

Chester Ismay and Albert Y. Kim

Statistical Inference via Data Science

Contents

1	Introduction	1
1.1	Introduction for students	1
1.1.1	What you will learn from this book	2
1.1.2	Data/science pipeline	5
1.1.3	Reproducible research	6
1.1.4	Final note for students	7
1.2	Introduction for instructors	7
1.2.1	Who is this book for?	8
1.3	Connect and contribute	10
1.4	About this book	11
1.5	About the authors	12
2	Getting Started with Data in R	15
2.1	What are R and RStudio?	15
2.1.1	Installing R and RStudio	16
2.1.2	Using R via RStudio	16
2.2	How do I code in R?	17
2.2.1	Basic programming concepts and terminology	18
2.2.2	Errors, warnings, and messages	19
2.2.3	Tips on learning to code	20
2.3	What are R packages?	21
2.3.1	Package installation	22
2.3.2	Package loading	24
2.3.3	Package use	25
2.4	Explore your first datasets	25
2.4.1	<code>nycflights13</code> package	26
2.4.2	<code>flights</code> data frame	26
2.4.3	Exploring data frames	27
2.4.4	Identification & measurement variables	30
2.4.5	Help files	31
2.5	Conclusion	32
2.5.1	Additional resources	32
2.5.2	What's to come?	32
I	Data Science via the tidyverse	35

3 Data Visualization	37
3.1 The Grammar of Graphics	38
3.1.1 Components of the Grammar	38
3.1.2 Gapminder data	39
3.1.3 Other components	41
3.1.4 ggplot2 package	41
3.2 Five Named Graphs - The 5NG	42
3.3 5NG#1: Scatterplots	42
3.3.1 Scatterplots via geom_point	43
3.3.2 Over-plotting	46
3.3.3 Summary	50
3.4 5NG#2: Linegraphs	50
3.4.1 Linegraphs via geom_line	52
3.4.2 Summary	53
3.5 5NG#3: Histograms	53
3.5.1 Histograms via geom_histogram	55
3.5.2 Adjusting the bins	58
3.5.3 Summary	60
3.6 Facets	60
3.7 5NG#4: Boxplots	62
3.7.1 Boxplots via geom_boxplot	65
3.7.2 Summary	68
3.8 5NG#5: Barplots	68
3.8.1 Barplots via geom_bar or geom_col	69
3.8.2 Must avoid pie charts!	73
3.8.3 Two categorical variables	74
3.8.4 Summary	78
3.9 Conclusion	80
3.9.1 Summary table	80
3.9.2 Argument specification	81
3.9.3 Additional resources	81
3.9.4 What's to come	82
4 Data Wrangling	85
4.1 The pipe operator: %>%	87
4.2 filter rows	88
4.3 summarize variables	91
4.4 group_by rows	94
4.4.1 Grouping by more than one variable	98
4.5 mutate existing variables	100
4.6 arrange and sort rows	104
4.7 join data frames	106

4.7.1	Matching “key” variable names	107
4.7.2	Different “key” variable names	108
4.7.3	Multiple “key” variables	110
4.7.4	Normal forms	110
4.8	Other verbs	111
4.8.1	<code>select</code> variables	111
4.8.2	<code>rename</code> variables	113
4.8.3	<code>top_n</code> values of a variable	114
4.9	Conclusion	115
4.9.1	Summary table	115
4.9.2	Additional resources	116
4.9.3	What’s to come?	117
5	Data Importing & “Tidy” Data	119
5.1	Importing data	120
5.1.1	Using the console	121
5.1.2	Using RStudio’s interface	122
5.2	Tidy data	123
5.2.1	Definition of “tidy” data	126
5.2.2	Converting to “tidy” data	129
5.2.3	<code>nycflights13</code> package	132
5.3	Case study: Democracy in Guatemala	133
5.4	Conclusion	136
5.4.1	<code>tidyverse</code> package	136
5.4.2	Additional resources	137
5.4.3	What’s to come?	138
II	Data Modeling via <code>moderndive</code>	141
6	Basic Regression	143
6.1	One numerical explanatory variable	145
6.1.1	Exploratory data analysis	146
6.1.2	Simple linear regression	155
6.1.3	Observed/fitted values and residuals	159
6.2	One categorical explanatory variable	162
6.2.1	Exploratory data analysis	163
6.2.2	Linear regression	170
6.2.3	Observed/fitted values and residuals	174
6.3	Related topics	175
6.3.1	Correlation is not necessarily causation	175
6.3.2	Best fitting line	177
6.3.3	<code>get_regression_x()</code> functions	181

6.4	Conclusion	184
6.4.1	Additional resources	184
6.4.2	What's to come?	184
7	Multiple Regression	185
7.1	One numerical & one categorical explanatory variable	186
7.1.1	Exploratory data analysis	187
7.1.2	Interaction model	190
7.1.3	Parallel slopes model	194
7.1.4	Observed/fitted values and residuals	198
7.2	Two numerical explanatory variables	200
7.2.1	Exploratory data analysis	201
7.2.2	Regression plane	206
7.2.3	Observed/fitted values and residuals	208
7.3	Related topics	209
7.3.1	Model selection	209
7.3.2	Correlation coefficient	213
7.3.3	Simpson's Paradox	213
7.4	Conclusion	216
7.4.1	Additional resources	216
7.4.2	What's to come?	216
III	Statistical Inference via infer	219
8	Sampling	221
8.1	Sampling bowl activity	221
8.1.1	What proportion of this bowl's balls are red?	222
8.1.2	Using the shovel once	222
8.1.3	Using the shovel 33 times	223
8.1.4	What did we just do?	226
8.2	Computer simulation of sampling	228
8.2.1	Using the virtual shovel once	229
8.2.2	Using the virtual shovel 33 times	232
8.2.3	Using the virtual shovel 1000 times	235
8.2.4	Using different shovels	237
8.3	Sampling framework	241
8.3.1	Terminology & notation	242
8.3.2	Statistical definitions	245
8.3.3	The moral of the story	248
8.4	Case study: Polls	251
8.5	Conclusion	256
8.5.1	Sampling scenarios	256

8.5.2	Central Limit Theorem	257
8.5.3	Normal distributions	257
8.5.4	Additional resources	260
8.5.5	What's to come?	260
9	Bootstrapping and Confidence Intervals	263
9.1	Pennies activity	265
9.1.1	What is the average year on US pennies in 2019?	265
9.1.2	Resampling once	270
9.1.3	Resampling 35 times	274
9.1.4	What did we just do?	277
9.2	Computer simulation of resampling	278
9.2.1	Virtually resampling once	278
9.2.2	Virtually resampling 35 times	280
9.2.3	Virtually resampling 1000 times	283
9.3	Understanding confidence intervals	285
9.3.1	Percentile method	287
9.3.2	Standard error method	288
9.4	Constructing confidence intervals	289
9.4.1	Original workflow	290
9.4.2	infer package workflow	291
9.4.3	Percentile method with infer	298
9.4.4	Standard error method with infer	300
9.5	Interpreting confidence intervals	301
9.5.1	Did the net capture the fish?	303
9.5.2	Precise & shorthand interpretation	309
9.5.3	Width of confidence intervals	311
9.6	Case study: Is yawning contagious?	314
9.6.1	Mythbusters study data	315
9.6.2	Sampling scenario	316
9.6.3	Constructing the confidence interval	317
9.6.4	Interpreting the confidence interval	323
9.7	Conclusion	325
9.7.1	Comparing bootstrap and sampling distributions	325
9.7.2	Theory-based confidence intervals	330
9.7.3	Additional resources	335
9.7.4	What's to come?	335
10	Hypothesis Testing	337
10.1	Promotions activity	338
10.1.1	Does gender affect promotions at bank?	338
10.1.2	Shuffling once	341

10.1.3 Shuffling 16 times	345
10.1.4 What did we just do?	346
10.2 Understanding hypothesis tests	347
10.3 Conducting hypothesis tests	351
10.3.1 <i>infer</i> package workflow	352
10.3.2 Comparison with confidence intervals	360
10.3.3 “There is only one test”	363
10.4 Interpreting hypothesis tests	364
10.4.1 Two possible outcomes	364
10.4.2 Types of errors	366
10.4.3 How do we choose alpha?	367
10.5 Case study: Are action or romance movies rated higher?	368
10.5.1 IMDb ratings data	369
10.5.2 Sampling scenario	371
10.5.3 Conducting the hypothesis test	373
10.6 Conclusion	379
10.6.1 Theory-based hypothesis tests	379
10.6.2 When inference is not needed	387
10.6.3 Problems with p-values	390
10.6.4 Additional resources	391
10.6.5 What’s to come	391
11 Inference for Regression	393
11.1 Regression refresher	393
11.1.1 Teaching evals analysis	394
11.1.2 Sampling scenario	396
11.2 Interpreting regression tables	397
11.2.1 Standard error	398
11.2.2 Test statistic	399
11.2.3 p-value	400
11.2.4 Confidence interval	400
11.2.5 How does R compute the table?	402
11.3 Conditions for inference for regression	404
11.3.1 Residuals refresher	404
11.3.2 Linearity of relationship	406
11.3.3 Independence of residuals	406
11.3.4 Normality of residuals	408
11.3.5 Equality of variance	410
11.3.6 What’s the conclusion?	412
11.4 Simulation-based inference for regression	413
11.4.1 Confidence interval for slope	413
11.4.2 Hypothesis test for slope	416

<i>Contents</i>	ix
11.5 Conclusion	420
11.5.1 Summary	420
11.5.2 Additional resources	421
11.5.3 What's to come	421
IV Conclusion	423
12 Tell the Story with Data	425
12.1 Case study: Seattle house prices	428
12.1.1 Exploratory data analysis (EDA)	429
12.1.2 log ₁₀ transformations	433
12.1.3 EDA Part II	436
12.1.4 Regression modeling	438
12.1.5 Making predictions	440
12.2 Case study: Effective data storytelling	442
12.2.1 Bechdel test for Hollywood gender representation . . .	442
12.2.2 US Births in 1999	443
12.2.3 Script of R code	445
Appendix	447
A Statistical Background	447
A.1 Basic statistical terms	447
A.1.1 Mean	447
A.1.2 Median	447
A.1.3 Standard deviation	447
A.1.4 Five-number summary	448
A.1.5 Distribution	448
A.1.6 Outliers	448
A.2 Normal distribution	448
Bibliography	451
Index	453

1

Introduction

Help! I'm new to R and RStudio and I need to learn about them!
However, I'm completely new to coding! What do I do?



If you're asking yourself this question, then you've come to the right place!
Start with our [Introduction for Students](#).

- *Are you an instructor hoping to use this book in your courses? Then head to Section 1.2 for more information on how to teach with this book.*
 - *Are you looking to connect with and contribute to ModernDive? Then head to Section 1.3 for information on how.*
 - *Are you curious about the publishing of this book? Then head to Section 1.4 for more information on the open-source technology, in particular R Markdown and the bookdown package.*
-

1.1 Introduction for students

This book assumes no prerequisites: no algebra, no calculus, and no prior programming/coding experience. This is intended to be a gentle introduction to the practice of analyzing data and answering questions using data the way data scientists, statisticians, data journalists, and other researchers would.

In Figure 1.1 we present a flowchart of what you'll cover in this book. You'll first get started with data in Chapter 2, where you'll learn about the difference between R and RStudio, start coding in R, understand what R packages are,

and explore your first dataset: all domestic departure flights from a New York City airport in 2013. Then

1. **Data science:** You'll assemble your data science toolbox using `tidyverse` packages. In particular:
 - Ch.3: Visualizing data via the `ggplot2` package.
 - Ch.4: Wrangling data via the `dplyr` package.
 - Ch.5: Understanding the concept of “tidy” data as a standardized data input format for all packages in the `tidyverse`
2. **Data modeling:** Using these data science tools and helper functions from the `moderndive` package, you'll start performing data modeling. In particular:
 - Ch.6: Constructing basic regression models.
 - Ch.7: Constructing multiple regression models.
3. **Statistical inference:** Once again using your newly acquired data science tools, we'll unpack statistical inference using the `infer` package. In particular:
 - Ch.8: Understanding the role that sampling variability plays in statistical inference using both tactile and virtual simulations of sampling from a “bowl” with an unknown proportion of red balls.
 - Ch.9: Building confidence intervals through the use of bootstrapping.
 - Ch.10: Conducting hypothesis tests through the use of permutation testing.
4. **Data modeling revisited:** Armed with your new understanding of statistical inference, you'll revisit and review the models you constructed in Ch.6 & Ch.7. In particular:
 - Ch.11: Interpreting both the statistical and practice significance of the results of the models.

We'll end with a discussion on what it means to “think with data” in Chapter 12 and present an example case study data analysis of house prices in Seattle.

1.1.1 What you will learn from this book

We hope that by the end of this book, you'll have learned

1. How to use R to explore data.
2. How to answer statistical questions using tools like confidence intervals and hypothesis tests.

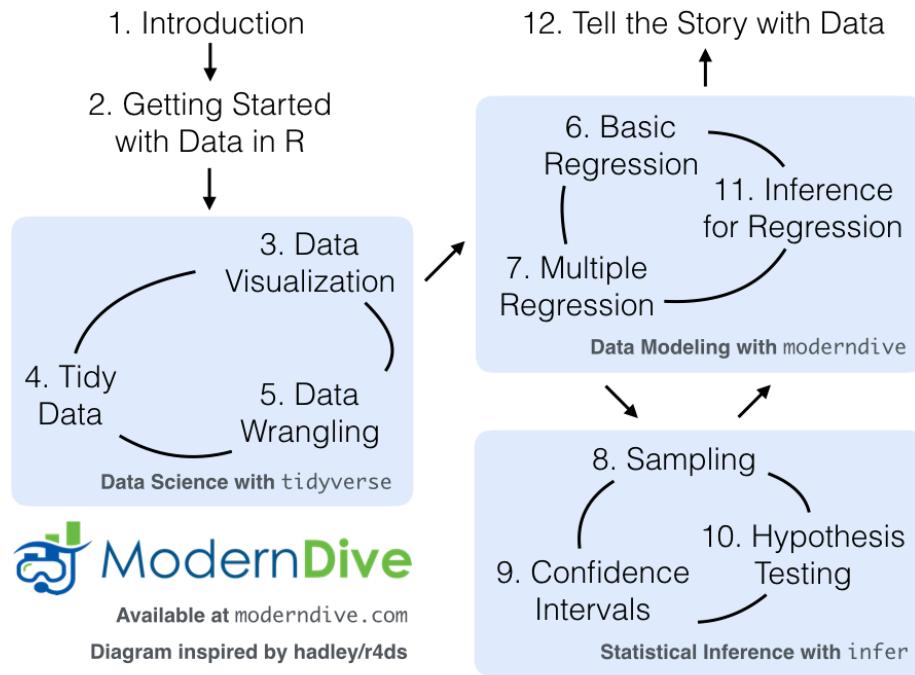


FIGURE 1.1: ModernDive Flowchart.

3. How to effectively create “data stories” using these tools.

What do we mean by data stories? We mean any analysis involving data that engages the reader in answering questions with careful visuals and thoughtful discussion, such as “How strong is the relationship between per capita income and crime in Chicago neighborhoods?”¹ and “How many f**ks does Quentin Tarantino give (as measured by the amount of swearing in his films)?”². Further discussions on data stories can be found in this Think With Google article³.

For other examples of data stories constructed by students like yourselves, look at the final projects for two courses that have previously used ModernDive:

¹http://rpubs.com/ry_lisa_elana/chicago

²https://ismayc.github.io/soc301_s2017/group_projects/group4.html

³<https://www.thinkwithgoogle.com/marketing-resources/data-measurement/tell-meaningful-stories-with-data/>

- Middlebury College MATH 116 Introduction to Statistical and Data Sciences⁴ using student collected data.
- Pacific University SOC 301 Social Statistics⁵ using data from the *fivethirtyeight* R package⁶.

This book will help you develop your “data science toolbox”, including tools such as data visualization, data formatting, data wrangling, and data modeling using regression. With these tools, you’ll be able to perform the entirety of the “data/science pipeline” while building data communication skills (see Subsection 1.1.2 for more details).

In particular, this book will lean heavily on data visualization. In today’s world, we are bombarded with graphics that attempt to convey ideas. We will explore what makes a good graphic and what the standard ways are to convey relationships with data. You’ll also see the use of visualization to introduce concepts like mean, median, standard deviation, distributions, etc. In general, we’ll use visualization as a way of building almost all of the ideas in this book.

To impart the statistical lessons in this book, we have intentionally minimized the number of mathematical formulas used and instead have focused on developing a conceptual understanding via data visualization, statistical computing, and simulations. We hope this is a more intuitive experience than the way statistics has traditionally been taught in the past and how it is commonly perceived.

Finally, you’ll learn the importance of literate programming. By this we mean you’ll learn how to write code that is useful not just for a computer to execute but also for readers to understand exactly what your analysis is doing and how you did it. This is part of a greater effort to encourage reproducible research (see Subsection 1.1.3 for more details). Hal Abelson coined the phrase that we will follow throughout this book:

“Programs must be written for people to read, and only incidentally for machines to execute.”

We understand that there may be challenging moments as you learn to pro-

⁴https://rudeboybert.github.io/MATH116/PS/final_project/final_project_outline.html#past_examples

⁵https://ismayc.github.io/soc301_s2017/group-projects/index.html

⁶<https://cran.r-project.org/web/packages/fivethirtyeight/vignettes/fivethirtyeight.html>

gram. Both of us continue to struggle and find ourselves often using web searches to find answers and reach out to colleagues for help. In the long run though, we all can solve problems faster and more elegantly via programming. We wrote this book as our way to help you get started and you should know that there is a huge community of R users that are always happy to help everyone along as well. This community exists in particular on the internet on various forums and websites such as stackoverflow.com⁷.

1.1.2 Data/science pipeline

You may think of statistics as just being a bunch of numbers. We commonly hear the phrase “statistician” when listening to broadcasts of sporting events. Statistics (in particular, data analysis), in addition to describing numbers like with baseball batting averages, plays a vital role in all of the sciences. You’ll commonly hear the phrase “statistically significant” thrown around in the media. You’ll see articles that say “Science now shows that chocolate is good for you.” Underpinning these claims is data analysis. By the end of this book, you’ll be able to better understand whether these claims should be trusted or whether we should be wary. Inside data analysis are many sub-fields that we will discuss throughout this book (though not necessarily in this order):

- data collection
- data wrangling
- data visualization
- data modeling
- inference
- correlation and regression
- interpretation of results
- data communication/storytelling

These sub-fields are summarized in what Grolemund and Wickham term the “Data/Science Pipeline”⁸ in Figure 1.2.

We will begin by digging into the gray **Understand** portion of the cycle with data visualization, then with a discussion on what is meant by tidy data and data wrangling, and then conclude by talking about interpreting and discussing the results of our models via **Communication**. These steps are vital to any statistical analysis. But why should you care about statistics? “Why did they make me take this class?”

There’s a reason so many fields require a statistics course. Scientific knowledge

⁷<https://stackoverflow.com/>

⁸<http://r4ds.had.co.nz/explore-intro.html>

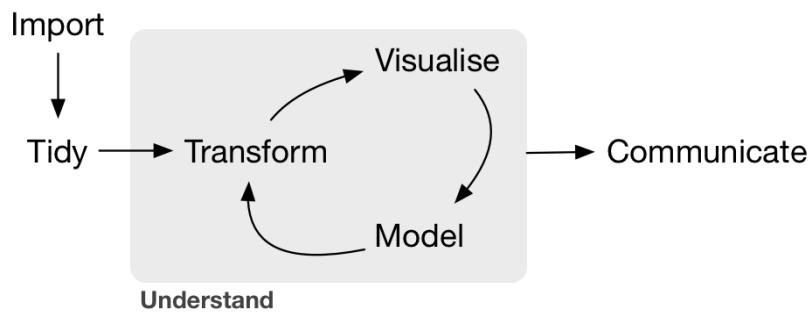


FIGURE 1.2: Data/Science Pipeline.

grows through an understanding of statistical significance and data analysis. You needn't be intimidated by statistics. It's not the beast that it used to be and, paired with computation, you'll see how reproducible research in the sciences particularly increases scientific knowledge.

1.1.3 Reproducible research

“The most important tool is the *mindset*, when starting, that the end product will be reproducible.” – Keith Baggerly

Another goal of this book is to help readers understand the importance of reproducible analyses. The hope is to get readers into the habit of making their analyses reproducible from the very beginning. This means we'll be trying to help you build new habits. This will take practice and be difficult at times. You'll see just why it is so important for you to keep track of your code and well-document it to help yourself later and any potential collaborators as well.

Copying and pasting results from one program into a word processor is not the way that efficient and effective scientific research is conducted. It's much more important for time to be spent on data collection and data analysis and not on copying and pasting plots back and forth across a variety of programs.

In traditional analyses if an error was made with the original data, we'd need to step through the entire process again: recreate the plots and copy-and-paste all of the new plots and our statistical analysis into your document. This is error prone and a frustrating use of time. We'll see how to use R Markdown

to get away from this tedious activity so that we can spend more time doing science.

“We are talking about *computational* reproducibility.” - Yihui Xie

Reproducibility means a lot of things in terms of different scientific fields. Are experiments conducted in a way that another researcher could follow the steps and get similar results? In this book, we will focus on what is known as **computational reproducibility**. This refers to being able to pass all of one’s data analysis, data-sets, and conclusions to someone else and have them get exactly the same results on their machine. This allows for time to be spent interpreting results and considering assumptions instead of the more error prone way of starting from scratch or following a list of steps that may be different from machine to machine.

1.1.4 Final note for students

At this point, if you are interested in instructor perspectives on this book, ways to contribute and collaborate, or the technical details of this book’s construction and publishing, then continue with the rest of the chapter below. Otherwise, let’s get started with R and RStudio in Chapter 2!

1.2 Introduction for instructors

This book is inspired by the following books:

- “Mathematical Statistics with Resampling and R” ([Chihara and Hesterberg, 2011](#)),
- “OpenIntro: Intro Stat with Randomization and Simulation” ([Diez et al., 2014](#)), and
- “R for Data Science” ([Grolemund and Wickham, 2016](#)).

The first book, while designed for upper-level undergraduates and graduate students, provides an excellent resource on how to use resampling to impart statistical concepts like sampling distributions using computation instead of

large-sample approximations and other mathematical formulas. The last two books are free options to learning introductory statistics and data science, providing an alternative to the many traditionally expensive introductory statistics textbooks.

When looking over the large number of introductory statistics textbooks that currently exist, we found that there wasn't one that incorporated many newly developed R packages directly into the text, in particular the many packages included in the `tidyverse`⁹ collection of packages, such as `ggplot2`, `dplyr`, `tidyr`, and `broom`. Additionally, there wasn't an open-source and easily reproducible textbook available that exposed new learners all of three of the learning goals listed at the outset of Subsection 1.1.1.

1.2.1 Who is this book for?

This book is intended for instructors of traditional introductory statistics classes using RStudio, either the desktop or server version, who would like to inject more data science topics into their syllabus. We assume that students taking the class will have no prior algebra, calculus, nor programming/coding experience.

Here are some principles and beliefs we kept in mind while writing this text. If you agree with them, this might be the book for you.

1. Blur the lines between lecture and lab

- With increased availability and accessibility of laptops and open-source non-proprietary statistical software, the strict dichotomy between lab and lecture can be loosened.
- It's much harder for students to understand the importance of using software if they only use it once a week or less. They forget the syntax in much the same way someone learning a foreign language forgets the rules. Frequent reinforcement is key.

2. Focus on the entire data/science research pipeline

- We believe that the entirety of Grolemund and Wickham's data/science pipeline¹⁰ should be taught.
- We believe in George Cobb's "minimizing prerequisites to research"¹¹: students should be answering questions with data as soon as possible.

3. It's all about the data

⁹<http://tidyverse.org/>

¹⁰<http://r4ds.had.co.nz/introduction.html>

¹¹<https://arxiv.org/abs/1507.05346>

- We leverage R packages for rich, real, and realistic data-sets that at the same time are easy-to-load into R, such as the `nycflights13` and `fivethirtyeight` packages.
 - We believe that data visualization is a gateway drug for statistics¹² and that the Grammar of Graphics as implemented in the `ggplot2` package is the best way to impart such lessons. However, we often hear: “You can’t teach `ggplot2` for data visualization in intro stats!” We, like David Robinson¹³, are much more optimistic.
 - `dplyr` has made data wrangling much more accessible¹⁴ to novices, and hence much more interesting data-sets can be explored.
4. **Use simulation/resampling to introduce statistical inference, not probability/mathematical formulas**
 - Instead of using formulas, large-sample approximations, and probability tables, we teach statistical concepts using resampling-based inference.
 - This allows for a de-emphasis of traditional probability topics, freeing up room in the syllabus for other topics. Bridges to these mathematical concepts are given as well to help with relation of these traditional topics with more modern approaches.
 5. **Don’t fence off students from the computation pool, throw them in!**
 - Computing skills are essential to working with data in the 21st century. Given this fact, we feel that to shield students from computing is to ultimately do them a disservice.
 - We are not teaching a course on coding/programming per se, but rather just enough of the computational and algorithmic thinking necessary for data analysis.
 6. **Complete reproducibility and customizability**
 - We are frustrated when textbooks give examples, but not the source code and the data itself. We give you the source code for all examples as well as the whole book!
 - Ultimately the best textbook is one you’ve written yourself. You know best your audience, their background, and their priorities. You know best your own style and the types of examples and problems you like best. Customization is the ultimate end. For more about how to make this book your own, see [About this Book](#).

¹²<http://escholarship.org/uc/item/84v3774z>

¹³http://varianceexplained.org/r/teach_ggplot2_to_beginners/

¹⁴<http://chance.amstat.org/2015/04/setting-the-stage/>

1.3 Connect and contribute

If you would like to connect with ModernDive, check out the following links:

- If you would like to receive periodic updates about ModernDive (roughly every 3 months), please sign up for our mailing list¹⁵.
- Contact Albert at albert.ys.kim@gmail.com¹⁶ and Chester at chester.ismay@gmail.com¹⁷.
- We're on Twitter at ModernDive¹⁸.

If you would like to contribute to ModernDive, there are many ways! Let's all work together to make this book as great as possible for as many students and instructors as possible!

- Please let us know if you find any errors, typos, or areas from improvement on our GitHub issues¹⁹ page.
- If you are familiar with GitHub and would like to contribute more, please see Section 1.4 below.

For example, we thank

- Dr. Andrew Heiss²⁰ for contributing Subsection 2.2.3 on “Errors, warnings, and messages”.
- Starry Zhou²¹ for her many edits to the book.

The authors would like to also thank Nina Sonneborn²², Kristin Bott²³, Dr. Jenny Smetzer²⁴, and the participants of our USCOTS 2017 workshop²⁵ for their feedback and suggestions. A special thanks goes to Dr. Yana Weinstein,

¹⁵<http://eepurl.com/cBkItf>

¹⁶<mailto:albert.ys.kim@gmail.com>

¹⁷<mailto:chester.ismay@gmail.com>

¹⁸<https://twitter.com/ModernDive>

¹⁹https://github.com/moderndive/moderndive_book/issues

²⁰<https://twitter.com/andrewheiss>

²¹<https://github.com/Starryz>

²²<https://github.com/nsonneborn>

²³<https://twitter.com/rhobott?lang=en>

²⁴<https://www.smith.edu/academics/faculty/jennifer-smetzer>

²⁵<https://www.causeweb.org/cause/uscot17/workshop/3>

cognitive psychological scientist and co-founder of The Learning Scientists²⁶, for her extensive contributions.

1.4 About this book

This book was written using RStudio’s bookdown²⁷ package by Yihui Xie (Xie, 2019). This package simplifies the publishing of books by having all content written in R Markdown²⁸. The bookdown/R Markdown source code for all versions of ModernDive is available on GitHub:

- **Latest published version** The most up-to-date release:
 - Version 0.5.0 released on February 24, 2019 (source code²⁹).
 - Available at ModernDive.com³⁰
- **Development version** The working copy of the next version which is currently being edited:
 - Preview of development version is available at <https://moderndive.netlify.com/>
 - Source code: Available on ModernDive’s GitHub repository page³¹
- **Previous versions** Older versions that may be out of date:
 - Version 0.4.0³² released on July 21, 2018 (source code³³)
 - Version 0.3.0³⁴ released on February 3, 2018 (source code³⁵)
 - Version 0.2.0³⁶ released on August 02, 2017 (source code³⁷)
 - Version 0.1.3³⁸ released on February 09, 2017 (source code³⁹)
 - Version 0.1.2⁴⁰ released on January 22, 2017 (source code⁴¹)

²⁶<http://www.learningscientists.org/yana-weinstein/>

²⁷<https://bookdown.org/>

²⁸http://rmarkdown.rstudio.com/html_document_format.html

²⁹https://github.com/moderndive/moderndive_book/releases/tag/v0.5.0

³⁰<https://moderndive.com/>

³¹https://github.com/moderndive/moderndive_book

³²[previous_versions/v0.4.0/index.html](https://github.com/moderndive/moderndive_book/releases/tag/v0.4.0)

³³https://github.com/moderndive/moderndive_book/releases/tag/v0.4.0

³⁴[previous_versions/v0.3.0/index.html](https://github.com/moderndive/moderndive_book/releases/tag/v0.3.0)

³⁵https://github.com/moderndive/moderndive_book/releases/tag/v0.3.0

³⁶[previous_versions/v0.2.0/index.html](https://github.com/moderndive/moderndive_book/releases/tag/v0.2.0)

³⁷https://github.com/moderndive/moderndive_book/releases/tag/v0.2.0

³⁸[previous_versions/v0.1.3/index.html](https://github.com/moderndive/moderndive_book/releases/tag/v0.1.3)

³⁹https://github.com/moderndive/moderndive_book/releases/tag/v0.1.3

⁴⁰[previous_versions/v0.1.2/index.html](https://github.com/moderndive/moderndive_book/releases/tag/v0.1.2)

⁴¹https://github.com/moderndive/moderndive_book/releases/tag/v0.1.2

Could this be a new paradigm for textbooks? Instead of the traditional model of textbook companies publishing updated *editions* of the textbook every few years, we apply a software design influenced model of publishing more easily updated *versions*. We can then leverage open-source communities of instructors and developers for ideas, tools, resources, and feedback. As such, we welcome your pull requests.

Finally, feel free to modify the book as you wish for your own needs, but please list the authors at the top of `index.Rmd` as “Chester Ismay, Albert Y. Kim, and YOU!”

1.5 About the authors

Who we are!

- Chester Ismay: Data Science Evangelist - DataRobot, Portland, OR, USA.
 - Email: chester.ismay@gmail.com⁴²
 - Webpage: <http://chester.rbind.io/>
 - Twitter: [old_man_chester](https://twitter.com/old_man_chester)⁴³
 - GitHub: <https://github.com/ismayc>
- Albert Y. Kim: Assistant Professor of Statistical & Data Sciences - Smith College, Northampton, MA, USA.
 - Email: albert.ys.kim@gmail.com⁴⁴
 - Webpage: <http://rudeboybert.rbind.io/>
 - Twitter: [rudeboybert](https://twitter.com/rudeboybert)⁴⁵
 - GitHub: <https://github.com/rudeboybert>

⁴² <mailto:chester.ismay@gmail.com>

⁴³ https://twitter.com/old_man_chester

⁴⁴ <mailto:albert.ys.kim@gmail.com>

⁴⁵ <https://twitter.com/rudeboybert>

Chester Ismay



Albert Y. Kim



2

Getting Started with Data in R

Before we can start exploring data in R, there are some key concepts to understand first:

1. What are R and RStudio?
2. How do I code in R?
3. What are R packages?

We'll introduce these concepts in upcoming Sections 2.1-2.3. If you are already somewhat familiar with these concepts, feel free to skip to Section 2.4 where we'll introduce our first data set: all domestic flights departing a New York City airport in 2013. This is a dataset we will explore in depth in this book.

2.1 What are R and RStudio?

For much of this book, we will assume that you are using R via RStudio. First time users often confuse the two. At its simplest R is like a car's engine while RStudio is like a car's dashboard.

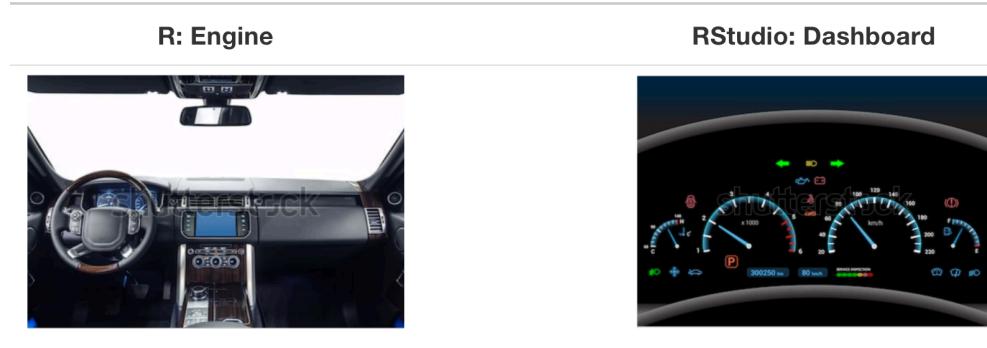


FIGURE 2.1: Analogy of difference between R and RStudio.

More precisely, R is a programming language that runs computations while

RStudio is an *integrated development environment (IDE)* that provides an interface by adding many convenient features and tools. So just as the way of having access to a speedometer, rearview mirrors, and a navigation system makes driving much easier, using RStudio's interface makes using R much easier as well.

2.1.1 Installing R and RStudio

Note about RStudio Server: If your instructor has provided you with a link and access to RStudio Server, then you can skip this section. We do recommend though after a few months of working on the RStudio Server that you return to these instructions.

You will first need to download and install both R and RStudio (Desktop version) on your computer.

1. **You must do this first:** Download and install R¹.
 - Click on the download link corresponding to your computer's operating system.
2. **You must do this second:** Download and install RStudio².
 - Scroll down to "Installers for Supported Platforms"
 - Click on the download link corresponding to your computer's operating system.

2.1.2 Using R via RStudio

Recall our car analogy from earlier. Much as we don't drive a car by interacting directly with the engine but rather by interacting with elements on the car's dashboard, we won't be using R directly but rather we will use RStudio's interface. After you install R and RStudio on your computer, you'll have two new programs AKA applications you can open. We will always work in RStudio and not R. Figure 2.2 shows what icon you should be clicking on your computer.

After you open RStudio, you should see the following in Figure 2.3.

¹<https://cran.r-project.org/>

²<https://www.rstudio.com/products/rstudio/download3/>

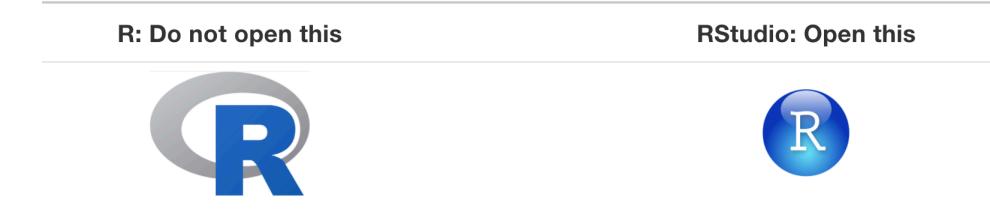


FIGURE 2.2: Icons of R versus RStudio on your computer.

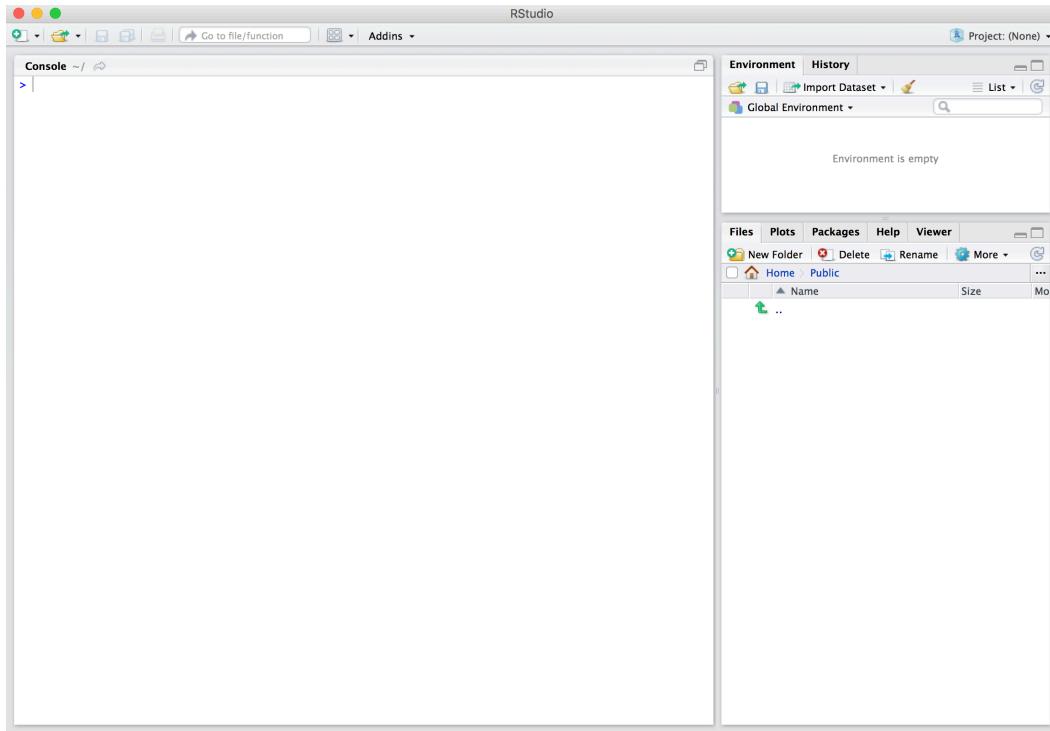


FIGURE 2.3: RStudio interface to R.

Note the three panes, which are three panels dividing the screen: The *Console pane*, the *Files pane*, and the *Environment pane*. Over the course of this chapter, you'll come to learn what purpose each of these panes serve.

2.2 How do I code in R?

Now that you're set up with R and RStudio, you are probably asking yourself “OK. Now how do I use R?” The first thing to note is that unlike other statisti-

cal software programs like Excel, STATA, or SAS that provide point and click³ interfaces, R is an interpreted language⁴. This means you have to enter in R commands written in R code. In other words, you have to code/program in R. Note that we'll use the terms “coding” and “programming” interchangeably in this book.

While it is not required to be a seasoned coder/computer programmer to use R, there is still a set of basic programming concepts that R users need to understand. Consequently, while this book is not a book on programming, you will still learn just enough of these basic programming concepts needed to explore and analyze data effectively.

2.2.1 Basic programming concepts and terminology

We now introduce some basic programming concepts and terminology that you'll learn as you go. Note that in this book, we will use a different font to distinguish regular font from `computer_code`.

It is important to note that while these tutorials serve as excellent introductions, a single pass through them is insufficient for long-term learning and retention. The ultimate tools for long-term learning and retention are “learning by doing” and repetition, something we will have you do over the course of the entire book and we encourage this process as much as possible as you learn any new skill.

- Basics:
 - Console: where you enter in commands
 - Objects: where values are saved, how to assign values to objects.
 - Data types: integers, doubles/numerics, logicals, characters.
- Vectors: a series of values. These are created using the `c()` function where `c()` stands for “combine” or “concatenate”. For example: `c(6, 11, 13, 31, 90, 92)`.
- Factors: *Categorical data* (as opposed to *numerical data*) are represented in R as `factors`.
- Data frames: Data frames are analogous to rectangular spreadsheets: they are representations of datasets in R where the rows correspond *observations* and the columns correspond to *variables* that describe the observations. We will revisit this later in Section 2.4.
- Conditionals:
 - Testing for equality in R using `==` (and not `=` which is typically used for

³https://en.wikipedia.org/wiki/Point_and_click

⁴https://en.wikipedia.org/wiki/Interpreted_language

- assignment). Ex: `2 + 1 == 3` compares `2 + 1` to `3` and is correct R syntax, while `2 + 1 = 3` is not and is incorrect R syntax.
- Boolean algebra: `TRUE/FALSE` statements and mathematical operators such as `<` (less than), `<=` (less than or equal), and `!=` (not equal to).
 - Logical operators: `&` representing “and”, `|` representing “or”. Ex: `(2 + 1 == 3) & (2 + 1 == 4)` returns `FALSE` while `(2 + 1 == 3) | (2 + 1 == 4)` returns `TRUE`.
 - Functions: Functions take in inputs (called *arguments*) and return outputs. You either manually specify a function’s arguments or use the function’s *defaults*.

This list is by no means an exhaustive list of all the programming concepts and terminology needed to become a savvy R user; such a list would be so large it wouldn’t be very useful, especially for novices. Rather, we feel this is the bare minimum you need to know before you get started; the rest we feel you can learn as you go. Remember that your knowledge of all of these concepts will build as you get better and better at “speaking R” and getting used to its syntax.

2.2.2 Errors, warnings, and messages

One slightly confusing part of R is how it reports errors, warnings, and messages. The default theme in RStudio colors errors, warnings, and messages in red, which makes them seem like you did something wrong. However, seeing red text in the console *is not always bad*.

R will show red text in the console in three different situations:

- **Errors:** When the red text is a legitimate error, it will be prefaced with “Error in...” and try to explain what went wrong. Generally when there’s an error, the code will not run. For example, as shown in Subsection 2.3.3 if you see `Error in ggplot(...) : could not find function "ggplot"`, it means that the `ggplot()` function is not accessible because the package was not loaded with `library(ggplot2)`, and thus you cannot use it.
- **Warnings:** When the red text is a warning, it will be prefaced with “Warning.” and try to explain why there’s a warning. Generally your code will still work, but with some caveats. For example, you see in Chapter 3 if you plot a scatterplot and one of the rows in your data frame is missing a value, you will see this warning: `Warning: Removed 1 rows containing missing values (geom_point)`. R will still make the scatterplot with all the remaining values, but it’s warning you that one of the points isn’t there.
- **Messages:** When the red text doesn’t start with either “Error” or “Warn-

ing”, it’s *just a friendly message*. You’ll see these messages when you load some packages like the `dplyr` package in Subsection 2.3.2, or when you read data saved in spreadsheet files with `read_csv()` as you’ll see in Chapter 5. These are helpful diagnostic messages and they don’t stop your code from working.

Remember, when you see red text in the console, *don’t panic*. It doesn’t necessarily mean anything is wrong.

- If the text starts with “Error”, figure out what’s causing it. Think of errors as a red traffic light: something is wrong!
- If the text starts with “Warning”, figure out if it’s something to worry about. For instance, if you get a warning about missing values in a scatterplot and you know there are missing values, you’re fine. If that’s surprising, look at your data and see what’s missing. Think of warnings as a yellow traffic light: everything is working fine, but watch out/pay attention.
- Otherwise the text is just a message. Read it, wave back at R, and thank it for talking to you. Think of messages as a green traffic light: everything is working fine.

2.2.3 Tips on learning to code

Learning to code/program is very much like learning a foreign language, it can be very daunting and frustrating at first. Such frustrations are very common and it is very normal to feel discouraged as you learn. However just as with learning a foreign language, if you put in the effort and are not afraid to make mistakes, anybody can learn.

Here are a few useful tips to keep in mind as you learn to program:

- **Remember that computers are not actually that smart:** You may think your computer or smartphone are “smart,” but really people spent a lot of time and energy designing them to appear “smart.” Rather you have to tell a computer everything it needs to do. Furthermore the instructions you give your computer can’t have any mistakes in them, nor can they be ambiguous in any way.
- **Take the “copy, paste, and tweak” approach:** Especially when learning your first programming language, it is often much easier to taking existing code that you know works and modify it to suit your ends, rather than trying to write new code from scratch. We call this the *copy, paste, and tweak* approach. So early on, we suggest not trying to write code from memory, but rather take existing examples we have provided you, then copy, paste, and tweak them to suit your goals. Don’t be afraid to play around!

- **The best way to learn to code is by doing:** Rather than learning to code for its own sake, we feel that learning to code goes much smoother when you have a goal in mind or when you are working on a particular project, like analyzing data that you are interested in.
 - **Practice is key:** Just as the only method to improving your foreign language skills is through practice, practice, and practice; so also the only method to improving your coding is through practice, practice, and practice. Don't worry however; we'll give you plenty of opportunities to do so!
-

2.3 What are R packages?

Another point of confusion with many new R users is the idea of an R package. R packages extend the functionality of R by providing additional functions, data, and documentation. They are written by a world-wide community of R users and can be downloaded for free from the internet. For example, among the many packages we will use in this book are:

- The `ggplot2` package for data visualization in Chapter 3.
- The `dplyr` package for data wrangling in Chapter 4.
- The `moderndive` package that accompanies this book.
- The `infer` package for “tidy” and transparent statistical inference in Chapters 9, 10, and 11.

A good analogy for R packages is they are like apps you can download onto a mobile phone:

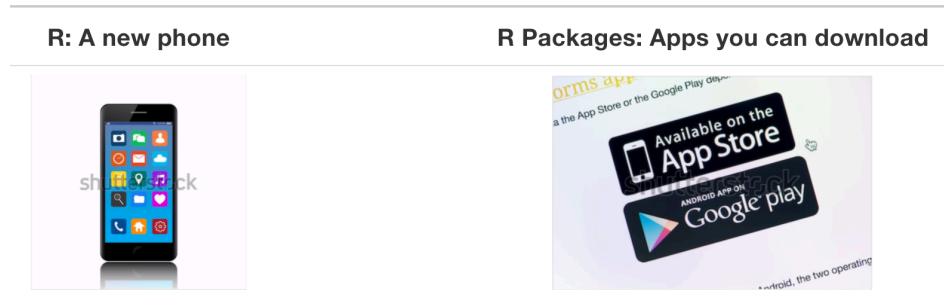


FIGURE 2.4: Analogy of R versus R packages.

So R is like a new mobile phone: while it has a certain amount of features when you use it for the first time, it doesn't have everything. R packages are

like the apps you can download onto your phone from Apple’s App Store or Android’s Google Play.

Let’s continue this analogy by considering the Instagram app for editing and sharing pictures. Say you have purchased a new phone and you would like to share a recent photo you have taken on Instagram. You need to:

1. *Install the app*: Since your phone is new and does not include the Instagram app, you need to download the app from either the App Store or Google Play. You do this once and you’re set. You might do this again in the future any time there is an update to the app.
2. *Open the app*: After you’ve installed Instagram, you need to open the app.

Once Instagram is open on your phone, you can then proceed to share your photo with your friends and family. The process is very similar for using an R package. You need to:

1. *Install the package*: This is like installing an app on your phone. Most packages are not installed by default when you install R and RStudio. Thus if you want to use a package for the first time, you need to install it first. Once you’ve installed a package, you likely won’t install it again unless you want to update it to a newer version.
2. *“Load” the package*: “Loading” a package is like opening an app on your phone. Packages are not “loaded” by default when you start RStudio on your computer; you need to “load” each package you want to use every time you start RStudio.

Let’s now show you how to perform these two steps for the `ggplot2` package for data visualization.

2.3.1 Package installation

Note about RStudio Server: If your instructor has provided you with a link and access to RStudio Server, you probably will not need to install packages, as they have likely been pre-installed for you by your instructor. That being said, it is still a good idea to know this process for later on when you are not using RStudio Server, but rather RStudio Desktop on your own computer.

There are two ways to install an R package. For example, to install the `ggplot2` package:

1. **Easy way:** In the Files pane of RStudio:
 - a) Click on the “Packages” tab.
 - b) Click on “Install” next to Update.
 - c) Type the name of the package under “Packages (separate multiple with space or comma):” In this case, type `ggplot2`.
 - d) Click “Install”.

