	.0006	.0006	.0006	.0005	.0005	.0005
−3.1	.0010	.0009	.0009	.0009	.0008	.0008	.0008	.0008	.0007	.0007
−3.0	.0013	.0013	.0013	.0012	.0012	.0011	.0011	.0011	.0010	.0010
−2.9	.0019	.0018	.0018	.0017	.0016	.0016	.0015	.0015	.0014	.0014
−2.8	.0026	.0025	.0024	.0023	.0023	.0022	.0021	.0021	.0020	.0019
−2.7	.0035	.0034	.0033	.0032	.0031	.0030	.0029	.0028	.0027	.0026
−2.6	.0047	.0045	.0044	.0043	.0041	.0040	.0039	.0038	.0037	.0036
−2.5	.0062	.0060	.0059	.0057	.0055	.0054	.0052	.0051	.0049	.0048
−2.4	.0082	.0080	.0078	.0075	.0073	.0071	.0069	.0068	.0066	.0064
−2.3	.0107	.0104	.0102	.0099	.0096	.0094	.0091	.0089	.0087	.0084
−2.2	.0139	.0136	.0132	.0129	.0125	.0122	.0119	.0116	.0113	.0110
−2.1	.0179	.0174	.0170	.0166	.0162	.0158	.0154	.0150	.0146	.0143
−2.0	.0228	.0222	.0217	.0212	.0207	.0202	.0197	.0192	.0188	.0183
−1.9	.0287	.0281	.0274	.0268	.0262	.0256	.0250	.0244	.0239	.0233
−1.8	.0359	.0351	.0344	.0336	.0329	.0322	.0314	.0307	.0301	.0294
−1.7	.0446	.0436	.0427	.0418	.0409	.0401	.0392	.0384	.0375	.0367
−1.6	.0548	.0537	.0526	.0516	.0505	.0495	.0485	.0475	.0465	.0455
−1.5	.0668	.0655	.0643	.0630	.0618	.0606	.0594	.0582	.0571	.0559
−1.4	.0808	.0793	.0778	.0764	.0749	.0735	.0721	.0708	.0694	.0681
−1.3	.0968	.0951	.0934	.0918	.0901	.0885	.0869	.0853	.0838	.0823
−1.2	.1151	.1131	.1112	.1093	.1075	.1056	.1038	.1020	.1003	.0985
−1.1	.1357	.1335	.1314	.1292	.1271	.1251	.1230	.1210	.1190	.1170
−1.0	.1587	.1562	.1539	.1515	.1492	.1469	.1446	.1423	.1401	.1379
−0.9	.1841	.1814	.1788	.1762	.1736	.1711	.1685	.1660	.1635	.1611
−0.8	.2119	.2090	.2061	.2033	.2005	.1977	.1949	.1922	.1894	.1867
−0.7	.2420	.2389	.2358	.2327	.2296	.2266	.2236	.2206	.2177	.2148
−0.6	.2743	.2709	.2676	.2643	.2611	.2578	.2546	.2514	.2483	.2451
−0.5	.3085	.3050	.3015	.2981	.2946	.2912	.2877	.2843	.2810	.2776
−0.4	.3446	.3409	.3372	.3336	.3300	.3264	.3228	.3192	.3156	.3121
−0.3	.3821	.3783	.3745	.3707	.3669	.3632	.3594	.3557	.3520	.3483
−0.2	.4207	.4168	.4129	.4090	.4052	.4013	.3974	.3936	.3897	.3859
−0.1	.4602	.4562	.4522	.4483	.4443	.4404	.4364	.4325	.4286	.4247
−0.0	.5000	.4960	.4920	.4880	.4840	.4801	.4761	.4721	.4681	.4641

Let's say we don't have a Z Score Table handy, and we don't have R. What are we to do? You can look online for a Z Score Calculator which should also give you the same answer. I always use Wolfram. There are so many Z Score Calculators out there... and only about half of them will give you the right answers. It's really sad! But Wolfram will give you the right answer, and it also asks you to specify what area you're looking for using very specific terminology. So I can ask Wolfram "What's the area under the normal curve to the left of z=0.29?" like this:



P[z < 0.29]

Assuming "P" is referring to statistics | Use as a variable instead

Input interpretation:

P(z < 0.29) where

z ≈	normal distribution	mean	$\mu = 0$
		standard deviation	$\sigma = 1$

P(A) is the probability of the event A

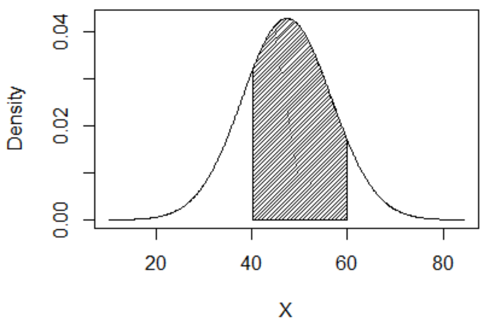
Result:

0.614092

The area is 0.614, or 61.4% - the same as we got from the Z Score Table and the `pnorm` calculation in R.

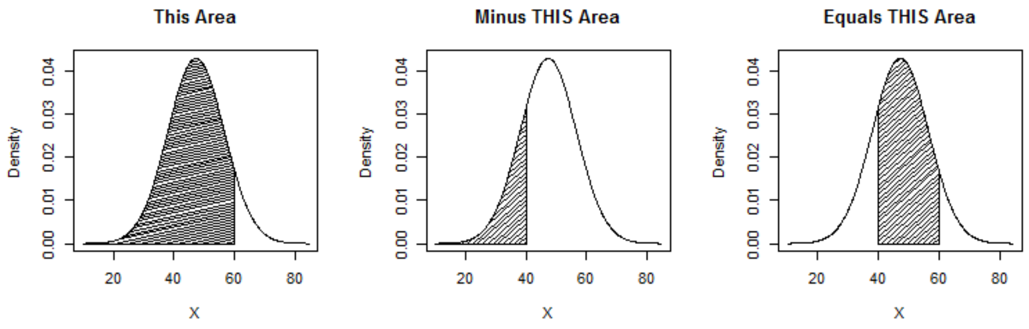
Let's Do Another Z Score Problem

Say, instead, we wanted to figure out what proportion of our students scored between 40 and 60. That means we want to find the area under N(47.4, 9.3) between x=40 and x=60. If we draw it, it will look like this:



```
shadenorm(between=c(40,60),color="black",mu=47.3,sig=9.3)
```

To calculate this area, we'll have to take *all the area to the left of 60* and subtract off *all the area to the left of 40*, because `pnorm` and Z Score Calculators don't let us figure out "areas in between two z values" directly. So let's do that. Graphically, we'll take the total area in the left graph below, and subtract off the area of the right graph in the middle, which will leave us with the area in the graph on the right:

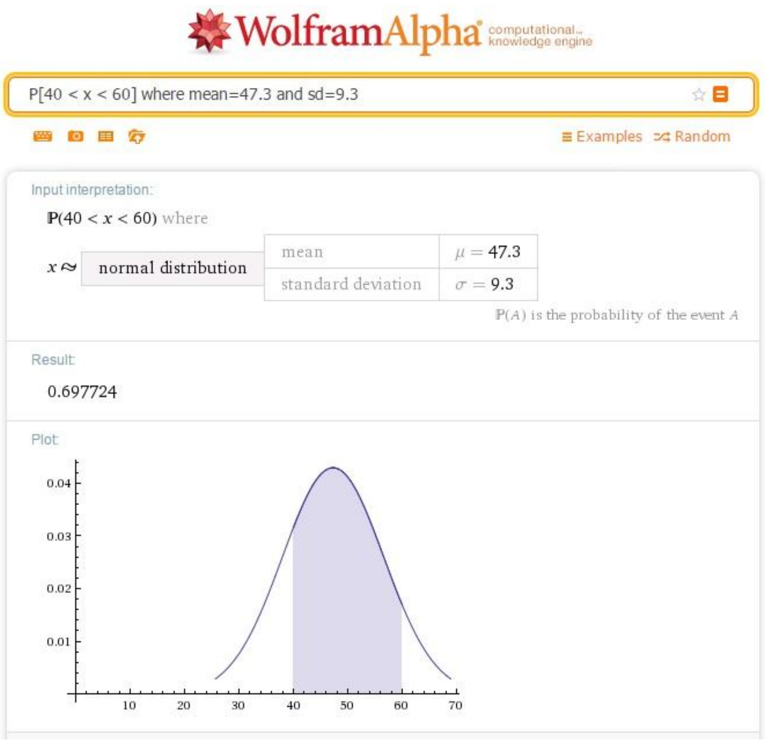


```
        par(mfrow=c(1,3))
        shadenorm(below=60,justbelow=TRUE,color="black",mu=47.3,sig=9.3)
        title("This Area")
        shadenorm(below=40,justbelow=TRUE,color="black",mu=47.3,sig=9.3)
        title("Minus THIS Area")
        shadenorm(between=c(40,60),color="black",mu=47.3,sig=9.3)
        title("Equals THIS Area")
```