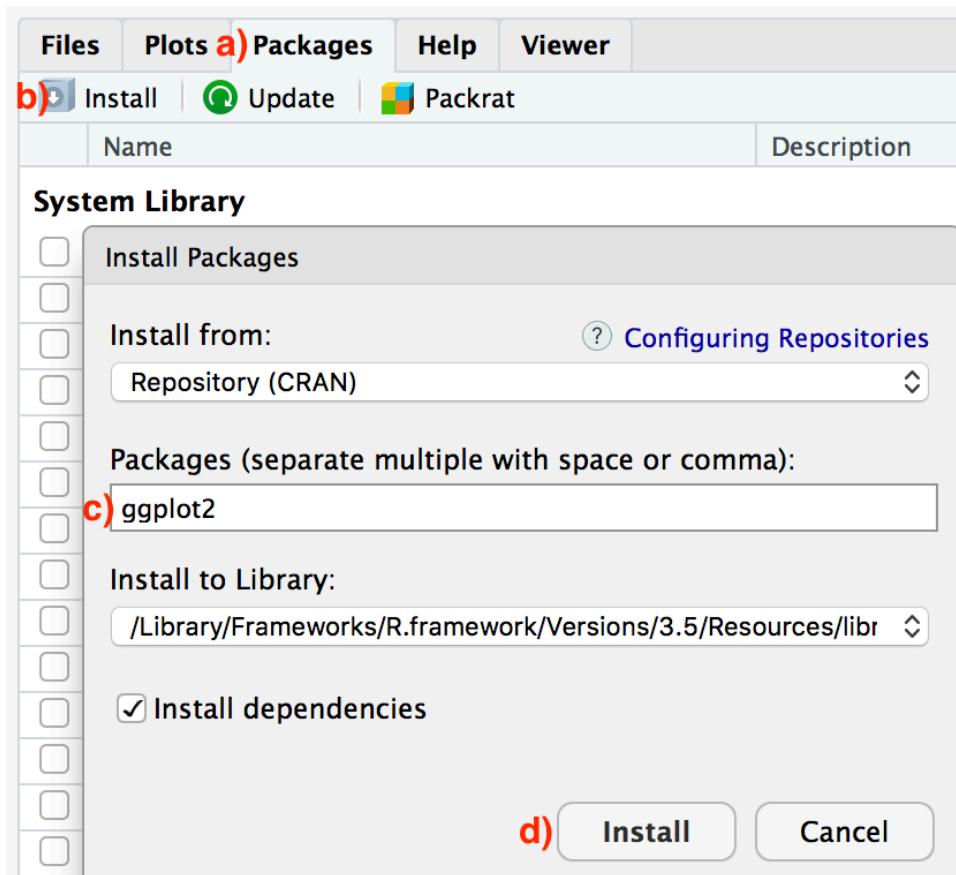


FIGURE 2.5: Installing packages in R the easy way.

1. **Slightly harder way:** An alternative but slightly less convenient way to install a package is by typing `install.packages("ggplot2")` in the Console pane of RStudio and pressing Return/Enter on your keyboard. Note you must include the quotation marks.

Much like an app on your phone, you only have to install a package once.

However, if you want to update an already installed package to a newer version, you need to re-install it by repeating the earlier steps.

Learning check

(LC2.1) Repeat the earlier installing steps, but for the `dplyr`, `nycflights13`, and `knitr` packages. This will install the earlier mentioned `dplyr` package, the `nycflights13` package containing data on all domestic flights leaving a NYC airport in 2013, and the `knitr` package for writing reports in R.

2.3.2 Package loading

Recall that after you've installed a package, you need to "load" it. In other words, open it. We do this by using the `library()` command. For example, to load the `ggplot2` package, run the following code in the Console pane. What do we mean by "run the following code"? Either type or copy & paste the following code into the Console pane and then hit the enter key.

```
library(ggplot2)
```

If after running the earlier code, a blinking cursor returns next to the `>` "prompt" sign, it means you were successful and the `ggplot2` package is now loaded and ready to use. If however, you get a red "error message" that reads...

```
Error in library(ggplot2) : there is no package called 'ggplot2'
```

... it means that you didn't successfully install it. In that case, go back to the previous subsection "Package installation" and install it.

Learning check

(LC2.2) "Load" the `dplyr`, `nycflights13`, and `knitr` packages as well by repeating the earlier steps.

2.3.3 Package use

One extremely common mistake new R users make when wanting to use particular packages is they forget to “load” them first by using the `library()` command we just saw. Remember: *you have to load each package you want to use every time you start RStudio*. If you don’t first “load” a package, but attempt to use one of its features, you’ll see an error message similar to:

```
Error: could not find function
```

R is telling you that you are trying to use a function in a package that has not yet been “loaded.” R doesn’t know where to find the function you are using. Almost all new users forget do this when starting out, and it is a little annoying to get used to doing it. However, you’ll remember with practice.

2.4 Explore your first datasets

Let’s put everything we’ve learned so far into practice and start exploring some real data! Data comes to us in a variety of formats, from pictures to text to numbers. Throughout this book, we’ll focus on datasets that are saved in “spreadsheet”-type format; this is probably the most common way data are collected and saved in many fields. Remember from Subsection 2.2.1 that these “spreadsheet”-type datasets are called *data frames* in R; we will focus on working with data saved as data frames throughout this book.

Let’s first load all the packages needed for this chapter, assuming you’ve already installed them. Read Section 2.3 for information on how to install and load R packages if you haven’t already.

```
library(nycflights13)
library(dplyr)
library(knitr)
```

At the beginning of all subsequent chapters in this text, we’ll always have a list of packages that you should have installed and loaded to work with that chapter’s R code.

2.4.1 `nycflights13` package

Many of us have flown on airplanes or know someone who has. Air travel has become an ever-present aspect in many people's lives. If you live in or are visiting a relatively large city and you walk around that city's airport, you see gates showing flight information from many different airlines. And you will frequently see that some flights are delayed because of a variety of conditions. Are there ways that we can avoid having to deal with these flight delays?

We'd all like to arrive at our destinations on time whenever possible. (Unless you secretly love hanging out at airports. If you are one of these people, pretend for the moment that you are very much anticipating being at your final destination.) Throughout this book, we're going to analyze data related to flights contained in the `nycflights13` package. Specifically, this package contains five data sets saved in five separate data frames with information about all domestic flights departing from New York City in 2013. These include Newark Liberty International (EWR), John F. Kennedy International (JFK), and LaGuardia (LGA) airports:

- `flights`: Information on all 336,776 flights
- `airlines`: A table matching airline names and their two letter IATA airline codes (also known as carrier codes) for 16 airline companies
- `planes`: Information about each of 3,322 physical aircraft used.
- `weather`: Hourly meteorological data for each of the three NYC airports. This data frame has 26,115 rows, roughly corresponding to the $365 \times 24 \times 3 = 26,280$ possible hourly measurements one can observe at three locations over the course of a year.
- `airports`: Airport names, codes, and locations for 1,458 destination airports.

2.4.2 `flights` data frame

We will begin by exploring the `flights` data frame that is included in the `nycflights13` package and getting an idea of its structure. Run the following code in your console (either by typing it or cutting & pasting it): it loads in the `flights` dataset into your Console. Note depending on the size of your monitor, the output may vary slightly.

```
flights
```

```
# A tibble: 336,776 x 19
  year month   day dep_time sched_dep_time dep_delay
  <int> <int> <int>    <int>          <int>      <dbl>
```

```
1 2013 1 1 517 515 2
2 2013 1 1 533 529 4
3 2013 1 1 542 540 2
4 2013 1 1 544 545 -1
5 2013 1 1 554 600 -6
6 2013 1 1 554 558 -4
7 2013 1 1 555 600 -5
8 2013 1 1 557 600 -3
9 2013 1 1 557 600 -3
10 2013 1 1 558 600 -2
# ... with 336,766 more rows, and 13 more variables:
#   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

Let's unpack this output:

- A `tibble`: `336,776 x 19`: A `tibble` is a kind of data frame used in R. This particular data frame has
 - `336,776` rows
 - `19` columns corresponding to `19` variables describing each observation
- `year month day dep_time sched_dep_time dep_delay arr_time` are different columns, in other words variables, of this data frame.
- We then have the first `10` rows of observations corresponding to `10` flights.
- `... with 336,766 more rows, and 11 more variables`: indicating to us that `336,766` more rows of data and `11` more variables could not fit in this screen.

Unfortunately, this output does not allow us to explore the data very well. Let's look at different tools to explore data frames.

2.4.3 Exploring data frames

Among the many ways of getting a feel for the data contained in a data frame such as `flights`, we present three functions that take as their “argument”, in other words their input, the data frame in question. We also include a fourth method for exploring one particular column of a data frame:

1. Using the `view()` function built for use in RStudio. We will use this the most.
2. Using the `glimpse()` function, which is included in the `dplyr` package.
3. Using the `kable()` function, which is included in the `knitr` package.

4. Using the `$` extraction operator to view a single variable in a data frame.

1. `view()`:

Run `View(flights)` in your Console in RStudio, either by typing it or cutting & pasting it into the Console pane, and explore this data frame in the resulting pop-up viewer. You should get into the habit of always `viewing` any data frames that come your way. Note the capital “V” in `view`. R is case-sensitive so you’ll receive an error if you run `view(flights)` instead of `View(flights)`.

Learning check

(LC2.3) What does any *ONE* row in this `flights` dataset refer to?

- A. Data on an airline
- B. Data on a flight
- C. Data on an airport
- D. Data on multiple flights

By running `View(flights)`, we see the different *variables* listed in the columns and we see that there are different types of variables. Some of the variables like `distance`, `day`, and `arr_delay` are what we will call *quantitative* variables. These variables are numerical in nature. Other variables here are *categorical*.

Note that if you look in the leftmost column of the `View(flights)` output, you will see a column of numbers. These are the row numbers of the dataset. If you glance across a row with the same number, say row 5, you can get an idea of what each row corresponds to. In other words, this will allow you to identify what object is being referred to in a given row. This is often called the *observational unit*. The observational unit in this example is an individual flight departing New York City in 2013. You can identify the observational unit by determining what “thing” is being measured or described by each of the variables. We’ll talk more about observational units in Section 2.4.4 on *identification* and *measurement* variables.

2. `glimpse()`:

The second way to explore a data frame is using the `glimpse()` function included in the `dplyr` package. Thus, you can only use the `glimpse()` function after you’ve loaded the `dplyr` package. This function provides us with an alternative method for exploring a data frame than the `View()` function:

```
glimpse(flights)
```

```
Observations: 336,776
Variables: 19
$ year           <int> 2013, 2013, 2013, 2013, 2013, 20...
$ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
$ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
$ dep_time        <int> 517, 533, 542, 544, 554, 554, 55...
$ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 60...
$ dep_delay       <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, ...
$ arr_time        <int> 830, 850, 923, 1004, 812, 740, 9...
$ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 8...
$ arr_delay       <dbl> 11, 20, 33, -18, -25, 12, 19, -1...
$ carrier         <chr> "UA", "UA", "AA", "B6", "DL", "U...
$ flight          <int> 1545, 1714, 1141, 725, 461, 1696...
$ tailnum         <chr> "N14228", "N24211", "N619AA", "N...
$ origin          <chr> "EWR", "LGA", "JFK", "JFK", "LGA...
$ dest            <chr> "IAH", "IAH", "MIA", "BQN", "ATL...
$ air_time        <dbl> 227, 227, 160, 183, 116, 150, 15...
$ distance        <dbl> 1400, 1416, 1089, 1576, 762, 719...
$ hour            <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, ...
$ minute          <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, ...
$ time_hour       <dttm> 2013-01-01 05:00:00, 2013-01-01...
```

We see that `glimpse()` will give you the first few entries of each variable in a row after the variable. In addition, the *data type* (see Subsection 2.2.1) of the variable is given immediately after each variable's name inside `< >`. Here, `int` and `dbl` refer to “integer” and “double”, which are computer coding terminology for quantitative/numerical variables. In contrast, `chr` refers to “character”, which is computer terminology for text data. Text data, such as the `carrier` or `origin` of a flight, are categorical variables. The `time_hour` variable is an example of one more type of data type: `dttm`. As you may suspect, this variable corresponds to a specific date and time of day. However, we won't work with dates in this class and leave it to a more advanced book on data science.

Learning check

(LC2.4) What are some examples in this dataset of **categorical** variables? What makes them different than **quantitative** variables?

3. `kable()`:

The final way to explore the entirety of a data frame is using the `kable()` function from the `knitr` package. Let's explore the different carrier codes for all the airlines in our dataset two ways. Run both of these lines of code in your Console:

```
airlines
kable(airlines)
```

At first glance, it may not appear that there is much difference in the outputs. However when using tools for document production such as R Markdown⁵, the latter code produces output that is much more legible and reader-friendly.

4. `$` operator

Lastly, the `$` operator allows us to extract and then explore a single variable within a data frame. For example, run the following in your console

```
airlines
airlines$name
```

We used the `$` operator to extract only the `name` variable and return it as a vector of length 16. We will only be occasionally exploring data frames using this operator, instead favoring the `view()` and `glimpse()` functions.

2.4.4 Identification & measurement variables

There is a subtle difference between the kinds of variables that you will encounter in data frames: *identification variables* and *measurement variables*. For example, let's explore the `airports` data frame by showing the output of `glimpse(airports)`:

```
glimpse(airports)
```

```
Observations: 1,458
Variables: 8
 $ faa    <chr> "04G", "06A", "06C", "06N", "09J", "0A9", ...
```

⁵<http://rmarkdown.rstudio.com/lesson-1.html>

```
$ name <chr> "Lansdowne Airport", "Moton Field Municip..."  
$ lat <dbl> 41.1, 32.5, 42.0, 41.4, 31.1, 36.4, 41.5,...  
$ lon <dbl> -80.6, -85.7, -88.1, -74.4, -81.4, -82.2,...  
$ alt <int> 1044, 264, 801, 523, 11, 1593, 730, 492, ...  
$ tz <dbl> -5, -6, -6, -5, -5, -5, -5, -5, -8, -...  
$ dst <chr> "A", "A", "A", "A", "A", "A", "A", "A", "...  
$ tzone <chr> "America/New_York", "America/Chicago", "A..."
```

The variables `faa` and `name` are what we will call *identification variables*: variables that uniquely identify each observational unit. They are mainly used to provide a unique name to each observational unit i.e. row, thereby allowing us to uniquely identify them. `faa` gives the unique code provided by the FAA for that airport, while the `name` variable gives the longer more natural name of the airport. The remaining variables (`lat`, `lon`, `alt`, `tz`, `dst`, `tzone`) are often called *measurement* or *characteristic* variables: variables that describe properties of each observational unit, in other words each observation in each row. For example, `lat` and `long` describe the latitude and longitude of each airport.

Furthermore, sometimes a single variable might not be enough to uniquely identify each observational unit: combinations of variables might be needed. While it is not an absolute rule, for organizational purposes it is considered good practice to have your identification variables in the left-most columns of your data frame.

Learning check

(LC2.5) What properties of the observational unit do each of `lat`, `lon`, `alt`, `tz`, `dst`, and `tzone` describe for the `airports` data frame? Note that you may want to use `?airports` to get more information.

(LC2.6) Provide the names of variables in a data frame with at least three variables in which one of them is an identification variable and the other two are not. In other words, create your own tidy data frame that matches these conditions.

2.4.5 Help files

Another nice feature of R is the help system. You can get help in R by entering `a ?` before the name of a function or data frame in question and you will be

presented with a page showing the documentation. For example, let's look at the help file for the `flights` data frame:

```
?flights
```

A help file should pop-up in the Help pane of RStudio. If you have questions about a function or data frame included in an R package, you should get in the habit of consulting the help file right away.

2.5 Conclusion

We've given you what we feel are the most essential concepts to know before you can start exploring data in R. Is this chapter exhaustive? Absolutely not. To try to include everything in this chapter would make the chapter so large it wouldn't be useful!

2.5.1 Additional resources

If you are completely new to the world of coding, R, and RStudio and feel you could benefit from a more detailed introduction, we suggest you check out ModernDive co-author Chester Ismay's "Getting used to R, RStudio, and R Markdown"⁶ short book (Ismay, 2016), which includes screencast recordings that you can follow along and pause as you learn. Furthermore, there is an introduction to R Markdown, a tool used for reproducible research in R.

2.5.2 What's to come?

As we stated earlier however, the best way to learn R is to learn by doing. We now start the "data science" portion of the book in Chapter 3 with what we feel is the most important tool in a data scientist's toolbox: data visualization. We will continue to explore the data included in the `nycflights13` package through data visualization. We'll see that data visualization is a powerful tool to add to our toolbox for data exploring that provides additional insight to what the `view()` and `glimpse()` functions can provide.

⁶<https://rbasics.netlify.com/>

1 Introduction

Chester Ismay
Patrick C. Kennedy
2018-05-23

This book was written to give people who are new to R, RStudio, and R Markdown the tools they need to begin making their own research reproducible. R is an open-source programming language that has seen its popularity grow tremendously in recent years, with developers adding new functionality via packages on a daily basis. RStudio is a graphical development environment that makes it easier to write and view the results of R code, and R Markdown provides an easy way to produce rich, fully-documented, reproducible analyses.

FIGURE 2.6: Preview of 'Getting Used to R, RStudio, and R Markdown'.

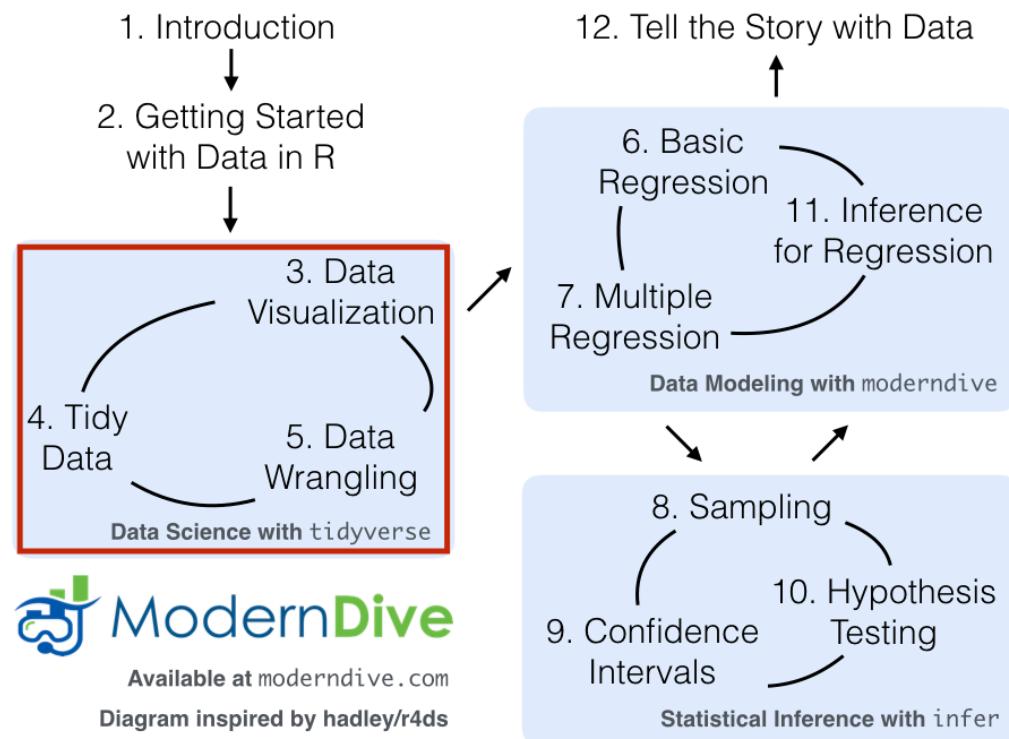


FIGURE 2.7: ModernDive flowchart.

Part I

Data Science via the tidyverse

3

Data Visualization

We begin the development of your data science toolbox with data visualization. By visualizing our data, we gain valuable insights that we couldn't initially see from just looking at the raw data in spreadsheet form. We will use the `ggplot2` package as it provides an easy way to customize your plots. `ggplot2` is rooted in the data visualization theory known as *The Grammar of Graphics* ([Wilkinson, 2005](#)), developed by Leland Wilkinson.

At the most basic level, graphics/plots/charts (we use these terms interchangeably in this book) provide a nice way for us to get a sense for how quantitative variables compare in terms of their center (where the values tend to be located) and their spread (how they vary around the center). Graphics should be designed to emphasize the findings and insight you want your audience to understand. This does however require a balancing act. On the one hand, you want to highlight as many meaningful relationships and interesting findings as possible; on the other you don't want to include so many as to overwhelm your audience.

As we will see, plots/graphics also help us to identify patterns and outliers in our data. We will see that a common extension of these ideas is to compare the *distribution* of one quantitative variable (i.e., what the spread of a variable looks like or how the variable is *distributed* in terms of its values) as we go across the levels of a different categorical variable.

Needed packages

Let's load all the packages needed for this chapter (this assumes you've already installed them). Read Section 2.3 for information on how to install and load R packages.

```
library(nycflights13)
library(ggplot2)
library(dplyr)
```

3.1 The Grammar of Graphics

We begin with a discussion of a theoretical framework for data visualization known as “The Grammar of Graphics,” which serves as the foundation for the `ggplot2` package. Think of how we construct sentences in English to form sentences by combining different elements, like nouns, verbs, particles, subjects, objects, etc. However, we can’t just combine these elements in any arbitrary order; we must do so following a set of rules known as a linguistic grammar. Similarly to a linguistic grammar, “The Grammar of Graphics” define a set of rules for constructing *statistical graphics* by combining different types of *layers*. This grammar was created by Leland Wilkinson ([Wilkinson, 2005](#)) and has been implemented in a variety of data visualization software including R.

3.1.1 Components of the Grammar

In short, the grammar tells us that:

A statistical graphic is a `mapping of data variables to aesthetic attributes of geometric objects`.

Specifically, we can break a graphic into the following three essential components:

1. `data`: the data set composed of variables that we map.
2. `geom`: the geometric object in question. This refers to the type of object we can observe in a plot. For example: points, lines, and bars.
3. `aes`: aesthetic attributes of the geometric object. For example, x/y position, color, shape, and size. Each assigned aesthetic attribute can be mapped to a variable in our data set.

You might be wondering why we wrote the terms `data`, `geom`, and `aes` in a computer code type font. We’ll see very shortly that we’ll specify the elements of the grammar in R using these terms. However, let’s first break down the grammar with an example.

3.1.2 Gapminder data

In February 2006, a statistician named Hans Rosling gave a TED talk titled “The best stats you’ve ever seen”¹ where he presented global economic, health, and development data from the website gapminder.org². For example, for the 142 countries included from 2007, let’s consider only the first 6 countries when listed alphabetically in Table 3.1.

TABLE 3.1: Gapminder 2007 Data: First 6 of 142 countries

Country	Continent	Life Expectancy	Population	GDP per Capita
Afghanistan	Asia	43.8	31889923	975
Albania	Europe	76.4	3600523	5937
Algeria	Africa	72.3	33333216	6223
Angola	Africa	42.7	12420476	4797
Argentina	Americas	75.3	40301927	12779
Australia	Oceania	81.2	20434176	34435

Each row in this table corresponds to a country in 2007. For each row, we have 5 columns:

1. **Country:** Name of country.
2. **Continent:** Which of the five continents the country is part of. (Note that “Americas” includes countries in both North and South America and that Antarctica is excluded.)
3. **Life Expectancy:** Life expectancy in years.
4. **Population:** Number of people living in the country.
5. **GDP per Capita:** Gross domestic product (in US dollars).

Now consider Figure 3.1, which plots this data for all 142 countries in the data.

Let’s view this plot through the grammar of graphics:

1. The `data` variable **GDP per Capita** gets mapped to the `x`-position aesthetic of the points.
2. The `data` variable **Life Expectancy** gets mapped to the `y`-position aesthetic of the points.
3. The `data` variable **Population** gets mapped to the `size` aesthetic of the points.

¹https://www.ted.com/talks/hans_rosling_shows_the_best_stats_you_ve_ever_seen

²http://www.gapminder.org/tools/#_locale_id=en;&chart-type=bubbles

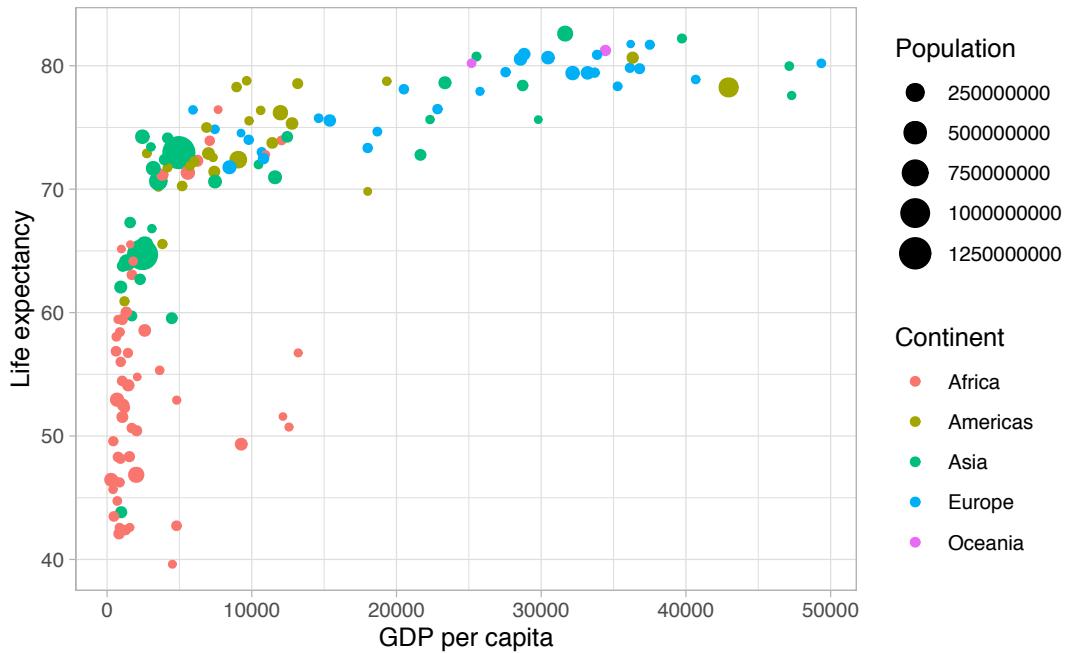


FIGURE 3.1: Life Expectancy over GDP per Capita in 2007.

4. The `data` variable **Continent** gets mapped to the `color` aesthetic of the points.

We'll see shortly that `data` corresponds to the particular data frame where our data is saved and a "data variable" corresponds to a particular column in the data frame. Furthermore, the type of `geometric` object considered in this plot are points. That being said, while in this example we are considering points, graphics are not limited to just points. Other plots involve lines while others involve bars.

Let's summarize the three essential components of the Grammar in Table 3.2.

TABLE 3.2: Summary of Grammar of Graphics for this plot

data variable	aes	geom
GDP per Capita	x	point
Life Expectancy	y	point
Population	size	point
Continent	color	point

3.1.3 Other components

There are other components of the Grammar of Graphics we can control as well. As you start to delve deeper into the Grammar of Graphics, you'll start to encounter these topics more frequently. In this book however, we'll keep things simple and only work with the two additional components listed below:

- `faceting` breaks up a plot into small multiples corresponding to the levels of another variable (Section 3.6)
- `position` adjustments for barplots (Section 3.8)

Other more complex components like `scales` and coordinate systems are left for a more advanced text such as R for Data Science³ (Grolmund and Wickham, 2016). Generally speaking, the Grammar of Graphics allows for a high degree of customization of plots and also a consistent framework for easily updating and modifying them.

3.1.4 ggplot2 package

In this book, we will be using the `ggplot2` package for data visualization, which is an implementation of the Grammar of Graphics for R (Wickham et al., 2019). As we noted earlier, a lot of the previous section was written in a computer code type font. This is because the various components of the Grammar of Graphics are specified in the `ggplot()` function included in the `ggplot2` package, which expects at a minimum as arguments (i.e. inputs):

- The data frame where the variables exist: the `data` argument.
- The mapping of the variables to aesthetic attributes: the `mapping` argument which specifies the `aesthetic` attributes involved.

After we've specified these components, we then add *layers* to the plot using the `+` sign. The most essential layer to add to a plot is the layer that specifies which type of `geometric` object we want the plot to involve: points, lines, bars, and others. Other layers we can add to a plot include layers specifying the plot title, axes labels, visual themes for the plots, and facets (which we'll see in Section 3.6).

Let's now put the theory of the Grammar of Graphics into practice.

³<http://r4ds.had.co.nz/data-visualisation.html#aesthetic-mappings>

3.2 Five Named Graphs - The 5NG

In order to keep things simple, we will only focus on five different types of graphics in this book, each with a commonly given name. We term these “five named graphs” the **5NG**:

1. scatterplots
2. linegraphs
3. boxplots
4. histograms
5. barplots

We will discuss some variations of these plots, but with this basic repertoire of graphics in your toolbox you can visualize a wide array of different variable types. Note that certain plots are only appropriate for categorical variables and while others are only appropriate for quantitative variables. You’ll want to quiz yourself often as we go along on which plot makes sense for a particular problem or data set.

3.3 5NG#1: Scatterplots

The simplest of the 5NG are *scatterplots*, also called bivariate plots. They allow you to visualize the relationship between two numerical variables. While you may already be familiar with scatterplots, let’s view them through the lens of the Grammar of Graphics. Specifically, we will visualize the relationship between the following two numerical variables in the `flights` data frame included in the `nycflights13` package:

1. `dep_delay`: departure delay on the horizontal “x” axis and
2. `arr_delay`: arrival delay on the vertical “y” axis

for Alaska Airlines flights leaving NYC in 2013. This requires paring down the data from all 336,776 flights that left NYC in 2013, to only the 714 *Alaska Airlines* flights that left NYC in 2013.

What this means computationally is: we’ll take the `flights` data frame, extract only the 714 rows corresponding to Alaska Airlines flights, and save this in a new data frame called `alaska_flights`. Run the code below to do this:

```
alaska_flights <- flights %>%
  filter(carrier == "AS")
```

For now we suggest you ignore how this code works; we'll explain this in detail in Chapter 4 when we cover data wrangling. However, convince yourself that this code does what it is supposed to by running `View(alaska_flights)`: it creates a new data frame with name `alaska_flights` consisting of only the 714 Alaska Airlines flights. Note the use of the `<-` operator here. This is what gives the name of `alaska_flights` to the data frame returned from the operation on the right-hand-side of it.

We'll see later in Chapter 4 on data wrangling that this code uses the `dplyr` package for data wrangling to achieve our goal: it takes the `flights` data frame and filters it to only return the rows where `carrier` is equal to "AS", Alaska Airlines' carrier code. Other examples of carrier codes include "AA" for American Airlines and "UA" for United Airlines. Recall from Section 2.2 that testing for equality is specified with `==` and not `=`. Fasten your seat belts and sit tight for now however, we'll introduce these ideas more fully in Chapter 4.

Learning check

(LC3.1) Take a look at both the `flights` and `alaska_flights` data frames by running `View(flights)` and `View(alaska_flights)`. In what respect do these data frames differ? For example, think about the number of rows in each dataset.

3.3.1 Scatterplots via geom_point

Let's now go over the code that will create the desired scatterplot, keeping in mind our discussion on the Grammar of Graphics in Section 3.1. We'll be using the `ggplot()` function included in the `ggplot2` package.

```
ggplot(data = alaska_flights, mapping = aes(x = dep_delay, y = arr_delay)) +
  geom_point()
```

Let's break this down piece-by-piece:

- Within the `ggplot()` function, we specify two of the components of the Grammar of Graphics as arguments (i.e. inputs):

1. The data frame to be `alaska_flights` by setting `data = alaska_flights`.
2. The aesthetic mapping by setting `aes(x = dep_delay, y = arr_delay)`. Specifically:
 - the variable `dep_delay` maps to the `x` position aesthetic
 - the variable `arr_delay` maps to the `y` position aesthetic
- We add a layer to the `ggplot()` function call using the `+` sign. The layer in question specifies the third component of the grammar: the geometric object. In this case the geometric object is set to points by specifying `geom_point()`.

After running the above code, you'll notice two outputs: a warning message and the graphic shown in Figure 3.2. Let's first unpack the warning message:

```
Warning: Removed 5 rows containing missing values
(geom_point).
```

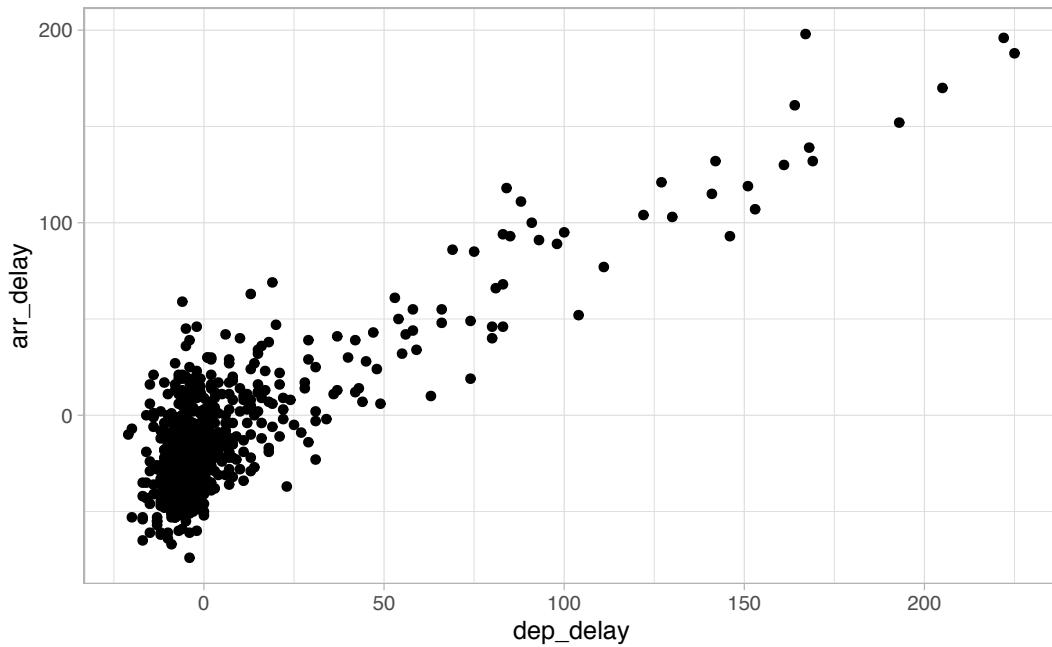


FIGURE 3.2: Arrival delays vs departure delays for Alaska Airlines flights from NYC in 2013.

After running the above code, R returns a warning message alerting us to the fact that 5 rows were ignored due to them being missing. For 5 rows either the value for `dep_delay` or `arr_delay` or both were missing (recorded in R as `NA`), and thus these rows were ignored in our plot. Turning our attention to the resulting scatterplot in Figure 3.2, we see that a positive relationship exists between `dep_delay` and `arr_delay`: as departure delays increase, arrival delays tend to also increase. We also note the large mass of points clustered near (0, 0).

Before we continue, let's consider a few more notes on the layers in the above code that generated the scatterplot:

- Note that the + sign comes at the end of lines, and not at the beginning. You'll get an error in R if you put it at the beginning.
- When adding layers to a plot, you are encouraged to start a new line after the + so that the code for each layer is on a new line. As we add more and more layers to plots, you'll see this will greatly improve the legibility of your code.
- To stress the importance of adding layers in particular the layer specifying the `geometric` object, consider Figure 3.3 where no layers are added. A not very useful plot!

```
ggplot(data = alaska_flights, mapping = aes(x = dep_delay, y = arr_delay))
```

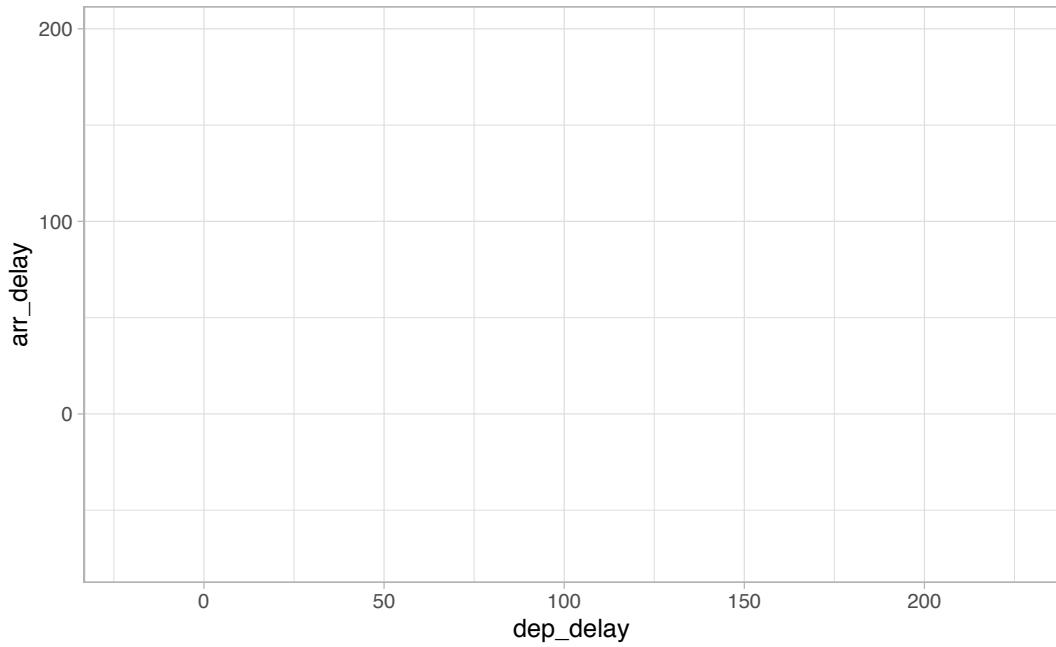


FIGURE 3.3: A plot with no layers.

Learning check

(LC3.2) What are some practical reasons why `dep_delay` and `arr_delay` have a positive relationship?

(LC3.3) What variables in the `weather` data frame would you expect to have a negative correlation (i.e. a negative relationship) with `dep_delay`? Why? Re-

member that we are focusing on numerical variables here. Hint: Explore the `weather` dataset by using the `view()` function.

(LC3.4) Why do you believe there is a cluster of points near (0, 0)? What does (0, 0) correspond to in terms of the Alaskan flights?

(LC3.5) What are some other features of the plot that stand out to you?

(LC3.6) Create a new scatterplot using different variables in the `alaska_flights` data frame by modifying the example above.

3.3.2 Over-plotting

The large mass of points near (0, 0) in Figure 3.2 can cause some confusion as it is hard to tell the true number of points that are plotted. This is the result of a phenomenon called *overplotting*. As one may guess, this corresponds to values being plotted on top of each other *over* and *over* again. It is often difficult to know just how many values are plotted in this way when looking at a basic scatterplot as we have here. There are two methods to address the issue of overplotting:

1. By adjusting the transparency of the points.
2. By adding a little random “jitter”, or random “nudges”, to each of the points.

Method 1: Changing the transparency

The first way of addressing overplotting is by changing the transparency of the points by using the `alpha` argument in `geom_point()`. By default, this value is set to 1. We can change this to any value between 0 and 1, where 0 sets the points to be 100% transparent and 1 sets the points to be 100% opaque. Note how the following code is identical to the code in Section 3.3 that created the scatterplot with overplotting, but with `alpha = 0.2` added to the `geom_point()`:

```
ggplot(data = alaska_flights, mapping = aes(x = dep_delay, y = arr_delay)) +
  geom_point(alpha = 0.2)
```

The key feature to note in Figure 3.4 is that the transparency of the points is cumulative: areas with a high-degree of overplotting are darker, whereas areas with a lower degree are less dark. Note furthermore that there is no `aes()` surrounding `alpha = 0.2`. This is because we are not mapping a variable to an

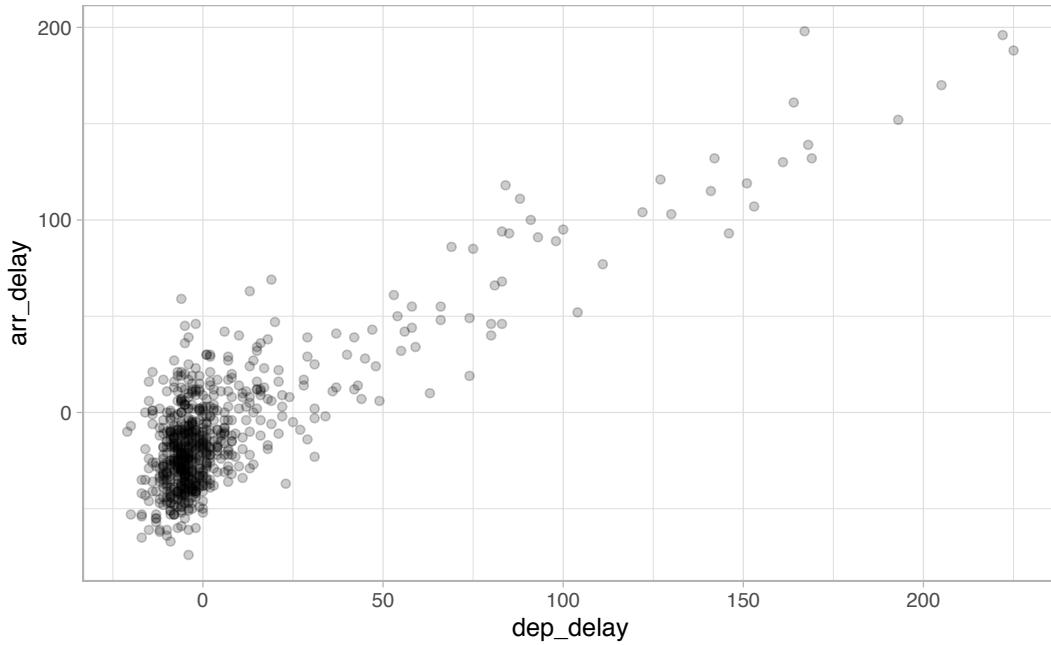


FIGURE 3.4: Arrival vs departure delays scatterplot with $\alpha = 0.2$.

aesthetic attribute, but rather merely changing the default setting of `alpha`. In fact, you'll receive an error if you try to change the second line above to read `geom_point(aes(alpha = 0.2))`.

Method 2: Jittering the points

The second way of addressing overplotting is by *jittering* all the points, in other words give each point a small nudge in a random direction. You can think of “jittering” as shaking the points around a bit on the plot. Let's illustrate using a simple example first. Say we have a data frame `jitter_example` with 4 rows of identical value 0 for both `x` and `y`:

```
# A tibble: 4 x 2
  x     y
  <dbl> <dbl>
1     0     0
2     0     0
3     0     0
4     0     0
```

We display the resulting scatterplot in Figure 3.5; observe that the 4 points are superimposed on top of each other. While we know there are 4 values being plotted, this fact might not be apparent to others.

In Figure 3.6 we instead display a *jittered scatterplot* where each point is given

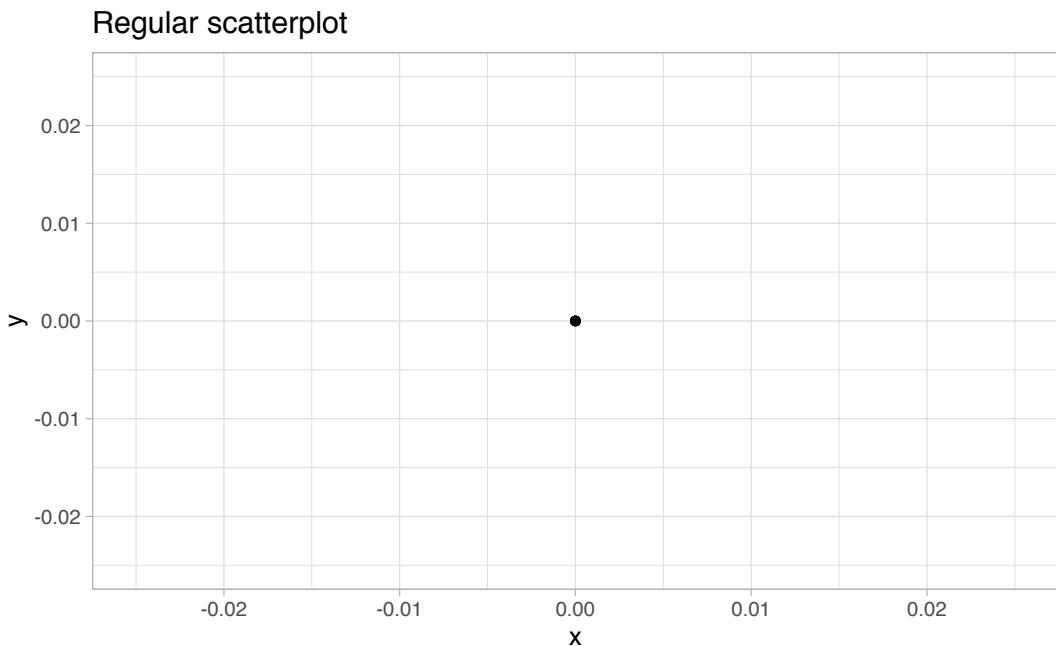


FIGURE 3.5: Regular scatterplot of example data.

a random “nudge.” It is now plainly evident that this plot involves four points. Keep in mind that jittering is strictly a visualization tool; even after creating a jittered scatterplot, the original values saved in `jitter_example` remain unchanged.

To create a jittered scatterplot, instead of using `geom_point()`, we use `geom_jitter()`. To specify how much jitter to add, we adjust the `width` and `height` arguments. This corresponds to how hard you’d like to shake the plot in units corresponding to those for both the horizontal and vertical variables (in this case minutes).

```
ggplot(data = alaska_flights, mapping = aes(x = dep_delay, y = arr_delay)) +
  geom_jitter(width = 30, height = 30)
```

Observe how the above code is identical to the code that created the scatterplot with overplotting in Subsection 3.3.1, but with `geom_point()` replaced with `geom_jitter()`.

The resulting plot in Figure 3.7 helps us a little bit in getting a sense for the overplotting, but with a relatively large data set like this one (714 flights), it can be argued that changing the transparency of the points by setting `alpha` proved more effective. In terms of how much jitter one should add using the `width` and `height` arguments, it is important to add just enough jitter to break

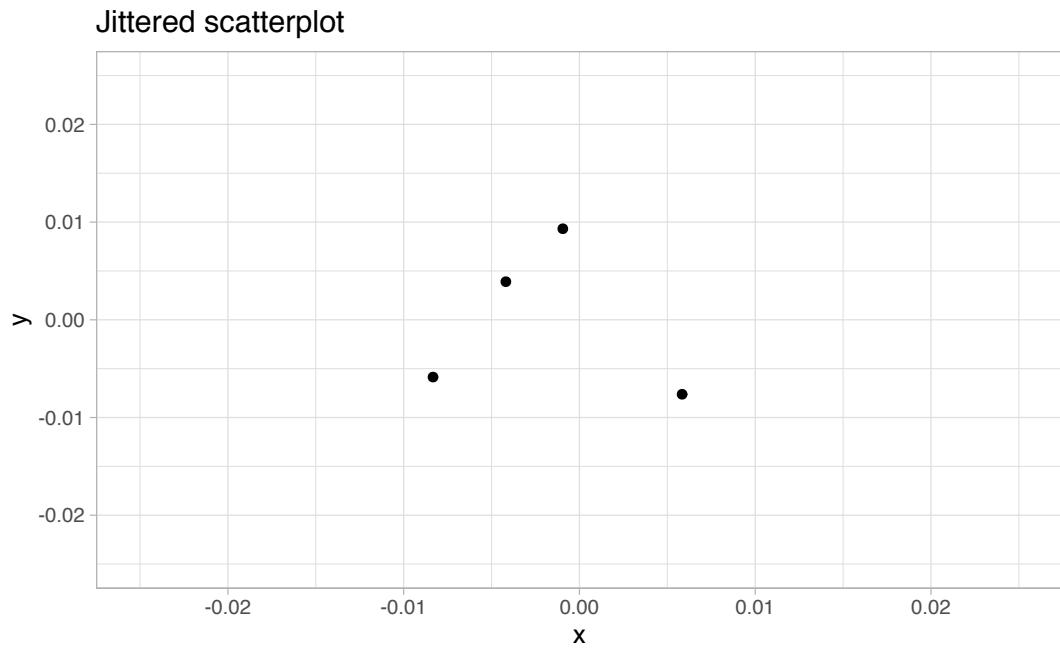


FIGURE 3.6: Jittered scatterplot of example data.

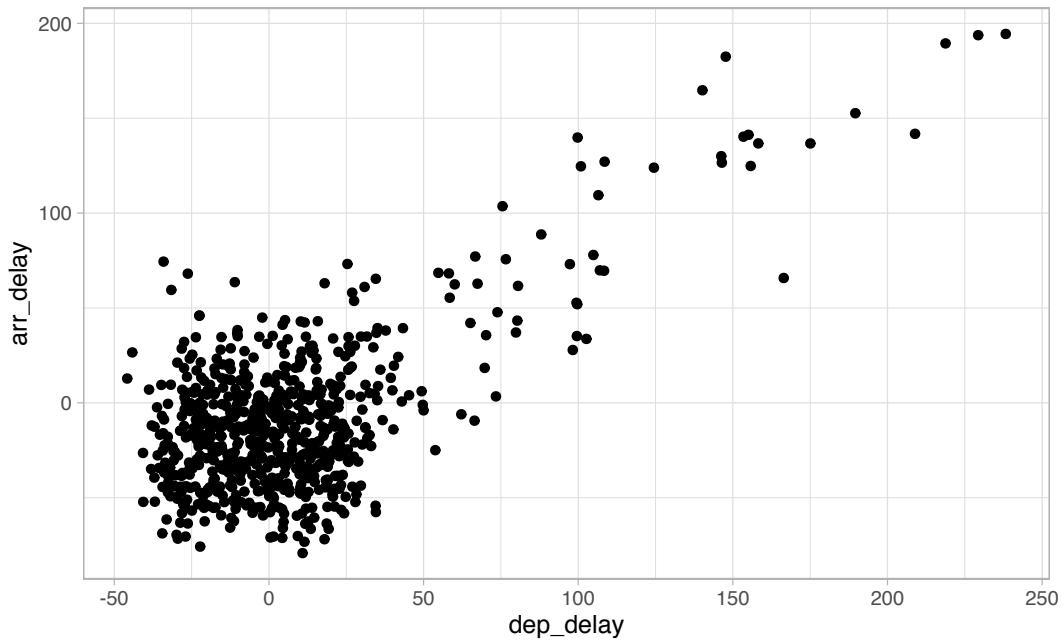


FIGURE 3.7: Arrival vs departure delays jittered scatterplot.

any overlap in points, but not so much that we completely alter the overall pattern in points.

When would it be better to use a jittered scatterplot? When would it be better to alter the points' transparency? There is no single right answer that applies to all situations. You need to make a subjective choice and own that choice. At the very least, when confronted with overplotting, we suggest making both types of plots and seeing which plot better emphasizes the point you are trying to make.

Learning check

(LC3.7) Why is setting the `alpha` argument value useful with scatterplots? What further information does it give you that a regular scatterplot cannot?

(LC3.8) After viewing the Figure 3.4 above, give an approximate range of arrival delays and departure delays that occur the most frequently. How has that region changed compared to when you observed the same plot without the `alpha = 0.2` set in Figure 3.2?

3.3.3 Summary

Scatterplots display the relationship between two numerical variables. They are among the most commonly used plots because they can provide an immediate way to see the trend in one variable versus another. However, if you try to create a scatterplot where either one of the two variables is not numerical, you might get strange results. Be careful!

With medium to large data sets, you may need to play around with the different modifications one can make to a scatterplot. This tweaking is often a fun part of data visualization, since you'll have the chance to see different relationships come about as you make subtle changes to your plots.

3.4 5NG#2: Linegraphs

The next of the five named graphs are linegraphs. Linegraphs show the relationship between two numerical variables when the variable on the x-axis, also

called the *explanatory* variable, is of a sequential nature; in other words there is an inherent ordering to the variable. The most common example of linegraphs have some notion of time on the x-axis: hours, days, weeks, years, etc. Since time is sequential, we connect consecutive observations of the variable on the y-axis with a line. Linegraphs that have some notion of time on the x-axis are also called *time series* plots. Linegraphs should be avoided when there is not a clear sequential ordering to the variable on the x-axis. Let's illustrate linegraphs using another data set in the `nycflights13` package: the `weather` data frame.

Let's get a sense for the `weather` data frame:

- Explore the `weather` data by running `View(weather)`.
- Run `?weather` to bring up the help file.

We can see that there is a variable called `temp` of hourly temperature recordings in Fahrenheit at weather stations near all three airports in New York City: Newark (origin code `EWR`), JFK, and La Guardia (`LGA`). Instead of considering hourly temperatures for all days in 2013 for all three airports however, for simplicity let's only consider hourly temperatures at only Newark airport for the first 15 days in January.

Recall in Section 3.3 we used the `filter()` function to only choose the subset of rows of `flights` corresponding to Alaska Airlines flights. We similarly use `filter()` here, but by using the `&` operator we only choose the subset of rows of `weather` where

1. The `origin` is "EWR" and
2. the `month` is January and
3. the `day` is between 1 and 15

```
early_january_weather <- weather %>%  
  filter(origin == "EWR" & month == 1 & day <= 15)
```

Learning check

(LC3.9) Take a look at both the `weather` and `early_january_weather` data frames by running `View(weather)` and `View(early_january_weather)`. In what respect do these data frames differ?

(LC3.10) `View()` the `flights` data frame again. Why does the `time_hour` variable uniquely identify the hour of the measurement whereas the `hour` variable does not?

3.4.1 Linegraphs via geom_line

Let's plot a linegraph of hourly temperatures in `early_january_weather` by using `geom_line()` instead of `geom_point()` like we did for scatterplots:

```
ggplot(data = early_january_weather, mapping = aes(x = time_hour, y = temp)) +
  geom_line()
```

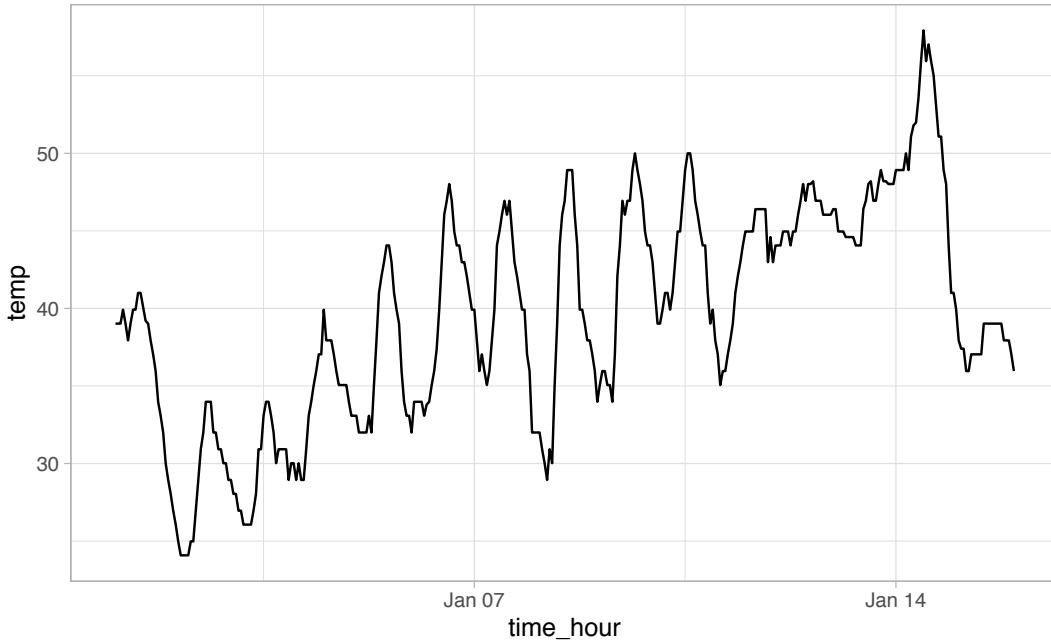


FIGURE 3.8: Hourly temperature in Newark for January 1-15, 2013.

Much as with the `ggplot()` code that created the scatterplot of departure and arrival delays for Alaska Airlines flights in Figure 3.2, let's break down the above code piece-by-piece in terms of the Grammar of Graphics:

- Within the `ggplot()` function call, we specify two of the components of the Grammar of Graphics as arguments:
 1. The data frame to be `early_january_weather` by setting `data = early_january_weather`
 2. The aesthetic mapping by setting `aes(x = time_hour, y = temp)`. Specifically:

- the variable `time_hour` maps to the `x` position aesthetic.
 - the variable `temp` maps to the `y` position aesthetic
- We add a layer to the `ggplot()` function call using the `+` sign. The layer in question specifies the third component of the grammar: the `geometric` object in question. In this case the geometric object is a `line`, set by specifying `geom_line()`.

Learning check

(LC3.11) Why should linegraphs be avoided when there is not a clear ordering of the horizontal axis?

(LC3.12) Why are linegraphs frequently used when time is the explanatory variable on the `x`-axis?

(LC3.13) Plot a time series of a variable other than `temp` for Newark Airport in the first 15 days of January 2013.

3.4.2 Summary

Linegraphs, just like scatterplots, display the relationship between two numerical variables. However it is preferred to use linegraphs over scatterplots when the variable on the `x`-axis (i.e. the explanatory variable) has an inherent ordering, like some notion of time.

3.5 5NG#3: Histograms

Let's consider the `temp` variable in the `weather` data frame once again, but unlike with the linegraphs in Section 3.4, let's say we don't care about the relationship of temperature to time, but rather we only care about how the values of `temp` *distribute*. In other words:

1. What are the smallest and largest values?
2. What is the “center” value?
3. How do the values spread out?
4. What are frequent and infrequent values?

One way to visualize this *distribution* of this single variable `temp` is to plot them on a horizontal line as we do in Figure 3.9:

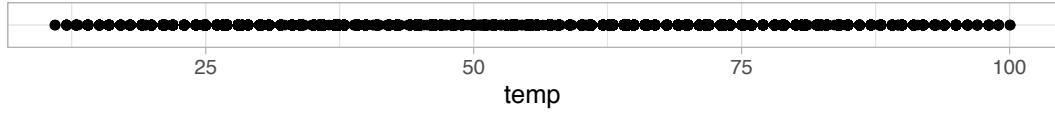


FIGURE 3.9: Plot of hourly temperature recordings from NYC in 2013.

This gives us a general idea of how the values of `temp` distribute: observe that temperatures vary from around 11°F up to 100°F. Furthermore, there appear to be more recorded temperatures between 40°F and 60°F than outside this range. However, because of the high degree of overlap in the points, it's hard to get a sense of exactly how many values are between, say, 50°F and 55°F.

What is commonly produced instead of the above plot is known as a *histogram*. A histogram is a plot that visualizes the *distribution* of a numerical value as follows:

1. We first cut up the x-axis into a series of *bins*, where each bin represents a range of values.
2. For each bin, we count the number of observations that fall in the range corresponding to that bin.
3. Then for each bin, we draw a bar whose height marks the corresponding count.

Let's drill-down on an example of a histogram, shown in Figure 3.10.

Observe that there are three bins of equal width between 30°F and 60°F, thus we have three bins of width 10°F each: one bin for the 30-40°F range, another bin for the 40-50°F range, and another bin for the 50-60°F range. Since:

1. The bin for the 30-40°F range has a height of around 5000, this histogram is telling us that around 5000 of the hourly temperature recordings are between 30°F and 40°F.
2. The bin for the 40-50°F range has a height of around 4300, this histogram is telling us that around 4300 of the hourly temperature recordings are between 40°F and 50°F.
3. The bin for the 50-60°F range has a height of around 3500, this histogram is telling us that around 3500 of the hourly temperature recordings are between 50°F and 60°F.

The remaining bins all have a similar interpretation.

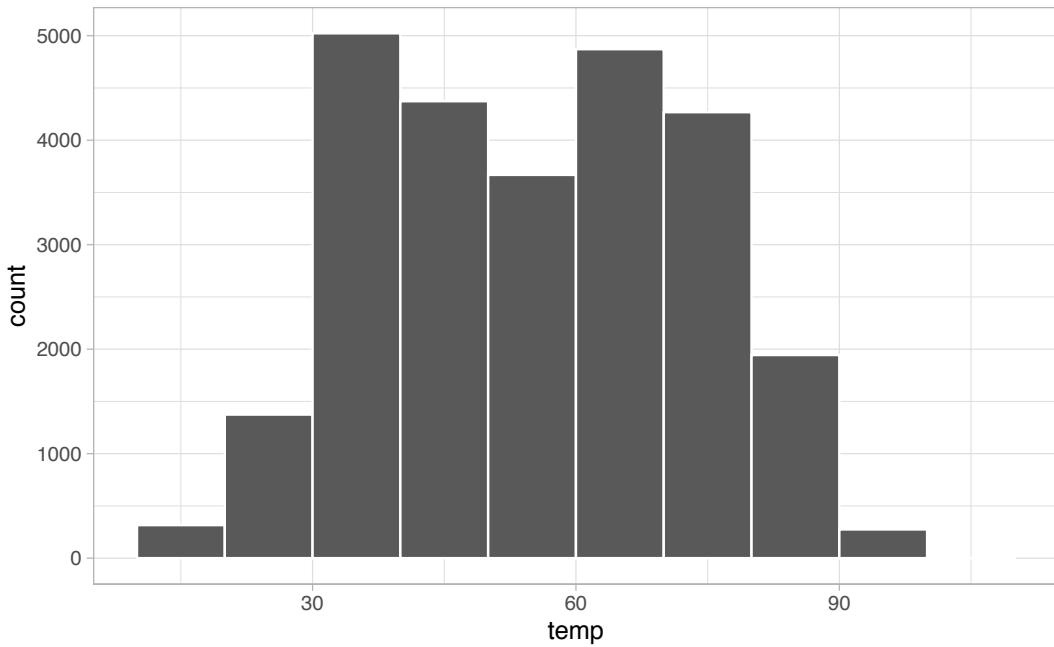


FIGURE 3.10: Example histogram.

3.5.1 Histograms via `geom_histogram`

Let's now present the `ggplot()` code to plot your first histogram! Unlike with scatterplots and linegraphs, there is now only one variable being mapped in `aes()`: the single numerical variable `temp`. The y-aesthetic of a histogram gets computed for you automatically. Furthermore, the geometric object layer is now a `geom_histogram()`.

```
ggplot(data = weather, mapping = aes(x = temp)) +
  geom_histogram()

`stat_bin()` using `bins = 30`. Pick better value with
`binwidth`.

Warning: Removed 1 rows containing non-finite values
(stat_bin).
```

Let's unpack the messages R sent us first. The first message is telling us that the histogram was constructed using `bins = 30`, in other words 30 equally spaced bins. This is known in computer programming as a default value; unless you override this default number of bins with a number you specify, R will choose 30 by default. We'll see in the next section how to change this default number of bins. The second message is telling us something similar to the warning

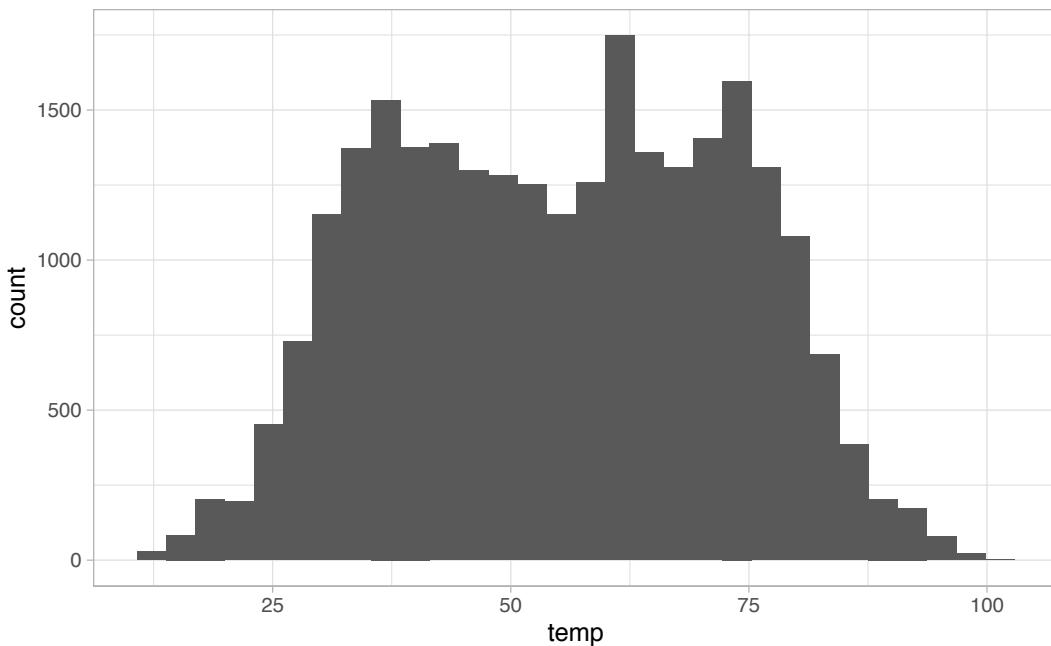


FIGURE 3.11: Histogram of hourly temperatures at three NYC airports.

message we received when we ran the code to create a scatterplot of departure and arrival delays for Alaska Airlines flights in Figure 3.2: that because one row has a missing `NA` value for `temp`, it was omitted from the histogram. R is just giving us a friendly heads up that this was the case.

Now's let's unpack the resulting histogram in Figure 3.11. Observe that values less than 25°F as well as values above 80°F are rather rare. However, because of the large number of bins, its hard to get a sense for which range of temperatures is covered by each bin; everything is one giant amorphous blob. So let's add white vertical borders demarcating the bins by adding a `color = "white"` argument to `geom_histogram()`:

```
ggplot(data = weather, mapping = aes(x = temp)) +
  geom_histogram(color = "white")
```

We can now better associate ranges of temperatures to each of the bins. We can also vary the color of the bars by setting the `fill` argument. Run `colors()` to see all 657 possible choice of colors!

```
ggplot(data = weather, mapping = aes(x = temp)) +
  geom_histogram(color = "white", fill = "steelblue")
```

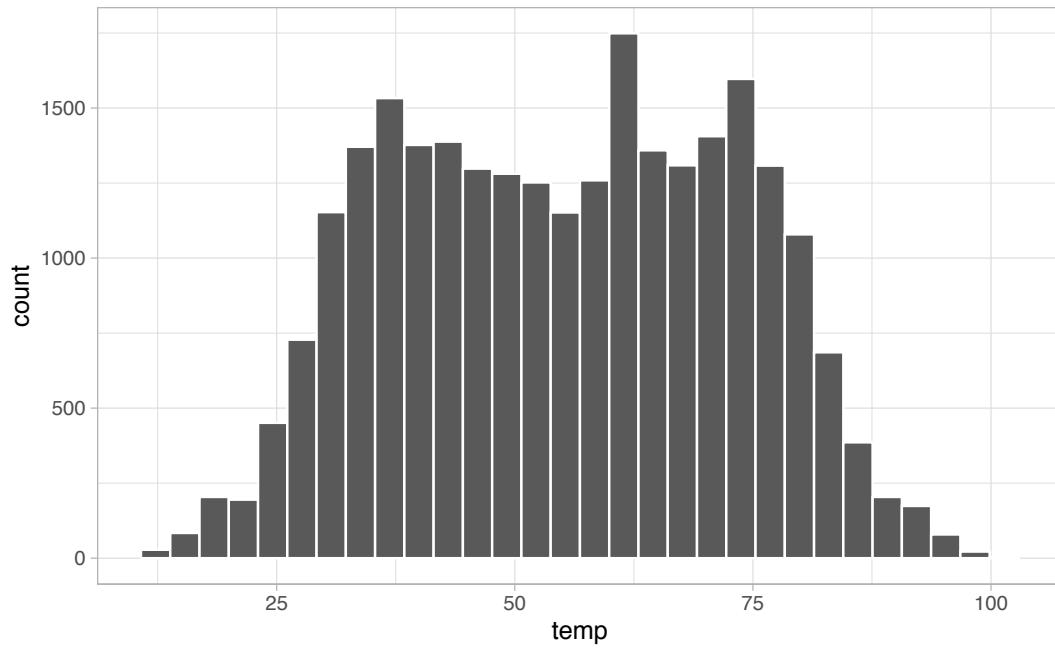


FIGURE 3.12: Histogram of hourly temperatures at three NYC airports with white borders.

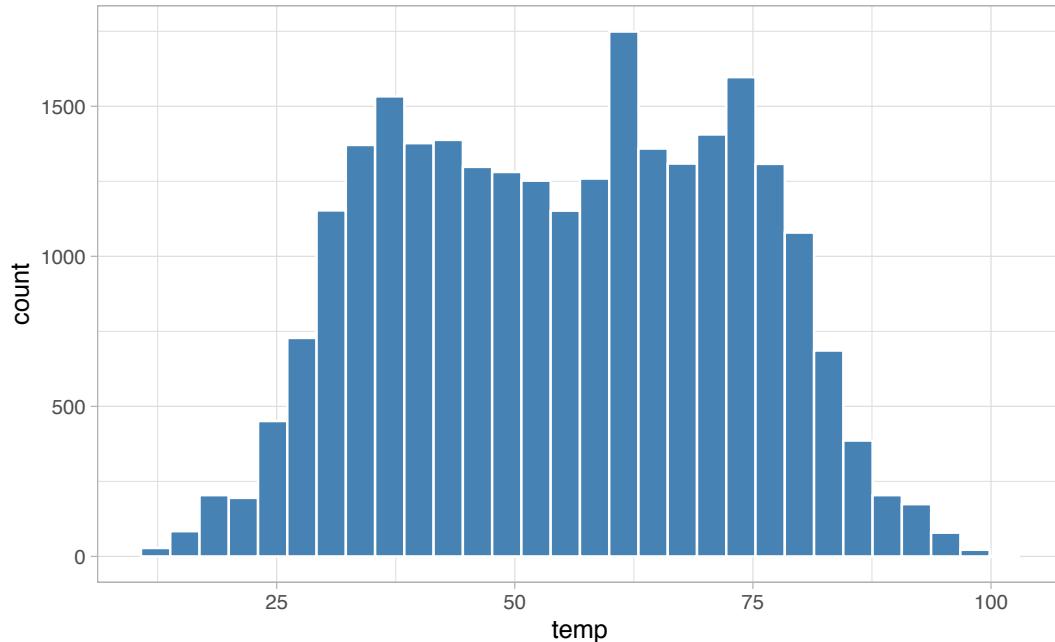


FIGURE 3.13: Histogram of hourly temperatures at three NYC airports with white borders.

3.5.2 Adjusting the bins

Observe in both Figure 3.12 and Figure 3.13 that in the 50-75°F range there appear to be roughly 8 bins. Thus each bin has width 25 divided by 8, or roughly 3.12°F which is not a very easily interpretable range to work with. Let's now adjust the number of bins in our histogram in one of two methods:

1. By adjusting the number of bins via the `bins` argument to `geom_histogram()`.
2. By adjusting the width of the bins via the `binwidth` argument to `geom_histogram()`.

Using the first method, we have the power to specify how many bins we would like to cut the x-axis up in. As mentioned in the previous section, the default number of bins is 30. We can override this default, to say 40 bins, as follows:

```
ggplot(data = weather, mapping = aes(x = temp)) +
  geom_histogram(bins = 40, color = "white")
```

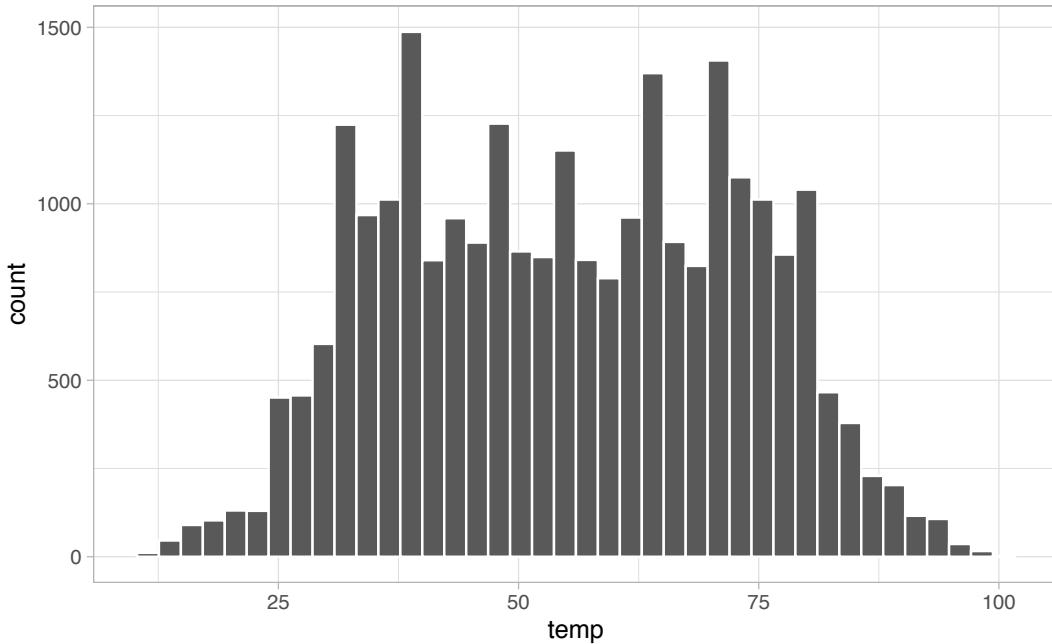


FIGURE 3.14: Histogram with 40 bins.

Using the second method, instead of specifying the number of bins, we specify the width of the bins by using the `binwidth` argument in the `geom_histogram()` layer. For example, let's set the width of each bin to be 10°F.

```
ggplot(data = weather, mapping = aes(x = temp)) +  
  geom_histogram(binwidth = 10, color = "white")
```

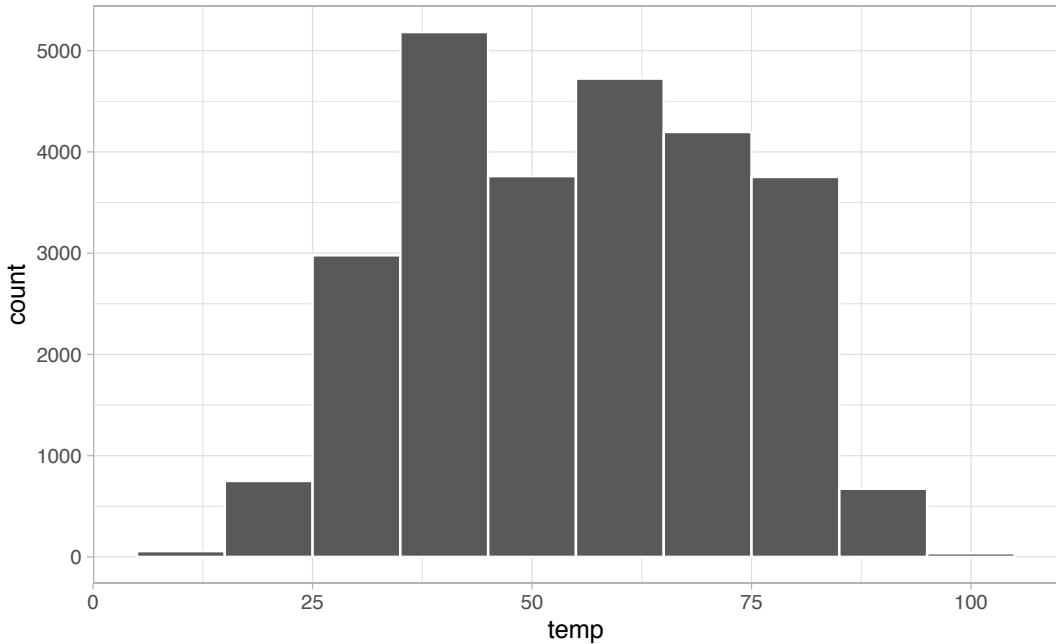


FIGURE 3.15: Histogram with binwidth 10.

Learning check

(LC3.14) What does changing the number of bins from 30 to 40 tell us about the distribution of temperatures?

(LC3.15) Would you classify the distribution of temperatures as symmetric or skewed?

(LC3.16) What would you guess is the “center” value in this distribution? Why did you make that choice?

(LC3.17) Is this data spread out greatly from the center or is it close? Why?

3.5.3 Summary

Histograms, unlike scatterplots and linegraphs, present information on only a single numerical variable. Specifically, they are visualizations of the distribution of the numerical variable in question.

3.6 Facets

Before continuing the 5NG, let's briefly introduce a new concept called *faceting*. Faceting is used when we'd like to split a particular visualization of variables by another variable. This will create multiple copies of the same type of plot with matching x and y axes, but whose content will differ.

For example, suppose we were interested in looking at how the histogram of hourly temperature recordings at the three NYC airports we saw in Section 3.5 differed by month. We would “split” this histogram by the 12 possible months in a given year, in other words plot histograms of `temp` for each `month`. We do this by adding `facet_wrap(~ month)` layer.

```
ggplot(data = weather, mapping = aes(x = temp)) +
  geom_histogram(binwidth = 5, color = "white") +
  facet_wrap(~ month)
```

Note the use of the tilde ~ before `month` in `facet_wrap()`. The tilde is required and you'll receive the error `Error in as.quoted(facets) : object 'month' not found` if you don't include it before `month` here. We can also specify the number of rows and columns in the grid by using the `nrow` and `ncol` arguments inside of `facet_wrap()`. For example, say we would like our faceted plot to have 4 rows instead of 3. Add the `nrow = 4` argument to `facet_wrap(~ month)`

```
ggplot(data = weather, mapping = aes(x = temp)) +
  geom_histogram(binwidth = 5, color = "white") +
  facet_wrap(~ month, nrow = 4)
```

Observe in both Figure 3.16 and Figure 3.17 that as we might expect in the

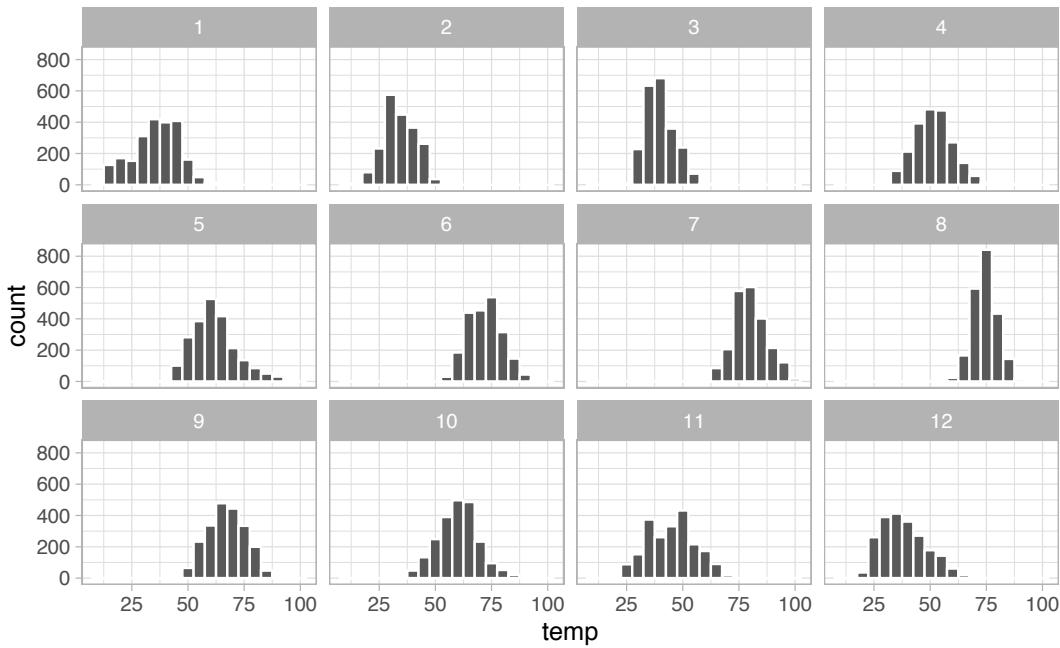


FIGURE 3.16: Faceted histogram of hourly temperatures by month.

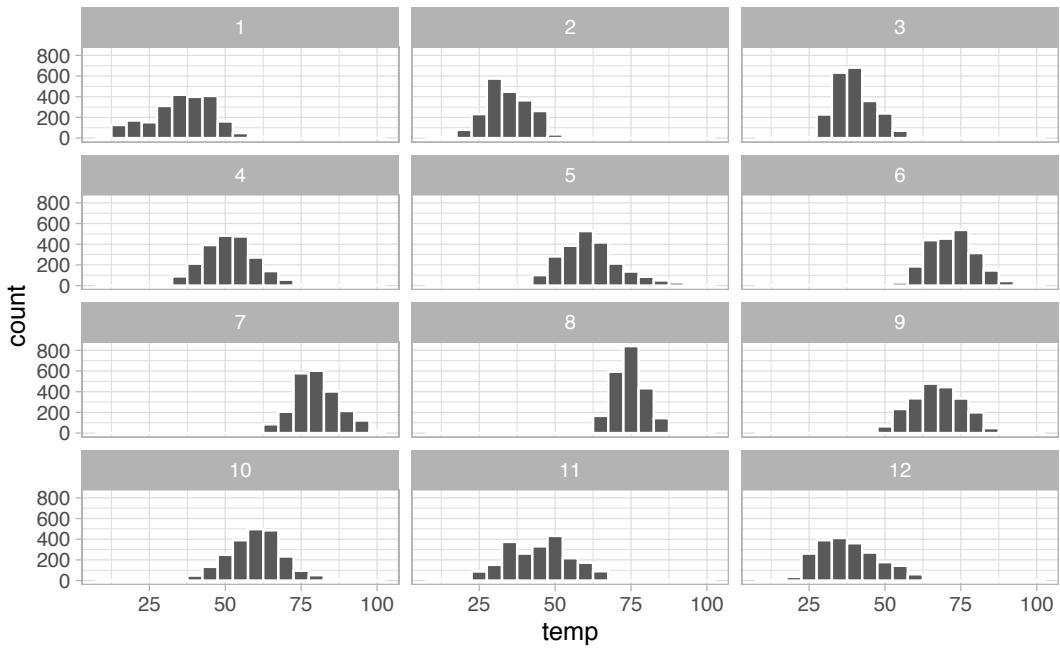


FIGURE 3.17: Faceted histogram with 4 instead of 3 rows.

Northern Hemisphere, temperatures tend to be higher in the summer months, while they tend to be lower in the winter.

Learning check

(LC3.18) What other things do you notice about the faceted plot above? How does a faceted plot help us see relationships between two variables?

(LC3.19) What do the numbers 1-12 correspond to in the plot above? What about 25, 50, 75, 100?

(LC3.20) For which types of data sets would these types of faceted plots not work well in comparing relationships between variables? Give an example describing the nature of these variables and other important characteristics.

(LC3.21) Does the `temp` variable in the `weather` data set have a lot of variability? Why do you say that?

3.7 5NG#4: Boxplots

While faceted histograms are one visualization that allows us to compare distributions of a numerical variable split by another variable, another visualization that achieves this same goal are *side-by-side boxplots*. A boxplot is constructed from the information provided in the *five-number summary* of a numerical variable (see Appendix A). To keep things simple for now, let's only consider hourly temperature recordings for the month of November in Figure 3.18.

These 2141 observations have the following five-number summary:

1. Minimum: 21.02°F
2. First quartile AKA 25th percentile: 35.96°F
3. Median AKA second quartile AKA 50th percentile: 44.96°F
4. Third quartile AKA 75th percentile: 51.98°F
5. Maximum: 71.06°F

Let's mark these 5 values with dashed horizontal lines in Figure 3.19.

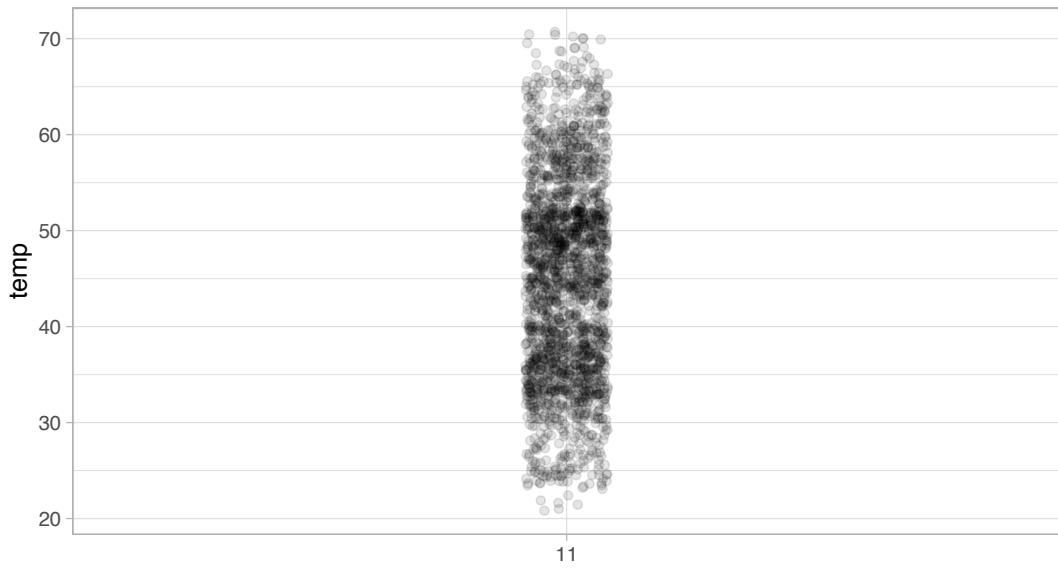


FIGURE 3.18: November temperatures represented as points.

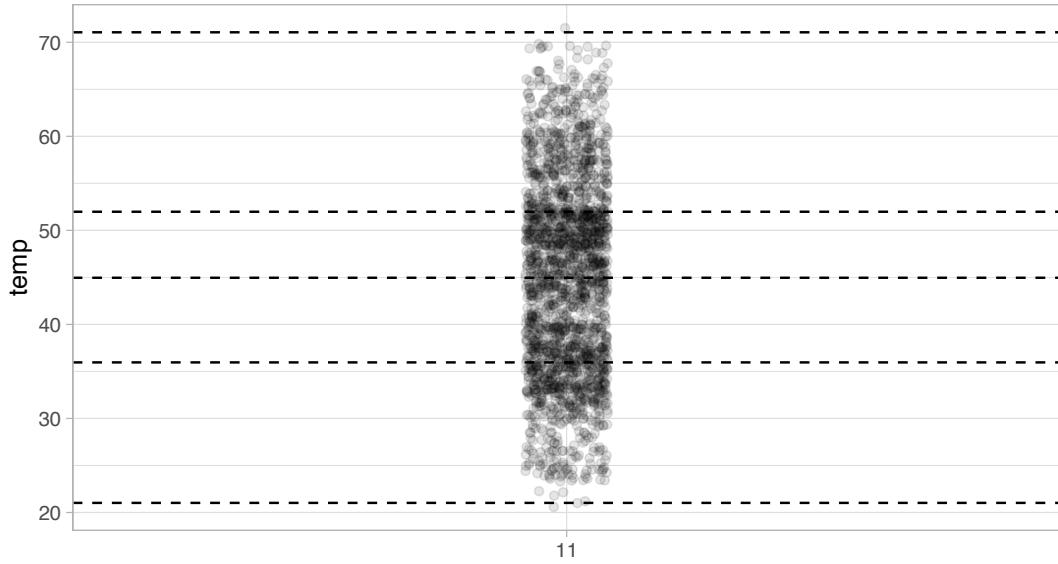


FIGURE 3.19: November temperatures with five-number summary.

Let's add the boxplot underneath these points and dashed horizontal lines in Figure 3.20.

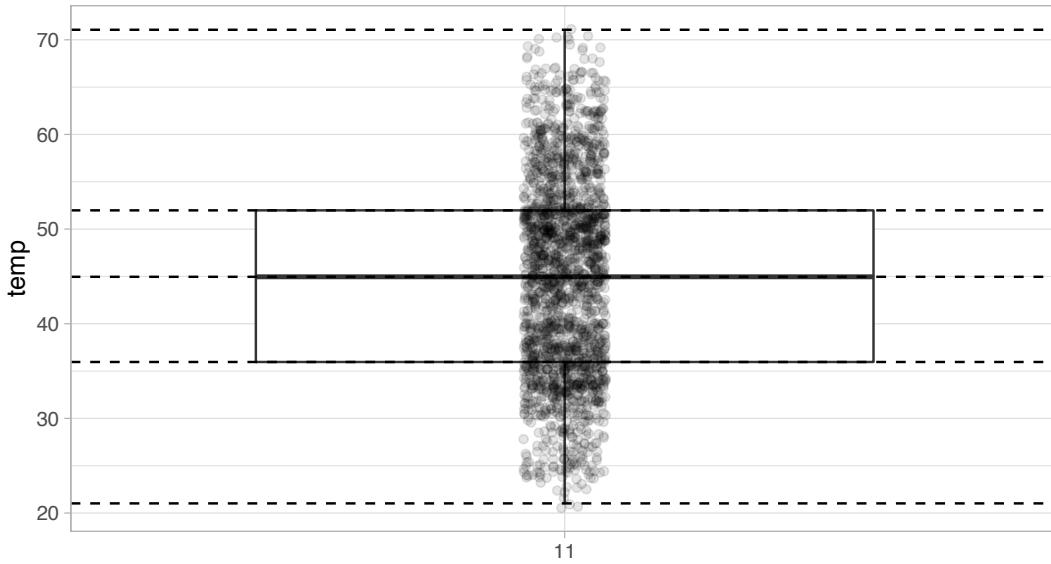


FIGURE 3.20: November temperatures with five-number summary and boxplot.

What the boxplot does summarize the 2141 points by emphasizing that:

1. 25% of points (about 534 observations) fall below the bottom edge of the box, which is the first quartile of 35.96°F. In other words 25% of observations were colder than 35.96°F.
2. 25% of points fall between the bottom edge of the box and the solid middle line, which is the median of 44.96°F. In other words 25% of observations were between 35.96 and 44.96°F and 50% of observations were colder than 44.96°F.
3. 25% of points fall between the solid middle line and the top edge of the box, which is the third quartile of 51.98°F. In other words 25% of observations were between 44.96 and 51.98°F and 75% of observations were colder than 51.98°F.
4. 25% of points fall over the top edge of the box. In other words 25% of observations were warmer than 51.98°F.
5. The middle 50% of points lie within the *interquartile range* between the first and third quartile of $51.98 - 35.96 = 16.02^{\circ}\text{F}$.

Lastly, for clarity's sake let's remove the points but keep the dashed horizontal lines in Figure 3.21.

We can now better see the *whiskers* of the boxplot. They stick out from either

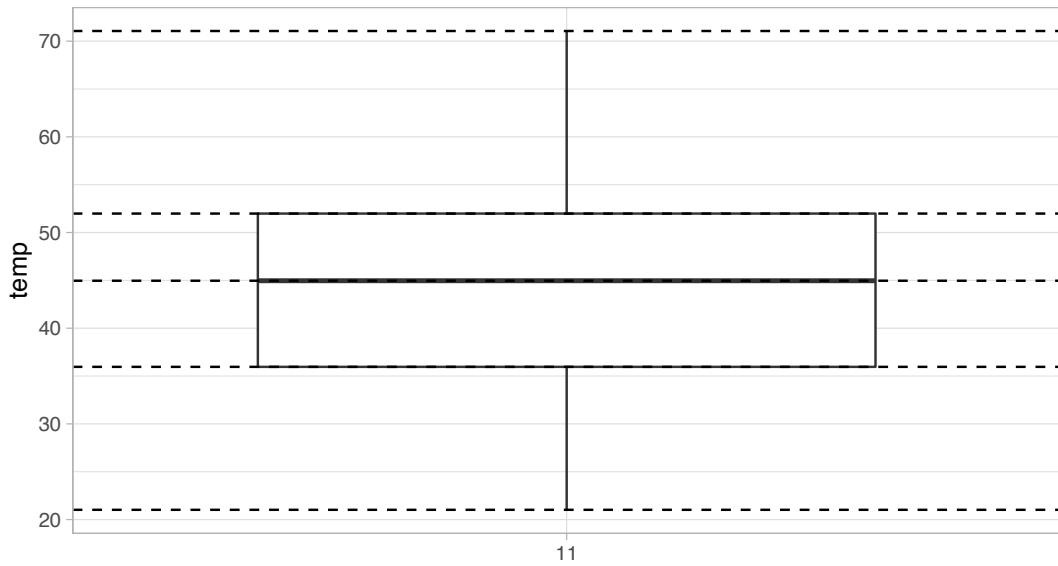


FIGURE 3.21: November temperatures boxplot.

end of the box all the way to the minimum and maximum observed temperatures of 21.02°F and 71.06°F respectively. However, the whiskers don't always extend to the smallest and largest observed values. They in fact can extend no more than $1.5 \times$ the interquartile range from either end of the box, in this case $1.5 \times 16.02^{\circ}\text{F} = 24.03^{\circ}\text{F}$ from either end of the box. Any observed values outside this whiskers get marked with points called *outliers*, which we'll see in the next section.

3.7.1 Boxplots via geom_boxplot

Let's now create a side-by-side boxplot of hourly temperatures split by the 12 months as we did above with the faceted histograms. We do this by mapping the `month` variable to the x-position aesthetic, the `temp` variable to the y-position aesthetic, and by adding a `geom_boxplot()` layer:

```
ggplot(data = weather, mapping = aes(x = month, y = temp)) +
  geom_boxplot()
```

Warning messages:

```
1: Continuous x aesthetic -- did you forget aes(group=...)?
2: Removed 1 rows containing non-finite values (stat_boxplot).
```

Observe in Figure 3.22 that this plot does not provide information about temperature separated by month. The warning messages clue us in as to why.

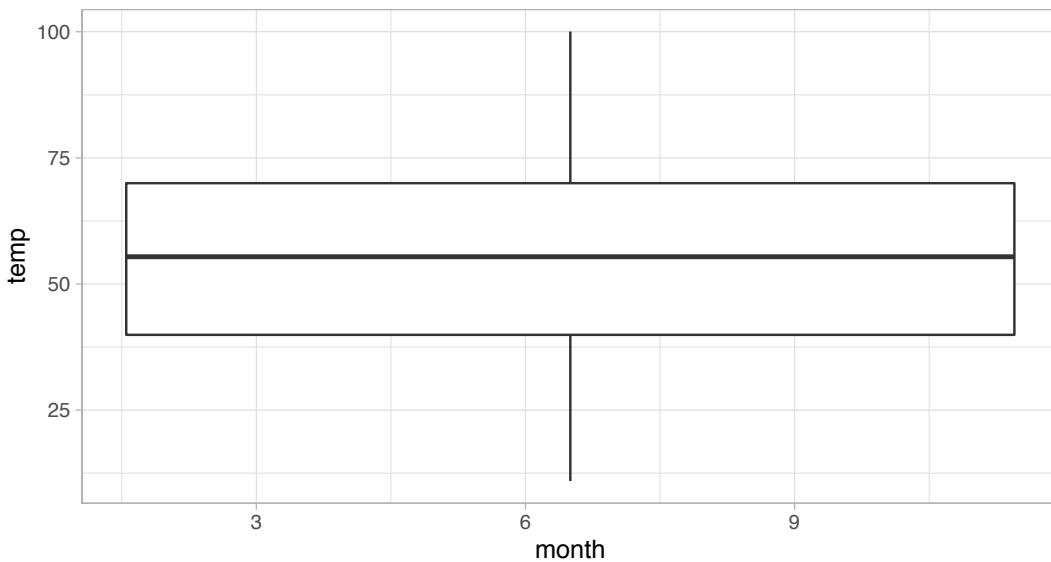


FIGURE 3.22: Invalid boxplot specification.

The second warning message is identical to the warning message when plotting a histogram of hourly temperatures: that one of the values was recorded as `NA` missing. However, the first warning message is telling us that we have a “continuous”, or numerical variable, on the x-position aesthetic. Boxplots however require a categorical variable on the x-axis.

We can convert the numerical variable `month` into a categorical variable by using the `factor()` function. So after applying `factor(month)`, `month` goes from having numerical values 1, 2, ..., 12 to having labels “1”, “2”, ..., “12.”