We can do this very easily with the `pnorm` command in R. The first part finds all of the area to the left of x=60, and the second part finds all of the area to the left of x=40. We subtract them to find the area in between:

```
> pnorm(60,mean=47.3,sd=9.3) - pnorm(40,mean=47.3,sd=9.3)
[1] 0.6977238
```

We can *also* do this in Wolfram as long as we know how to ask for the answer:


P[40 < x < 60] where mean=47.3 and sd=9.3

Input interpretation:

P(40 < x < 60) where

x ≈ normal distribution

mean	$\mu = 47.3$
standard deviation	$\sigma = 9.3$

P(A) is the probability of the event A

Result:

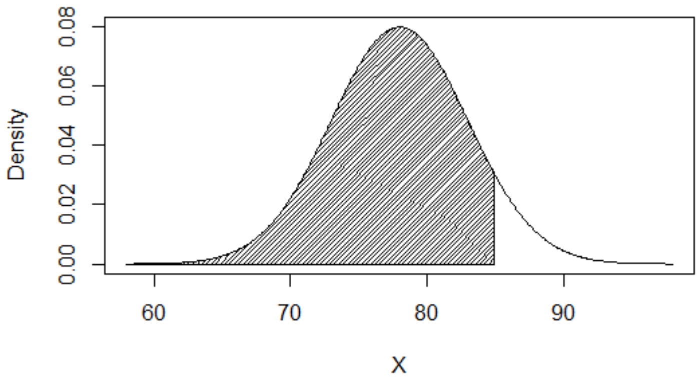
0.697724

Plot:

All of the methods give us the same answer: 69.7% of all the test scores are between x=40 and x=60. I would really have preferred that my class did better than this! Fortunately, these scores are from a pre-test taken at the beginning of the semester, which means this represents the knowledge about statistics that they come to me with. Looks like I have a completely green field of minds in front of me... not a bad thing.

Let's Go Back to That Problem From the Beginning

So in the beginning of the chapter, we were talking about an example where WE scored an 85 on a certification exam where all of the test scores were normally distributed with N(78,5). Clearly we did well, but we want to know: what percentage of all test-takers did we score higher than? Now that we know about `pnorm`, this is easy to figure out, by drawing `shadenorm(below=85,justbelow=TRUE,color="black",mu=78,sig=5)`:



From the picture, we can see that we scored higher than at least half of all the test-takers. Using `pnorm`, we can tell exactly what the area underneath the curve is:

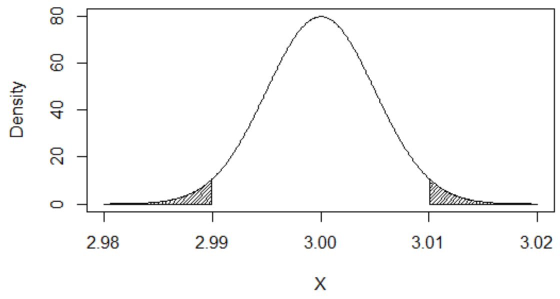
```
> pnorm(85,mean=78,sd=5)
[1] 0.9192433
```

Want to double check? Calculate the z-score associated with 85 for this particular normal distribution, head to Wolfram, and ask it to calculate P[z < whatever z score you calculated].