```
ggplot(data = weather, mapping = aes(x = factor(month), y = temp)) +
  geom_boxplot()
```

The resulting Figure 3.23 shows 12 separate “box and whiskers” plots with the features we saw earlier focusing only on November:

- The “box” portions of this visualization represent the 1st quartile, the median AKA the 2nd quartile, and the 3rd quartile.
- The height of each box, i.e. the value of the 3rd quartile minus the value of the 1st quartile, is the *interquartile range*. It is a measure of spread of the middle 50% of values, with longer boxes indicating more variability.
- The “whisker” portions of these plots extend out from the bottoms and tops of the boxes and represent points less than the 25th percentile and greater than the 75th percentiles respectively. They’re set to extend out no more than $1.5 \times IQR$ units away from either end of the boxes. We say “no

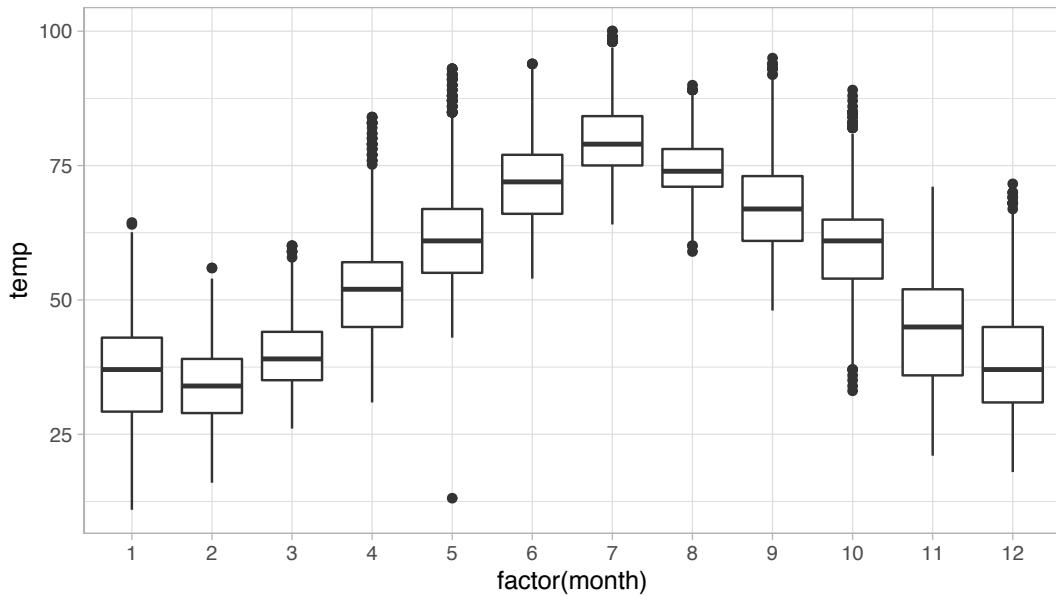


FIGURE 3.23: Temperature by month side-by-side boxplot.

more than” because the ends of the whiskers have to correspond to observed temperatures. The length of these whiskers show how the data outside the middle 50% of values vary, with longer whiskers indicating more variability.

- The dots representing values falling outside the whiskers are called *outliers*. These can be thought of as anomalous values.

It is important to keep in mind that the definition of an outlier is somewhat arbitrary and not absolute. In this case, they are defined by the length of the whiskers, which are no more than $1.5 \times IQR$ units long. Looking at this plot we can see, as expected, that summer months (6 through 8) have higher median temperatures as evidenced by the higher solid lines in the middle of the boxes. We can easily compare temperatures across months by drawing imaginary horizontal lines across the plot. Furthermore, the height of the 12 boxes as quantified by the interquartile ranges are informative too; they tell us about variability, or spread, of temperatures recorded in a given month.

Learning check

(LC3.22) What does the dot at the bottom of the plot for May correspond to? Explain what might have occurred in May to produce this point.

(LC3.23) Which months have the highest variability in temperature? What reasons can you give for this?

(LC3.24) We looked at the distribution of the numerical variable `temp` split by the numerical variable `month` that we converted to a categorical variable using the `factor()` function. Why would a boxplot of `temp` split by the numerical variable `pressure` similarly converted to a categorical variable using the `factor()` not be informative?

(LC3.25) Boxplots provide a simple way to identify outliers. Why may outliers be easier to identify when looking at a boxplot instead of a faceted histogram?

3.7.2 Summary

Side-by-side boxplots provide us with a way to compare and contrast the distribution of a quantitative variable across multiple levels of another categorical variable. One can see where the median falls across the different groups by looking at the center line in the boxes. To see how spread out the variable is across the different groups, look at both the width of the box and also how far the whiskers stretch out away from the box. Outliers are even more easily identified when looking at a boxplot than when looking at a histogram as they are marked with points.

3.8 5NG#5: Barplots

Both histograms and boxplots are tools to visualize the distribution of numerical variables. Another common task is visualize the distribution of a categorical variable. This is a simpler task, as we are simply counting different categories, also known as *levels*, of a categorical variable. Often the best way to visualize these different counts, also known as *frequencies*, is with a barplot (also known as a barchart).

One complication, however, is how your data is represented: is the categorical variable of interest “pre-counted” or not? For example, run the following code that manually creates two data frames representing a collection of fruit: 3 apples and 2 oranges.

```
fruits <- tibble(
  fruit = c("apple", "apple", "orange", "apple", "orange")
)
fruits_counted <- tibble(
  fruit = c("apple", "orange"),
  number = c(3, 2)
)
```

We see both the `fruits` and `fruits_counted` data frames represent the same collection of fruit. Whereas `fruits` just lists the fruit individually...

```
# A tibble: 5 × 1
  fruit
  <chr>
1 apple
2 apple
3 orange
4 apple
5 orange
```

... `fruits_counted` has a variable `count` which represents pre-counted values of each fruit.

```
# A tibble: 2 × 2
  fruit   number
  <chr>   <dbl>
1 apple     3
2 orange    2
```

Depending on how your categorical data is represented, you'll need to use add a different `geom` layer to your `ggplot()` to create a barplot, as we now explore.

3.8.1 Barplots via `geom_bar` or `geom_col`

Let's generate barplots using these two different representations of the same basket of fruit: 3 apples and 2 oranges. Using the `fruits` data frame where all 5 fruits are listed individually in 5 rows, we map the `fruit` variable to the x-position aesthetic and add a `geom_bar()` layer.

```
ggplot(data = fruits, mapping = aes(x = fruit)) +
  geom_bar()
```

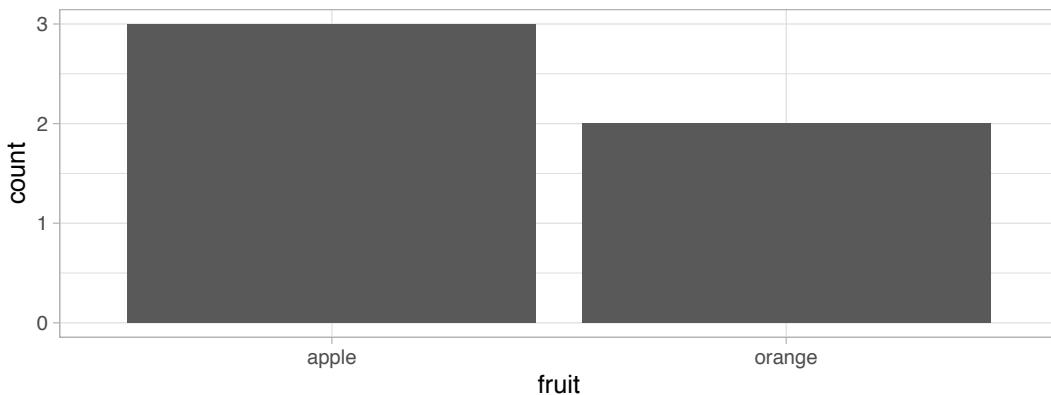


FIGURE 3.24: Barplot when counts are not pre-counted.

However, using the `fruits_counted` data frame where the fruit have been “pre-counted”, we map the `fruit` variable to the x-position aesthetic as with `geom_bar()`, but we also map the `count` variable to the y-position aesthetic, and add a `geom_col()` layer.

```
ggplot(data = fruits_counted, mapping = aes(x = fruit, y = number)) +
  geom_col()
```

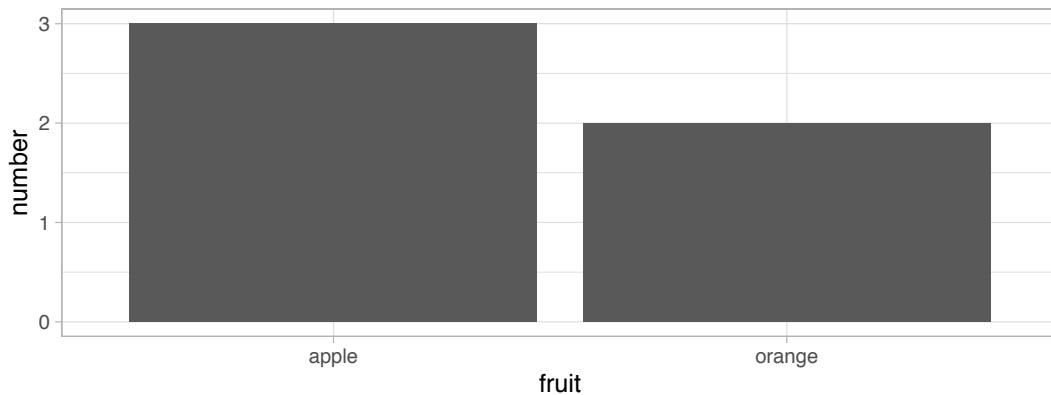


FIGURE 3.25: Barplot when counts are pre-counted.

Compare the barplots in Figures 3.24 and 3.25. They are identical because they reflect count of the same five fruit. However depending on how our data is saved, either pre-counted or not, we must add a different `geom` layer. When the categorical variable whose distribution you want to visualize is:

- Is not pre-counted in your data frame: use `geom_bar()`.
- Is pre-counted in your data frame, use `geom_col()` with the y-position aesthetic mapped to the variable that has the counts.

Let's now go back to the `flights` data frame in the `nycflights13` package and visualize the distribution of the categorical variable `carrier`. In other words, let's visualize the number of domestic flights out of the three New York City airports each airline company flew in 2013. Recall from Section 2.4.3 when you first explored the `flights` data frame you saw that each row corresponds to a flight. In other words the `flights` data frame is more like the `fruits` data frame than the `fruits_counted` data frame above, and thus we should use `geom_bar()` instead of `geom_col()` to create a barplot. Much like a `geom_histogram()`, there is only one variable in the `aes()` aesthetic mapping: the variable `carrier` gets mapped to the `x`-position.

```
ggplot(data = flights, mapping = aes(x = carrier)) +
  geom_bar()
```

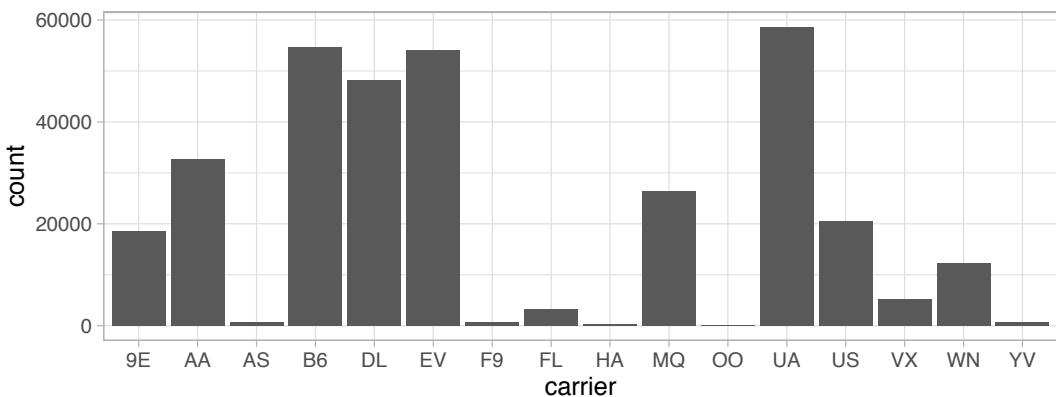


FIGURE 3.26: Number of flights departing NYC in 2013 by airline using `geom_bar()`.

Observe in Figure 3.26 that United Air Lines (UA), JetBlue Airways (B6), and ExpressJet Airlines (EV) had the most flights depart New York City in 2013. If you don't know which airlines correspond to which carrier codes, then run `view(airlines)` to see a directory of airlines. For example: AA is American Airlines; B6 is JetBlue Airways; DL is Delta Airlines; EV is ExpressJet Airlines; MQ is Envoy Air; while UA is United Airlines.

Alternatively, say you had a data frame `flights_counted` where the number of flights for each `carrier` was pre-counted like in Table 3.3.

TABLE 3.3: Number of flights pre-counted for each carrier.

carrier	number
9E	18460
AA	32729

AS	714
B6	54635
DL	48110
EV	54173
F9	685
FL	3260
HA	342
MQ	26397
OO	32
UA	58665
US	20536
VX	5162
WN	12275
YV	601

In order to create a barplot visualizing the distribution of the categorical variable `carrier` in this case, we would use `geom_col()` instead with `x` mapped to `carrier` and `y` mapped to `number` as seen below. The resulting barplot would be identical to Figure 3.26.

```
ggplot(data = flights_table, mapping = aes(x = carrier, y = number)) +  
  geom_col()
```

Learning check

(LC3.26) Why are histograms inappropriate for visualizing categorical variables?

(LC3.27) What is the difference between histograms and barplots?

(LC3.28) How many Envoy Air flights departed NYC in 2013?

(LC3.29) What was the seventh highest airline in terms of departed flights from NYC in 2013? How could we better present the table to get this answer quickly?

3.8.2 Must avoid pie charts!

Unfortunately, one of the most common plots seen today for categorical data is the pie chart. While they may seem harmless enough, they actually present a problem in that humans are unable to judge angles well. As Naomi Robbins describes in her book “Creating More Effective Graphs” (Robbins, 2013), we overestimate angles greater than 90 degrees and we underestimate angles less than 90 degrees. In other words, it is difficult for us to determine relative size of one piece of the pie compared to another.

Let’s examine the same data used in our previous barplot of the number of flights departing NYC by airline in Figure 3.26, but this time we will use a pie chart in Figure 3.27.

Try to answer the following questions:

- How much larger the portion of the pie is for ExpressJet Airlines (`EV`) compared to US Airways (`US`),
- What the third largest carrier is in terms of departing flights, and
- How many carriers have fewer flights than United Airlines (`UA`)?

While it is quite difficult to answer these questions when looking at the pie chart in Figure 3.27, we can much more easily answer these questions using the barchart in Figure 3.26. This is true since barplots present the information in a way such that comparisons between categories can be made with single horizontal lines, whereas pie charts present the information in a way such that comparisons between categories must be made by comparing angles.

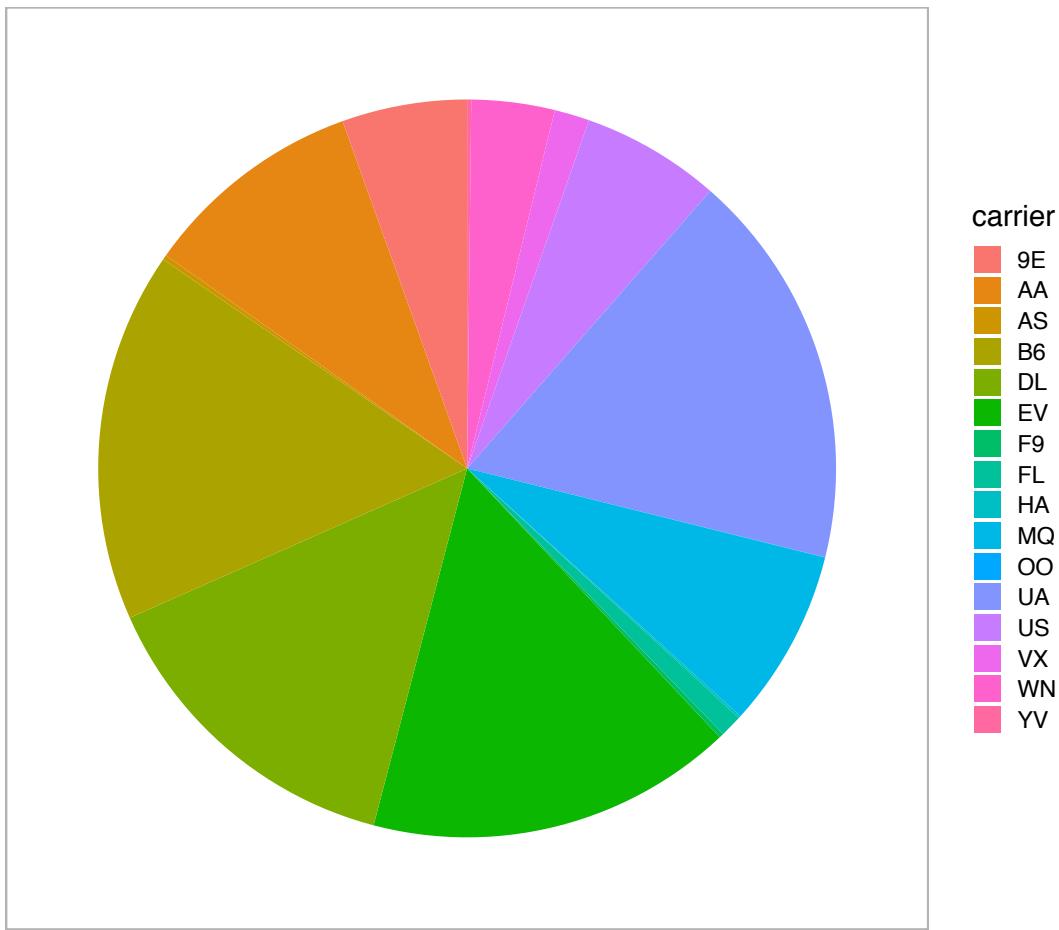


FIGURE 3.27: The dreaded pie chart.

Learning check

(LC3.30) Why should pie charts be avoided and replaced by barplots?

(LC3.31) Why do you think people continue to use pie charts?

3.8.3 Two categorical variables

Barplots are the go-to way to visualize the frequency of different categories, or levels, of a single categorical variable. Another use of barplots is to visualize the *joint* distribution of two categorical variables at the same time. Let's examine

the *joint* distribution of outgoing domestic flights from NYC by `carrier` and `origin`, or in other words the number of flights for each `carrier` and `origin` combination. For example, the number of WestJet flights from `JFK`, the number of WestJet flights from `LGA`, the number of WestJet flights from `EWR`, the number of American Airlines flights from `JFK`, and so on. Recall the `ggplot()` code that created the barplot of `carrier` frequency in Figure 3.26:

```
ggplot(data = flights, mapping = aes(x = carrier)) +
  geom_bar()
```

We can now map the additional variable `origin` by adding a `fill = origin` inside the `aes()` aesthetic mapping; the `fill` aesthetic of any bar corresponds to the color used to fill the bars.

```
ggplot(data = flights, mapping = aes(x = carrier, fill = origin)) +
  geom_bar()
```

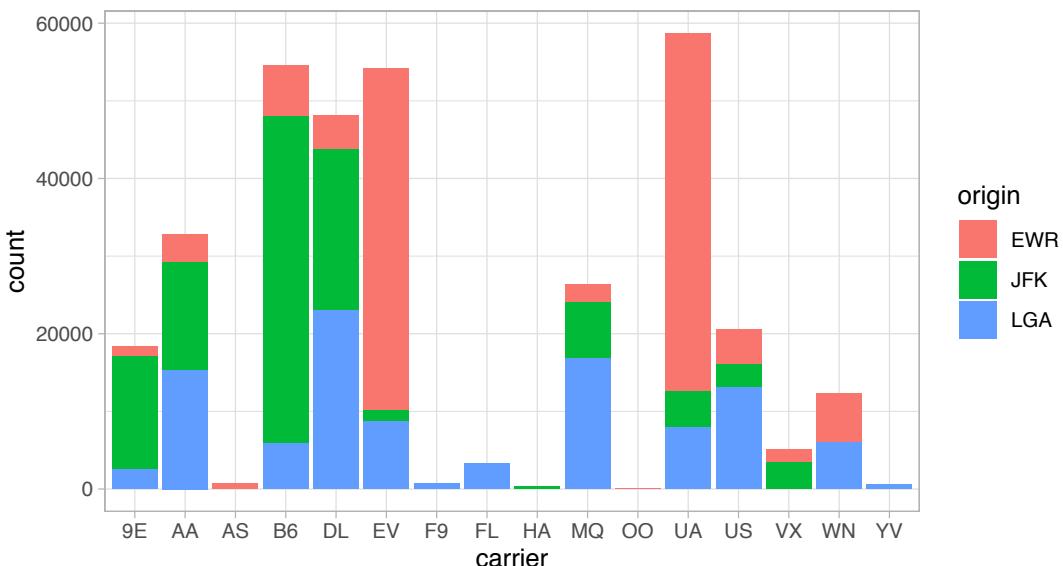


FIGURE 3.28: Stacked barplot comparing the number of flights by carrier and origin.

Figure 3.28 is an example of a *stacked barplot*. While simple to make, in certain aspects it is not ideal. For example, it is difficult to compare the heights of the different colors between the bars, corresponding to comparing the number of flights from each `origin` airport between the carriers.

Before we continue, let's address some common points of confusion amongst

new R users. First, note that `fill` is another aesthetic mapping much like `x-position`; thus it must be included within the parentheses of the `aes()` mapping. The following code, where the `fill` aesthetic is specified outside the `aes()` mapping will yield an error. This is a fairly common error that new `ggplot` users make:

```
ggplot(data = flights, mapping = aes(x = carrier), fill = origin) +
  geom_bar()
```

Second, the `fill` aesthetic corresponds to the color used to fill the bars, while the `color` aesthetic corresponds to the color of the outline of the bars. Observe in Figure 3.29 that mapping `origin` to `color` and not `fill` yields grey bars with different colored outlines.

```
ggplot(data = flights, mapping = aes(x = carrier, color = origin)) +
  geom_bar()
```

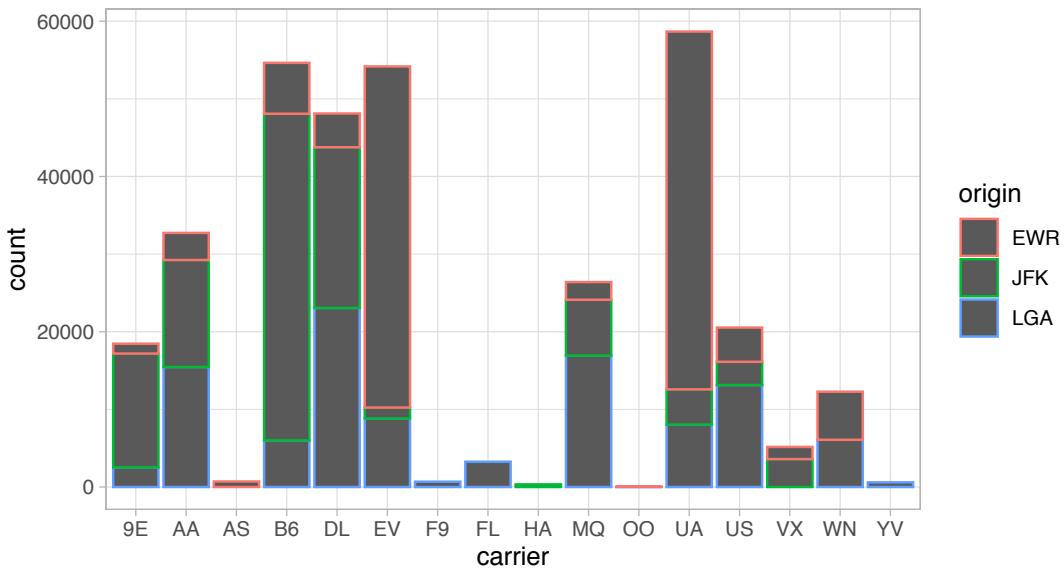


FIGURE 3.29: Stacked barplot with color aesthetic used instead of fill.

Learning check

(LC3.32) What kinds of questions are not easily answered by looking at the above figure?

(LC3.33) What can you say, if anything, about the relationship between

airline and airport in NYC in 2013 in regards to the number of departing flights?

Another alternative to stacked barplots are *side-by-side barplots*, also known as a *dodged barplot*. The code to created a side-by-side barplot is identical to the code to create a stacked barplot, but with a `position = "dodge"` argument added to `geom_bar()`. In other words, we are overriding the default barplot type, which is a stacked barplot, and specifying it to be a side-by-side barplot.

```
ggplot(data = flights, mapping = aes(x = carrier, fill = origin)) +
  geom_bar(position = "dodge")
```

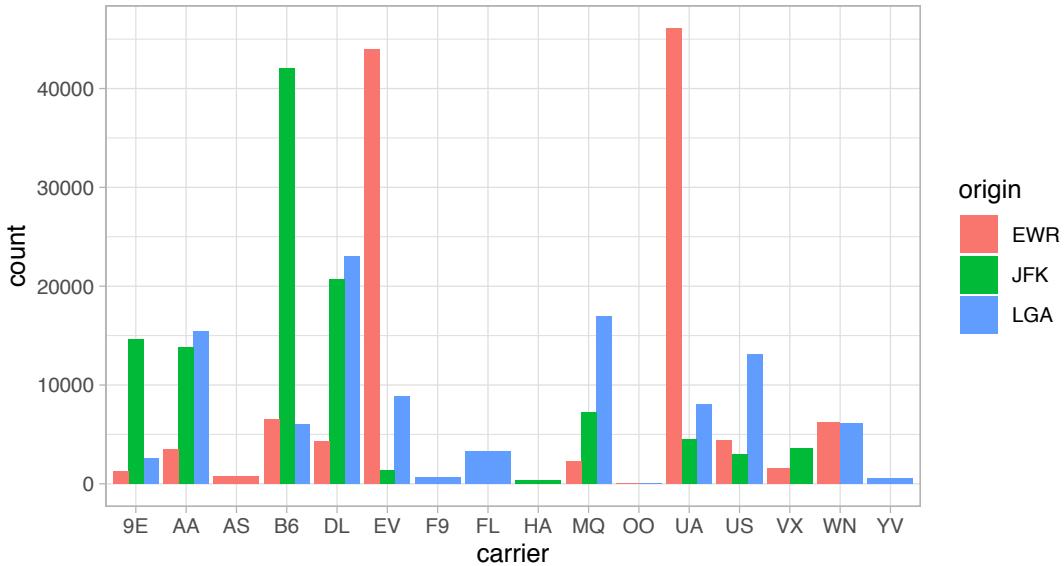


FIGURE 3.30: Side-by-side (AKA dodged) barplot comparing the number of flights by carrier and origin.

Learning check

(LC3.34) Why might the side-by-side (AKA dodged) barplot be preferable to a stacked barplot in this case?

(LC3.35) What are the disadvantages of using a side-by-side (AKA dodged) barplot, in general?

Lastly, another type of barplot is a *faceted barplot*. Recall in Section 3.6 we visualized the distribution of hourly temperatures at the 3 NYC airports *split* by month using facets. We apply the same principle to our barplot visualizing the frequency of `carrier` split by `origin`: instead of mapping `origin`

```
ggplot(data = flights, mapping = aes(x = carrier)) +  
  geom_bar() +  
  facet_wrap(~ origin, ncol = 1)
```

Learning check

(LC3.36) Why is the faceted barplot preferred to the side-by-side and stacked barplots in this case?

(LC3.37) What information about the different carriers at different airports is more easily seen in the faceted barplot?

3.8.4 Summary

Barplots are the preferred way of displaying the distribution of a categorical variable, or in other words the frequency with which the different categories called *levels* occur. They are easy to understand and make it easy to make comparisons across levels. When trying to visualize two categorical variables, you have many options: stacked barplots, side-by-side barplots, and faceted barplots. Depending on what aspect of the joint distribution you are trying to emphasize, you will need to make a choice between these three types of barplots.

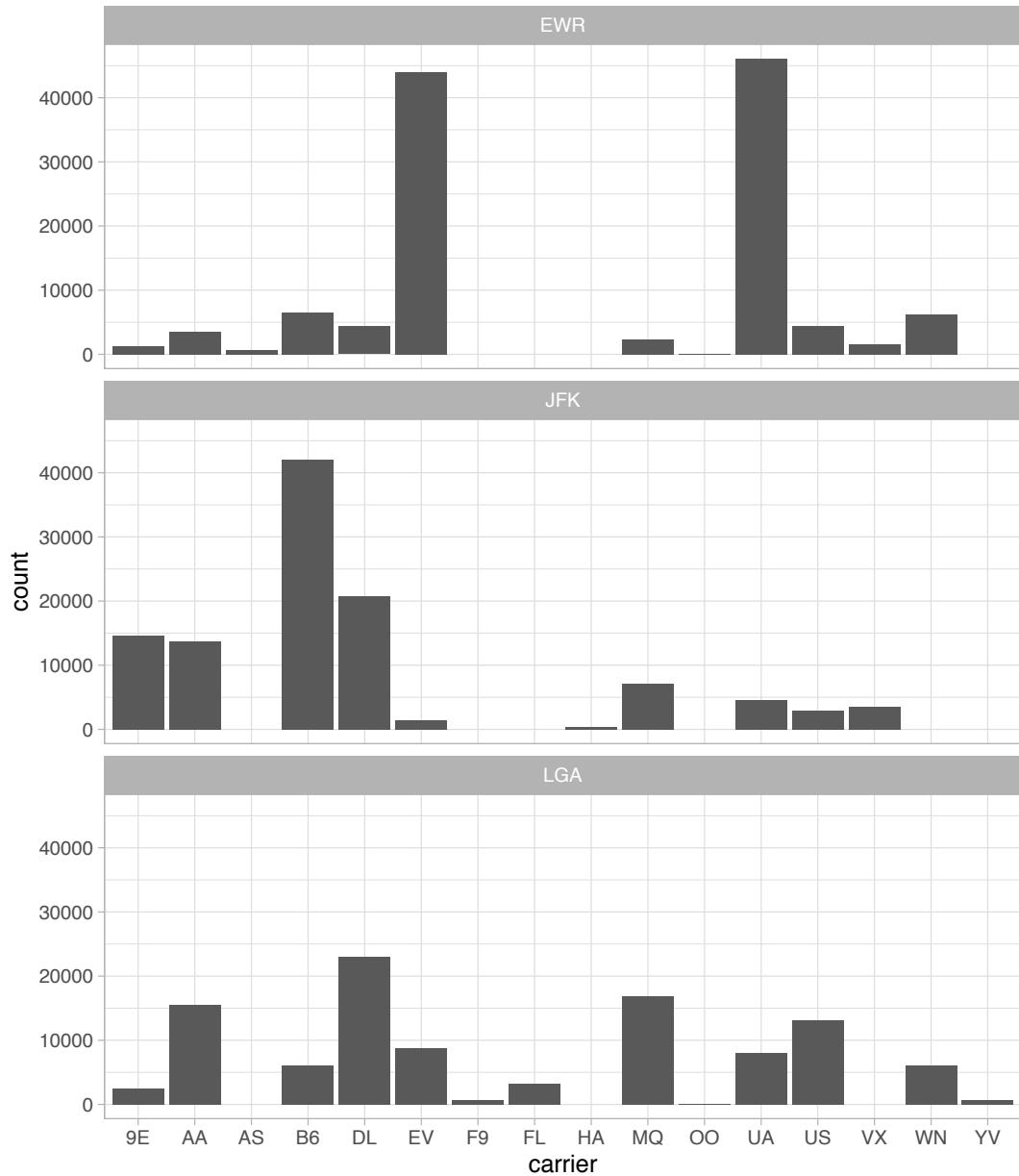


FIGURE 3.31: Faceted barplot comparing the number of flights by carrier and origin.

3.9 Conclusion

3.9.1 Summary table

Let's recap all five of the Five Named Graphs (5NG) in Table 3.4 summarizing their differences. Using these 5NG, you'll be able to visualize the distributions and relationships of variables contained in a wide array of datasets. This will be even more the case as we start to map more variables to more of each geometric object's aesthetic attribute options, further unlocking the awesome power of the `ggplot2` package.

TABLE 3.4: Summary of 5NG

X1	Named graph	Shows	Geometric object	Notes
1	Scatterplot	Relationship between 2 numerical variables	'geom_point()'	
2	Linegraph	Relationship between 2 numerical variables	'geom_line()'	Used when there is a sequential order to x-variable e.g. time
3	Histogram	Distribution of 1 numerical variable	'geom_histogram()'	Facetted histograms show the distribution of 1 numerical variable split by values of another variable
4	Boxplot	Distribution of 1 numerical variable split by 1 categorical variable	'geom_boxplot()'	
5	Barplot	Distribution of 1 categorical variable	'geom_bar()' when counts are not pre-counted, 'geom_col()' when counts are pre-counted	Stacked, side-by-side, and faceted barplots show the *joint* distribution of 2 categorical variables

3.9.2 Argument specification

Run the following two segments of code. First this:

```
ggplot(data = flights, mapping = aes(x = carrier)) +  
  geom_bar()
```

then this:

```
ggplot(flights, aes(x = carrier)) +  
  geom_bar()
```

You'll notice that both code segments create the same barplot, even though in the second segment we omitted the `data =` and `mapping =` code argument names. This is because the `ggplot()` by default assumes that the `data` argument comes first and the `mapping` argument comes second. So as long as you specify the data frame in question first and the `aes()` mapping second, you can omit the explicit statement of the argument names `data =` and `mapping =`.

Going forward for the rest of this book, all `ggplot()` will be like the second segment above: with the `data =` and `mapping =` explicit naming of the argument omitted and the default ordering of arguments respected.

3.9.3 Additional resources

An R script file of all R code used in this chapter is available here⁴.

If you want to further unlock the power of the `ggplot2` package for data visualization, we suggest you that you check out RStudio's "Data Visualization with ggplot2" cheatsheet. This cheatsheet summarizes much more than what we've discussed in this chapter, in particular the many more than the 5 `geom` geometric objects we covered in this Chapter, while providing quick and easy to read visual descriptions.

You can access this cheatsheet by going to the RStudio Menu Bar -> Help -> Cheatsheets -> "Data Visualization with ggplot2":

⁴[scripts/03-visualization.R](#)

Data Visualization with ggplot2 :: CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data set**, a **coordinate system**, and **geoms**—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like `size`, `color`, and `x` and `y` locations.

Complete the template below to build a graph.

```
ggplot(data = <DATA>) +
  geom<function>(mapping = aes<variables>,
  stat = <STAT>, position = <POSITION>)
  +>
  <COORDINATE FUNCTION>
  +>
  <FACET FUNCTION>
  +>
  <SCALE FUNCTION>
  +>
  <THEME FUNCTION>
```

Required: `geom`.
Not required, sensible defaults: `stat`, `position`.

ggplot(data = mpg, aes(y = cyl, x = weight)) +
 begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings **data** **geom**

ggplot(x = cyl, y = mpg, geom = "point")
 Creates a complete plot with given data, geom, and mappings. Supersedes other defaults.
 last_plot() Returns the last plot.

ggplot("plot.png", width = 5, height = 5) Saves last plot as 5x5 file named "plot.png" in working directory.
 Matches file type to file extension.

Geoms Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

- a <- ggplot(economics, aes(date, unemploy))
- b <- ggplot(mpg, aes(cty, hwy))
- a + geom_blank()
- ([no data, no coordinate system])
- b + geom_curve(aes(yend = lat + 1, xend = long + 1), curvature=0.2, xend, yend, alpha, angle, color, curvature, linetype, size)
- a + geom_rect(xmin=long - 1, xmax=long + 1, ymin=lat - 1, ymax=lat + 1, alpha, color, fill, group, linetype, size)
- b + geom_polygon(aes(group = group), x, y, alpha, color, fill, group, linetype, size)
- a + geom_ribbon(aes(min = unemploy - 900, max = unemploy + 900), x, y, alpha, color, fill, group, linetype, size)
- b + geom_point(), x, y, alpha, color, fill, shape, stroke
- e + geom_quartile(), x, y, alpha, color, group, linetype, size, weight
- e + geom_rug(sides = "bl"), x, y, alpha, color, linetype, size
- e + geom_smooth(method = lm), x, y, alpha, color, fill, group, linetype, size, weight
- e + geom_text(aes(label = cty), nudge_x = 0, nudge_y = 1, check_overlap = TRUE), x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

TWO VARIABLES

continuous x , continuous y

- e + geom_label(aes(label = cyl), nudge_x = 1, nudge_y = 1, check_overlap = TRUE), x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust
- B + geom_bump()
- C + geom_jitter(height = 2, width = 2)
- x, y, alpha, color, fill, shape, size
- e + geom_point(), x, y, alpha, color, fill, shape, size, stroke
- e + geom_quartile(), x, y, alpha, color, group, linetype, size, weight
- e + geom_rug(sides = "bl"), x, y, alpha, color, linetype, size
- e + geom_smooth(method = lm), x, y, alpha, color, fill, group, linetype, size, weight
- e + geom_text(aes(label = cty), nudge_x = 0, nudge_y = 1, check_overlap = TRUE), x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

continuous bivariate distribution

- h <- ggplot(diamonds, aes(carat, price))
- h + geom_bind2(binwidth = c(0.25, 25), x, y, alpha, color, fill, linetype, size, weight)
- h + geom_density2d()
- x, y, alpha, colour, group, linetype, size
- h + geom_hex()
- x, y, alpha, colour, fill, size

continuous function

- i <- ggplot(economics, aes(date, unemploy))
- i + geom_area()
- i + geom_line()
- i + geom_step(direction = "hv")
- i + geom_line()

visualizing error

- df <- data.frame(grp = c("A", "B"), fit = 4.5, se = 1.2)
- j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))
- j + geom_crossbar(aes(seen = 2))
- x, y, ymax, ymin, alpha, color, fill, group, linetype, size
- j + geom_errorbar()
- j + geom_errorbar()
- j + geom_linerange()
- x, ymin, ymax, alpha, color, group, linetype, size
- j + geom_pointrange()
- x, y, ymax, ymin, alpha, color, fill, group, linetype, size

maps

- data <- data.frame(murder = USArrests\$Murder, state = USArrests\$State, state_name = USArrests\$State)
- map <- map_data("state")
- k <- ggplot(data, aes(fill = murder))
- k + geom_map(aes(map_id = state), map = map)
- k + expand_limits(map_id = map\$long, y = map\$lat), map_id, alpha, color, fill, linetype, size
- l + geom_raster(aes(fill = z), x, y, z, alpha, color, group, linetype, size, weight)
- l + geom_contour(aes(z = z))
- x, y, z, alpha, color, group, linetype, size, weight
- l + geom_hex(aes(fill = z))
- x, y, alpha, color, fill, linetype, size, width

THREE VARIABLES

- seals <- with(seals, sort(dates, long^2 + date_lat^2))
- l <- ggplot(seals, aes(long))
- l + geom_hex()
- x, y, z, alpha, color, group, linetype, size, weight
- l + geom_raster(aes(fill = z), x, y, alpha, fill, interpolate = FALSE)
- x, y, alpha, fill
- l + geom_hex(aes(fill = z))
- x, y, alpha, color, fill, linetype, size, width



FIGURE 3.32: Data Visualization with ggplot2 cheatsheat.

3.9.4 What's to come

Recall in Figure 3.2 in Section 3.3 we visualized the relationship between departure delay and arrival delay for Alaska Airlines flights. This necessitated paring down the `flights` data frame to a new data frame `alaska_flights` consisting of only `carrier == AS` flights first:

```
alaska_flights <- flights %>%
  filter(carrier == "AS")

ggplot(data = alaska_flights, mapping = aes(x = dep_delay, y = arr_delay)) +
  geom_point()
```

Furthermore recall in Figure 3.8 in Section 3.4 we visualized hourly temperature recordings at Newark airport only for the first 15 days of January 2013. This necessitated paring down the `weather` data frame to a new data frame `early_january_weather` consisting of hourly temperature recordings only for `origin == "EWR"`, `month == 1`, and day less than or equal to 15 first:

```
early_january_weather <- weather %>%  
  filter(origin == "EWR" & month == 1 & day <= 15)  
  
ggplot(data = early_january_weather, mapping = aes(x = time_hour, y = temp)) +  
  geom_line()
```

These two code segments were a preview of Chapter 4 on data wrangling where we'll delve further into the `dplyr` package. Data wrangling is the process of transforming and modifying existing data with the intent of making it more appropriate for analysis purposes. For example, the two code segments used the `filter()` function to create new data frames (`alaska_flights` and `early_january_weather`) by choosing only a subset of rows of existing data frames (`flights` and `weather`). In this next chapter, we'll formally introduce the `filter()` and other data wrangling functions as well as the *pipe operator* `%>%` which allows you to combine multiple data wrangling actions into a single sequential *chain* of actions. On to Chapter 4 on data wrangling!

4

Data Wrangling

So far in our journey, we've seen how to look at data saved in data frames using the `glimpse()` and `view()` functions in Chapter 2 on and how to create data visualizations using the `ggplot2` package in Chapter 3. In particular we studied what we term the “five named graphs” (5NG):

1. scatterplots via `geom_point()`
2. linegraphs via `geom_line()`
3. boxplots via `geom_boxplot()`
4. histograms via `geom_histogram()`
5. barplots via `geom_bar()` or `geom_col()`

We created these visualizations using the “Grammar of Graphics”, which maps variables in a data frame to the aesthetic attributes of one the above 5 geometric objects. We can also control other aesthetic attributes of the geometric objects such as the size and color as seen in the Gapminder data example in Figure 3.1.

Recall however in Section 3.9.4 we discussed that for two of our visualizations we needed transformed/modified versions of existing data frames. Recall for example the scatterplot of departure and arrival delay *only* for Alaska Airlines flights. In order to create this visualization, we needed to first pare down the `flights` data frame to a new data frame `alaska_flights` consisting of only `carrier == "AS"` flights using the `filter()` function.

```
alaska_flights <- flights %>%
  filter(carrier == "AS")

ggplot(data = alaska_flights, mapping = aes(x = dep_delay, y = arr_delay)) +
  geom_point()
```

In this chapter, we'll introduce a series of functions from the `dplyr` package that will allow you to take a data frame and

1. `filter()` its existing rows to only pick out a subset of them. For example, the `alaska_flights` data frame above.
2. `summarize()` one of its columns/variables with a *summary statistic*. Examples include the median and interquartile range of temperatures as we saw in Section 3.7 on boxplots.
3. `group_by()` its rows. In other words assign different rows to be part of the same *group* and report summary statistics for each group separately. For example, say perhaps you don't want a single overall average departure delay `dep_delay` for all three `origin` airports combined, but rather three separate average departure delays, one for each of the three `origin` airports.
4. `mutate()` its existing columns/variables to create new ones. For example, convert hourly temperature recordings from °F to °C.
5. `arrange()` its rows. For example, sort the rows of `weather` in ascending or descending order of `temp`.
6. `join()` it with another data frame by matching along a “key” variable. In other words, merge these two data frames together.

Notice how we used `computer code` font to describe the actions we want to take on our data frames. This is because the `dplyr` package for data wrangling that we'll introduce in this chapter has intuitively verb-named functions that are easy to remember.

We'll start by introducing the pipe operator `%>%`, which allows you to combine multiple data wrangling verb-named functions into a single sequential *chain* of actions.

Needed packages

Let's load all the packages needed for this chapter (this assumes you've already installed them). If needed, read Section 2.3 for information on how to install and load R packages.

```
library(dplyr)
library(ggplot2)
library(nycflights13)
```

4.1 The pipe operator: %>%

Before we start data wrangling, let's first introduce a very nifty tool that gets loaded along with the `dplyr` package: the pipe operator `%>%`. Say you would like to perform a hypothetical sequence of operations on a hypothetical data frame `x` using hypothetical functions `f()`, `g()`, and `h()`:

1. Take `x` *then*
2. Use `x` as an input to a function `f()` *then*
3. Use the output of `f(x)` as an input to a function `g()` *then*
4. Use the output of `g(f(x))` as an input to a function `h()`

One way to achieve this sequence of operations is by using nesting parentheses as follows:

```
h(g(f(x)))
```

The above code isn't so hard to read since we are applying only three functions: `f()`, then `g()`, then `h()`. However, you can imagine that this can get progressively harder and harder to read as the number of functions applied in your sequence increases. This is where the pipe operator `%>%` comes in handy. `%>%` takes one output of one function and then "pipes" it to be the input of the next function. Furthermore, a helpful trick is to read `%>%` as "then." For example, you can obtain the same output as the above sequence of operations as follows:

```
x %>%  
  f() %>%  
  g() %>%  
  h()
```

You would read this above sequence as:

1. Take `x` *then*
2. Use this output as the input to the next function `f()` *then*
3. Use this output as the input to the next function `g()` *then*
4. Use this output as the input to the next function `h()`

So while both approaches above would achieve the same goal, the latter is much more human-readable because you can read the sequence of operations

line-by-line. But what are the hypothetical `x`, `f()`, `g()`, and `h()`? Throughout this chapter on data wrangling:

- The starting value `x` will be a data frame. For example: `flights`.
- The sequence of functions, here `f()`, `g()`, and `h()`, will be a sequence of any number of the six data wrangling verb-named functions we listed in the introduction to this chapter. For example: `filter(carrier == "AS")`.
- The result will be the transformed/modified data frame that you want. For example: a data frame consisting of only the subset of rows in `flights` corresponding to Alaska Airlines flights.

Much like when adding layers to a `ggplot()` using the `+` sign at the end of lines, you form a single *chain* of data wrangling operations by combining verb-named functions into a single sequence with pipe operators `%>%` at the end of lines. So continuing our example involving Alaska Airlines flights, we form a chain using the pipe operator `%>%` and save the resulting data frame in `alaska_flights`:

```
alaska_flights <- flights %>%
  filter(carrier == "AS")
```

Keep in mind, there are many more advanced data wrangling functions than just the six listed in the introduction to this chapter; you'll see some examples of these near in Section 4.8. However, just with these six verb-named functions you'll be able to perform a broad array of data wrangling tasks for the rest of this book.

4.2 `filter` rows

Subset Observations (Rows)



FIGURE 4.1: Diagram of `filter()` rows operation.

The `filter()` function here works much like the “Filter” option in Microsoft Excel; it allows you to specify criteria about the values of a variables in your

dataset and then filters out only those rows that match that criteria. We begin by focusing only on flights from New York City to Portland, Oregon. The `dest` code (or airport code) for Portland, Oregon is "`PDX`". Run the following and look at the resulting “spreadsheet”-like view to ensure that only flights heading to Portland are chosen here:

```
portland_flights <- flights %>%
  filter(dest == "PDX")
View(portland_flights)
```

Note the following:

- The ordering of the commands:
 - Take the `flights` data frame `flights` *then*
 - `filter` the data frame so that only those where the `dest` equals "`PDX`" are included.
- We test for equality using the double equal sign `==` and not a single equal sign `=`. In other words `filter(dest = "PDX")` will yield an error. This is a convention across many programming languages. If you are new to coding, you'll probably forget to use the double equal sign `==` a few times before you get the hang of it.

You can use other mathematical operations beyond just `==` to form criteria:

- `>` corresponds to “greater than”
- `<` corresponds to “less than”
- `>=` corresponds to “greater than or equal to”
- `<=` corresponds to “less than or equal to”
- `!=` corresponds to “not equal to”. The `!` is used in many programming languages to indicate “not”.

Furthermore, you can combine multiple criteria together using operators that make comparisons:

- `|` corresponds to “or”
- `&` corresponds to “and”

To see many of these in action, let's filter `flights` for all rows that:

- Departed from JFK airport and
- Were heading to Burlington, Vermont ("`BTV`") or Seattle, Washington ("`SEA`") and
- Departed in the months of October, November, or December.

Run the following:

```
btv_sea_flights_fall <- flights %>%
  filter(origin == "JFK" & (dest == "BTV" | dest == "SEA") & month >= 10)
View(btv_sea_flights_fall)
```

Note that even though colloquially speaking one might say “all flights leaving Burlington, Vermont *and* Seattle, Washington,” in terms of computer operations, we really mean “all flights leaving Burlington, Vermont *or* leaving Seattle, Washington.” For a given row in the data, `dest` can be “BTV”, “SEA”, or something else, but not “BTV” and “SEA” at the same time. Furthermore, note the careful use of parentheses around the `dest == "BTV" | dest == "SEA"`.

We can often skip the use of `&` and just separate our conditions with a comma. In other words the code above will return the identical output `btv_sea_flights_fall` as this code below:

```
btv_sea_flights_fall <- flights %>%
  filter(origin == "JFK", (dest == "BTV" | dest == "SEA"), month >= 10)
View(btv_sea_flights_fall)
```

Let’s present another example that uses the `!` “not” operator to pick rows that *don’t* match a criteria. As mentioned earlier, the `!` can be read as “not.” Here we are filtering rows corresponding to flights that didn’t go to Burlington, VT or Seattle, WA.

```
not_BTV_SEA <- flights %>%
  filter(!(dest == "BTV" | dest == "SEA"))
View(not_BTV_SEA)
```

Again, note the careful use of parentheses around the `(dest == "BTV" | dest == "SEA")`. If we didn’t use parentheses as follows:

```
flights %>%
  filter(!dest == "BTV" | dest == "SEA")
```

We would be returning all flights not headed to “BTV” *or* those headed to “SEA”, which is an entirely different resulting data frame.

Now say we have a large list of airports we want to filter for, say `BTV`, `SEA`, `PDX`, `SFO`, and `BDL`. We could continue to use the `|` (“or”) operator as so:

```
many_airports <- flights %>%
  filter(dest == "BTV" | dest == "SEA" | dest == "PDX" |
        dest == "SFO" | dest == "BDL")
View(many_airports)
```

but as we progressively include more airports, this will get unwieldy. A slightly shorter approach uses the `%in%` operator:

```
many_airports <- flights %>%
  filter(dest %in% c("BTV", "SEA", "PDX", "SFO", "BDL"))
View(many_airports)
```

What this code is doing is filtering `flights` for all flights where `dest` is in the list of airports `c("BTV", "SEA", "PDX", "SFO", "BDL")`. Recall from Chapter 2 that the `c()` function “combines” or “concatenates” values in a vector of values. Both outputs of `many_airports` are the same, but as you can see the latter takes much less time to code.

As a final note we point out that `filter()` should often be among the first verbs you apply to your data. This cleans your dataset to only those rows you care about, or put differently, it narrows down the scope of your data frame to just the observations your care about.

Learning check

(LC4.1) What’s another way of using the “not” operator `!` to filter only the rows that are not going to Burlington VT nor Seattle WA in the `flights` data frame? Test this out using the code above.

4.3 summarize variables

The next common task when working with data is to return *summary statistics*: a single numerical value that summarizes a large number of values, for example the mean/average or the median. Other examples of summary statistics that might not immediately come to mind include the sum, the smallest value

AKA the minimum, the largest value AKA the maximum, and the standard deviation; they are all summaries of a large number of values.



FIGURE 4.2: Diagram of summarize() rows.



FIGURE 4.3: Diagram of concept of a summary function.

Let's calculate the mean and the standard deviation of the temperature variable `temp` in the `weather` data frame included in the `nycflights13` package (See Appendix A). We'll do this in one step using the `summarize()` function from the `dplyr` package and save the results in a new data frame `summary_temp` with columns/variables `mean` and the `std_dev`. Note you can also use the UK spelling of "summarise" using the `summarise()` function.

As shown in Figures 4.2 and 4.3, the `weather` data frame's many rows will be collapsed into a single row of just the summary values, in this case the mean and standard deviation:

```
summary_temp <- weather %>%
  summarize(mean = mean(temp), std_dev = sd(temp))
summary_temp
```

```
# A tibble: 1 × 2
  mean std_dev
  <dbl>   <dbl>
1     NA      NA
```

Why are the values returned `NA`? As we saw in Section 3.3.1 when creating the scatterplot of departure and arrival delays for `alaska_flights`, `NA` is how R encodes *missing values* where `NA` indicates “not available” or “not applicable.” If a value for a particular row and a particular column does not exist, `NA` is stored instead. Values can be missing for many reasons. Perhaps the data was collected but someone forgot to enter it? Perhaps the data was not collected at all because it was too difficult? Perhaps there was an erroneous value that someone entered that has been correct to read as missing? You’ll often encounter issues with missing values when working with real data.

Going back to our `summary_temp` output above, by default any time you try to calculate a summary statistic of a variable that has one or more `NA` missing values in R, then `NA` is returned. To work around this fact, you can set the `na.rm` argument to `TRUE`, where `rm` is short for “remove”; this will ignore any `NA` missing values and only return the summary value for all non-missing values.

The code below computes the mean and standard deviation of all non-missing values of `temp`. Notice how the `na.rm=TRUE` are used as arguments to the `mean()` and `sd()` functions individually, and not to the `summarize()` function.

```
summary_temp <- weather %>%
  summarize(mean = mean(temp, na.rm = TRUE),
            std_dev = sd(temp, na.rm = TRUE))
summary_temp
```

```
# A tibble: 1 x 2
  mean std_dev
  <dbl>   <dbl>
1  55.3    17.8
```

However, one needs to be cautious whenever ignoring missing values as we’ve done above. In the upcoming Learning Checks we’ll consider the possible ramifications of blindly sweeping rows with missing values “under the rug.” This is in fact why the `na.rm` argument to any summary statistic function in R has is set to `FALSE` by default; in other words, do not ignore rows with missing values by default. R is alerting you to the presence of missing data and you should be mindful of this missingness and any potential causes of this missingness throughout your analysis.

What are other summary statistic functions can we use inside the `summarize()` verb? As seen in Figure 4.3, you can use any function in R that takes many values and returns just one. Here are just a few:

- `mean()`: the mean AKA the average

- `sd()`: the standard deviation, which is a measure of spread
- `min()` and `max()`: the minimum and maximum values respectively
- `IQR()`: Interquartile range
- `sum()`: the sum
- `n()`: a count of the number of rows/observations in each group. This particular summary function will make more sense when `group_by()` is covered in Section 4.4.

Learning check

(LC4.2) Say a doctor is studying the effect of smoking on lung cancer for a large number of patients who have records measured at five year intervals. She notices that a large number of patients have missing data points because the patient has died, so she chooses to ignore these patients in her analysis. What is wrong with this doctor’s approach?

(LC4.3) Modify the above `summarize` function to create `summary_temp` to also use the `n()` summary function: `summarize(count = n())`. What does the returned value correspond to?

(LC4.4) Why doesn’t the following code work? Run the code line by line instead of all at once, and then look at the data. In other words, run `summary_temp <- weather %>% summarize(mean = mean(temp, na.rm = TRUE))` first.

```
summary_temp <- weather %>%
  summarize(mean = mean(temp, na.rm = TRUE)) %>%
  summarize(std_dev = sd(temp, na.rm = TRUE))
```

4.4 `group_by` rows

Say instead of the a single mean temperature for the whole year, you would like 12 mean temperatures, one for each of the 12 months separately? In other words, we would like to compute the mean temperature split by month AKA sliced by month AKA aggregated by month. We can do this by “grouping” temperature observations by the values of another variable, in this case by the 12 values of the variable `month`. Run the following code:



FIGURE 4.4: Diagram of `group_by()` and `summarize()`.

```
summary_monthly_temp <- weather %>%
  group_by(month) %>%
  summarize(mean = mean(temp, na.rm = TRUE),
            std_dev = sd(temp, na.rm = TRUE))
summary_monthly_temp
```

```
# A tibble: 12 x 3
  month   mean  std_dev
  <dbl> <dbl>    <dbl>
1     1  35.6    10.2
2     2  34.3     6.98
3     3  39.9     6.25
4     4  51.7     8.79
5     5  61.8     9.68
6     6  72.2     7.55
7     7  80.1     7.12
8     8  74.5     5.19
9     9  67.4     8.47
10    10  60.1     8.85
11    11  45.0    10.4
12    12  38.4     9.98
```

This code is identical to the previous code that created `summary_temp`, but with an extra `group_by(month)` added before the `summarize()`. Grouping the `weather` dataset by `month` and then applying the `summarize()` functions yields a data frame that displays the mean and standard deviation temperature split by the 12 months of the year.

It is important to note that the `group_by()` function doesn't change data frames by itself. Rather it changes the *meta-data*, or data about the data, specifically the group structure. It is only after we apply the `summarize()` function that

the data frame changes. For example, let's consider the `diamonds` data frame included in the `ggplot2` package. Run this code, specifically in the console:

```
diamonds
```

```
# A tibble: 53,940 x 10
  carat cut     color clarity depth table price     x     y
  <dbl> <ord>   <ord> <ord>    <dbl> <dbl> <int> <dbl> <dbl>
1 0.23 Ideal    E      SI2      61.5    55    326  3.95  3.98
2 0.21 Premium  E      SI1      59.8    61    326  3.89  3.84
3 0.23 Good     E      VS1      56.9    65    327  4.05  4.07
4 0.290 Premium I      VS2      62.4    58    334  4.2   4.23
5 0.31 Good     J      SI2      63.3    58    335  4.34  4.35
6 0.24 Very     G~ J    VVS2     62.8    57    336  3.94  3.96
7 0.24 Very     G~ I    VVS1     62.3    57    336  3.95  3.98
8 0.26 Very     G~ H    SI1      61.9    55    337  4.07  4.11
9 0.22 Fair     E      VS2      65.1    61    337  3.87  3.78
10 0.23 Very    G~ H    VS1      59.4    61   338   4     4.05
# ... with 53,930 more rows, and 1 more variable: z <dbl>
```

Observe that the first line of the output reads `# A tibble: 53,940 x 10`. This is an example of meta-data, in this case the number of observations/rows and variables/columns in `diamonds`. The actual data itself are the subsequent table of values.

Now let's pipe the `diamonds` data frame into `group_by(cut)`. Run this code, specifically in the console:

```
diamonds %>%
  group_by(cut)
```

```
# A tibble: 53,940 x 10
# Groups:   cut [5]
  carat cut     color clarity depth table price     x     y
  <dbl> <ord>   <ord> <ord>    <dbl> <dbl> <int> <dbl> <dbl>
1 0.23 Ideal    E      SI2      61.5    55    326  3.95  3.98
2 0.21 Premium  E      SI1      59.8    61    326  3.89  3.84
3 0.23 Good     E      VS1      56.9    65    327  4.05  4.07
4 0.290 Premium I      VS2      62.4    58    334  4.2   4.23
5 0.31 Good     J      SI2      63.3    58    335  4.34  4.35
6 0.24 Very     G~ J    VVS2     62.8    57    336  3.94  3.96
7 0.24 Very     G~ I    VVS1     62.3    57    336  3.95  3.98
8 0.26 Very     G~ H    SI1      61.9    55    337  4.07  4.11
```

```

9 0.22 Fair E VS2      65.1    61   337  3.87  3.78
10 0.23 Very G~ H VS1     59.4    61   338   4     4.05
# ... with 53,930 more rows, and 1 more variable: z <dbl>

```

Observe that now there is additional meta-data: # Groups: cut [5] indicating that the grouping structure meta-data has been set based on the 5 possible values AKA levels of the categorical variable cut: "Fair", "Good", "Very Good", "Premium", "Ideal". On the other hand observe that the data has not changed: it is still a table of $53,940 \times 10$ values.

Only by combining a `group_by()` with another data wrangling operation, in this case `summarize()` will the actual data be transformed.

```

diamonds %>%
  group_by(cut) %>%
  summarize(avg_price = mean(price))

```

```

# A tibble: 5 x 2
  cut      avg_price
  <ord>     <dbl>
1 Fair      4359.
2 Good      3929.
3 Very Good 3982.
4 Premium   4584.
5 Ideal     3458.

```

If we would like to remove this group structure meta-data, we can pipe the resulting data frame into the `ungroup()` function. Observe how the # Groups: cut [5] meta-data is no longer present. Run this code, specifically in the console:

```

diamonds %>%
  group_by(cut) %>%
  ungroup()

```

```

# A tibble: 53,940 x 10
  carat cut      color clarity depth table price      x      y
  <dbl> <ord>    <ord> <ord>    <dbl> <dbl> <int> <dbl> <dbl>
1 0.23 Ideal    E      SI2      61.5    55   326  3.95  3.98
2 0.21 Premium  E      SI1      59.8    61   326  3.89  3.84
3 0.23 Good     E      VS1      56.9    65   327  4.05  4.07
4 0.290 Premium I      VS2      62.4    58   334  4.2    4.23
5 0.31 Good     J      SI2      63.3    58   335  4.34  4.35
6 0.24 Very G~ J      VVS2     62.8    57   336  3.94  3.96

```

```

7 0.24 Very G~ I     VVS1      62.3    57    336  3.95  3.98
8 0.26 Very G~ H     SI1       61.9    55    337  4.07  4.11
9 0.22 Fair   E      VS2       65.1    61    337  3.87  3.78
10 0.23 Very G~ H    VS1       59.4    61    338  4      4.05
# ... with 53,930 more rows, and 1 more variable: z <dbl>

```

Let's now revisit the `n()` counting summary function we introduced in the previous section. For example, suppose we'd like to count how many flights departed each of the three airports in New York City:

```

by_origin <- flights %>%
  group_by(origin) %>%
  summarize(count = n())
by_origin

```

```

# A tibble: 3 × 2
  origin  count
  <chr>   <int>
1 EWR     120835
2 JFK     111279
3 LGA     104662

```

We see that Newark ("EWR") had the most flights departing in 2013 followed by "JFK" and lastly by LaGuardia ("LGA"). Note there is a subtle but important difference between `sum()` and `n()`; While `sum()` returns the sum of a numerical variable, `n()` returns counts of the the number of rows/observations.

4.4.1 Grouping by more than one variable

You are not limited to grouping by one variable! Say you wanted to know the number of flights leaving each of the three New York City airports *for each month*, we can also group by a second variable `month`: `group_by(origin, month)`. We see there are 36 rows to `by_origin_monthly` because there are 12 months for 3 airports (EWR, JFK, and LGA).

```

by_origin_monthly <- flights %>%
  group_by(origin, month) %>%
  summarize(count = n())
by_origin_monthly

```

```

# A tibble: 36 × 3
# Groups:   origin [3]
  origin  month  count
  <chr>   <dbl> <int>
1 EWR     1       120835
2 EWR     2       111279
3 EWR     3       104662
4 EWR     4       104662
5 EWR     5       104662
6 EWR     6       104662
7 EWR     7       104662
8 EWR     8       104662
9 EWR     9       104662
10 EWR    10      104662
11 EWR    11      104662
12 EWR    12      104662
13 JFK     1       111279
14 JFK     2       111279
15 JFK     3       104662
16 JFK     4       104662
17 JFK     5       104662
18 JFK     6       104662
19 JFK     7       104662
20 JFK     8       104662
21 JFK     9       104662
22 JFK    10      104662
23 JFK    11      104662
24 JFK    12      104662
25 LGA     1       104662
26 LGA     2       104662
27 LGA     3       104662
28 LGA     4       104662
29 LGA     5       104662
30 LGA     6       104662
31 LGA     7       104662
32 LGA     8       104662
33 LGA     9       104662
34 LGA    10      104662
35 LGA    11      104662
36 LGA    12      104662

```

```
origin month count
<chr> <int> <int>
1 EWR      1  9893
2 EWR      2  9107
3 EWR      3 10420
4 EWR      4 10531
5 EWR      5 10592
6 EWR      6 10175
7 EWR      7 10475
8 EWR      8 10359
9 EWR      9  9550
10 EWR     10 10104
# ... with 26 more rows
```

Why do we `group_by(origin, month)` and not `group_by(origin)` and then `group_by(month)`? Let's investigate:

```
by_origin_monthly_incorrect <- flights %>%
  group_by(origin) %>%
  group_by(month) %>%
  summarize(count = n())
by_origin_monthly_incorrect
```

```
# A tibble: 12 x 2
  month count
  <int> <int>
1     1 27004
2     2 24951
3     3 28834
4     4 28330
5     5 28796
6     6 28243
7     7 29425
8     8 29327
9     9 27574
10    10 28889
11    11 27268
12    12 28135
```

What happened here is that the second `group_by(month)` overrode the group structure meta-data of the first `group_by(origin)`, so that in the end we are only grouping by `month`. The lesson here is if you want to `group_by()` two or

more variables, you should include all these variables in a single `group_by()` function call.

Learning check

(LC4.5) Recall from Chapter 3 when we looked at plots of temperatures by months in NYC. What does the standard deviation column in the `summary_monthly_temp` data frame tell us about temperatures in New York City throughout the year?

(LC4.6) What code would be required to get the mean and standard deviation temperature for each day in 2013 for NYC?

(LC4.7) Recreate `by_monthly_origin`, but instead of grouping via `group_by(origin, month)`, group variables in a different order `group_by(month, origin)`. What differs in the resulting dataset?

(LC4.8) How could we identify how many flights left each of the three airports for each `carrier`?

(LC4.9) How does the `filter()` operation differ from a `group_by()` followed by a `summarize()`?

4.5 `mutate` existing variables

Make New Variables

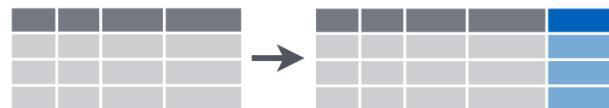


FIGURE 4.5: Diagram of `mutate()` columns.

Another common transformation of data is to create/compute new variables based on existing ones. For example, say you are more comfortable thinking of

temperature in degrees Celsius °C and not degrees Fahrenheit °F. The formula to convert temperatures from °F to °C is:

$$\text{temp in C} = \frac{\text{temp in F} - 32}{1.8}$$

We can apply this formula to the `temp` variable using the `mutate()` function, which takes existing variables and mutates them to create new ones.

```
weather <- weather %>%
  mutate(temp_in_C = (temp - 32) / 1.8)
View(weather)
```

Note that we have overwritten the original `weather` data frame with a new version that now includes the additional variable `temp_in_C`. In other words, the `mutate()` command outputs a new data frame which then gets saved over the original `weather` data frame. Furthermore, note how in `mutate()` we used `temp_in_C = (temp - 32) / 1.8` to create a new variable `temp_in_C`.

Why did we overwrite the data frame `weather` instead of assigning the result to a new data frame like `weather_new`, but on the other hand why did we *not* overwrite `temp`, but instead created a new variable called `temp_in_C`? As a rough rule of thumb, as long as you are not losing original information that you might need later, it's acceptable practice to overwrite existing data frames. On the other hand, had we used `mutate(temp = (temp - 32) / 1.8)` instead of `mutate(temp_in_C = (temp - 32) / 1.8)`, we would have overwritten the original variable `temp` and lost its values.

Let's compute average monthly temperatures in both °F and °C using the similar `group_by()` and `summarize()` code as in the previous section.

```
summary_monthly_temp <- weather %>%
  group_by(month) %>%
  summarize(mean_temp_in_F = mean(temp, na.rm = TRUE),
            mean_temp_in_C = mean(temp_in_C, na.rm = TRUE))
summary_monthly_temp
```

```
# A tibble: 12 x 3
  month mean_temp_in_F mean_temp_in_C
  <dbl>      <dbl>      <dbl>
1     1        35.6       2.02
2     2        34.3       1.26
3     3        39.9       4.38
```

4	4	51.7	11.0
5	5	61.8	16.6
6	6	72.2	22.3
7	7	80.1	26.7
8	8	74.5	23.6
9	9	67.4	19.7
10	10	60.1	15.6
11	11	45.0	7.22
12	12	38.4	3.58

Let's consider another example. Passengers are often frustrated when their flights depart late, but change their mood a bit if pilots can make up some time during the flight to get them to their destination close to the original arrival time. This is commonly referred to as "gain" and we will create this variable using the `mutate()` function.

```
flights <- flights %>%
  mutate(gain = dep_delay - arr_delay)
```

Let's take a look at only the `dep_delay`, `arr_delay`, and the resulting `gain` variables for the first 5 rows in our updated `flights` data frame:

TABLE 4.1: Subset of flights data frame

dep_delay	arr_delay	gain
2	11	-9
4	20	-16
2	33	-31
-1	-18	17
-6	-25	19

The flight in the first row departed 2 minutes late but arrived 11 minutes late, so its "gained time in the air" is actually a loss of 9 minutes, hence its `gain` is `-9`. Contrast this to the flight in the fourth row which departed a minute early (`dep_delay` of `-1`) but arrived 18 minutes early (`arr_delay` of `-18`), so its "gained time in the air" is 17 minutes, hence its `gain` is `+17`.

Let's look at summary measures of this `gain` variable and even plot it in the form of a histogram:

```
gain_summary <- flights %>%
```

```
summarize(  
  min = min(gain, na.rm = TRUE),  
  q1 = quantile(gain, 0.25, na.rm = TRUE),  
  median = quantile(gain, 0.5, na.rm = TRUE),  
  q3 = quantile(gain, 0.75, na.rm = TRUE),  
  max = max(gain, na.rm = TRUE),  
  mean = mean(gain, na.rm = TRUE),  
  sd = sd(gain, na.rm = TRUE),  
  missing = sum(is.na(gain))  
)  
gain_summary
```

```
# A tibble: 1 x 8  
  min     q1 median     q3   max   mean     sd missing  
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int>  
1 -196    -3     7    17   109   5.66  18.0    9430
```

We've recreated the `summary` function we saw in Chapter 3 here using the `summarize()` function in `dplyr`.

```
ggplot(data = flights, mapping = aes(x = gain)) +  
  geom_histogram(color = "white", bins = 20)
```

We can also create multiple columns at once and even refer to columns that were just created in a new column. Hadley and Garrett produce one such example in Chapter 5 of “R for Data Science” (Grolmund and Wickham, 2016):

```
flights <- flights %>%  
  mutate(  
    gain = dep_delay - arr_delay,  
    hours = air_time / 60,  
    gain_per_hour = gain / hours  
)
```

Learning check

(LC4.10) What do positive values of the `gain` variable in `flights` correspond to? What about negative values? And what about a zero value?

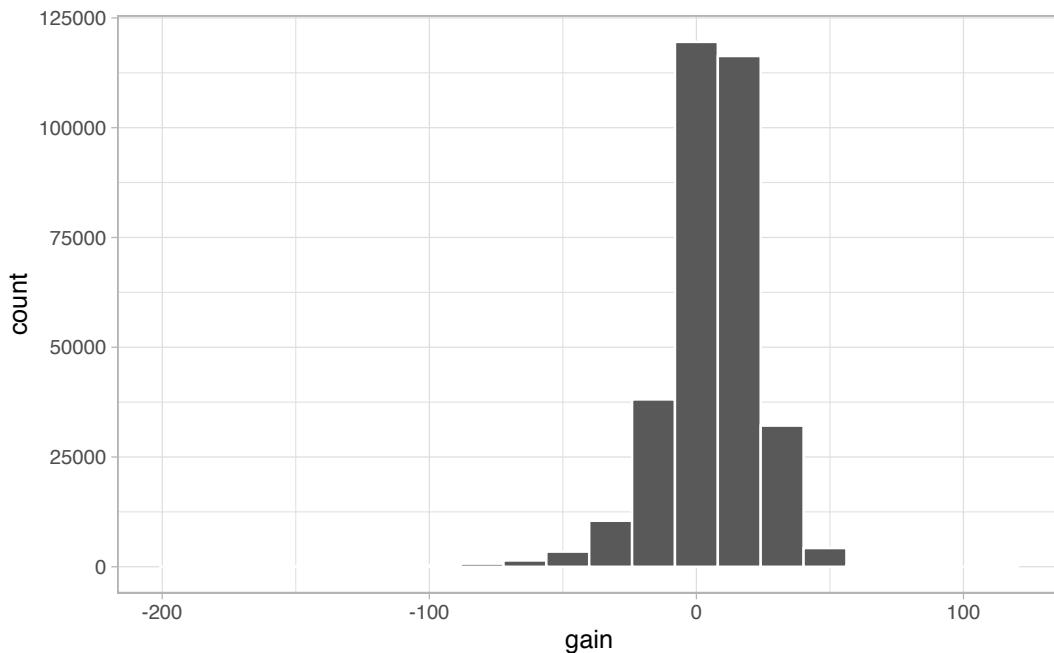


FIGURE 4.6: Histogram of gain variable.

(LC4.11) Could we create the `dep_delay` and `arr_delay` columns by simply subtracting `dep_time` from `sched_dep_time` and similarly for arrivals? Try the code out and explain any differences between the result and what actually appears in `flights`.

(LC4.12) What can we say about the distribution of `gain`? Describe it in a few sentences using the plot and the `gain_summary` data frame values.

4.6 `arrange` and sort rows

One of the most common tasks people working with data would like to perform is sort the data frame's rows in alphanumeric order of the values in a variable/column. For example, when calculating a median by hand requires you to first sort the data from the smallest to highest in value and then identify the “middle” value. The `dplyr` package has a function called `arrange()` that we will use to sort/reorder a data frame's rows according to the values of the

specified variable. This is often used after we have used the `group_by()` and `summarize()` functions as we will see.

Let's suppose we were interested in determining the most frequent destination airports for all domestic flights departing from New York City in 2013:

```
freq_dest <- flights %>%
  group_by(dest) %>%
  summarize(num_flights = n())
freq_dest

# A tibble: 105 x 2
  dest   num_flights
  <chr>     <int>
1 ABQ        254
2 ACK        265
3 ALB        439
4 ANC         8
5 ATL       17215
6 AUS       2439
7 AVL        275
8 BDL        443
9 BGR        375
10 BHM       297
# ... with 95 more rows
```

Observe that by default the rows of the resulting `freq_dest` data frame are sorted in alphabetical order of `dest` destination. Say instead we would like to see the same data, but sorted from the most to the least number of flights `num_flights` instead:

```
freq_dest %>%
  arrange(num_flights)

# A tibble: 105 x 2
  dest   num_flights
  <chr>     <int>
1 LEX        1
2 LGA        1
3 ANC         8
4 SBN        10
5 HDN        15
```

```
6 MTJ          15
7 EYW          17
8 PSP          19
9 JAC          25
10 BZN         36
# ... with 95 more rows
```

This is actually giving us the opposite of what we are looking for: the rows are sorted with the least frequent destination airports displayed first. To switch the ordering to be descending instead of ascending we use the `desc()` function, which is short for “descending”:

```
freq_dest %>%
  arrange(desc(num_flights))
```

```
# A tibble: 105 x 2
  dest   num_flights
  <chr>     <int>
1 ORD      17283
2 ATL      17215
3 LAX      16174
4 BOS      15508
5 MCO      14082
6 CLT      14064
7 SFO      13331
8 FLL      12055
9 MIA      11728
10 DCA     9705
# ... with 95 more rows
```

In other words, `arrange()` sorts in ascending order by default unless you override this default behavior by using `desc()`.

4.7 join data frames

Another common data transformation task is “joining” or “merging” two different datasets. For example, in the `flights` data frame the variable `carrier` lists the carrier code for the different flights. While the corresponding airline names for "UA" and "AA" might be somewhat easy to guess (United and Ameri-

can Airlines), what airlines have codes? "vx", "HA", and "B6"? This information is provided in a separate data frame `airlines`.

```
View(airlines)
```

We see that in `airports`, `carrier` is the carrier code while `name` is the full name of the airline company. Using this table, we can see that "vx", "HA", and "B6" correspond to Virgin America, Hawaiian Airlines, and JetBlue respectively. However, wouldn't it be nice to have all this information in a single data frame instead of two separate data frames? We can do this by "joining" i.e. "merging" the `flights` and `airlines` data frames.

Note that the values in the variable `carrier` in the `flights` data frame match the values in the variable `carrier` in the `airlines` data frame. In this case, we can use the variable `carrier` as a *key variable* to match the rows of the two data frames. Key variables are almost always identification variables that uniquely identify the observational units as we saw in Subsection 2.4.4. This ensures that rows in both data frames are appropriately matched during the join. Hadley and Garrett ([Gromelund and Wickham, 2016](#)) created the following diagram to help us understand how the different datasets are linked by various key variables:

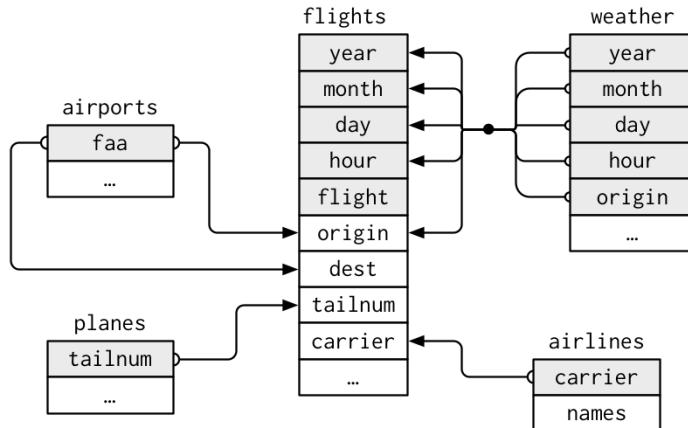


FIGURE 4.7: Data relationships in `nycflights13` from 'R for Data Science'.

4.7.1 Matching “key” variable names

In both the `flights` and `airlines` data frames, the key variable we want to join/merge/match the rows of the two data frames by have the same name: `carriers`. We make use of the `inner_join()` function to join the two data frames, where the rows will be matched by the variable `carrier`.

```
flights_joined <- flights %>%
  inner_join(airlines, by = "carrier")
View(flights)
View(flights_joined)
```

Observe that the `flights` and `flights_joined` data frames are identical except that `flights_joined` has an additional variable `name` whose values correspond to the airline company names drawn from the `airlines` data frame.

A visual representation of the `inner_join()` is given below (Grolmund and Wickham, 2016). There are other types of joins available (such as `left_join()`, `right_join()`, `outer_join()`, and `anti_join()`), but the `inner_join()` will solve nearly all of the problems you'll encounter in this book.

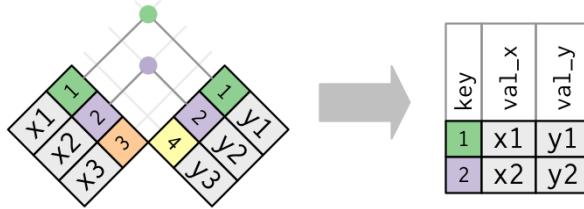


FIGURE 4.8: Diagram of inner join from 'R for Data Science'.

4.7.2 Different “key” variable names

Say instead you are interested in the destinations of all domestic flights departing NYC in 2013 and ask yourself:

- “What cities are these airports in?”
- “Is “ORD” Orlando?”
- “Where is “FLL”?”

The `airports` data frame contains airport codes:

```
View(airports)
```

However, looking at both the `airports` and `flights` frames and the visual representation of the relations between these data frames in Figure 4.8 above, we see that in:

- the `airports` data frame the airport code is in the variable `faa`
- the `flights` data frame the airport codes are in the variables `origin` and `dest`

So to join these two data frames so that we can identify the destination cities for example, our `inner_join()` operation will use the `by = c("dest" = "faa")` argument, which allows us to join two data frames where the key variable has a different name:

```
flights_with_airport_names <- flights %>%
  inner_join(airports, by = c("dest" = "faa"))
View(flights_with_airport_names)
```

Let's construct the sequence of commands that computes the number of flights from NYC to each destination, but also includes information about each destination airport:

```
named_dests <- flights %>%
  group_by(dest) %>%
  summarize(num_flights = n()) %>%
  arrange(desc(num_flights)) %>%
  inner_join(airports, by = c("dest" = "faa")) %>%
  rename(airport_name = name)
named_dests
```

```
# A tibble: 101 x 9
  dest num_flights airport_name   lat    lon    alt    tz
  <chr>     <int> <chr>       <dbl>  <dbl> <int>  <dbl>
1 ORD        17283 Chicago Oha~  42.0   -87.9   668   -6
2 ATL        17215 Hartsfield ~  33.6   -84.4  1026   -5
3 LAX        16174 Los Angeles~  33.9  -118.    126   -8
4 BOS        15508 General Edw~  42.4   -71.0    19   -5
5 MCO        14082 Orlando Intl  28.4   -81.3    96   -5
6 CLT        14064 Charlotte D~  35.2   -80.9   748   -5
7 SFO        13331 San Francis~  37.6  -122.     13   -8
8 FLL        12055 Fort Lauder~  26.1   -80.2     9   -5
9 MIA        11728 Miami Intl   25.8   -80.3     8   -5
10 DCA       9705 Ronald Reag~  38.9   -77.0    15   -5
# ... with 91 more rows, and 2 more variables: dst <chr>,
#   tzone <chr>
```

In case you didn't know, "ORD" is the airport code of Chicago O'Hare airport and "FLL" is the main airport in Fort Lauderdale, Florida, which we can now see in the `airport_name` variable in the resulting `named_dests` data frame.

4.7.3 Multiple “key” variables

Say instead we are in a situation where we need to join by multiple variables. For example, in Figure 4.7 above we see that in order to join the `flights` and `weather` data frames, we need more than one key variable: `year`, `month`, `day`, `hour`, and `origin`. This is because the combination of these 5 variables act to uniquely identify each observational unit in the `weather` data frame: hourly weather recordings at each of the 3 NYC airports.

We achieve this by specifying a vector of key variables to join by using the `c()` function for “combine” or “concatenate” that we saw earlier:

```
flights_weather_joined <- flights %>%
  inner_join(weather, by = c("year", "month", "day", "hour", "origin"))
View(flights_weather_joined)
```

Learning check

(LC4.13) Looking at Figure 4.7, when joining `flights` and `weather` (or, in other words, matching the hourly weather values with each flight), why do we need to join by all of `year`, `month`, `day`, `hour`, and `origin`, and not just `hour`?

(LC4.14) What surprises you about the top 10 destinations from NYC in 2013?

4.7.4 Normal forms

The data frames included in the `nycflights13` package are in a form that minimizes redundancy of data. For example, the `flights` data frame only saves the `carrier` code of the airline company; it does not include the actual name of the airline. For example the first row of `flights` has `carrier` equal to `UA`, but does it does not include the airline name “United Air Lines Inc.” The names of the airline companies are included in the `name` variable of the `airlines` data frame. In order to have the airline company name included in `flights`, we could join these two data frames as follows:

```
joined_flights <- flights %>%
  inner_join(airlines, by = "carrier")
View(joined_flights)
```

We are capable of performing this join because each of the data frames have *keys* in common to relate one to another: the `carrier` variable in both the `flights` and `airlines` data frames. The *key* variable(s) that we join are often *identification variables* we mentioned previously.

This is an important property of what's known as **normal forms** of data. The process of decomposing data frames into less redundant tables without losing information is called **normalization**. More information is available on Wikipedia¹.

Learning check

(LC4.15) What are some advantages of data in normal forms? What are some disadvantages?

4.8 Other verbs

Here are some other useful data wrangling verbs that might come in handy:

- `select()` only a subset of variables/columns
- `rename()` variables/columns to have new names
- Return only the `top_n()` values of a variable

4.8.1 `select` variables

We've seen that the `flights` data frame in the `nycflights13` package contains 19 different variables. You can identify the names of these 19 variables by running the `glimpse()` function from the `dplyr` package:

¹https://en.wikipedia.org/wiki/Database_normalization

Subset Variables (Columns)

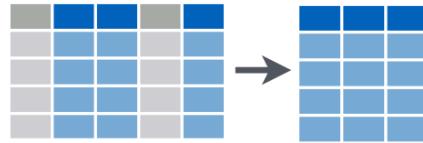


FIGURE 4.9: Diagram of `select()` columns.

```
glimpse(flights)
```

However, say you only need two of these variables, say `carrier` and `flight`. You can `select()` these two variables:

```
flights %>%
  select(carrier, flight)
```

This function makes exploring data frames with a very large number of variables easier for humans to process by restricting consideration to only those we care about, like our example with `carrier` and `flight` above. This might make viewing the dataset using the `view()` spreadsheet viewer more digestible. However, as far as the computer is concerned, it doesn't care how many additional variables are in the data frame in question, so long as `carrier` and `flight` are included.

Let's say instead you want to drop i.e. deselect certain variables. For example, take the variable `year` in the `flights` data frame. This variable isn't quite a "variable" in the sense that all the values are `2013` i.e. it doesn't change. Say you want to remove the `year` variable from the data frame; we can deselect `year` by using the `-` sign:

```
flights_no_year <- flights %>%
  select(-year)
glimpse(flights_no_year)
```

Another way of selecting columns/variables is by specifying a range of columns:

```
flight_arr_times <- flights %>%
  select(month:day, arr_time:sched_arr_time)
flight_arr_times
```

The `select()` function can also be used to reorder columns in combination with the `everything()` helper function. Let's suppose we'd like the `hour`, `minute`, and `time_hour` variables, which appear at the end of the `flights` dataset, to appear immediately after the `year`, `month`, and `day` variables while keeping the rest of the variables. In the code below `everything()` picks up all remaining variables.

```
flights_reorder <- flights %>%
  select(year, month, day, hour, minute, time_hour, everything())
glimpse(flights_reorder)
```

Lastly, the helper functions `starts_with()`, `ends_with()`, and `contains()` can be used to select variables/column that match those conditions. For example:

```
flights_begin_a <- flights %>%
  select(starts_with("a"))
flights_begin_a
```

```
flights_delays <- flights %>%
  select(ends_with("delay"))
flights_delays
```

```
flights_time <- flights %>%
  select(contains("time"))
flights_time
```

4.8.2 `rename` variables

Another useful function is `rename()`, which as you may have guessed renames one column to another name. Suppose we want `dep_time` and `arr_time` to be `departure_time` and `arrival_time` instead in the `flights_time` data frame:

```
flights_time_new <- flights %>%
  select(contains("time")) %>%
```

```
rename(departure_time = dep_time,
       arrival_time = arr_time)
glimpse(flights_time_new)
```

Note that in this case we used a single = sign within the `rename()`, for example `departure_time = dep_time`. This is because we are not testing for equality like we would using ==, but instead we want to assign a new variable `departure_time` to have the same values as `dep_time` and then delete the variable `dep_time`. It's easy to forget if the new name comes before or after the equals sign. We usually remember this as "New Before, Old After" or NBOA.

4.8.3 `top_n` values of a variable

We can also return the top `n` values of a variable using the `top_n()` function. For example, we can return a data frame of the top 10 destination airports using the example from Section 4.7.2. Observe that we set the number of values to return to `n = 10` and `wt = num_flights` to indicate that we want the rows corresponding to the top 10 values of `num_flights`. See the help file for `top_n()` by running `?top_n` for more information.

```
named_dests %>%
  top_n(n = 10, wt = num_flights)
```

Let's further `arrange()` these results in descending order of `num_flights`:

```
named_dests %>%
  top_n(n = 10, wt = num_flights) %>%
  arrange(desc(num_flights))
```

Learning check

(LC4.16) What are some ways to select all three of the `dest`, `air_time`, and `distance` variables from `flights`? Give the code showing how to do this in at least three different ways.

(LC4.17) How could one use `starts_with`, `ends_with`, and `contains` to select columns from the `flights` data frame? Provide three different examples in total: one for `starts_with`, one for `ends_with`, and one for `contains`.

(LC4.18) Why might we want to use the `select` function on a data frame?

(LC4.19) Create a new data frame that shows the top 5 airports with the largest arrival delays from NYC in 2013.

4.9 Conclusion

4.9.1 Summary table

Let's recap our data wrangling verbs in Table 4.2. Using these verbs and the pipe `%>%` operator from Section 4.1, you'll be able to write easily legible code to perform almost all the data wrangling necessary for the rest of this book.

TABLE 4.2: Summary of data wrangling verbs

Verb	Data wrangling operation
<code>'filter()'</code>	Pick out a subset of rows
<code>'summarize()'</code>	Summarize many values to one using a summary statistic function like <code>'mean()'</code> , <code>'median()'</code> , etc.
<code>'group_by()'</code>	Add grouping structure to rows in data frame. Note this does not change values in data frame.
<code>'mutate()'</code>	Create new variables by mutating existing ones
<code>'arrange()'</code>	Arrange rows of a data variable in ascending (default) or <code>'desc'</code> ending order
<code>'inner_join()'</code>	Join/merge two data frames, matching rows by a key variable

Learning check

(LC4.20) Let's now put your newly acquired data wrangling skills to the test!

An airline industry measure of a passenger airline's capacity is the available seat miles², which is equal to the number of seats available multiplied by the number of miles or kilometers flown summed over all flights. So for example say an airline had 2 flights using a plane with 10 seats that flew 500 miles and 3 flights using a plane with 20 seats that flew 1000 miles, the available seat miles would be $2 \times 10 \times 500 + 3 \times 20 \times 1000 = 70,000$ seat miles.

²https://en.wikipedia.org/wiki/Available_seat_miles

Using the datasets included in the `nycflights13` package, compute the available seat miles for each airline sorted in descending order. After completing all the necessary data wrangling steps, the resulting data frame should have 16 rows (one for each airline) and 2 columns (airline name and available seat miles). Here are some hints:

1. **Crucial:** Unless you are very confident in what you are doing, it is worthwhile to not starting to code right away. Rather first sketch out on paper all the necessary data wrangling steps not using exact code, but rather high-level *pseudocode* that is informal yet detailed enough to articulate what you are doing. This way you won't confuse *what* you are trying to do (the algorithm) with *how* you are going to do it (writing `dplyr` code).
2. Take a close look at all the datasets using the `view()` function: `flights`, `weather`, `planes`, `airports`, and `airlines` to identify which variables are necessary to compute available seat miles.
3. Figure 4.7 above showing how the various datasets can be joined will also be useful.
4. Consider the data wrangling verbs in Table 4.2 as your toolbox!

4.9.2 Additional resources

An R script file of all R code used in this chapter is available here³.

If you want to further unlock the power of the `dplyr` package for data wrangling, we suggest you that you check out RStudio's "Data Transformation with `dplyr`" cheatsheet. This cheatsheet summarizes much more than what we've discussed in this chapter, in particular more-intermediate level and advanced data wrangling functions, while providing quick and easy to read visual descriptions.

You can access this cheatsheet by going to the RStudio Menu Bar -> Help -> Cheatsheets -> "Data Transformation with `dplyr`".

On top of data wrangling verbs and examples we presented in this section, if you'd like to see more examples of using the `dplyr` package for data wrangling check out Chapter 5⁴ of Garrett Grolemund and Hadley Wickham's and Garrett's book ([Grolemund and Wickham, 2016](#)).

³[scripts/04-wrangling.R](#)

⁴<http://r4ds.had.co.nz/transform.html>

Data Transformation with dplyr :: CHEAT SHEET

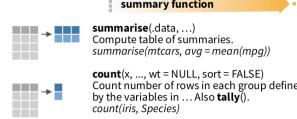


dplyr functions work with pipes and expect **tidy data**. In tidy data:



Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

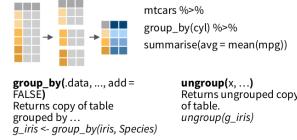


VARIATIONS

`summarise_all()` - Apply funs to every column.
`summarise_at()` - Apply funs to specific columns.
`summarise_if()` - Apply funs to all cols of one type.

Group Cases

Use `group_by()` to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

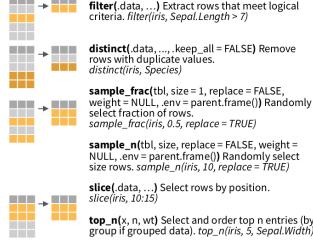


RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with `browseVignettes(package = c('dplyr', 'tibble'))` • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2017-03

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.



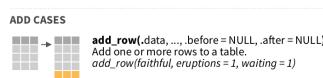
Logical and boolean operators to use with filter()

< <= is.na() %in% | xor()
> >= !is.na() ! &

See ?base::logical and ?Comparison for help.

ARRANGE CASES

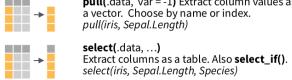
ADD CASES



Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.



Use these helpers with select(), e.g. `select(iris, starts_with("Sepal"))`

`contains(match)` `num_range(prefix, range)` ; e.g. `mpg:cyl`
`ends_with(match)` `one_of(...)`
`matches(match)` `starts_with(match)`

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

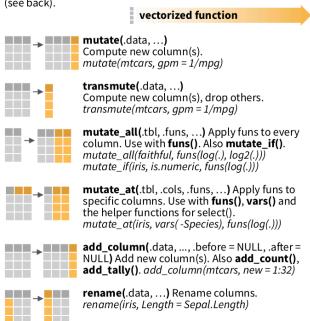


FIGURE 4.10: Data Transformation with dplyr cheatsheet.

4.9.3 What's to come?

So far in this book, we've explored, visualized, and wrangled data saved in data frames that are in spreadsheet-type format: rectangular with a certain number of rows corresponding to observations and a certain number of columns corresponding to variables describing the observations.

We'll see in Chapter 5 that there are actually two ways to represent data in spreadsheet-type rectangular format: 1) "wide" format and 2) "tall/narrow" format also known in R circles as "tidy" format. While the distinction between "tidy" and non-"tidy" formatted data is very subtle, it has very large implications for whether or not we can use the `ggplot2` package for data visualization and the `dplyr` package for data wrangling.

Furthermore, we've only explored, visualized, and wrangled data saved within R packages. What if you have spreadsheet data saved in a Microsoft Excel, Google Sheets, or "Comma-Separated Values" (CSV) file that you would like to analyze? In Chapter 5, we'll show you how to import this data into R using the `readr` package.

5

Data Importing & “Tidy” Data

In Subsection 2.2.1 we introduced the concept of a data frame: a rectangular spreadsheet-like representation of data in R where the rows correspond to observations and the columns correspond to variables describing each observation. In Section 2.4, we started exploring our first data frame: the `flights` data frame included in the `nycflights13` package. In Chapter 3 we created visualizations based on the data included in `flights` and other data frames such as `weather`. In Chapter 4, we learned how to wrangle data, in other words take existing data frames and transform/modify them to suit our analysis goals.

In this final chapter of the “Data Science via the tidyverse” portion of the book, we extend some of these ideas by discussing a type of data formatting called “tidy” data. You will see that having data stored in “tidy” format is about more than what the colloquial definition of the term “tidy” might suggest: having your data “neatly organized.” Instead, we define the term “tidy” in a more rigorous fashion, outlining a set of rules by which data can be stored, and the implications of these rules for analyses.

Although knowledge of this type of data formatting was not necessary for our treatment of data visualization in Chapter 3 and data wrangling in Chapter 4 since all the data was already in “tidy” format, we’ll now see this format is actually essential to using the tools we covered in these two chapters. Furthermore, it will also be useful for all subsequent chapters in this book when we cover regression and statistical inference. First however, we’ll show you how to import spreadsheet data for use in R.

Needed packages

Let’s load all the packages needed for this chapter (this assumes you’ve already installed them). If needed, read Section 2.3 for information on how to install and load R packages.

```
library(dplyr)
library(ggplot2)
library(readr)
```

```
library(tidyr)
library(nycflights13)
library(fivethirtyeight)
```

5.1 Importing data

Up to this point, we’ve almost entirely used data stored inside of an R package. Say instead you have your own data saved on your computer or somewhere online? How can you analyze this data in R? Spreadsheet data is often saved in one of the following formats:

- A *Comma Separated Values* .csv file. You can think of a .csv file as a bare-bones spreadsheet where:
 - Each line in the file corresponds to one row of data/one observation.
 - Values for each line are separated with commas. In other words, the values of different variables are separated by commas.
 - The first line is often, but not always, a *header* row indicating the names of the columns/variables.
- An Excel .xlsx file. This format is based on Microsoft’s proprietary Excel software. As opposed to a bare-bones .csv files, .xlsx Excel files contain a lot of meta-data, or put more simply, data about the data. (Recall we saw a previous example of meta-data in Section 4.4 when adding “group structure” meta-data to a data frame by using the `group_by()` verb.) Some examples of spreadsheet meta-data include the use of bold and italic fonts, colored cells, different column widths, and formula macros.
- A Google Sheets¹ file, which is a “cloud” or online-based way to work with a spreadsheet. Google Sheets allows you to download your data in both comma separated values .csv and Excel .xlsx formats however: go to the Google Sheets menu bar -> File -> Download as -> Select “Microsoft Excel” or “Comma-separated values.”

We’ll cover two methods for importing .csv and .xlsx spreadsheet data in R: one using the R console and the other using RStudio’s graphical user interface, abbreviated a GUI.

¹<https://www.google.com/sheets/about/>

5.1.1 Using the console

First, let's import a Comma Separated Values .csv file of data directly off the internet. The .csv file `dem_score.csv` accessible at https://moderndive.com/data/dem_score.csv contains ratings of the level of democracy in different countries spanning 1952 to 1992. Let's use the `read_csv()` function from the `readr` package to read it off the web, import it into R, and save it in a data frame called `dem_score`.