You Don't Need All the Data

In the examples above, we figured out what normal model to use based on the characteristics of our data set. However, sometimes, you might just be *told* what the characteristics of the population are - and asked to figure out what proportion of the population has values that fall above, below, or between certain outcomes. For example, let's say we are responsible for buying manufactured parts from one of our suppliers, to use in assemblies that we sell to our customers. To work in our assembly, each part has to be within 0.01 inches of the target length of 3.0 inches. If our supplier tells us that the population of their parts has a mean length of 3.0 inches with a standard deviation of 0.005 inches, what proportion of the parts that we buy can we expect to *not be able to use*? (This has implications for how many parts we order, and what price we will negotiate with our supplier.)

To solve this problem, we need to **draw a picture**. We know that the length of the parts is distributed as N(3.0, 0.005). We can't use parts that are shorter than (3.0 - 0.01 = 2.99 inches), nor can we use parts that are longer than (3.0 + 0.01 = 3.01 inches). This picture is drawn with `shadenorm(below=2.99,above=3.01,color="black",mu=3,sig=0.005)`:



What proportion of the area is contained within these tails, which represent the proportion of parts we won't be able to use? Because the normal model is symmetric, as long as we can find the area under the curve inside *one* of those tails, we can just multiply what we get by two to get the area in *both* of the tails together.

Since pnorm always gives us the area to the *left* of a certain point, let's use it to find out the area in the left tail. First, let's calculate a z score for x=2.99:

$$z = \frac{x-\mu}{\sigma} = \frac{2.99-3.00}{0.005} = -2$$

Using the 68-95-99.7 rule, we know the area we're looking for will be about 5% (since 95% of the area is contained *inside* z=-2 and z=+2). So let's look up to see what the area is *exactly*, multiplying by 2 since we need to include the area in both tails:

```
> pnorm(-2) * 2
[1] 0.04550026
```

We can also ask `pnorm` for the area directly, without having to compute the z score. Notice how we give `pnorm` the x value at the boundary of the left tail, since we know `pnorm` gives us everything to the left of a particular x value:

```
> pnorm(2.99,mean=3,sd=0.005) * 2
[1] 0.04550026
```

All methods agree. Approximately 4.5% of the parts that we order won't be within our required specifications.

If this was a real problem we were solving for our employer, though, the hard part would be yet to come: how are we going to use this knowledge? Does it still make sense to buy our parts from this supplier, or would we be better off considering other alternatives? Should we negotiate a price discount? Solving problems in statistics can be useful, but sometimes the bigger problem comes after you've done the calculations.

Now What?

Here are some useful resources that talk more about the concepts in this chapter:

- My favorite picture of z-scores superimposed on the normal model is here. Print it out! Carry it with you! It is tremendously valuable. http://en.wikipedia.org/wiki/Standard_score
- You can find out more about the 68-95-99.7 rule at http://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule
- Like I said before, I am not a mathematician, so I didn't go into depth about the math behind the normal pdf or cdf (or values that can be derived from those equations). If you want to know more, Wolfram has an excellent page that goes into depth at http://mathworld.wolfram.com/NormalDistribution.html

Notice that in *all* of the examples from this chapter, we've used our model of a population to answer questions about the population. But if we're only able to select a *small sample* of items from our population (usually less than 30), we aren't going to be able to get a really good sense of the variability within the population. We will have to adjust our normal model to account for the fact that we only have limited knowledge of the variability within the population: and to do that, we use the *t distribution*.

Appendix K: Overview of Inference Tests

Type of Data

	Normal or Nearly Normal	Not Normal	Binomial (Proportions)	Variances
Compare data within one group to a standard, target, or recommended value	One sample t-test	Wilcoxon Rank Sum test	One proportion z-test (or Exact Binomial test)	Chi-square test for one variance
Compare data within two groups (where observations are unpaired)	Two sample t-test	Mann-Whitney U test	Two proportion z-test, Chi-square test of independence (or Fisher's Exact test if counts in cells < 5)	F Test for Homogeneity of Variances
Compare data within two groups (where observations are paired)	Paired t-test	Wilcoxon Rank Sum test	McNemar's test	Bonett's Test
Compare data between many groups	One-Way Analysis of Variance (ANOVA)	Kruskal-Wallis test	Chi-square test of independence (or Fisher's Exact test if counts in cells < 5)	Levene's Test (or Bartlett's if source data is normal)

Index