```
library(readr)
dem_score <- read_csv("https://moderndive.com/data/dem_score.csv")
dem_score
```

```
# A tibble: 96 x 10
  country `1952` `1957` `1962` `1967` `1972` `1977` `1982` 
  <chr>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl> 
1 Albania     -9      -9      -9      -9      -9      -9      -9  
2 Argent~     -9      -1      -1      -9      -9      -9      -8  
3 Armenia     -9      -7      -7      -7      -7      -7      -7  
4 Austra~     10      10      10      10      10      10      10  
5 Austria     10      10      10      10      10      10      10  
6 Azerba~     -9      -7      -7      -7      -7      -7      -7  
7 Belarus     -9      -7      -7      -7      -7      -7      -7  
8 Belgium     10      10      10      10      10      10      10  
9 Bhutan      -10     -10     -10     -10     -10     -10     -10 
10 Bolivia     -4      -3      -3      -4      -7      -7      8  
# ... with 86 more rows, and 2 more variables:
#   `1987` <dbl>, `1992` <dbl>
```

In this `dem_score` data frame, the minimum value of `-10` corresponds to a highly autocratic nation whereas a value of `10` corresponds to a highly democratic nation. We'll revisit the `dem_score` data frame in a case study in the upcoming Section 5.3.

Note that the `read_csv()` function included in the `readr` package is different than the `read.csv()` function that comes installed with R by default. While the difference in the names might seem near meaningless (an `_` instead of a `.`), the `read_csv()` function is in our opinion easier to use since it can more easily read data off the web and generally imports data at a much faster speed.

5.1.2 Using RStudio’s interface

Let’s read in the exact same data saved in Excel format, but this time via RStudio’s graphical interface instead of via the R console. First download the Excel file `dem_score.xlsx` by going here, then

1. Go to the Files panel of RStudio.
2. Navigate to the directory i.e. folder on your computer where the downloaded `dem_score.xlsx` Excel file is saved.
3. Click on `dem_score.xlsx`.
4. Click “Import Dataset...”

At this point you should see an image like this:

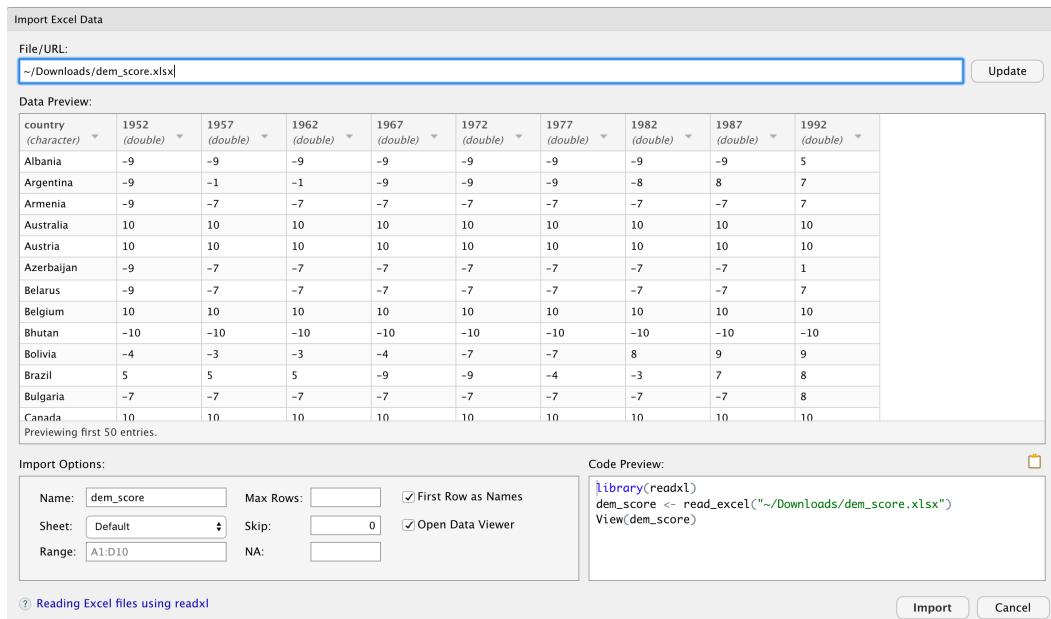


FIGURE 5.1: Reading in an Excel file to R.

After clicking on the “Import” button on the bottom right RStudio, RStudio will save this spreadsheet’s data in a data frame called `dem_score` and display its contents in the spreadsheet viewer. Furthermore, note in the bottom right of the above image there exists a “Code Preview”: you can copy and paste this code to reload your data again later automatically instead of repeating the above manual point-and-click process.

5.2 Tidy data

Let's now switch gears and learn about the concept of "tidy" data format by starting with a motivating example. Let's consider the `drinks` data frame included in the `fivethirtyeight` data. Run the following:

```
drinks
```

```
# A tibble: 193 x 5
  country beer_servings spirit_servings wine_servings
  <chr>      <int>          <int>          <int>
1 Afghan~         0            0            0
2 Albania        89           132           54
3 Algeria        25            0           14
4 Andorra       245           138          312
5 Angola        217            57           45
6 Antigu~       102           128           45
7 Argent~       193            25          221
8 Armenia        21            179           11
9 Austra~       261            72          212
10 Austria       279            75          191
# ... with 183 more rows, and 1 more variable:
#   total_litres_of_pure_alcohol <dbl>
```

After reading the help file by running `?drinks`, we see that `drinks` is a data frame containing results from a survey of the average number of servings of beer, spirits, and wine consumed for 193 countries. This data was originally reported on the data journalism website FiveThirtyEight.com in Mona Chalabi's article "Dear Mona Followup: Where Do People Drink The Most Beer, Wine And Spirits?"²

Let's apply some of the data wrangling verbs we learned in Chapter 4 on the `drinks` data frame. Let's

1. `filter()` the `drinks` data frame to only consider 4 countries (the United States, China, Italy, and Saudi Arabia) then
2. `select()` all columns except `total_litres_of_pure_alcohol` by using - sign, then

²<https://fivethirtyeight.com/features/dear-mona-followup-where-do-people-drink-the-most-beer-wine-and-spirits/>

3. `rename()` the variables `beer_servings`, `spirit_servings`, and `wine_servings` to `beer`, `spirit`, and `wine` respectively

and save the resulting data frame in `drinks_smaller`.

```
drinks_smaller <- drinks %>%
  filter(country %in% c("USA", "China", "Italy", "Saudi Arabia")) %>%
  select(-total_litres_of_pure_alcohol) %>%
  rename(beer = beer_servings, spirit = spirit_servings, wine = wine_servings)
drinks_smaller
```

```
# A tibble: 4 x 4
  country      beer  spirit  wine
  <chr>       <int>   <int>   <int>
1 China        79    192     8
2 Italy         85     42    237
3 Saudi Arabia  0      5     0
4 USA          249   158    84
```

Using the `drinks_smaller` data frame, how would we create the side-by-side AKA dodged barplot in Figure 5.2? Recall we saw barplots displaying two categorical variables in Section 3.8.3.

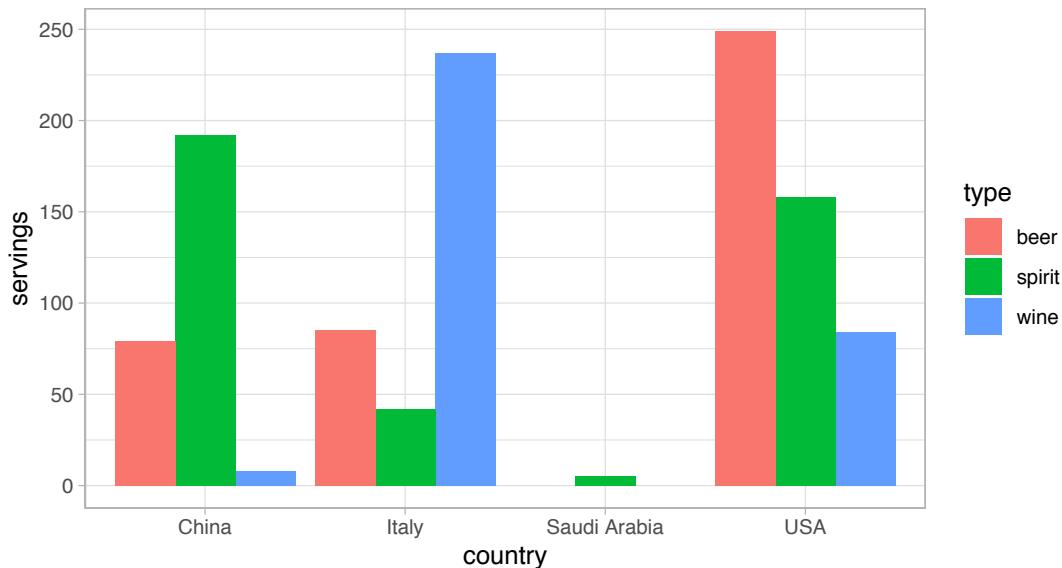


FIGURE 5.2: Comparing alcohol consumption in 4 countries.

Let’s break down the Grammar of Graphics:

1. The categorical variable `country` with four levels (China, Italy, Saudi Arabia, USA) would have to be mapped to the `x`-position of the bars.
2. The numerical variable `servings` would have to be mapped to the `y`-position of the bars, in other words the height of the bars.
3. The categorical variable `type` with three levels (beer, spirit, wine) who have to be mapped to the `fill` color of the bars.

Observe however that `drinks_smaller` has *three separate variables* for `beer`, `spirit`, and `wine`, whereas in order to recreate the side-by-side AKA dodged barplot in Figure 5.2 we would need a *single variable* `type` with three possible values: `beer`, `spirit`, and `wine`, which we would then map to the `fill` aesthetic. In other words, for us to be able to create the barplot in Figure 5.2, our data frame would have to look like this:

drinks_smaller_tidy

```
# A tibble: 12 x 3
  country     type   servings
  <chr>      <chr>    <int>
1 China       beer      79
2 Italy        beer      85
3 Saudi Arabia beer      0
4 USA          beer     249
5 China       spirit    192
6 Italy        spirit    42
7 Saudi Arabia spirit    5
8 USA          spirit   158
9 China       wine      8
10 Italy       wine     237
11 Saudi Arabia wine      0
12 USA          wine     84
```

Let's compare the `drinks_smaller_tidy` with the `drinks_smaller` data frame from earlier:

drinks_smaller

```
# A tibble: 4 x 4
  country     beer   spirit   wine
  <chr>      <int>  <int>  <int>
1 China        79    192      8
2 Italy        85     42     237
```

3 Saudi Arabia	0	5	0
4 USA	249	158	84

Observe that while `drinks_smaller` and `drinks_smaller_tidy` are both rectangular in shape and contain the same 12 numerical values (3 alcohol types \times 4 countries), they are formatted differently. `drinks_smaller` is formatted in what's known as "wide"³ format, whereas `drinks_smaller_tidy` is formatted in what's known as "long/narrow"⁴. In the context of using R, long/narrow format is also known as "tidy" format. Furthermore, in order to use the `ggplot2` and `dplyr` packages for data visualization and data wrangling, your input data frames *must* be in "tidy" format. So all non-"tidy" data must be converted to "tidy" format first.

Before we show you how to convert non-"tidy" data frames like `drinks_smaller` to "tidy" data frames like `drinks_smaller_tidy`, let's go over the explicit definition of "tidy" data.

5.2.1 Definition of “tidy” data

You have surely heard the word "tidy" in your life:

- “Tidy up your room!”
- “Please write your homework in a tidy way so that it is easier to grade and to provide feedback.”
- Marie Kondo's best-selling book *The Life-Changing Magic of Tidying Up: The Japanese Art of Decluttering and Organizing*⁵ and Netflix TV series *Tidying Up with Marie Kondo*⁶.
- “I am not by any stretch of the imagination a tidy person, and the piles of unread books on the coffee table and by my bed have a plaintive, pleading quality to me - ‘Read me, please!’ ” - Linda Grant

What does it mean for your data to be "tidy"? While "tidy" has a clear English meaning of "organized", "tidy" in the context of data science using R means that your data follows a standardized format. We will follow Hadley Wickham's definition of *tidy data* here (Wickham, 2014):

A dataset is a collection of values, usually either numbers (if quantitative)

³https://en.wikipedia.org/wiki/Wide_and_narrow_data

⁴https://en.wikipedia.org/wiki/Wide_and_narrow_data#Narrow

⁵https://www.amazon.com/Life-Changing-Magic-Tidying-Decluttering-Organizing/dp/1607747308/ref=sr_1_1?ie=UTF8&qid=1469400636&sr=8-1&keywords=tidying+up

⁶<https://www.netflix.com/title/80209379>

or strings AKA text data (if qualitative). Values are organised in two ways. Every value belongs to a variable and an observation. A variable contains all values that measure the same underlying attribute (like height, temperature, duration) across units. An observation contains all values measured on the same unit (like a person, or a day, or a city) across attributes.

Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types. In *tidy data*:

1. Each variable forms a column.
 2. Each observation forms a row.
 3. Each type of observational unit forms a table.
-

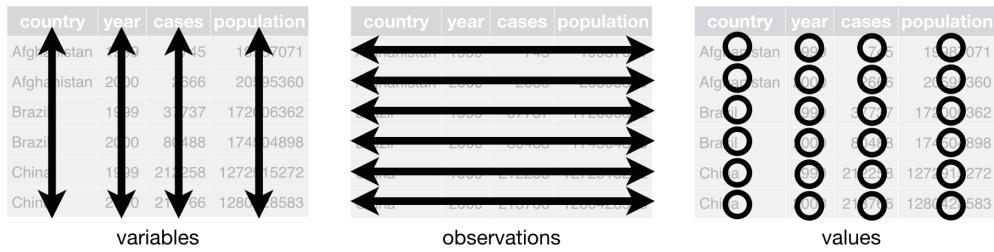


FIGURE 5.3: Tidy data graphic from 'R for Data Science'.

For example, say you have the following table of stock prices in Table 5.1:

TABLE 5.1: Stock Prices (Non-Tidy Format)

Date	Boeing Stock Price	Amazon Stock Price	Google Stock Price
2009-01-01	\$173.55	\$174.90	\$174.34
2009-01-02	\$172.61	\$171.42	\$170.04

Although the data are neatly organized in a rectangular spreadsheet-type format, they are not in tidy format because while there are three variables corresponding to three unique pieces of information (Date, Stock Name, and Stock Price), there are not three columns. In “tidy” data format each variable should be its own column, as shown in Table 5.2. Notice that both tables present the same information, but in different formats.

TABLE 5.2: Stock Prices (Tidy Format)

Date	Stock Name	Stock Price
2009-01-01	Boeing	\$173.55
2009-01-02	Boeing	\$172.61
2009-01-01	Amazon	\$174.90
2009-01-02	Amazon	\$171.42
2009-01-01	Google	\$174.34
2009-01-02	Google	\$170.04

Now we have the requisite three columns Date, Stock Name, and Stock Price. On the other hand, consider the data in Table 5.3.

TABLE 5.3: Date, Boeing Price, Weather Data

Date	Boeing Price	Weather
2009-01-01	\$173.55	Sunny
2009-01-02	\$172.61	Overcast

In this case, even though the variable “Boeing Price” occurs just like in our non-“tidy” data in Table 5.1, the data *is* “tidy” since there are three variables corresponding to three unique pieces of information: Date, Boeing stock price, and the weather that particular day.

Learning check

(LC5.1) What are common characteristics of “tidy” data frames?

(LC5.2) What makes “tidy” data frames useful for organizing data?

5.2.2 Converting to “tidy” data

In this book so far, you’ve only seen data frames that were already in “tidy” format. Furthermore for the rest of this book, you’ll mostly only see data frames that are already in “tidy” format as well. This is not always the case however with data in the wild. If your original data frame is in wide i.e. non-“tidy” format and you would like to use the `ggplot2` package for data visualization or the `dplyr` package for data wrangling, you will first have to convert it “tidy” format using the `gather()` function in the `tidyverse` package ([Wickham and Henry, 2019](#)).

Going back to our `drinks_smaller` data frame from earlier:

```
drinks_smaller
```

```
# A tibble: 4 x 4
  country     beer  spirit   wine
  <chr>      <int> <int>    <int>
1 China        79    192      8
2 Italy        85     42     237
3 Saudi Arabia  0      5      0
4 USA         249    158     84
```

We convert it to “tidy” format by using the `gather()` function from the `tidyverse` package as follows:

```
drinks_smaller_tidy <- drinks_smaller %>%
  gather(key = type, value = servings, -country)
drinks_smaller_tidy
```

```
# A tibble: 12 x 3
  country     type   servings
  <chr>      <chr>    <int>
1 China      beer       79
2 Italy      beer       85
3 Saudi Arabia beer       0
4 USA        beer      249
5 China      spirit     192
6 Italy      spirit      42
7 Saudi Arabia spirit      5
8 USA        spirit     158
9 China      wine        8
10 Italy     wine      237
```

```
11 Saudi Arabia wine          0
12 USA           wine         84
```

We set the arguments to `gather()` as follows:

1. `key` is the name of the column/variable in the new “tidy” frame that contains the column names of the original data frame that you want to tidy. Observe how we set `key = type` and in the resulting `drinks_smaller_tidy` the column `type` contains the three types of alcohol `beer`, `spirit`, and `wine`.
2. `value` is the name of the column/variable in the “tidy” frame that contains the rows and columns of values in the original data frame you want to tidy. Observe how we set `value = servings` and in the resulting `drinks_smaller_tidy` the column `value` contains the $4 \times 3 = 12$ numerical values.
3. The third argument are the columns you either want to or don’t want to tidy. Observe how we set this to `-country` indicating that we don’t want to tidy the `country` variable in `drinks_smaller` and rather only `beer`, `spirit`, and `wine`.

The third argument is a little nuanced, so let’s consider another example. Note the code below is very similar, but now the third argument specifies which columns we’d want to tidy `c(beer, spirit, wine)`, instead of the columns we don’t want to tidy `-country`. Note the use of `c()` to create a vector of the columns in `drinks_smaller` that we’d like to tidy. If you run the code below, you’ll see that the resulting `drinks_smaller_tidy` is the same.

```
drinks_smaller_tidy <- drinks_smaller %>%
  gather(key = type, value = servings, c(beer, spirit, wine))
drinks_smaller_tidy
```

With our `drinks_smaller_tidy` “tidy” format data frame, we can now produce a side-by-side AKA dodged barplot using `geom_col()` and not `geom_bar()`, since we would like to map the `servings` variable to the y-aesthetic of the bars.

```
ggplot(drinks_smaller_tidy,
       aes(x = country, y = servings, fill = type)) +
  geom_col(position = "dodge")
```

Converting “wide” format data to “tidy” format often confuses new R users. The only way to learn to get comfortable with the `gather()` function is with practice, practice, and more practice. For example, see the examples in the

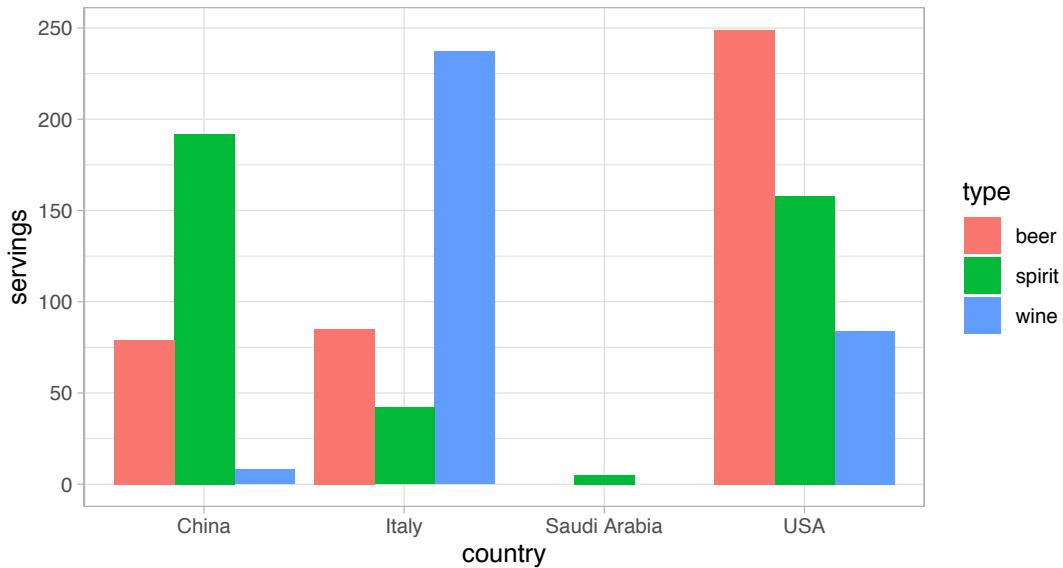


FIGURE 5.4: Comparing alcohol consumption in 4 countries.

bottom of the help file for `gather()` by running `?gather`. We'll show another example of using `gather()` to convert a "wide" formatted data frame to "tidy" format in Section 5.3. For other examples of converting a dataset into "tidy" format, check out the different functions available for data tidying and a case study using data from the World Health Organization in R for Data Science⁷ (Grolmund and Wickham, 2016).

Learning check

(LC5.3) Take a look the `airline_safety` data frame included in the `fivethirtyeight` data package. Run the following:

```
airline_safety
```

After reading the help file by running `?airline_safety`, we see that `airline_safety` is a data frame containing information on different airlines companies' safety records. This data was originally reported on the data journalism website FiveThirtyEight.com in Nate Silver's article "Should Travelers Avoid Flying Airlines That Have Had Crashes in the Past?"⁸. Let's ignore the `incl_reg_subsidiaries` and `avail_seat_km_per_week` variables for simplicity:

⁷<http://r4ds.had.co.nz/tidy-data.html>

⁸<https://fivethirtyeight.com/features/should-travelers-avoid-flying-airlines-that-have-had-crashes-in-the-past/>

```
airline_safety_smaller <- airline_safety %>%
  select(-c(incl_reg_subsidiaries, avail_seat_km_per_week))
airline_safety_smaller
```

```
# A tibble: 56 x 7
  airline incidents_85_99 fatal_accidents~ fatalities_85_99
  <chr>          <int>           <int>           <int>
1 Aer Li~         2              0              0
2 Aerofl~        76             14             128
3 Aeroli~         6              0              0
4 Aerome~         3              1              64
5 Air Ca~         2              0              0
6 Air Fr~        14             4              79
7 Air In~         2              1             329
8 Air Ne~         3              0              0
9 Alaska~         5              0              0
10 Alital~        7              2              50
# ... with 46 more rows, and 3 more variables:
#   incidents_00_14 <int>, fatal_accidents_00_14 <int>,
#   fatalities_00_14 <int>
```

This data frame is not in “tidy” format. How would you convert this data frame to be in “tidy” format, in particular so that it has a variable `incident_type_years` indicating the incident type/year and a variable `count` of the counts?

5.2.3 `nycflights13` package

Recall the `nycflights13` package with data about all domestic flights departing from New York City in 2013 that we introduced in Section 2.4 and used extensively in Chapter 3 on data visualization and Chapter 4 on data wrangling. Let’s revisit the `flights` data frame by running `View(flights)`. We saw that `flights` has a rectangular shape with each of its 336,776 rows corresponding to a flight and each of its 22 columns corresponding to different characteristics/measurements of each flight. This matches exactly with our definition of “tidy” data from above.

1. Each variable forms a column.
2. Each observation forms a row.

But what about the third property of “tidy” data?

3. Each type of observational unit forms a table.
-

Recall that we also saw in Section 2.4.3 that the observational unit for the `flights` data frame is an individual flight. In other words, the rows of the `flights` data frame refer to characteristics/measurements of individual flights. Also included in the `nycflights13` package are other data frames with their rows representing different observational units (Wickham, 2018):

- `airlines`: translation between two letter IATA carrier codes and names (16 in total). i.e. the observational unit is an airline company.
- `planes`: construction information about each of 3,322 planes used. i.e. the observational unit is an aircraft.
- `weather`: hourly meteorological data (about 8705 observations) for each of the three NYC airports. i.e. the observational unit is an hourly measurement.
- `airports`: airport names and locations. i.e. the observational unit is an airport.

The organization of the information into these five data frames follow the third “tidy” data property: observations corresponding to the same observational unit should be saved in the same table i.e. data frame. You could think of this property as the old English expression: “birds of a feather flock together.”

5.3 Case study: Democracy in Guatemala

In this section, we’ll show you another example of how to convert a data frame that isn’t in “tidy” format i.e. “wide” format, to a data frame that is in “tidy” format i.e. “long/narrow” format. We’ll do this using the `gather()` function from the `tidyverse` package again. Furthermore, we’ll make use of some of the `ggplot2` data visualization and `dplyr` data wrangling tools you learned in Chapters 3 and 4.

Let’s use the `dem_score` data frame we imported in Section 5.1, but focus on only data corresponding to Guatemala.

```
guat_dem <- dem_score %>%
  filter(country == "Guatemala")
guat_dem

# A tibble: 1 x 10
  country `1952` `1957` `1962` `1967` `1972` `1977` `1982` 
  <chr>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl> 
1 Guatem~     2      -6      -5      3      1      -3      -7 
# ... with 2 more variables: `1987` <dbl>, `1992` <dbl>
```

Now let’s produce a *time-series plot* showing how the democracy scores have changed over the 40 years from 1952 to 1992 for Guatemala. Recall that we saw time-series plot in Section 3.4 on creating linegraphs using `geom_line()`. Let’s lay out the Grammar of Graphics we saw in Section 3.1.

First we know we need to set `data = guat_dem` and use a `geom_line()` layer, but what is the aesthetic mapping of variables. We’d like to see how the democracy score has changed over the years, so we need to map:

- `year` to the x-position aesthetic and
- `democracy_score` to the y-position aesthetic

Now we are stuck in a predicament, much like with our `drinks_smaller` example in Section 5.2. We see that we have a variable named `country`, but its only value is "Guatemala". We have other variables denoted by different year values. Unfortunately, the `guat_dem` data frame is not “tidy” and hence is not in the appropriate format to apply the Grammar of Graphics and thus we cannot use the `ggplot2` package. We need to take the values of the columns corresponding to years in `guat_dem` and convert them into a new “key” variable called `year`. Furthermore, we’d like to take the democracy scores on the inside of the table and turn them into a new “value” variable called `democracy_score`. Our resulting data frame will thus have three columns: `country`, `year`, and `democracy_score`.

Recall that the `gather()` function in the `tidyr` package can complete this task for us:

```
guat_dem_tidy <- guat_dem %>%
  gather(key = year, value = democracy_score, -country)
guat_dem_tidy
```

```
# A tibble: 9 x 3
  country   year  democracy_score
  <chr>     <chr>        <dbl>
1 Guatem~   1952       -6.00
2 Guatem~   1957       -5.00
3 Guatem~   1962       -5.00
4 Guatem~   1967        3.00
5 Guatem~   1972        1.00
6 Guatem~   1977       -3.00
7 Guatem~   1982       -7.00
8 Guatem~   1987        2.00
9 Guatem~   1992        1.00
```

1 Guatemala 1952	2
2 Guatemala 1957	-6
3 Guatemala 1962	-5
4 Guatemala 1967	3
5 Guatemala 1972	1
6 Guatemala 1977	-3
7 Guatemala 1982	-7
8 Guatemala 1987	3
9 Guatemala 1992	3

We set the arguments to `gather()` as follows:

1. `key` is the name of the column/variable in the new “tidy” frame that contains the column names of the original data frame that you want to tidy. Observe how we set `key = year` and in the resulting `guat_dem_tidy` the column `year` contains the years where the Guatemala’s democracy score were measured.
2. `value` is the name of the column/variable in the “tidy” frame that contains the rows and columns of values in the original data frame you want to tidy. Observe how we set `value = democracy_score` and in the resulting `guat_dem_tidy` the column `democracy_score` contains the $1 \times 9 = 9$ democracy scores.
3. The third argument are the columns you either want to or don’t want to tidy. Observe how we set this to `-country` indicating that we don’t want to tidy the `country` variable in `guat_dem` and rather only 1952 through 1992.

However, observe in the output for `guat_dem_tidy` that the `year` variable is of type `chr` or character. Before we can plot this variable on the x-axis, we need to convert it into a numerical variable using the `as.numeric()` function within the `mutate()` function, which we saw in Section 4.5 on mutating existing variables to create new ones.

```
guat_dem_tidy <- guat_dem_tidy %>%
  mutate(year = as.numeric(year))
```

We can now create the plot to show how the democracy score of Guatemala changed from 1952 to 1992 using a `geom_line()`:

```
ggplot(guat_dem_tidy, aes(x = year, y = democracy_score)) +
  geom_line() +
  labs(x = "Year", y = "Democracy Score")
```



FIGURE 5.5: Democracy scores in Guatemala 1952-1992.

Learning check

(LC5.4) Convert the `dem_score` data frame into a tidy data frame and assign the name of `dem_score_tidy` to the resulting long-formatted data frame.

(LC5.5) Read in the life expectancy data stored at https://moderndive.com/data/le_mess.csv and convert it to a tidy data frame.

5.4 Conclusion

5.4.1 `tidyverse` package

Notice at the beginning of the chapter we loaded the following four packages, which are among the four of the most frequently used R packages for data science:

```
library(dplyr)  
library(ggplot2)
```

```
library(readr)
library(tidyr)
```

There is a much quicker way to load these packages than by individually loading them as we did above: by installing and loading the `tidyverse` package. The `tidyverse` package acts as an “umbrella” package whereby installing/loading it will install/load multiple packages at once for you. So after installing the `tidyverse` package as you would a normal package, running this:

```
library(tidyverse)
```

would be the same as running this:

```
library(ggplot2)
library(dplyr)
library(tidyr)
library(readr)
library(purrr)
library(tibble)
library(stringr)
library(forcats)
```

You’ve seen the first 4 of the these packages: `ggplot2` for data visualization, `dplyr` for data wrangling, `tidyr` for converting data to “tidy” format, and `readr` for importing spreadsheet data into R. The remaining packages (`purrr`, `tibble`, `stringr`, and `forcats`) are left for a more advanced book; check out R for Data Science⁹ to learn about these packages.

The `tidyverse` “umbrella” package gets its name from the fact that all functions in all its constituent packages are designed so that all inputs/argument data frames are in “tidy” format and all output data frames are in “tidy” format as well. This standardization of input and output data frames makes transitions between the various functions in these packages as seamless as possible.

5.4.2 Additional resources

An R script file of all R code used in this chapter is available here¹⁰.

⁹<http://r4ds.had.co.nz/>

¹⁰[scripts/05-tidy.R](#)

If you want to learn more about using the `readr` and `tidyverse` package, we suggest you that you check out RStudio’s “Data Import” cheatsheet. You can access this cheatsheet by going to RStudio’s cheatsheet page¹¹ and searching for “Data Import Cheat Sheet”.

Data Import :: CHEAT SHEET

R's `tidyverse` is built around `tidy` data stored in `tibbles`, which are enhanced data frames.



The front side of this sheet shows how to read text files into R with `readr`.

The reverse side shows how to create tibbles with `tibble` and to layout tidy data with `tidyR`.

OTHER TYPES OF DATA

Try one of the following packages to import other types of files

- `haven` - SPSS, Stata, and SAS files
- `readxl` - excel files (.xls and .xlsx)
- `DBI` - databases
- `jsonlite` - json
- `xmll2` - XML
- `httr` - Web APIs
- `rvest` - HTML (Web Scraping)

Save Data

Save `x`, an R object, to `path`, a file path, as:

```
## Comma delimited file
write_csv(x, path, na = "NA", append = FALSE,
          col_names = lappend)

## File with arbitrary delimiter
write_delim(x, path, delim = ";", na = "NA",
            append = FALSE, col_names = lappend)

## CSV for excel
write_excel_csv(x, path, na = "NA", append =
    FALSE, col_names = lappend)

## String to file
write_file(x, path, append = FALSE)

## String vector to file, one element per line
write_lines(x, path, na = "NA", append = FALSE)

## Object to RDS file
write_rds(x, path, compress = c("none", "gz",
                                "bz2", "xz"))

## Tab delimited files
write_tsv(x, path, na = "NA", append = FALSE,
          col_names = lappend)
```



Read Tabular Data

- These functions share the common arguments:

```
read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(),
      quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000,
      n_max), progress = interactive())
```

Comma Delimited Files
`read_csv("file.csv")`
 To make file.csv run:
`write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")`

Semi-colon Delimited Files
`read_csv2("file.csv")`
`write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")`

Files with Any Delimiter
`read_delim(file.txt", delim = "|")`
`write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")`

Fixed Width Files
`read_fwf("file.fwf", col_positions = c(1, 3, 5))`
`write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")`

Tab Delimited Files
`read_tsv("file.tsv")` Also `read_table()`.
`write_file(x = "t\bt\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")`

USEFUL ARGUMENTS

Example file
`write_file(a,b,c\n1,2,3\n4,5,NA,"file.csv")` **1 2 3** **Skip lines**
`4 5 NA` `read_csv(f, skip = 1)`

No header
`read_csv(f, col_names = FALSE)` **A B C** **Read in a subset**
`1 2 3` `read_csv(f, n_max = 1)`

Provide header
`read_csv(f, col_names = c("x", "y", "z"))` **A B C** **Missing Values**
`NA 2 3` `read_csv(f, na = c("1", ""))`
`4 5 NA`

Read Non-Tabular Data

Read a file into a single string

`read_file(file, locale = default_locale())`

Read each line into its own string

`read_lines(file, skip = 0, n_max = -1L, na = character(),
 locale = default_locale(), progress = interactive())`

Read Apache style log files

`read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive())`

Read a file into a raw vector

`read_file_raw(file)`

Read each line into a raw vector

`read_lines_raw(file, skip = 0, n_max = -1L,
 progress = interactive())`

Data types



`readr` functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:
## col_1: age = col_integer(), age is an integer
## sex = col_character(), sex is a character
## earn = col_double() earn is a double(numeric) character
```

1. Use `problems()` to diagnose problems.
`x <- read_csv("file.csv"); problems(x)`

2. Use a `col_` function to guide parsing.

- `col_guess()` - the default
- `col_character()`
- `col_double(), col_euro_double()`
- `col_datetime(format = "")` Also `col_date(format = "")`, `col_time(format = "")`
- `col_factor(levels, ordered = FALSE)`
- `col_integer()`
- `col_logical()`
- `col_number(), col_numeric()`
- `col_skip()`

`x <- read_csv("file.csv", col_types = cols(
 A = col_double(),
 B = col_logical(),
 C = col_factor()))`

3. Else, read in as character vectors then parse with a `parse_` function.

- `parse_guess()`
- `parse_character()`
- `parse_datetime()` Also `parse_date()` and `parse_time()`
- `parse_double()`
- `parse_factor()`
- `parse_integer()`
- `parse_logical()`
- `parse_number()`

`x$A <- parse_number(x$A)`

FIGURE 5.6: Data Import cheatsheet.

5.4.3 What's to come?

Congratulations! We've completed the “Data Science via the tidyverse” portion of this book! We'll now move to the “data modeling” portion in Chapters 6 and 7, where you'll leverage your data visualization and wrangling skills to model relationships between different variables in data frames. However, we're going to leave the Chapter 11 on “Inference for Regression” until after we've covered statistical inference. Onwards and upwards!

¹¹ <https://www.rstudio.com/resources/cheatsheets/>

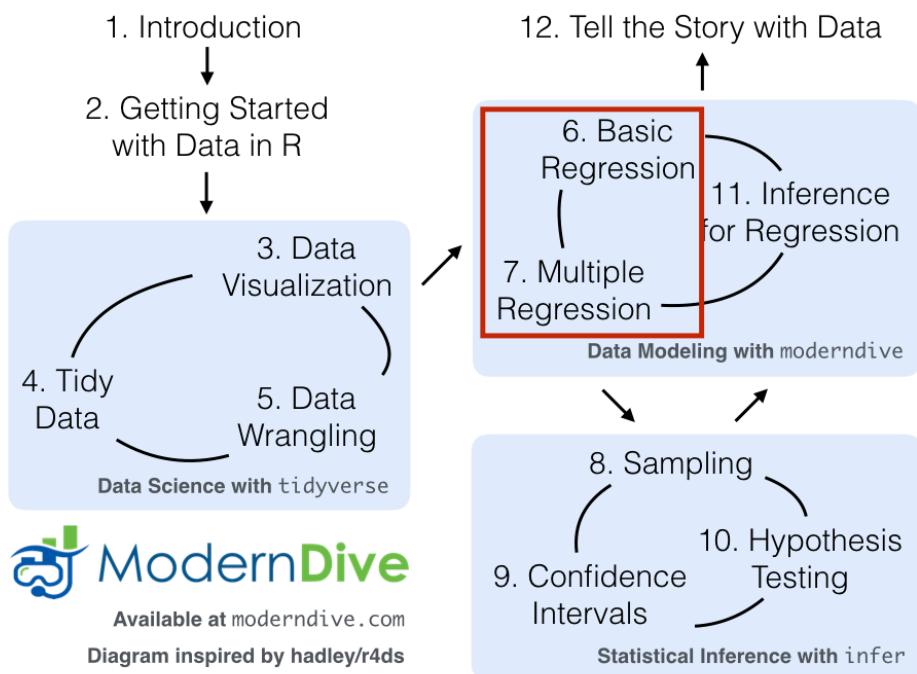


FIGURE 5.7: ModernDive flowchart - On to Part II!

Part II

Data Modeling via moderndive

6

Basic Regression

Now that we are equipped with data visualization skills from Chapter 3, data wrangling skills from Chapter 4, and an understanding of “tidy” data format from Chapter 5, let’s now proceed with data modeling. The fundamental premise of data modeling is to make explicit the relationship between:

- an outcome variable y , also called a dependent variable or response variable and
- an explanatory/predictor variable x , also called an independent variable or covariate.

Another way to state this is using mathematical terminology: we will model the outcome variable y “as a function” of the explanatory/predictor variable x . When we say “function” here, we aren’t referring to a function like the `ggplot()` function, but rather as a mathematical function. Why do we have two different labels, explanatory and predictor, for the variable x ? That’s because even though the two terms are used interchangeably, roughly speaking data modeling serves one of two purposes:

1. **Modeling for explanation:** When you want to explicitly describe and quantify the relationship between an outcome variable y and a set of explanatory variables x , determine the significance of any relationships, and have measures summarizing these relationships.
2. **Modeling for prediction:** When you want to predict an outcome variable y based on the information contained in a set of predictor variables. Unlike modeling for explanation however, you don’t care so much about understanding how all the variables relate and interact with one another, but rather only whether you can make good predictions about y using the information in predictor variables x .

For example, say you are interested in an outcome variable y of whether patients develop lung cancer and information x on their risk factors, such as smoking habits, age, and socioeconomic status. If we are modeling for explanation, we would be interested describing and quantifying the effects of the different risk factors. One reason could be because you want to design an intervention to reduce lung cancer incidence in a population, such as targeting

smokers of a specific age group with advertising for smoking cessation programs. If we are modeling for prediction however, we wouldn't care so much about understanding how all the individual risk factors contribute to lung cancer incidence, but rather only whether we can make good predictions of who will contract lung cancer.

In this book, we'll focus on modeling for explanation and hence refer to x . If you are interested in learning about modeling for prediction, we suggest you check out books and courses on machine learning. Furthermore, while there exists many techniques for modeling, such as tree-based models and neural networks, in this book we'll focus on one particular technique: *linear regression*, one of the most commonly-used and easy-to-understand approaches to modeling.

Linear regression involves a *numerical* outcome variable y and explanatory variables x that are either *numerical* or *categorical*. Furthermore, the relationship between y and x is assumed to be linear, or in other words, a line. However we'll see that what constitutes a "line" will vary depending on the nature of your x explanatory variables. We'll study

- In Chapter 6 on basic regression, we'll only consider models with a single explanatory variable x
 1. In Section 6.1, the explanatory variable will be numerical. This scenario is known as *simple linear regression*.
 2. In Section 6.2, the explanatory variable will be categorical.
- In Chapter 7 on multiple regression, we'll consider models with two explanatory variables x_1 and x_2 :
 1. In Section 7.2, we'll have one numerical and one categorical explanatory variable. In particular, we'll consider two possible models in this case: *interaction* and *parallel slopes* models.
 2. In Section 7.1, we'll have two numerical explanatory variables.
- In Chapter 11 on inference for regression, we'll revisit our regression models and analyze the results using the tools for "statistical inference" you'll develop in Chapters 8, 9, and 10 on sampling, confidence intervals, and hypothesis test/p-values respectively.

Let's now begin with basic regression, which are linear regression models with a single explanatory variable x . We'll also discuss important statistical concepts like the correlation coefficient, that "correlation isn't necessarily causation", and what it means for a line to be "best fitting."

Needed packages

Let's now load all the packages needed for this chapter (this assumes you've already installed them). In this chapter we introduce some new packages:

1. The `tidyverse` “umbrella” package. Recall from our discussion in Section 5.4.1 that loading the `tidyverse` package by running `library(tidyverse)` loads the following commonly used data science packages all at once:
 - `ggplot2` for data visualization
 - `dplyr` for data wrangling
 - `tidyr` for converting data to “tidy” format
 - `readr` for importing spreadsheet data into R
 - As well as the more advanced `purrr`, `tibble`, `stringr`, and `forcats` packages
2. The `moderndive` package of datasets and functions for tidyverse-friendly introductory linear regression.
3. The `skimr` package which provides a simple to use summary function that can be used with pipes and displays nicely in the console.

If needed, read Section 2.3 for information on how to install and load R packages.

```
library(tidyverse)
library(moderndive)
library(skimr)
library(gapminder)
```

6.1 One numerical explanatory variable

Why do some professors and instructors at universities and colleges receive high teaching evaluations from students while others don't? Are there differences in teaching evaluations between instructors of different demographic groups? Could there be an impact due to student biases? These are all questions that are of interest to university/college administrators, as teaching evaluations are among the many criteria considered in determining which instructors and professors get promoted.

Researchers at the University of Texas in Austin, Texas (UT Austin) tried to

answer the following research question: what factors can explain differences in instructor's teaching evaluation scores? To this end, they collected instructor and course information on 463 courses. A full description of the study can be found at [openintro.org¹](https://www.openintro.org/stat/data/?data=evals).

In this section, we'll keep things simple for now and try to explain differences in instructor teaching scores as a function of one numerical variable: the instructor's "beauty score"; we'll describe how this score was computed shortly. Could it be that instructors with higher beauty scores also have higher teaching evaluations? Could it be instead that instructors with higher beauty scores tend to have lower teaching evaluations? Or could it be there is no relationship between beauty score and teaching evaluations? We'll answer these questions by modeling the relationship between teaching scores and "beauty scores" using *simple linear regression* where we have:

1. A numerical outcome variable y , the instructor's teaching score and
2. A single numerical explanatory variable x , the instructor's beauty score.

6.1.1 Exploratory data analysis

The data on the 463 courses at the UT Austin can be found in the `evals` data frame included in the `moderndive` package. However, to keep things simple, let's `select()` only the subset of the variables we'll consider in this chapter, and save this data in a new data frame called `eval_ch6`:

```
evals_ch6 <- evals %>%
  select(ID, score, bty_avg, age)
```

A crucial step before doing any kind of analysis or modeling is performing an *exploratory data analysis*, or EDA for short. Exploratory data analysis gives you a sense of the distributions of the individual variables in your data, whether there are outliers and/or missing values, and most importantly help inform how to build your model. Here are three common steps in an exploratory data analysis.

1. Most crucially: Looking at the raw data values.
2. Computing summary statistics, like means, medians, and interquartile ranges.
3. Creating data visualizations.

¹<https://www.openintro.org/stat/data/?data=evals>

Let's perform the first common step in an exploratory data analysis: looking at the raw data values. Unfortunately, many analysts ignore the first step. Because this step seems so trivial, many analysts often ignore it. However, getting an early sense of what your raw data looks like can often prevent many larger issues down the road. You can do this by using RStudio's spreadsheet viewer or by using the `glimpse()` function as introduced in Section 2.4.3 on exploring data frames:

```
glimpse(evals_ch6)
```

```
Observations: 463
Variables: 4
$ ID      <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ...
$ score   <dbl> 4.7, 4.1, 3.9, 4.8, 4.6, 4.3, 2.8, 4.1, ...
$ bty_avg <dbl> 5.00, 5.00, 5.00, 5.00, 3.00, 3.00, 3.0...
$ age     <int> 36, 36, 36, 36, 59, 59, 51, 51, 40, ...
```

Observe that `Observations: 463` indicates that there are 463 rows/observations in `evals_ch6`, where each row corresponds to one observed course at UT Austin. It is important to note that the *observational unit* are individual courses and not individual instructors. The observational unit is the thing that is being measured or with which the variables provide characteristics for. Since instructors often teach more than one course in an academic year, the same instructor can often appear more than once in the data. Hence there are fewer than 463 unique instructors being represented in `evals_ch6`. We'll revisit this idea in Chapter 11, when we talk about the “independence assumption” for inference for regression.

While a full description of all the variables included in `evals` can be found at [openintro.org²](https://openintro.org/stat/data/?data=evals) and by reading the associated help file by running `?evals` in the Console, let's fully describe the 4 variables we selected in `evals_ch6`:

1. `ID`: An identification variable used to distinguish between the 1 through 463 courses in the dataset.
2. `score`: A numerical variable of the course instructor's average teaching score, where the average is computed from the evaluation scores from all students in that course. Teaching scores of 1 are lowest and 5 are highest. This is the outcome variable y of interest.
3. `bty_avg`: A numerical variable of the course instructor's average “beauty” score, where the average is computed from a separate panel

²<https://www.openintro.org/stat/data/?data=evals>

of 6 students. “Beauty” scores of 1 are lowest and 10 are highest. This is the explanatory variable x of interest.

4. `age`: A numerical variable of the course instructor’s age. This will be another explanatory variable x we’ll study later.

An alternative way to look at the raw data values is by choosing a random sample of the courses, i.e. rows in `evals_ch6` by piping it into the `sample_n()` function from the `dplyr` package. Here we set the `size` argument to be 5, indicating that we want a random sample of 5 rows. We display the results Table 6.1. Note due to the random nature of the sampling, you will likely end up with a different subset of 5 rows.

```
evals_ch6 %>%
  sample_n(size = 5)
```

TABLE 6.1: A random sample of 5 out of the 463 courses at UT Austin

ID	score	bty_avg	age
129	3.7	3.00	62
109	4.7	4.33	46
28	4.8	5.50	62
434	2.8	2.00	62
330	4.0	2.33	64

Now that we’ve looked at the raw values in our `evals_ch6` data frame and obtained a sense of the data, let’s move on to next common step in an exploratory data analysis: computing summary statistics. Let’s start by computing the mean and median of our numerical outcome variable `score` and our numerical explanatory variable `bty_avg` beauty score:

```
evals_ch6 %>%
  summarize(mean_bty_avg = mean(bty_avg), mean_score = mean(score),
            median_bty_avg = median(bty_avg), median_score = median(score))

# A tibble: 1 x 4
  mean_bty_avg mean_score median_bty_avg median_score
  <dbl>        <dbl>        <dbl>        <dbl>
1       4.42      4.17       4.33        4.3
```

However, what if we want other summary statistics as well, such as the standard deviation (a measure of spread), the minimum and maximum values, and various percentiles? Typing out all these summary statistic functions in the

earlier `summarize()` would be long and tedious. Instead, let's use the very convenient `skim()` function from the `skimr` package. This function takes in a data frame, "skims" it, and returns commonly used summary statistics. Let's take our `evals_ch6` data frame, `select()` only the outcome and explanatory variables teaching `score` and `bty_avg`, and pipe it into the `skim()` function:

```
evals_ch6 %>%
  select(score, bty_avg) %>%
  skim()
```

```
Skim summary statistics
n obs: 463
n variables: 2

— Variable type:numeric —
variable missing complete    n  mean   sd   p0   p25   p50   p75   p100
bty_avg      0     463 463 4.42 1.53 1.67 3.17 4.33 5.5 8.17
score        0     463 463 4.17 0.54 2.3  3.8  4.3  4.6  5
```

(Note that for formatting purposes, the inline histogram that is usually printed with `skim()` has been removed.)

For our two numerical variables teaching `score` and beauty score `bty_avg` it returns:

- `missing`: the number of missing values
- `complete`: the number of non-missing or complete values
- `n`: the total number of values
- `mean`: the mean AKA average
- `sd`: the standard deviation
- `p0`: the 0th percentile: the value at which 0% of observations are smaller than it AKA the *minimum* value
- `p25`: the 25th percentile: the value at which 25% of observations are smaller than it AKA the *1st quartile*
- `p50`: the 50th percentile: the value at which 50% of observations are smaller than it AKA the *2nd quartile* and more commonly the *median*
- `p75`: the 75th percentile: the value at which 75% of observations are smaller than it AKA the *3rd quartile*
- `p100`: the 100th percentile: the value at which 100% of observations are smaller than it AKA the *maximum* value

Looking at the earlier output we get an idea of how the values of both variables distribute. For example, the mean teaching score was 4.17 out of 5 whereas the mean beauty score was 4.42 out of 10. Furthermore, the middle 50% of teaching

scores were between 3.80 and 4.6 (the first and third quartiles) whereas the middle 50% of beauty scores were between 3.17 and 5.5 out of 10.

However, the `skim()` function only returns what are known as *univariate* summary statistics: functions that take a single variable and return some summary of that variable. However, there also exist *bivariate* summary statistics: functions that take in two variables and return some summary of those two variables. In particular, when the two variables are numerical we can compute the *correlation coefficient*. Generally speaking, *coefficients* are quantitative expressions of a specific property of a phenomenon. A *correlation coefficient* is a quantitative expression of the *strength of the linear relationship between two numerical variables* whose value range between -1 and 1 where:

- -1 indicates a perfect *negative relationship*: As the value of one variable goes up, the value of the other variable tends to go down.
- 0 indicates no relationship: The values of both variables go up/down independently of each other.
- +1 indicates a perfect *positive relationship*: As the value of one variable goes up, the value of the other variable tends to go up as well.

Figure 6.1 gives examples of 9 different correlation coefficient values for hypothetical numerical variables x and y . For example, observe that for a correlation coefficient of -0.75 there is still a negative linear relationship between x and y , it is not as strong as the negative linear relationship between x and y when the correlation coefficient is -0.9 or -1.

The correlation coefficient can be computed using the `get_correlation()` function in the `moderndive` package, where in this case the inputs to the function are the two numerical variables from which we want to calculate the correlation coefficient. We place the name of the response variable on the left hand side of the ~ and the explanatory variable on the right hand side of the “tilde” in R’s formula notation. We will use this same “formula” syntax with regression later in this chapter.

```
evals_ch6 %>%
  get_correlation(formula = score ~ bty_avg)
```

```
# A tibble: 1 × 1
  correlation
  <dbl>
1 0.187
```

An alternative way to compute the correlation coefficient is to use the `cor()` function within a `summarize()`:

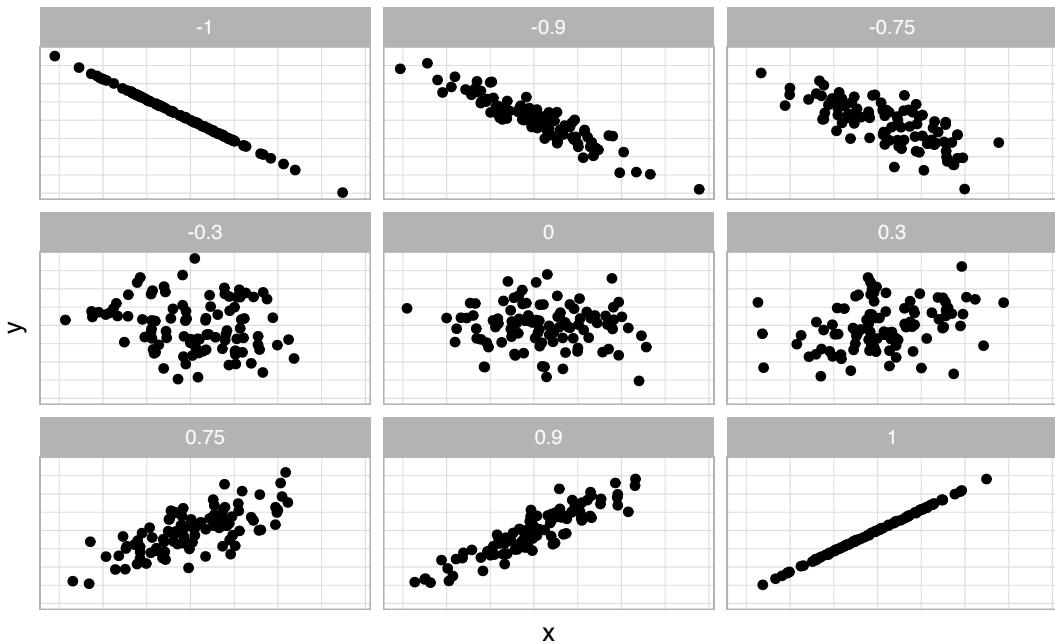


FIGURE 6.1: Different correlation coefficients.

```
evals_ch6 %>%
  summarize(correlation = cor(score, bty_avg))
```

```
# A tibble: 1 × 1
  correlation
  <dbl>
1 0.187
```

In our case, the correlation coefficient of 0.187 indicates that the relationship between teaching evaluation score and beauty average is “weakly positive.” There is a certain amount of subjectivity in interpreting correlation coefficients, especially those that aren’t close to the extreme values of -1, 0, and 1. To develop intuition in interpreting correlation coefficients see play the “Guess the Correlation” 1980’s style video game in Subsection 6.4.1.

Let’s now perform the last of the three common steps in an exploratory data analysis: creating data visualizations. Since both the `score` and `bty_avg` variables are numerical, a scatterplot is an appropriate graph to visualize this data. Let’s do this using `geom_point()` and display the result in Figure 6.2. Furthermore, let’s highlight the 6 points in the top right of the visualization in orange.

```
ggplot(evals_ch6, aes(x = bty_avg, y = score)) +
  geom_point() +
  labs(x = "Beauty Score", y = "Teaching Score",
       title = "Scatterplot of relationship of teaching and beauty scores")
```

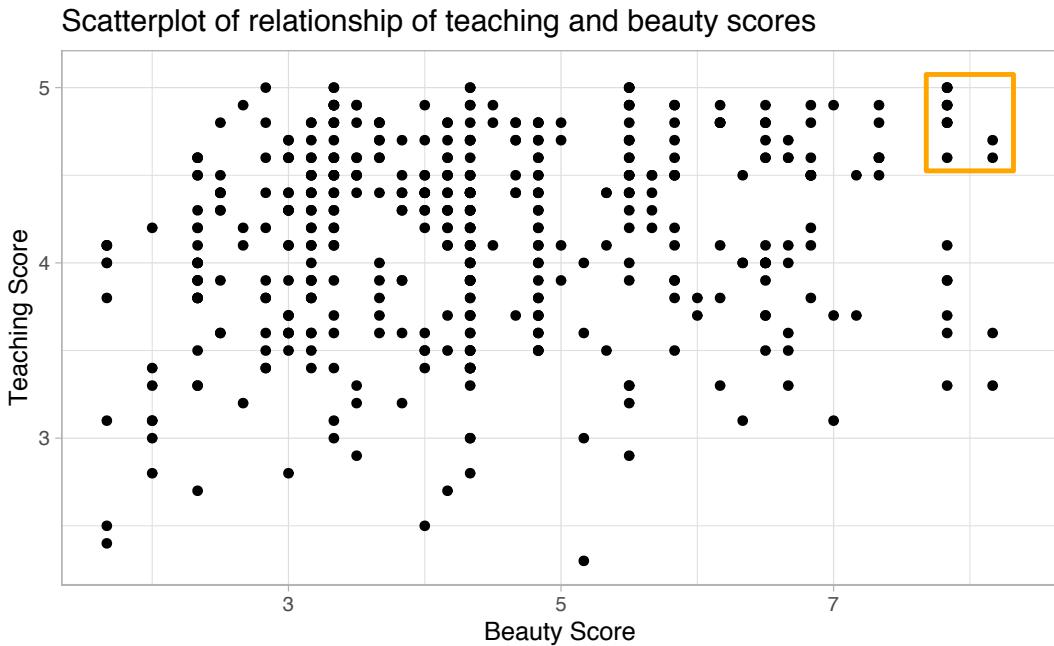


FIGURE 6.2: Instructor evaluation scores at UT Austin.

Observe the following:

1. Most “beauty” scores lie between 2 and 8.
2. Most teaching scores lie between 3 and 5.
3. While opinions may vary, it is our opinion that the relationship between teaching score and beauty score appears weakly positive. This is consistent with our earlier computed correlation coefficient of 0.187.

Furthermore, there appear to be 6 points in the top-right of this plot highlighted in the orange box. However, this is not the case as this plot suffers from *overplotting*. Recall from Subsection 3.3.2 that overplotting occurs when several points are stacked directly on top of each other, thereby obscuring their number. So while it may appear that there are only 6 points in the orange box, there are actually more. This fact is only apparent when using `geom_jitter()` in place of `geom_point()`. We display the resulting plot in Figure 6.3 along with the same orange box as in Figure 6.3.

```
ggplot(evals_ch6, aes(x = bty_avg, y = score)) +
  geom_jitter() +
  labs(x = "Beauty Score", y = "Teaching Score",
       title = "Scatterplot of relationship of teaching and beauty scores")
```

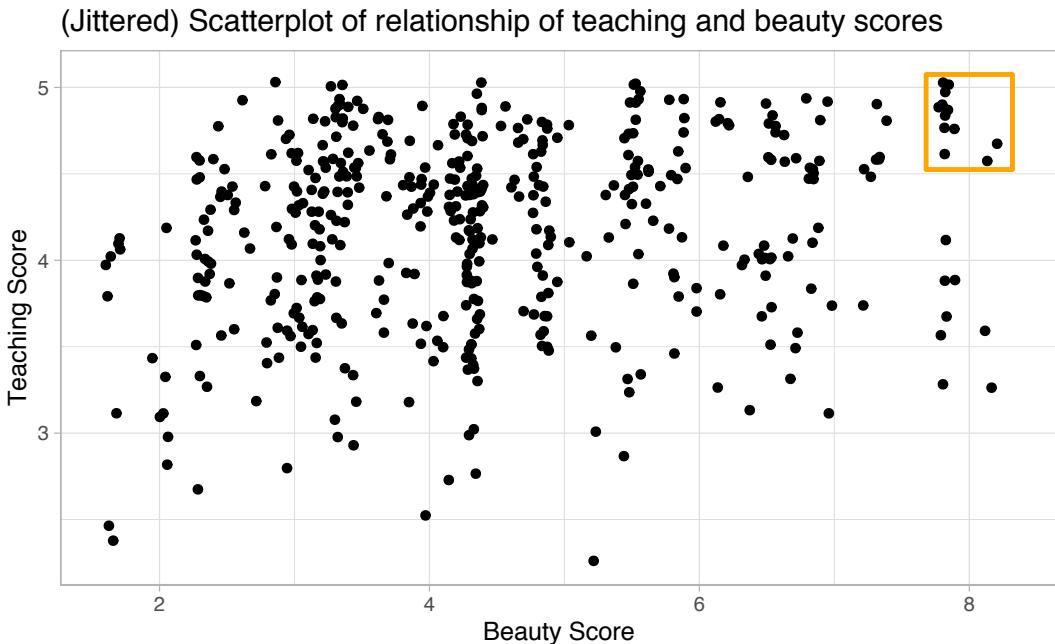


FIGURE 6.3: Instructor evaluation scores at UT Austin.

It is now apparent that there are 12 points in the area highlighted in orange and not 6 as originally suggested in Figure 6.2. Recall from Section 3.3.2 on overplotting that jittering adds a little random “nudge” to each of the points to break up these ties. Furthermore, jittering is strictly a visualization tool; it does not alter the original values in the data frame `evals_ch6`. To keep things simple going forward however, we’ll only present regular scatterplots rather than their jittered counterparts.

Let’s build on theunjittered scatterplot in Figure 6.2 by adding a “best-fitting” line; of all possible lines we can draw on this scatterplot, its the line that “best” fits through the cloud of points. We do this by adding a new `geom_smooth(method = "lm", se = FALSE)` layer to the `ggplot()` code that created the scatterplot in Figure 6.2. The `method = lm` argument sets the line to be a “linear model” while the `se = FALSE` argument suppresses “standard error” uncertainty bars.

```
ggplot(evals_ch6, aes(x = bty_avg, y = score)) +
  geom_point() +
  labs(x = "Beauty Score", y = "Teaching Score",
       title = "Relationship between teaching and beauty scores") +
  geom_smooth(method = "lm", se = FALSE)
```

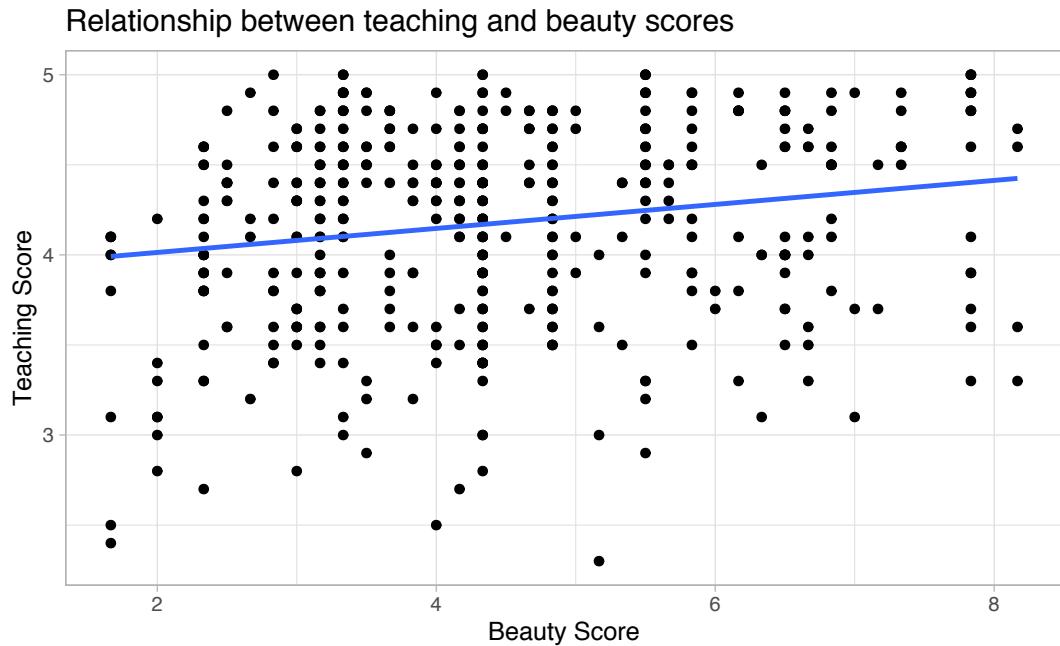


FIGURE 6.4: Regression line.

The blue line in the resulting Figure 6.4 is called a “regression line”. The regression line is a visual summary of the relationship between two numerical variables, in our case the outcome variable `score` and the explanatory variable `bty_avg`. The positive slope of the blue line is consistent with our earlier observed correlation coefficient of 0.187 suggesting that there is a positive relationship between these two variables: as instructors have higher beauty scores so also do they have receive higher teaching evaluations. We’ll see later however that while the correlation coefficient and the slope of a regression line always have the same sign (positive or negative), they do not necessarily have the same value.

Furthermore, a regression line is “best” fitting in that it minimizes some mathematical criteria. We present this mathematical criteria in Subsection 6.3.2, but we suggest you read this subsection only after reading the rest of this section on regression with one numerical explanatory variable.

Learning check

(LC6.1) Conduct a new exploratory data analysis with the same outcome variable y being `score` but with `age` as the new explanatory variable x . Remember, this involves three things:

- a) Looking at the raw data values.
- b) Computing summary statistics.
- c) Creating data visualizations.

What can you say about the relationship between age and teaching scores based on this exploration?

6.1.2 Simple linear regression

You may recall from secondary school / high school algebra, the equation of a line is $y = a + b \cdot x$ which is defined by two coefficients a and b (recall from earlier that coefficients are “quantitative expressions of a specific property of a phenomenon”): the intercept coefficient a i.e. the value of y when $x = 0$ and the slope coefficient b for x i.e. the increase in y for every increase of one in x .

However, when defining a regression line like the blue regression line in Figure 6.4, we use slightly different notation: the equation of the regression line is $\hat{y} = b_0 + b_1 \cdot x$ where the intercept coefficient is b_0 i.e. the value of \hat{y} when $x = 0$. The slope coefficient is b_1 for x i.e. the increase in \hat{y} for every increase of one in x . Why do we put a “hat” on top of the y ? It’s a form of notation commonly used in regression to indicate that we have a “fitted value”, or the value of y on the regression line for a given x value; we’ll discuss this more in the upcoming Subsection 6.1.3.

We know that the blue regression line in Figure 6.4 has a positive slope b_1 corresponding to our explanatory x variable `bty_avg`. Why? Because as instructors have higher `bty_avg` scores, so also do they tend to have higher teaching evaluation `scores`. However, what is the specific numerical value of the slope b_1 ? What about the intercept b_0 ? Let’s compute these two values by hand, but rather let’s use a computer!

We can obtain the intercept b_0 and slope b_1 for `btg_avg` by outputting a *linear regression table*. This is done in two steps:

1. We first “fit” the linear regression model using the `lm()` function and save it in `score_model`.
2. We get the regression table by applying the `get_regression_table()` from the `moderndive` package to `score_model`.

```
# Fit regression model:
score_model <- lm(score ~ bty_avg, data = evals_ch6)
# Get regression table:
get_regression_table(score_model)
```

TABLE 6.2: Linear regression table

term	estimate	std_error	statistic	p_value	lower_ci	upper_ci
intercept	3.880	0.076	50.96	0	3.731	4.030
bty_avg	0.067	0.016	4.09	0	0.035	0.099

Let’s first focus on interpreting the regression table output in Table 6.2 and then later we’ll revisit the code that produced it. In the `estimate` column of Table 6.2 are the intercept $b_0 = 3.88$ and the slope $b_1 = 0.067$ for `bty_avg` and thus the equation of the blue regression line in Figure 6.4 follows (note that the \cdot symbol is equivalent to the \times “multiply by” mathematical symbol; we use the \cdot symbol in this book as it is a little cleaner):

$$\begin{aligned}\hat{y} &= b_0 + b_1 \cdot x \\ \widehat{\text{score}} &= b_0 + b_{\text{bty_avg}} \cdot \text{bty_avg} \\ &= 3.880 + 0.067 \cdot \text{bty_avg}\end{aligned}$$

The intercept $b_0 = 3.8803$ is the value average teaching score $\hat{y} = \widehat{\text{score}}$ for those courses where the instructor had a beauty score `bty_avg` of 0. In other words, it’s where the line intersects the y axis when $x = 0$. Note however that while the intercept of the regression line has a mathematical interpretation, it has no *practical* interpretation since observing a `bty_avg` of 0 is impossible; it is the average of six panelists’ beauty score ranging from 1 to 10. Furthermore, looking at the scatterplot with the regression line in Figure 6.4, no instructors had a beauty score anywhere near 0.

Of greater interest is the slope $b_1 = b_{\text{bty_avg}}$ of $+0.067$, as this summarizes the relationship between teaching score and beauty score. Note that the sign is positive suggesting a positive relationship between these two variables, meaning teachers with higher beauty scores also tend to have higher teaching scores. Recall from earlier that the correlation coefficient is 0.187: they

both have the same positive sign, but have a different value. Recall further that the correlation's interpretation is the "strength of linear association". The slope's interpretation is a little different:

For every increase of 1 unit in `bty_avg`, there is an *associated* increase of, *on average*, 0.0666 units of `score`.

We only state that there is an *associated* increase and not necessarily a *causal* increase. For example, perhaps it's not that higher beauty scores directly cause higher teaching scores per se. Instead it could be that individuals from wealthier backgrounds tend to have stronger educational backgrounds and hence have higher teaching scores, but that these wealthy individuals also have higher beauty scores. In other words, just because two variables are strongly associated doesn't mean that one necessarily causes the other. This is summed up in the often quoted phrase "correlation is not necessarily causation." We discuss this idea further in Subsection 6.3.1.

Furthermore, we say that this associated increase is *on average* 0.067 units of teaching `score` because you might have two instructors whose `bty_avg` score differ by 1 unit, but their difference in teaching scores isn't necessarily 0.067. What the slope of 0.067 is across all courses, the *average* difference in teaching score between two instructors whose beauty scores differ by one is 0.067.

Now that we've learned how to compute the equation for the blue regression line in Figure 6.4 using the values in the `estimate` column of Table 6.2 and how to interpret the resulting the intercept and slope, let's revisit the code that generated this table:

```
# Fit regression model:  
score_model <- lm(score ~ bty_avg, data = evals_ch6)  
# Get regression table:  
get_regression_table(score_model)
```

First, we "fit" the linear regression model to the `data` using the `lm()` function and save this to `score_model`. When we say "fit", we mean "find the best fitting line to this data." `lm()` stands for "linear model" and is used as follows: `lm(y ~ x, data = data_frame_name)` where:

- `y` is the outcome variable, followed by a tilde (~); this is likely the key to the left of the "1" key on your keyboard. In our case, `y` is set to `score`.

- x is the explanatory variable. In our case, x is set to `bty_avg`.
- The combination of $y \sim x$ is called a *model formula* (note the order of y and x). In our case, the model formula is `score ~ bty_avg`. We saw such model formulas earlier as well when we computed the correlation coefficient using the `get_correlation()` function in Subsection 6.1.1.
- `data_frame_name` is the name of the data frame that contains the variables y and x . In our case, `data_frame_name` is the `evals_ch6` data frame.

Second, we take the saved model in `score_model` and apply the `get_regression_table()` function from the `moderndive` package to it to obtain the regression table in Table 6.2. This function is an example of what's known as a *wrapper function* in computer programming, which takes other pre-existing functions and "wraps" them into a single function that hides its inner workings. This concept is illustrated in Figure 6.5.

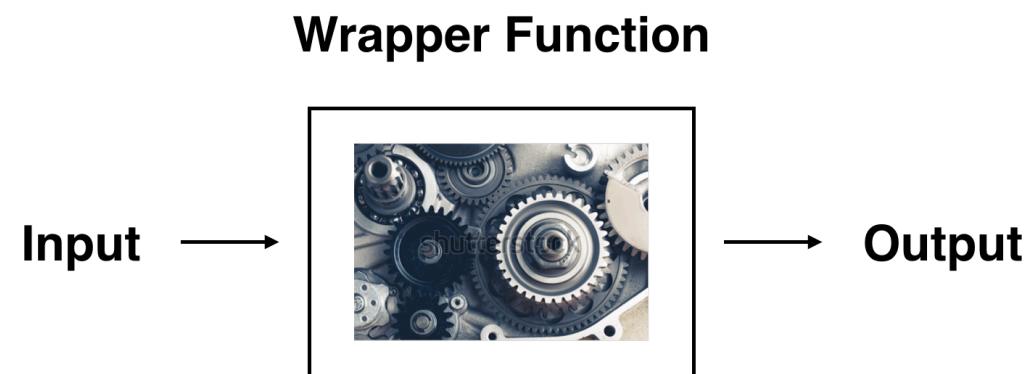


FIGURE 6.5: The concept of a wrapper function.

So all you need to worry about is the what the inputs look like and what the outputs look like; you leave all the other details "under the hood of the car." In our regression modeling example, the `get_regression_table()` function takes as input a saved `lm()` linear regression and returns as output a data frame of the regression table with information on the intercept and slope of the regression line. If you're interested in learning more about the `get_regression_table()` function's design and inner-workings, check out Subsection 6.3.3.

Lastly, you might be wondering what remaining 5 columns in Table 6.2 are: `std_error`, `statistic`, `p_value`, `lower_ci` and `upper_ci`? They are the "standard error", "test statistic", "p-value", "lower 95% confidence interval bound", and "upper 95% confidence interval bound." They tell us about both the *statistical significance* and *practical significance* of our results. You can think of this loosely as the "meaningfulness" of our results from a statistical perspective.

We are going to put aside these ideas for now and revisit them in Chapter 11 on (statistical) inference for regression, after we've had a chance to cover:

- Standard errors in Chapter 8
- Confidence intervals in Chapter 9
- Hypothesis testing and p-values in Chapter 10

Learning check

(LC6.2) Fit a new simple linear regression using `lm(score ~ age, data = evals_ch6)` where `age` is the new explanatory variable x . Get information about the “best-fitting” line from the regression table by applying the `get_regression_table()` function. How do the regression results match up with the results from your earlier exploratory data analysis?

6.1.3 Observed/fitted values and residuals

We just saw how to get the value of the intercept and the slope of a regression line from the `estimate` column of a regression table generated by `get_regression_table()`. Now instead say we want information on individual observations. For example, let's focus on 21st of the 463 courses in the `evals_ch6` data frame in Table 6.3:

TABLE 6.3: Data for the 21st course out of 463

ID	score	bty_avg	age
21	4.9	7.33	31

What is the value \hat{y} on the blue line regression line corresponding to this instructor's `bty_avg` beauty score of 7.333? In Figure 6.6 we mark three values corresponding to the instructor for this 21st course:

- Red circle: The *observed value* $y = 4.9$ is this course's instructor's actual teaching score.
- Red square: The *fitted value* \hat{y} is value on the regression line for $x = \text{bty_avg} = 7.333$. This value is computed using the intercept and slope in the previous regression table:

$$\hat{y} = b_0 + b_1 \cdot x = 3.88 + 0.067 \cdot 7.333 = 4.369$$

- Blue arrow: The length of this arrow is the *residual* and is computed by subtracting the fitted value \hat{y} from the observed value y . The residual can be thought of as the error or “lack of fit”. In the case of this course’s instructor, it is $y - \hat{y} = 4.9 - 4.369 = 0.531$.

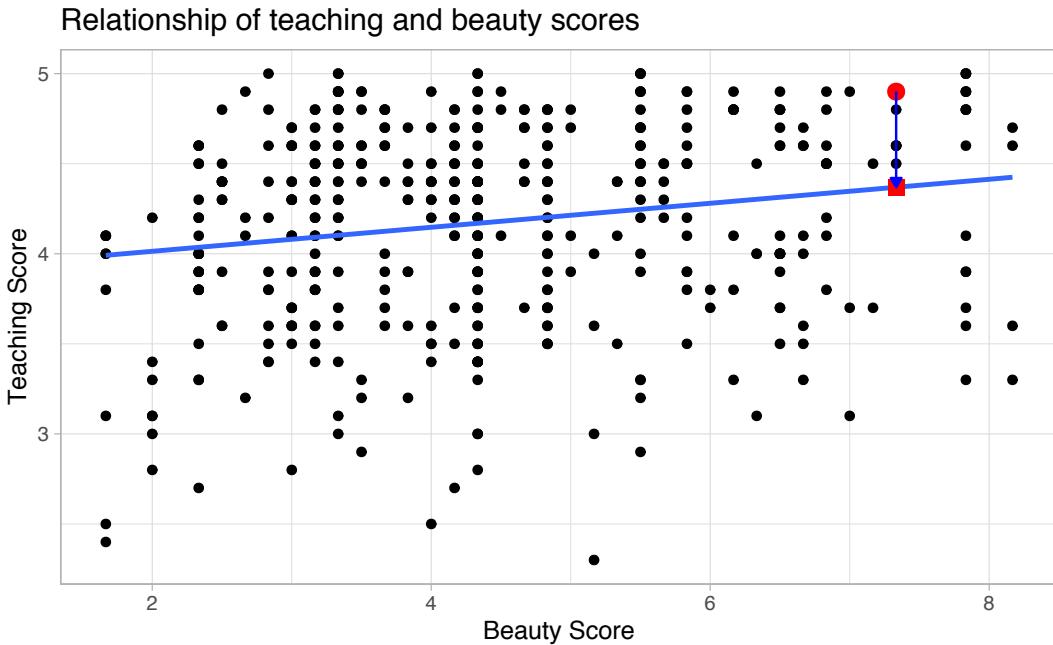


FIGURE 6.6: Example of observed value, fitted value, and residual.

Now say we want to compute both the

1. the fitted value $\hat{y} = b_0 + b_1 \cdot x$ and
2. the residual $y - \hat{y}$

not only for the instructor of the 21st course, but for all 463 courses in the study? Recall that each course corresponds to one of the 463 rows in the `evals_ch6` data frame and also one of the 463 points in the regression plot in Figure 6.6.

We could repeat the previous calculations we performed by hand 463 times, but that would be tedious and time consuming. Instead, let’s use the computer using the `get_regression_points()` function included in the `moderndive` package. Just like the `get_regression_table()` function, the `get_regression_points()` function is a “wrapper” function; however it returns a different output. Let’s apply the `get_regression_points()` function to `score_model`, which is where we saved our `lm()` model in the previous section. In Table 6.4 we present only the results of the 21st through 24th courses for brevity’s sake.

TABLE 6.4: Regression points (for only the 21st through 24th courses)

ID	score	bty_avg	score_hat	residual
21	4.9	7.33	4.37	0.531
22	4.6	7.33	4.37	0.231
23	4.5	7.33	4.37	0.131
24	4.4	5.50	4.25	0.153

```
regression_points <- get_regression_points(score_model)
regression_points
```

Let's inspect the individual columns and match them with the elements of Figure 6.6:

- The `score` column represents the observed outcome variable y i.e. the y -position of the 463 black points.
- The `bty_avg` column represents the values of the explanatory variable x i.e. the x -position of the 463 black points.
- The `score_hat` column represents the fitted values \hat{y} i.e. the corresponding value on the blue regression line for the 463 x values.
- The `residual` column represents the residuals $y - \hat{y}$ i.e. the 463 vertical distances between the 463 black points and the blue regression line.

Just as we did for the instructor of the 21st course in the `evals_ch6` dataset (in the first row of the table above), let's repeat the calculations for the instructor of the 24th course (in the fourth row of Table 6.4 above):

- `score = 4.4` is the observed teaching `score` y for this course's instructor.
- `bty_avg = 5.50` is the value of the explanatory variable `bty_avg` x for this course's instructor.
- `score_hat = 4.25 = 3.88 + 0.067 · 5.50` is the fitted value \hat{y} on the blue regression line for this course's instructor.
- `residual = 0.153 = 4.4 - 4.25` is the value of the residual for this instructor. In other words, the model was off by 0.153 teaching score units for this course's instructor.

At this point we suggest you read Section 6.3.2 in which we define what we mean by “best” for “best-fitting” regression lines: it is the line that minimizes the *sum of squared residuals*.

Learning check

(LC6.3) Generate a data frame of the residuals of the model where you used age as the explanatory x variable.

6.2 One categorical explanatory variable

It's an unfortunate truth that life expectancy is not the same across all countries in the world. International development agencies are very interested in studying these differences in life expectancy in the hopes of identifying where governments should allocate resources to address this problem. In this section, we'll explore differences in life expectancy in two ways:

1. Differences between continents: Are there significant differences in average life expectancy between the five continents of the world: Africa, the Americas, Asia, Europe, and Oceania?
2. Differences within continents: How does life expectancy vary within the world's five continents? For example, is the spread of life expectancy among the countries of Africa larger than the spread of life expectancy among the countries of Asia?

To answer such questions, we'll use the `gapminder` data frame included in the `gapminder` package. This dataset has international development statistics such as life expectancy, GDP per capita, and population for 142 countries for 5-year intervals between 1952 and 2007. Recall we visualized this data in Figure 3.1 in Subsection 3.1.2 on the “Grammar of Graphics”.

We'll use this data for basic linear regression again but now using an explanatory variable x that is categorical, as opposed to the numerical explanatory variable model we saw in Section 6.1 on instructor teaching and beauty scores. In this section, we'll model the relationship between

1. A numerical outcome variable y , a country's life expectancy and
2. A single categorical explanatory variable x , the continent the country is a part of.

When the explanatory variable x is categorical, the concept of a “best-fitting” regression line is a little different than the one we saw previously in Section 6.1 where the explanatory variable x was numerical. We’ll study these differences shortly in Subsection 6.2.2, but first we conduct our exploratory data analysis.

6.2.1 Exploratory data analysis

The data on the 142 countries can be found in the `gapminder` data frame included in the `gapminder` package. However, to keep things simple, let’s `filter()` for only observations/rows corresponding to the year 2007, `select()` only the subset of the variables we’ll consider in this chapter, and save this data in a new data frame called `gapminder2007`:

```
library(gapminder)
gapminder2007 <- gapminder %>%
  filter(year == 2007) %>%
  select(country, lifeExp, continent, gdpPercap)
```

Recall from Section 6.1.1 that there are three common steps in an exploratory data analysis:

1. Most crucially: Looking at the raw data values.
2. Computing summary statistics, like means, medians, and interquartile ranges.
3. Creating data visualizations.

Let’s perform the first common step in an exploratory data analysis: looking at the raw data values. You can do this by using RStudio’s spreadsheet viewer or by using the `glimpse()` command as introduced in Section 2.4.3 on exploring data frames:

```
glimpse(gapminder2007)
```

```
Observations: 142
Variables: 4
$ country    <fct> Afghanistan, Albania, Algeria, Angola...
$ lifeExp    <dbl> 43.8, 76.4, 72.3, 42.7, 75.3, 81.2, 7...
$ continent  <fct> Asia, Europe, Africa, Africa, America...
$ gdpPercap   <dbl> 975, 5937, 6223, 4797, 12779, 34435, ...
```

Observe that `Observations: 142` indicates that there are 142 rows/observations

in `gapminder2007`, where each row corresponds to one country. In other words, the *observational unit* are individual countries. Furthermore, observe that the variable `continent` is of type `<fct>`, which stands for “factor,” which is R’s way of encoding categorical variables.

While a full description of all the variables included in `gapminder` can be found by reading the associated help file by running `?gapminder` in the Console, let’s fully describe the 4 variables we selected in `gapminder2007`:

1. `country`: An identification variable used to distinguish the 142 countries in the dataset.
2. `lifeExp`: A numerical variable of that country’s life expectancy at birth. This is the outcome variable y of interest.
3. `continent`: A categorical variable with 5 levels i.e. possible categories: Africa, Asia, Americas, Europe, and Oceania. This is the explanatory variable x of interest.
4. `gdpPercap`: A numerical variable of that country’s GDP per capita in US inflation-adjusted dollars that we’ll use as another outcome variable y in the Learning Check at the end of this section.

Furthermore, let’s look at a random sample of 5 out of the 142 countries in Table 6.5. Note due to the random nature of the sampling, you will likely end up with a different subset of 5 rows.

```
gapminder2007 %>%
  sample_n(size = 5)
```

TABLE 6.5: Random sample of 5 out of 142 countries

country	lifeExp	continent	gdpPercap
Togo	58.4	Africa	883
Sao Tome and Principe	65.5	Africa	1598
Congo, Dem. Rep.	46.5	Africa	278
Lesotho	42.6	Africa	1569
Bulgaria	73.0	Europe	10681

Now that we’ve looked at the raw values in our `gapminder2007` data frame and obtained a sense of the data, let’s move on to next common step in an exploratory data analysis: computing summary statistics. Let’s once again apply the `skim()` function from the `skimr` package. Recall from our previous EDA that this function takes in a data frame, “skims” it, and returns commonly used summary statistics. Let’s take our `gapminder2007` data frame, `select()` only the

outcome and explanatory variables `lifeExp` and `continent`, and pipe it into the `skim()` function:

```
gapminder2007 %>%
  select(lifeExp, continent) %>%
  skim()

Skim summary statistics
n obs: 142
n variables: 2

— Variable type:factor —————
variable missing complete n n_unique top_counts ordered
continent      0     142 142      5 Afr: 52, Asi: 33, Eur: 30, Ame: 25 FALSE

— Variable type:numeric —————
variable missing complete n mean sd p0 p25 p50 p75 p100
lifeExp        0     142 142 67.01 12.07 39.61 57.16 71.94 76.41 82.6
```

The `skim()` output now reports summaries for categorical variables (`variable type:factor`) separately from the numerical variables (`variable type:numeric`). For the categorical variable `continent` it now reports:

- `missing`, `complete`, `n` which are the number of missing, complete, and total number of values as before.
- `n_unique`: The number of unique levels to this variable, corresponding to Africa, Asia, Americas, Europe, and Oceania.
- `top_counts`: In this case the top four counts: `Africa` has 52 corresponding to its 52 countries, `Asia` has 33, `Europe` has 30, and `Americas` has 25. Not displayed is `Oceania` with 2 countries.
- `ordered`: This tells us whether the categorical variable is “ordinal”: whether there is encoded hierarchy (like low, medium, high). In this case, it is not ordered.

Turning our attention to the summary statistics of the numerical variable `lifeExp`, we observe that the global median life expectancy is 71.94, or in other words half of the world’s countries (71 countries) will have a life expectancy less than 71.94. The mean life expectancy of 67.01 is lower however. Why is the mean life expectancy lower than the median?

We can answer this question via the last of the three common steps in an exploratory data analysis: creating data visualizations. Let’s visualize the distribution of our outcome variable $y = \text{lifeExp}$ in Figure 6.7.

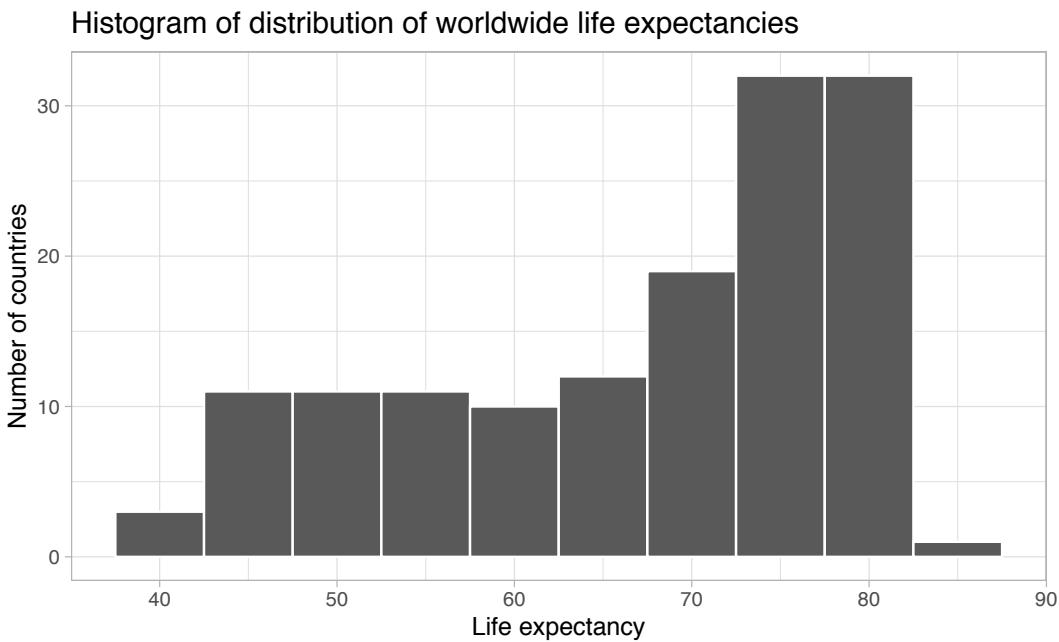


FIGURE 6.7: Histogram of Life Expectancy in 2007.

We see that this data is *left-skewed*, also known as *negatively skewed*: there are a few countries with very low life expectancy that are bringing down the mean life expectancy. However, the median is less sensitive to the effects of such outliers, hence the median is greater than the mean in this case.

Remember however, that we want to compare life expectancies both between continents and within continents. In other words, our visualizations need to incorporate some notion of the variable `continent`. We can do this easily with a faceted histogram. Recall from Section 3.6 that facets allow us to split a visualization by the different values of another variable. We display the resulting visualization in Figure 6.8 by adding a `facet_wrap(~ continent, nrow = 2)` layer.

```
ggplot(gapminder2007, aes(x = lifeExp)) +
  geom_histogram(binwidth = 5, color = "white") +
  labs(x = "Life expectancy", y = "Number of countries",
       title = "Histogram of distribution of worldwide life expectancies") +
  facet_wrap(~ continent, nrow = 2)
```

We observe that unfortunately the distribution of African life expectancies is much lower than the other continents while in Europe life expectancies tend to be higher and do not vary as much. Both Asia and Africa have the most variation in life expectancies. There is the least variation in Oceania, but this

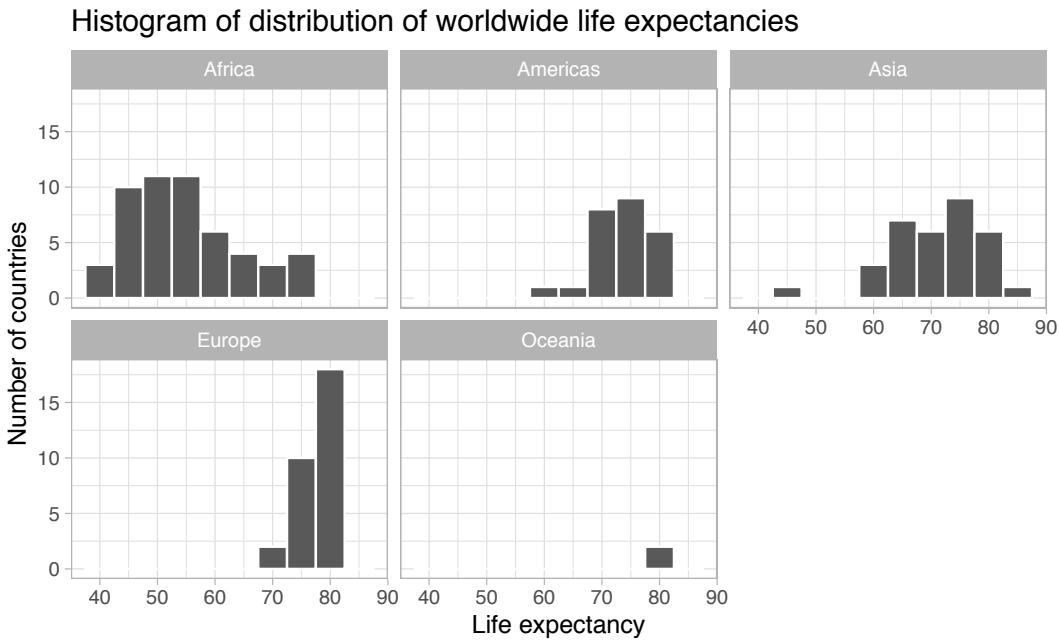


FIGURE 6.8: Life expectancy in 2007.

greatly influenced by the fact that there are only two countries in Oceania: Australia and New Zealand.

Recall than an alternative visualization of the distribution of a numerical variable split by a categorical variable is by using a side-by-side boxplot via a `geom_boxplot()`; we map the categorical variable `continent` to the *x*-axis and the different life expectancies within each continent on the *y*-axis.

```
ggplot(gapminder2007, aes(x = continent, y = lifeExp)) +
  geom_boxplot() +
  labs(x = "Continent", y = "Life expectancy (years)",
       title = "Life expectancy by continent")
```

Some people prefer comparing the distributions of a numerical variable between different levels of a categorical variable using a boxplot instead of a faceted histogram as we can make quick comparisons with imaginary horizontal lines. For example, observe in Figure 6.9 that Oceania clearly tends to have the highest life expectancies given that we could draw an imaginary horizontal line at $y = 80$. Furthermore, as we observed with in faceted histogram in Figure 6.8, Africa and Asia have the biggest variation in life expectancy as evidenced by their large interquartile ranges i.e. the height of the boxes.

It's important to remember however that the solid lines in the middle of the

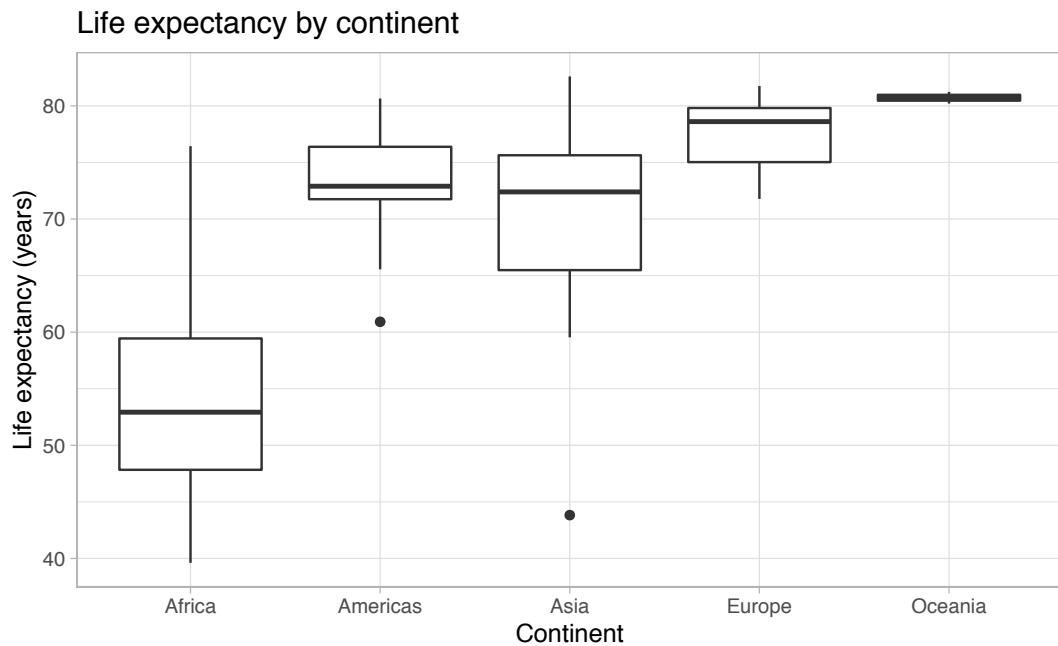


FIGURE 6.9: Life expectancy in 2007.

TABLE 6.6: Life expectancy by continent

continent	median	mean
Africa	52.9	54.8
Americas	72.9	73.6
Asia	72.4	70.7
Europe	78.6	77.6
Oceania	80.7	80.7

boxes correspond to the medians (i.e. the middle value) rather than the mean (the average). So, for example, if you look at Asia, the solid line denotes the median life expectancy of around 72 years. This indicates to us that half of all countries in Asia have a life expectancy below 72 years whereas half of all countries in Asia have a life expectancy above 72 years.

Let's compute the median and mean life expectancy for each continent with a little more data wrangling and display the results in Table 6.6.

```
lifeExp_by_continent <- gapminder2007 %>%
  group_by(continent) %>%
  summarize(median = median(lifeExp), mean = mean(lifeExp))
```

Observe the order of the second column `median` life expectancy: Africa is lowest, the Americas and Asia are next with similar medians, then Europe, then Asia. This ordering corresponds to the ordering of the solid black lines inside the boxes in our side-by-side boxplot in Figure 6.9. Let's now turn our attention to the values in the third column `mean`. Using Africa as a *baseline for comparison*, let's start making relative comparisons of life expectancies for the other continents:

1. The mean life expectancy of the Americas is $73.6 - 54.8 = 18.8$ years higher.
2. The mean life expectancy of Asia is $70.7 - 54.8 = 15.9$ years higher.
3. The mean life expectancy of Europe is $77.6 - 54.8 = 22.8$ years higher.
4. The mean life expectancy of Oceania is $80.7 - 54.8 = 25.9$ years higher.

Let's put these values Table 6.7, which we'll revisit later on in this section.

TABLE 6.7: Mean life expectancy by continent and relative differences from mean for Africa.

continent	mean	Difference relative to Africa
Africa	54.8	0.0
Americas	73.6	18.8
Asia	70.7	15.9
Europe	77.6	22.8
Oceania	80.7	25.9

Learning check

(LC6.4) Conduct a new exploratory data analysis with the same explanatory variable x being `continent` but with `gdpPerCap` as the new outcome variable y . Remember, this involves three things:

1. Most crucially: Looking at the raw data values.
2. Computing summary statistics, like means, medians, and interquartile ranges.
3. Creating data visualizations.

What can you say about the differences in GDP per capita between continents based on this exploration?

6.2.2 Linear regression

In Subsection 6.1.2 we introduced simple linear regression which involves modeling the relationship between a numerical outcome variable y and a numerical explanatory variable x . In our life expectancy example, we now instead have a categorical explanatory variable x continent. However our model will not yield a “best-fitting” regression line like in Figure 6.4, but rather “offsets relative to a baseline for comparison.”

As we did in Section 6.1.2 when studying the relationship between teaching scores and beauty scores, let’s output the regression table for this model. Recall that this is done in two steps:

1. We first “fit” the linear regression model using the `lm(y~x, data)` function and save it in `lifeExp_model`.
2. We get the regression table by applying the `get_regression_table()` from the `moderndive` package to `lifeExp_model`.

```
# Fit regression model:
lifeExp_model <- lm(lifeExp ~ continent, data = gapminder2007)
# Get regression table:
get_regression_table(lifeExp_model)
```

TABLE 6.8: Linear regression table

term	estimate	std_error	statistic	p_value	lower_ci	upper_ci
intercept	54.8	1.02	53.45	0	52.8	56.8
continentAmericas	18.8	1.80	10.45	0	15.2	22.4
continentAsia	15.9	1.65	9.68	0	12.7	19.2
continentEurope	22.8	1.70	13.47	0	19.5	26.2
continentOceania	25.9	5.33	4.86	0	15.4	36.5

Let’s once again focus on the values in the `term` and `estimate` columns of Table 6.8. Why are there now 5 rows? Let’s break them down one-by-one:

1. `intercept` here corresponds to the mean life expectancy of countries in Africa of 54.8 years.

2. `continentAmericas` corresponds to countries in the `continent` of the Americas and the value $+18.8$ is the same difference in mean life expectancy relative to Africa we displayed in Table 6.7. In other words, the mean life expectancy of countries in the Americas is $54.8 + 18.8 = 73.6$.
3. `continentAsia` corresponds to countries in the `continent` of Asia and the value $+15.9$ is the same difference in mean life expectancy relative to Africa we displayed in Table 6.7. In other words, the mean life expectancy of countries in Asia is $54.8 + 15.9 = 70.7$.
4. `continentEurope` corresponds to countries in the `continent` of Europe and the value $+22.8$ is the same difference in mean life expectancy relative to Africa we displayed in Table 6.7. In other words, the mean life expectancy of countries in Europe is $54.8 + 22.8 = 77.6$.
5. `continentOceania` corresponds to countries in the `continent` of Oceania and the value $+25.9$ is the same difference in mean life expectancy relative to Africa we displayed in Table 6.7. In other words, the mean life expectancy of countries in the Oceania is $54.8 + 25.9 = 80.7$.

In other words, the 5 values in the `estimate` correspond to:

1. The “baseline for comparison” continent Africa (the intercept).
2. Four “offsets” from this baseline for the remaining 4 continents: the Americas, Asia, Europe, and Oceania.

You might be asking at this point why was Africa chosen as the “baseline for comparison” group. For no other reason than it comes first alphabetically of the five continents; by default R arranges factors/categorical variables in alphanumeric order. However you can change this baseline group to be another continent if you manipulate the variable `continent`’s factor “levels” using the `forcats` package. See Chapter 15³ of Garrett Grolemund and Hadley Wickham’s book (Grolemund and Wickham, 2016) for examples.

Let’s now write the equation for our fitted values $\hat{y} = \widehat{\text{life exp}}$.

$$\begin{aligned}\hat{y} = \widehat{\text{life exp}} &= b_0 + b_{\text{Amer}} \cdot \mathbb{1}_{\text{Amer}}(x) + b_{\text{Asia}} \cdot \mathbb{1}_{\text{Asia}}(x) + \\ &\quad b_{\text{Euro}} \cdot \mathbb{1}_{\text{Euro}}(x) + b_{\text{Ocean}} \cdot \mathbb{1}_{\text{Ocean}}(x) \\ &= 54.8 + 18.8 \cdot \mathbb{1}_{\text{Amer}}(x) + 15.9 \cdot \mathbb{1}_{\text{Asia}}(x) + \\ &\quad 22.8 \cdot \mathbb{1}_{\text{Euro}}(x) + 25.9 \cdot \mathbb{1}_{\text{Ocean}}(x)\end{aligned}$$

Whoa! That looks very daunting! Don’t fret however, as once you understand

³<https://r4ds.had.co.nz/factors.html>

what all the elements mean, things simply greatly. First, $\mathbb{1}_A(x)$ is what's known in mathematics as an "indicator function" that takes only one of two possible values, 0 and 1, where

$$\mathbb{1}_A(x) = \begin{cases} 1 & \text{if } x \text{ is in } A \\ 0 & \text{if otherwise} \end{cases}$$

In a statistical modeling context this is also known as a "dummy variable". In our case, let's consider the first such indicator variable; this indicator function returns 1 if a country is in the Americas, 0 otherwise.

$$\mathbb{1}_{\text{Amer}}(x) = \begin{cases} 1 & \text{if country } x \text{ is in the Americas} \\ 0 & \text{otherwise} \end{cases}$$

Second, b_0 corresponds to the intercept as before; in this case its the mean life expectancy of all countries in Africa. Third, the b_{Amer} , b_{Asia} , b_{Euro} , and b_{Ocean} represent the 4 "offsets relative to the baseline for comparison" in the regression table output in Table 6.8: `continentAmericas`, `continentAsia`, `continentEurope`, and `continentOceania`.

Let's put this all together and compute the fitted value $\widehat{y} = \widehat{\text{life exp}}$ for a country in Africa. Since the country is in Africa, all four indicator functions $\mathbb{1}_{\text{Amer}}(x)$, $\mathbb{1}_{\text{Asia}}(x)$, $\mathbb{1}_{\text{Euro}}(x)$, and $\mathbb{1}_{\text{Ocean}}(x)$ will equal 0, and thus:

$$\begin{aligned} \widehat{\text{life exp}} &= b_0 + b_{\text{Amer}} \cdot \mathbb{1}_{\text{Amer}}(x) + b_{\text{Asia}} \cdot \mathbb{1}_{\text{Asia}}(x) + \\ &\quad b_{\text{Euro}} \cdot \mathbb{1}_{\text{Euro}}(x) + b_{\text{Ocean}} \cdot \mathbb{1}_{\text{Ocean}}(x) \\ &= 54.8 + 18.8 \cdot \mathbb{1}_{\text{Amer}}(x) + 15.9 \cdot \mathbb{1}_{\text{Asia}}(x) + \\ &\quad 22.8 \cdot \mathbb{1}_{\text{Euro}}(x) + 25.9 \cdot \mathbb{1}_{\text{Ocean}}(x) \\ &= 54.8 + 18.8 \cdot 0 + 15.9 \cdot 0 + 22.8 \cdot 0 + 25.9 \cdot 0 \\ &= 54.8 \end{aligned}$$

In other words, all that's left is the intercept b_0 corresponding to the average life expectancy of African countries of 54.8 years. Next, say we are considering a country in the Americas. In this case only the indicator function $\mathbb{1}_{\text{Amer}}(x)$ for the Americas will equal 1, while all the others will equal 0, and thus:

$$\begin{aligned} \widehat{\text{life exp}} &= 54.8 + 18.8 \cdot \mathbb{1}_{\text{Amer}}(x) + 15.9 \cdot \mathbb{1}_{\text{Asia}}(x) + 22.8 \cdot \mathbb{1}_{\text{Euro}}(x) + \\ &\quad 25.9 \cdot \mathbb{1}_{\text{Ocean}}(x) \\ &= 54.8 + 18.8 \cdot 1 + 15.9 \cdot 0 + 22.8 \cdot 0 + 25.9 \cdot 0 \\ &= 54.8 + 18.8 \\ &= 72.9 \end{aligned}$$

which is the mean life expectancy for countries in the Americas of 72.9 years we computed in the previous subsection in Table 6.7. Note the “offset from the baseline for comparison” here is +18.8 years. Let’s do one more. Say we are considering a country in Asia. In this case only the indicator function $\mathbb{1}_{\text{Asia}}(x)$ for Asia will equal 1, while all the others will equal 0, and thus:

$$\begin{aligned}\widehat{\text{life exp}} &= 54.8 + 18.8 \cdot \mathbb{1}_{\text{Amer}}(x) + 15.9 \cdot \mathbb{1}_{\text{Asia}}(x) + 22.8 \cdot \mathbb{1}_{\text{Euro}}(x) + \\ &\quad 25.9 \cdot \mathbb{1}_{\text{Ocean}}(x) \\ &= 54.8 + 18.8 \cdot 0 + 15.9 \cdot 1 + 22.8 \cdot 0 + 25.9 \cdot 0 \\ &= 54.8 + 15.9 \\ &= 70.7\end{aligned}$$

which is the mean life expectancy for countries in the Americas of 72.9 years we computed in the previous subsection in Table 6.7. Note the “offset from the baseline for comparison” here is +15.9 years.

Let’s generalize this idea a bit. If we fit a linear regression model using a categorical explanatory variable x that has k levels i.e. possible categories, a regression model will return an intercept and $k - 1$ “offsets.” In our case, since there are $k = 5$ continents, the regression model returns an intercept corresponding to the baseline for comparison Africa and $k - 1 = 4$ offsets corresponding to the Americas, Asia, Europe, and Oceania.

Phew! That was a lot of work! Understanding a regression table output when you’re using a categorical explanatory variable is a topic that many new regression practitioners struggle with. The only real remedy for these struggles is practice, practice, practice. However, once you equip yourselves with an understanding of how to create regression models using categorical explanatory variables, you’ll be able to incorporate many new variables in your models, given the large amount of the world’s data that is categorical. If you feel like you’re still struggling at this point however, we suggest you closely compare Tables 6.7 and 6.8 and note how you can compute all the values from one table using the values in the other.

Learning check

(LC6.5) Fit a new linear regression using `lm(gdpPercap ~ continent, data = gapminder2007)` where `gdpPercap` is the new outcome variable y . Get information about the “best-fitting” line from the regression table by applying the `get_regression_table()` function. How do the regression results match up with the results from your previous exploratory data analysis?

6.2.3 Observed/fitted values and residuals

Recall in Subsection 6.1.3, we defined the following three concepts:

1. Observed values y , or the observed value of the outcome variable
2. Fitted values \hat{y} , or the value on the regression line for a given x value
3. Residuals $y - \hat{y}$, or the error between the observed value and the fitted value

We obtained these values and other values using the `get_regression_points()` function from the `moderndive` package. This time however, let's add an `ID = "country"` argument: this is telling the function to use the variable `country` in `gapminder2007` as an identification variable in the output. This will help contextualize our analysis by matching values to countries.

```
regression_points <- get_regression_points(lifeExp_model, ID = "country")
regression_points
```

TABLE 6.9: Regression points (First 10 out of 142 countries)

country	lifeExp	continent	lifeExp_hat	residual
Afghanistan	43.8	Asia	70.7	-26.900
Albania	76.4	Europe	77.6	-1.226
Algeria	72.3	Africa	54.8	17.495
Angola	42.7	Africa	54.8	-12.075
Argentina	75.3	Americas	73.6	1.712
Australia	81.2	Oceania	80.7	0.515
Austria	79.8	Europe	77.6	2.180
Bahrain	75.6	Asia	70.7	4.907
Bangladesh	64.1	Asia	70.7	-6.666
Belgium	79.4	Europe	77.6	1.792

Observe in Table 6.9

- `lifeExp_hat` are the fitted values $\hat{y} = \widehat{\text{lifeexp}}$.
- If you look closely, there are only 5 possible values for `lifeExp_hat`. These correspond to the 5 mean life expectancies for the 5 continents we displayed in Table 6.7: Africa 54.8, the Americas 73.6, Asia 70.7, Europe 77.6, and Oceania 80.7

- The `residual` column is simply $y - \hat{y} = \text{lifeexp} - \text{lifeexp_hat}$. These values can be interpreted as the deviation of a country's life expectancy from its continent's average life expectancy. For example, look at the first row of Table 6.9 corresponding to Afghanistan. The residual of $y - \hat{y} = 43.8 - 70.7 = -26.9$ is indicating that Afghanistan's life expectancy is a whopping 26.9 years lower than the mean life expectancy of all Asia countries. This can in part be explained by the many years of war that country has suffered.

Learning check

(LC6.6) Using either the sorting functionality of RStudio's spreadsheet viewer or using the data wrangling tools you learned in Chapter 4, identify the 5 countries with the 5 smallest (most negative) residuals? What do these negative residuals say about their life expectancy relative to their continents?

(LC6.7) Repeat the above, but identify the 5 countries with the 5 largest (most positive) residuals. What do these positive residuals say about their life expectancy relative to their continents?

6.3 Related topics

6.3.1 Correlation is not necessarily causation

Recall that in Section 6.1 we've been very cautious when interpreting regression slope coefficients. We always discussed the “associated” effect of an explanatory variable x on an outcome variable y . For example our statement from Subsection 6.1.2 that “for every increase of 1 unit in `bty_avg`, there is an *associated* increase of, *on average*, 18.802 units of `score`.” We include the term “associated” to be extra careful not suggest we are making a *causal* statement. So while beauty score `bty_avg` is positively correlated with teaching score, we can't necessarily make any statements about beauty scores' direct causal effects on teaching score without more information on how this study was conducted.

For example, let's say an instructor has their `bty_avg` reevaluated after taking steps to try to boost their “beauty” score like changing their wardrobe.

Does this mean that they will suddenly become a better instructor? Will they necessarily start getting higher teaching scores from students? Maybe?

Here is another example: a not-so-great medical doctor goes through their medical records and finds that patients who slept with their shoes on tended to wake up more with headaches. So this doctor declares “Sleeping with shoes on cause headaches!”



FIGURE 6.10: Does sleeping with shoes on cause headaches?

However as some of you might have guessed, if someone is sleeping with their shoes on, it’s likely because they are intoxicated from alcohol. Furthermore, higher levels of drinking leads to more hangovers, and hence more headaches. In this instance, alcohol is what’s known as a *confounding/lurking* variable. It “lurks” behind the scenes, confounding or making less apparent, the causal effect (if any) of “sleeping with shoes on” with waking up with a headache. We can summarize this notion in Figure 6.11 with a *causal graph* where:

- Y is an *outcome* variable; here “waking up with a headache.”
- X is a *treatment* variable whose causal effect we are interested in; here “sleeping with shoes on.”

To study the relationship between Y and X, we could use a regression model where the outcome variable is set to Y and the explanatory variable is set to be X, as you’ve been doing throughout this chapter. However, Figure 6.11 also includes a third variable with arrows pointing at both X and Y:

- Z is a *confounding* variable that affects both X & Y thus “confounding” their relationship; here “alcohol”

Alcohol will both cause people to be more likely to sleep with their shoes on as well as be more likely to wake up with a headache. Thus any regression model of the relationship between X and Y needs to also use Z as an explanatory variable as well. In other words our doctor needs to take into account who had been drinking the night before. We’ll start covering multiple regression models that allows us to incorporate more than one variable in the next chapter.

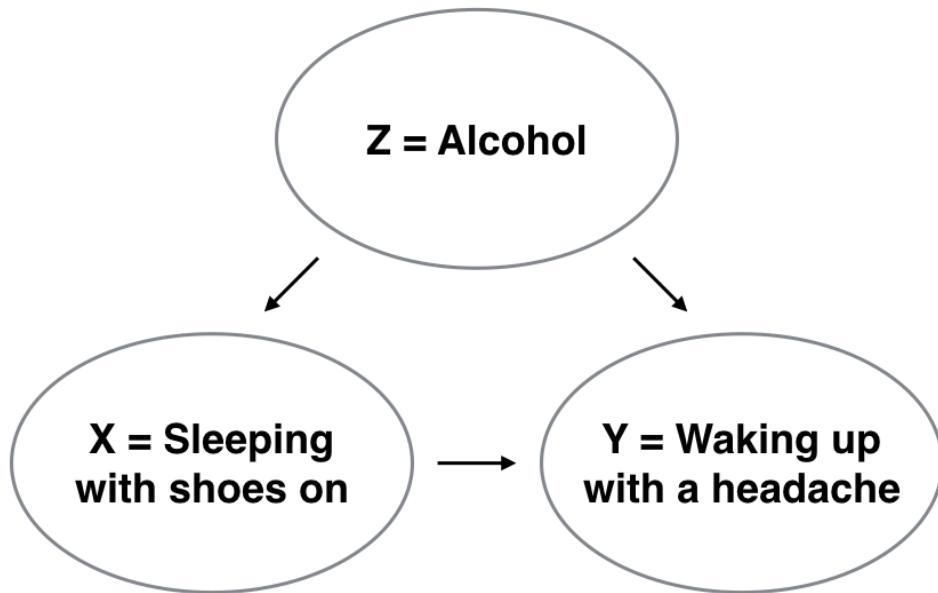


FIGURE 6.11: Causal graph.

Establishing causation is a tricky problem and frequently takes either carefully designed experiments or methods to control for the effects of potential confounding variables. Both these approaches attempt to either take them into account all possible confounding variables as best they can or negate their impact. This allows researchers to focus only on the relationship between the outcome variable Y and the treatment variable X.

As you read news stories, be careful to not fall into the trap of thinking the correlation necessarily implies causation. Check out Spurious Correlations⁴ for some examples of rather comical examples of variables that are correlated, but definitely are not causally related.

6.3.2 Best fitting line

Regression lines are also known as “best fitting” lines. But what do we mean by best? Let’s unpack the criteria that is used by regression to determine best. Recall the plot in Figure 6.6 where for a instructor with a beauty score of $x = 7.333$ we mark with

- Red circle: The *observed value* $y = 4.9$ is this course’s instructor’s actual teaching score.
- Red square: The *fitted value* \hat{y} is value on the regression line for $x = \text{bty_avg}$

⁴<http://www.tylervigen.com/spurious-correlations>

$= 7.333$. This value is computed using the intercept and slope in the previous regression table:

$$\hat{y} = b_0 + b_1 \cdot x = 54.806 + 18.802 \cdot 7.333 = 4.369$$

- Blue arrow: The length of this arrow is the *residual* and is computed by subtracting the fitted value \hat{y} from the observed value y . The residual can be thought of as the error or “lack of fit”. In the case of this course’s instructor, it is $y - \hat{y} = 4.9 - 4.369 = 0.531$.

We redisplay this information in the top-left plot of Figure 6.12. Furthermore, let’s repeat this for three more arbitrarily chosen course’s instructors:

1. A course whose instructor had a beauty score $x = 2.333$ and teaching score $y = 2.7$. The residual in this case is $2.7 - 4.036 = -1.336$, which we mark with a new blue arrow in the top-right plot.
2. A course whose instructor had a beauty score $x = 3.667$ and teaching score $y = 4.4$. The residual in this case is $4.4 - 4.125 = 0.2753$, which we mark with a new blue arrow in the bottom-left plot.
3. A course whose instructor had a beauty score $x = 6$ and teaching score $y = 3.8$. The residual in this case is $3.8 - 4.28 = -0.4802$, which we mark with a new blue arrow in the bottom-right plot.

Now say we repeated this process of computing residuals for all 463 courses’ instructors, then we squared all the residuals, and then we summed them. We call this quantity the “sum of squared residuals” and it is a measure of the “lack-of-fit” of a model to a cloud of point where larger values indicate a bigger “lack of fit.” If the blue regression line perfectly fits the cloud of points, then the sum of squared residuals is 0. This is because if the blue regression line fits all the points perfectly, then the fitted value \hat{y} equals the observed value y in all cases, and hence the residual $y - \hat{y} = 0$ in all cases, and the sum of a large number of 0’s is still 0.

Furthermore, of all possible lines we can draw through the cloud of 463 points, the blue regression line minimizes this value. In other words, the blue regression and its corresponding fitted values \hat{y} minimizes the sum of the squared residuals:

$$\sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Let’s use our data wrangling tools from Chapter 4 to compute the sum of squared residuals quantity exactly:

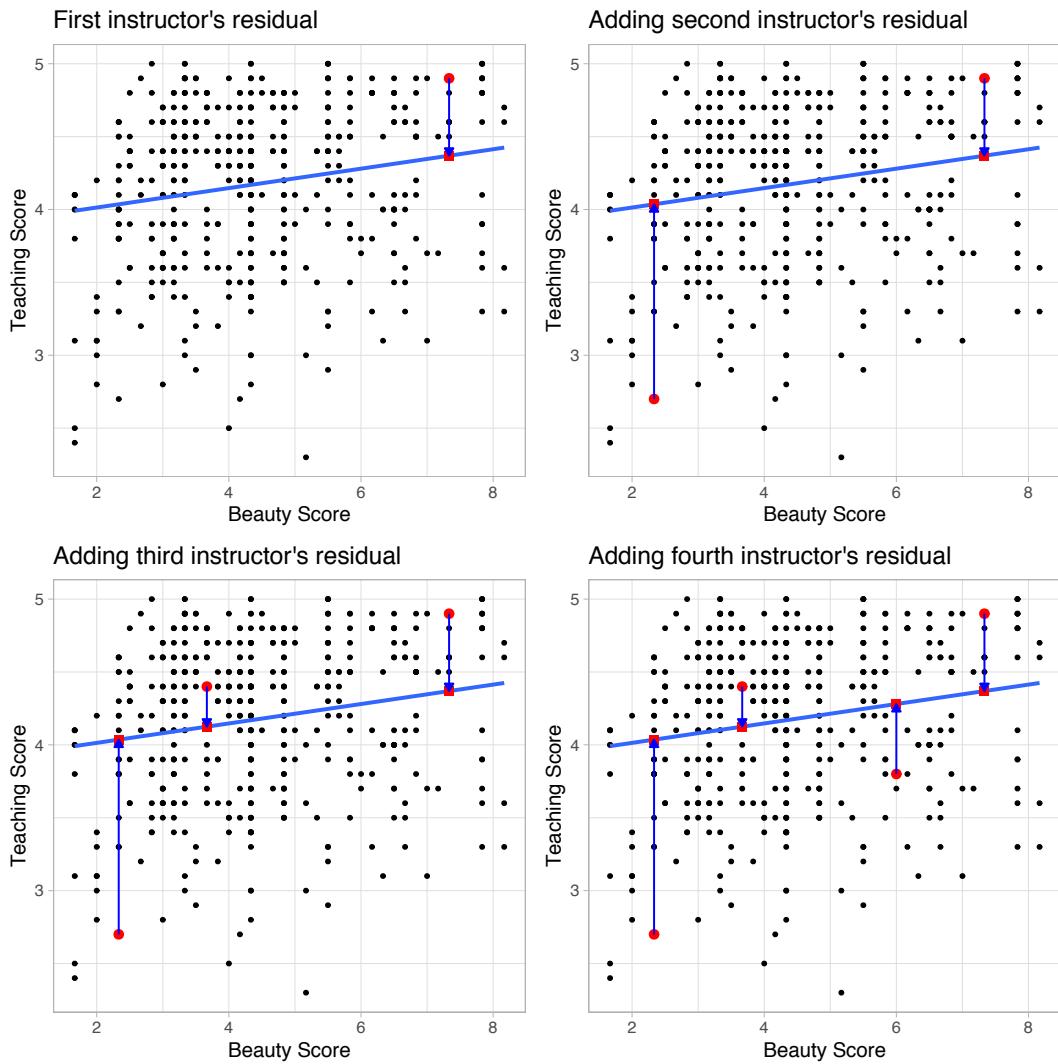


FIGURE 6.12: Example of observed value, fitted value, and residual.

```
# Fit regression model:
score_model <- lm(score ~ bty_avg, data = evals_ch6)

# Get regression points:
regression_points <- get_regression_points(score_model)
regression_points
```

```
# A tibble: 463 x 5
  ID score bty_avg score_hat residual
  <int> <dbl>    <dbl>      <dbl>     <dbl>
1     1     4.7      5        4.21     0.486
```

```

2     2   4.1   5      4.21  -0.114
3     3   3.9   5      4.21  -0.314
4     4   4.8   5      4.21   0.586
5     5   4.6   3      4.08   0.52
6     6   4.3   3      4.08   0.22
7     7   2.8   3      4.08  -1.28
8     8   4.1   3.33   4.10  -0.002
9     9   3.4   3.33   4.10  -0.702
10    10  4.5   3.17   4.09   0.409
# ... with 453 more rows

```

```

# Compute sum of squared residuals
regression_points %>%
  mutate(squared_residuals = residual^2) %>%
  summarize(sum_of_squared_residuals = sum(squared_residuals))

```

```

# A tibble: 1 × 1
sum_of_squared_residuals
<dbl>
1                  132.

```

Any other line draw in the figure would yield a sum of squared residuals greater than 132; this is a mathematically guaranteed fact that can be proven via calculus and linear algebra. That's why alternative names for the linear regression line are the **best fitting line** as well as the **least-squares line**. But why do we square the residuals (i.e. the arrow lengths)? We do this so that positive and negative deviations of the same amount are treated equally.

Learning check

(LC6.8) Note in the following plot there are 3 points marked with black dots along with:

- The “best” fitting regression line in blue
- An arbitrarily chosen line in dashed red
- Another arbitrarily chosen line in dashed green

Compute the sum of squared residuals by hand for each line and show that of these three lines, the regression line in blue has the smallest value.

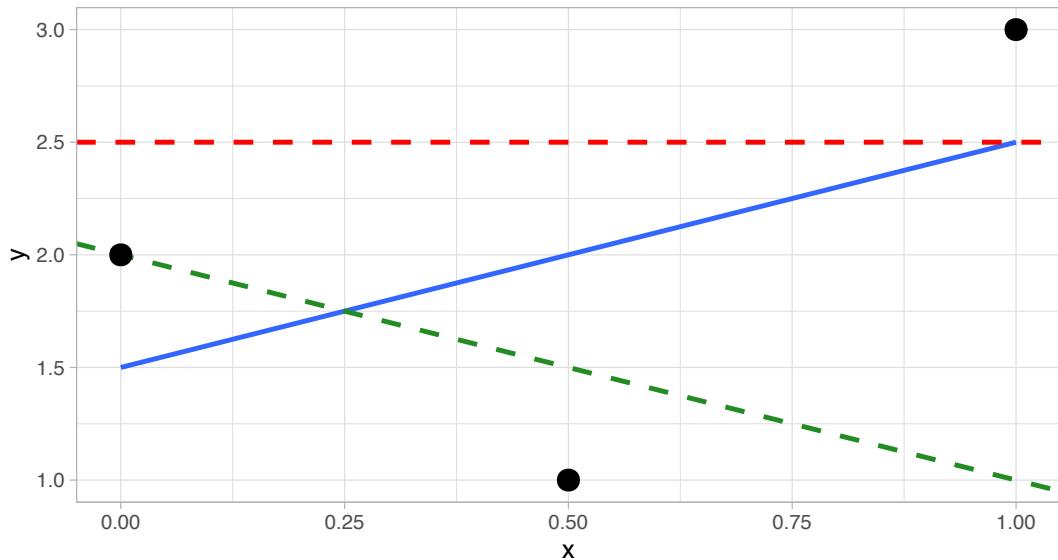


FIGURE 6.13: Regression line and two others.

6.3.3 `get_regression_x()` functions

Recall in this chapter we introduced two functions from the `moderndive` package:

1. `get_regression_table()` function that returns a regression table in Subsection 6.1.2 and the
2. `get_regression_points()` function that returns point-by-point information from a regression model In Subsection 6.1.3.

What is going on behind the scenes with the `get_regression_table()` and `get_regression_points()`? We mentioned that these were examples of *wrapper functions*: functions that takes other pre-existing functions and “wraps” them in a single function that hide the user from its inner workings. This way all the user needs to worry about is what the input looks like and what the output looks like. In this subsection we’ll “get under the hood” of these functions and see how the “engine” of these wrapper functions work.

Recall our two step process to generated a regression table from Subsection 6.1.2:

```
# Fit regression model:
score_model <- lm(score ~ bty_avg, data = evals_ch6)
# Get regression table:
get_regression_table(score_model)
```

TABLE 6.10: Regression table.

term	estimate	std_error	statistic	p_value	lower_ci	upper_ci
intercept	3.880	0.076	50.96	0	3.731	4.030
bty_avg	0.067	0.016	4.09	0	0.035	0.099

The `get_regression_table()` wrapper function takes two pre-existing functions in other R packages

- the `tidy()` function from the `broom` package⁵ and
- the `clean_names()` function from the `janitor` package⁶

and “wraps” them in a single function that takes in a saved `lm()` linear model model, here `score_model`, and returns a regression table saved as a “tidy” data frame. Here is how we used the `tidy()` and `clean_names()` functions:

```
library(broom)
library(janitor)
score_model %>%
  tidy(conf.int = TRUE) %>%
  mutate_if(is.numeric, round, digits = 3) %>%
  clean_names() %>%
  rename(lower_ci = conf_low,
        upper_ci = conf_high)
```

Attaching package: 'janitor'

The following objects are masked from 'package:stats':

`chisq.test`, `fisher.test`

TABLE 6.11: Regression table using `tidy()` from `broom` package.

term	estimate	std_error	statistic	p_value	lower_ci	upper_ci
(Intercept)	3.880	0.076	50.96	0	3.731	4.030
bty_avg	0.067	0.016	4.09	0	0.035	0.099

Yikes! That's a lot of code! So in order to simplify your lives, we made the

⁵<https://broom.tidyverse.org/>

⁶<https://github.com/sfirke/janitor>

editorial decision to “wrap” all the code into `get_regression_table()`, freeing you from the need to understand the inner workings of the function. (Note that the `mutate_if()` function is from the `dplyr` package and applies the `round()` function with 3 significant digits precision only to those variables that are numerical.)

Similarly, the `get_regression_points()` function is another wrapper function, but this time returning information about a regression like the fitted values, observed values, and the residuals. `get_regression_points()` uses the `augment()` function in the `broom` package⁷ instead of the `tidy()` function as with `get_regression_table()`:

```
library(broom)
library(janitor)
score_model %>%
  augment() %>%
  mutate_if(is.numeric, round, digits = 3) %>%
  clean_names() %>%
  select(-c("se_fit", "hat", "sigma", "cooksdi", "std_resid"))
```

TABLE 6.12: Regression points using `augment()` from `broom` package.

	score	bty_avg	fitted	resid
	4.7	5.00	4.21	0.486
	4.1	5.00	4.21	-0.114
	3.9	5.00	4.21	-0.314
	4.8	5.00	4.21	0.586
	4.6	3.00	4.08	0.520
	4.3	3.00	4.08	0.220
	2.8	3.00	4.08	-1.280
	4.1	3.33	4.10	-0.002
	3.4	3.33	4.10	-0.702
	4.5	3.17	4.09	0.409

In this case, it outputs only the variables of interest to people learning regression: the outcome variable y (`score`), all explanatory/predictor variables (`bty_avg`), all resulting `fitted` values \hat{y} used by applying the equation of the regression line to `bty_avg`, and the `residual` $y - \hat{y}$.

If you’re even more curious about how these and other wrapper functions work, take a look at the source code for these functions on GitHub⁸.

⁷<https://broom.tidyverse.org/>

⁸https://github.com/moderndive/moderndive/blob/master/R/regression_functions.R

6.4 Conclusion

6.4.1 Additional resources

An R script file of all R code used in this chapter is available here⁹.

As we suggested in Subsection 6.1.1, interpreting coefficients that are not close to the extreme values of -1 and 1 can be subjective. To develop your sense of correlation coefficients, we suggest you play the following 80's-style video game called "Guess the correlation" at <http://guessthecorrelation.com/>.

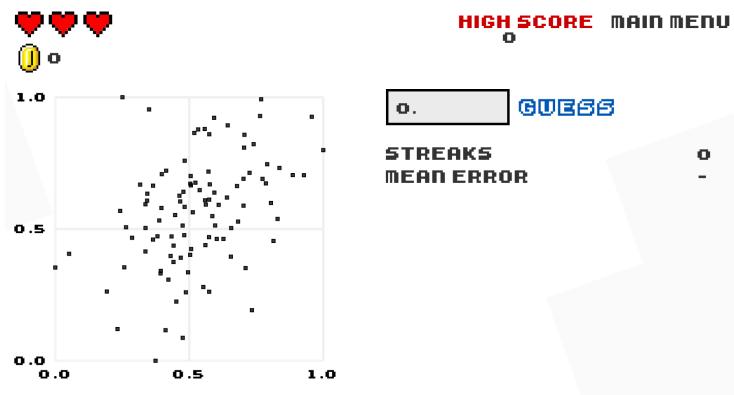


FIGURE 6.14: Preview of "Guess the Correlation" Game.

6.4.2 What's to come?

In this chapter, you've studied what we like to term "basic regression" where you only have one explanatory variable. In Chapter 7, we'll study *multiple regression* where our regression models can have more than one explanatory variable! In particular, we'll consider two scenarios: regression models with one numerical and one categorical explanatory variables and regression models with two numerical explanatory variables. This will allow you to construct more sophisticated and powerful models in the hopes of better explaining your outcome variable y of interest.

⁹[scripts/06-regression.R](#)

7

Multiple Regression

In Chapter 6 we introduced ideas related to modeling for explanation, in particular that the goal of modeling is make explicit the relationship between some outcome variable y and some explanatory variable x . While there are many approaches to modeling, we focused on one particular technique: *linear regression*, one of the most commonly-used and easy-to-understand approaches to modeling. Furthermore to keep things simple we only considered models with one explanatory x variable that was either numerical in Section 6.1 or categorical in Section 6.2.

In this chapter on multiple regression we'll start considering models that include more than one explanatory variable x . You can imagine when trying to model a particular outcome variable, like teaching evaluation scores as in Section 6.1 or life expectancy as in Section 6.2, that it would be very useful to include more than just one explanatory variable's worth of information.

Since our regression models will now consider more than one explanatory variable, the interpretation of the associated effect of any one explanatory variable must be made in conjunction with the other explanatory variables included in your model. Let's begin!

Needed packages

Let's load all the packages needed for this chapter (this assumes you've already installed them). Recall from our discussion in Section 5.4.1 that loading the `tidyverse` package by running `library(tidyverse)` loads the following commonly used data science packages all at once:

- `ggplot2` for data visualization
- `dplyr` for data wrangling
- `tidyverse` for converting data to “tidy” format
- `readr` for importing spreadsheet data into R
- As well as the more advanced `purrr`, `tibble`, `stringr`, and `forcats` packages

If needed, read Section 2.3 for information on how to install and load R packages.

```
library(tidyverse)
library(moderndive)
library(skimr)
library(ISLR)
```

7.1 One numerical & one categorical explanatory variable

Let's revisit the instructor evaluation data we introduced in Section 6.1, where we studied the relationship between instructor evaluation scores (as given by students) and their “beauty” scores for instructors teaching courses at the UT Austin; the variable `teaching score` was a numerical outcome variable y and the variable `beauty score bty_avg` was a numerical explanatory x variable.

In this section we are going to consider a different model. Our outcome variable will still be teaching score, but now including two different explanatory variables: age and gender. Could it be that instructors who are older receive better teaching evaluations from students? Or could it instead be that younger instructors receive better evaluations? Are there differences in evaluations given by students for instructors of different genders? We'll answer these questions by modeling the relationship between these variables using *multiple regression* where we have:

1. A numerical outcome variable y , as before the instructor's teaching score and
2. Two explanatory variables:
 1. A numerical explanatory variable x_1 , the instructor's age
 2. A categorical explanatory variable x_2 , the instructor's binary gender (male or female).

It is important to note that at the time of this study, due to then commonly held beliefs about gender, this variable was often recorded as a binary. While the results of a model that oversimplifies gender this way may be imperfect, we still found the results to be very pertinent and relevant today. An eminent statistician by the name George E.P. Box summarizes our thinking very nicely: “All models are wrong, but some are useful.”¹.

¹https://en.wikipedia.org/wiki/All_models_are_wrong

7.1.1 Exploratory data analysis

The data on the 463 courses at the UT Austin can be found in the `evals` data frame included in the `moderndive` package. However, to keep things simple, let's `select()` only the subset of the variables we'll consider in this chapter, and save this data in a new data frame called `eval_ch7`. Note that these are different than the variables chosen in Chapter 6.

```
evals_ch7 <- evals %>%  
  select(ID, score, age, gender)
```

Recall the three common steps in an exploratory data analysis we saw in Section 6.1.1

1. Looking at the raw data values.
2. Computing summary statistics, like means, medians, and interquartile ranges.
3. Creating data visualizations.

Let's first look at the raw data values both either looking at `evals_ch7` RStudio's spreadsheet viewer or using the `glimpse()` function

```
glimpse(evals_ch7)
```

```
Observations: 463  
Variables: 4  
 $ ID      <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1...  
 $ score   <dbl> 4.7, 4.1, 3.9, 4.8, 4.6, 4.3, 2.8, 4.1, ...  
 $ age     <int> 36, 36, 36, 36, 59, 59, 59, 51, 51, 40, ...  
 $ gender  <fct> female, female, female, female, male, ma...
```

Let's also display a random sample of 5 rows of the 463 rows corresponding to different courses in Table 7.1. Remember due to the random nature of the sampling, you will likely end up with a different subset of 5 rows.

```
evals_ch7 %>%  
  sample_n(size = 5)
```

TABLE 7.1: A random sample of 5 out of the 463 courses at UT Austin

ID	score	age	gender
129	3.7	62	male
109	4.7	46	female
28	4.8	62	male
434	2.8	62	male
330	4.0	64	male

Now that we've looked at the raw values in our `evals_ch7` data frame and obtained a sense of the data, let's move on to next common step in an exploratory data analysis: computing summary statistics. As we did in our exploratory data analyses in Sections 6.1.1 and 6.2.1 from the previous chapter, let's use the `skim()` function from the `skimr` package, being sure to only `select()` the variables of interest of model:

```
evals_ch7 %>%
  select(score, age, gender) %>%
  skim()

Skim summary statistics
n obs: 463
n variables: 3

— Variable type:factor —————
variable missing complete n n_unique top_counts ordered
  gender      0       463 463          2 mal: 268, fem: 195, NA: 0 FALSE

— Variable type:integer —————
variable missing complete n mean sd p0 p25 p50 p75 p100
  age        0       463 463 48.37 9.8 29  42   48   57   73

— Variable type:numeric —————
variable missing complete n mean sd p0 p25 p50 p75 p100
  score       0       463 463 4.17 0.54 2.3  3.8  4.3  4.6   5
```

Observe for example that we have no missing data, courses taught by 268 male vs 195 female instructors, and average age of 48.37. Recall however that each row in our data represents a particular course and that instructors can teach more than one course. Therefore the average age of the unique instructors may differ.

Furthermore, let's compute the correlation between our two numerical vari-

ables: `score` and `age`. Recall from Section 6.1.1 that correlation coefficients only exist between numerical variables. We observe that they are weakly negatively correlated.

```
evals_ch7 %>%
  get_correlation(formula = score ~ age)
```

```
# A tibble: 1 × 1
  correlation
  <dbl>
1 -0.107
```

Let's now perform the last of the three common steps in an exploratory data analysis: creating data visualizations. Given that the outcome variable `score` and explanatory variable `age` are both numerical, we'll use a scatterplot to display their relationship. How can we incorporate the categorical variable `gender` however? By mapping the variable `gender` to the color aesthetic and creating a *colored* scatterplot! The following code is very similar to the code that created the scatterplot of teaching score and beauty score in Figure 6.2, but with `color = gender` added to the `aes()`.

```
ggplot(evals_ch7, aes(x = age, y = score, color = gender)) +
  geom_point() +
  labs(x = "Age", y = "Teaching Score", color = "Gender") +
  geom_smooth(method = "lm", se = FALSE)
```

In the resulting Figure 7.1, observe that `ggplot` assigns a default red/blue color scheme to the points and lines associated with each of the two levels of `gender`: `female` and `male`. Furthermore the `geom_smooth(method = "lm", se = FALSE)` layer automatically fits a different regression line for each group since we have provided `color = gender` in the aesthetic mapping. This allows for all subsequent geometries to have the same aesthetic mappings.

We notice some interesting trends:

1. There are almost no women faculty over the age of 60 as evidenced by lack of red dots above $x = 60$.
2. While both regression lines are negatively sloped with age (i.e. older instructors tend to have lower scores), the slope for age for the female instructors is *more* negative. In other words, female instructors are paying a harsher penalty for their age than the male instructors.

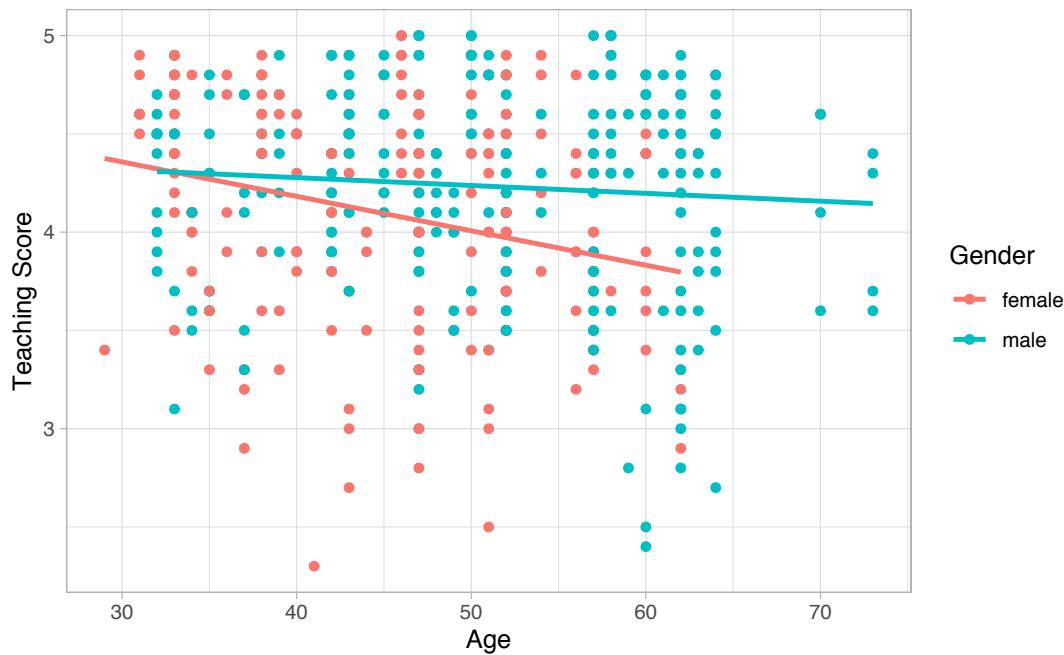


FIGURE 7.1: Colored scatterplot of relationship of teaching and beauty scores.

7.1.2 Interaction model

Let's now quantify the relationship of our outcome variable y and two explanatory variables using one type of multiple regression model known as an "interaction model." Unfortunately, we don't have enough context at this point to explain where the term "interaction" comes from; we'll explain why statisticians use this term at the end of this section.

In particular, we'll write out the equation of the two regression lines in Figure 7.1 using the values from a regression table. Before we do this however, let's go over a brief refresher of regression when you have a categorical explanatory variable x .

Recall in Section 6.2.2 we fit a regression model for countries' life expectancy as a function of which continent the country was in. In other words we had a numerical outcome variable $y = \text{lifeExp}$ and a categorical explanatory variable $x = \text{continent}$ which had 5 levels: Africa, Americas, Asia, Europe, and Oceania. Let's redisplay the regression table you saw in Table 6.8:

TABLE 7.2: Regression table for life expectancy as a function of continent.

term	estimate	std_error	statistic	p_value	lower_ci	upper_ci
intercept	54.8	1.02	53.45	0	52.8	56.8
continentAmericas	18.8	1.80	10.45	0	15.2	22.4
continentAsia	15.9	1.65	9.68	0	12.7	19.2
continentEurope	22.8	1.70	13.47	0	19.5	26.2
continentOceania	25.9	5.33	4.86	0	15.4	36.5

Recall our interpretations of the `estimate` column. Since `Africa` was the “baseline for comparison” group since Africa comes first alphabetically, the `intercept` term corresponds to the mean life expectancy for all countries in Africa of 54.8 years. The other 4 values of `estimate` correspond to “offsets” relative to the baseline group. So for example, the “offset” corresponding to the Americas is +18.8 versus the baseline for comparison group Africa i.e. the average life expectancy for countries in the Americas is 18.8 years *higher*. Thus the mean life expectancy for all countries in the Americas is $54.8 + 18.8 = 73.6$. The same interpretation holds for Asia, Europe, and Oceania.

Going back to our multiple regression model for teaching `score` using `age` and `gender` in Figure 7.1, we generate the regression table using the same two step approach from Chapter 6: we first “fit” the model using the `lm()` “linear model” function and then we apply the `get_regression_table()` function. This time however our model formula won’t be of form $y \sim x$, but rather of form $y \sim x_1 * x_2$. In other words our two explanatory variables x_1 and x_2 are separated by a $*$ sign:

```
# Fit regression model:
score_model_interaction <- lm(score ~ age * gender, data = evals_ch7)
# Get regression table:
get_regression_table(score_model_interaction)
```

TABLE 7.3: Regression table for interaction model.

term	estimate	std_error	statistic	p_value	lower_ci	upper_ci
intercept	4.883	0.205	23.80	0.000	4.480	5.286
age	-0.018	0.004	-3.92	0.000	-0.026	-0.009
gendermale	-0.446	0.265	-1.68	0.094	-0.968	0.076
age:gendermale	0.014	0.006	2.45	0.015	0.003	0.024

Looking at the regression table output in Table 7.3, we see there are four rows

of values in the `estimate` column. While it is not immediately apparent, using these four values we can write out the equations of both the red and blue lines in Figure 7.1. Let's build these up.

First, since the word `female` is alphabetically before `male`, female instructors are the “baseline for comparison” group. Therefore `intercept` is the intercept and `age` is the slope for age *for only the female instructors*. In other words, the red regression line in Figure 7.1 has intercept 4.883 and slope for age of -0.018. Remember that for this particular data, while the intercept has a mathematical interpretation, it has no *practical* interpretation since there can't be any instructors with age = 0.

What about the intercept and slope for age of the male instructors? In other words the blue line in Figure 7.1? This is where our notion of “offsets” comes into play once again. The value for `gendermale` of -0.446 is not the intercept for the male instructors, but rather the *offset* (or difference) in intercept for male instructors relative to female instructors. Therefore, the intercept for the male instructors is `intercept + gendermale` = $4.883 + (-0.446) = 4.883 - 0.446 = 4.437$.

Similarly, `age:gendermale` = 0.014 is not the slope for age for the male instructors, but rather the *offset* (or difference) in slope for the male instructors. Therefore, the slope for age for the male instructors is `age + age:gendermale` = $-0.018 + 0.014 = -0.004$. Therefore the blue regression line in Figure 7.1 has intercept 4.437 and slope for age of -0.004.

Let's summarize these values in Table 7.4 and focus on the two slopes for age:

TABLE 7.4: Comparison of female and male intercepts and age slopes

Gender	Intercept	Slope for age
Female instructors	4.88	-0.018
Male instructors	4.44	-0.004

Since the slope for age for the female instructors was -0.018, it means that on average, a female instructor who is a year older would have a teaching score that is 0.018 units **lower**. For the male instructors however, the corresponding associated decrease was on average only 0.004 units. While both slopes for age were negative, the slope for age for the female instructors is *more negative*. This is consistent with our observation from Figure 7.1, that this model is suggesting that age is impacts teaching scores more for female instructors.

Let's now write the equation for our regression lines, which we can use to compute our fitted values $\hat{y} = \widehat{\text{score}}$.

$$\begin{aligned}\hat{y} = \widehat{\text{score}} &= b_0 + b_{\text{age}} \cdot \text{age} + b_{\text{male}} \cdot \mathbb{1}_{\text{is male}}(x) + b_{\text{age,male}} \cdot \text{age} \cdot \mathbb{1}_{\text{is male}} \\ &= 4.883 - 0.018 \cdot \text{age} - 0.446 \cdot \mathbb{1}_{\text{is male}}(x) + 0.014 \cdot \text{age} \cdot \mathbb{1}_{\text{is male}}\end{aligned}$$

Whoa! That's even more daunting than the equation you saw for the life expectancy as a function of continent in Section 6.2.2! However if you recall what an "indicator function" AKA "dummy variable" does, the equation simplifies greatly. In the above equation, we have one indicator function of interest:

$$\mathbb{1}_{\text{is male}}(x) = \begin{cases} 1 & \text{if instructor } x \text{ is male} \\ 0 & \text{otherwise} \end{cases}$$

Second, let's match coefficients in the above equation with values in the `estimate` column in our regression table in Table 7.3:

1. b_0 is the `intercept` = 4.883 for the female instructors
2. b_{age} is the slope for `age` = -0.018 for the female instructors
3. b_{male} is the offset in intercept for the male instructors
4. $b_{\text{age,male}}$ is the offset in slope for `age` for the male instructors

Let's put this all together and compute the fitted value $\hat{y} = \widehat{\text{score}}$ for female instructors. Since for female instructors $\mathbb{1}_{\text{is male}}(x) = 0$, the above equation becomes

$$\begin{aligned}\hat{y} = \widehat{\text{score}} &= b_0 + b_{\text{age}} \cdot \text{age} + b_{\text{male}} \cdot \mathbb{1}_{\text{is male}}(x) + b_{\text{age,male}} \cdot \text{age} \cdot \mathbb{1}_{\text{is male}} \\ &= 4.883 - 0.018 \cdot \text{age} - 0.446 \cdot \mathbb{1}_{\text{is male}}(x) + 0.014 \cdot \text{age} \cdot \mathbb{1}_{\text{is male}} \\ &= 4.883 - 0.018 \cdot \text{age} - 0.446 \cdot 0 + 0.014 \cdot \text{age} \cdot 0 \\ &= 4.883 - 0.018 \cdot \text{age} - 0 + 0 \\ &= 4.883 - 0.018 \cdot \text{age}\end{aligned}$$

which is the equation of the red regression line in Figure 7.1 corresponding to the female instructors. Correspondingly, since for male instructors $\mathbb{1}_{\text{is male}}(x) = 1$, the above equation becomes

$$\begin{aligned}\hat{y} = \widehat{\text{score}} &= 4.883 - 0.018 \cdot \text{age} - 0.446 \cdot \mathbb{1}_{\text{is male}}(x) + 0.014 \cdot \text{age} \cdot \mathbb{1}_{\text{is male}} \\ &= 4.883 - 0.018 \cdot \text{age} - 0.446 \cdot 1 + 0.014 \cdot \text{age} \cdot 1 \\ &= 4.883 - 0.018 \cdot \text{age} - 0.446 + 0.014 \cdot \text{age} \\ &= (4.883 - 0.446) + (-0.018 + 0.014) * \text{age} \\ &= 4.437 - 0.004 \cdot \text{age}\end{aligned}$$

which is the equation of the blue regression line in Figure 7.1 corresponding to the male instructors.

Phew! That was a lot of arithmetic! Don't fret however, this is as hard as modeling will get in this book. If you're still a little unsure about using indicator functions and using categorical explanatory variables, we *highly* suggest you re-read Section 6.2.2 which involves only a single categorical explanatory variable and thus is much simpler.

Before we end this section, we explain why we refer to this type of model as an "interaction model." The $b_{\text{age,male}}$ term in the equation for the fitted value $\hat{y} = \widehat{\text{score}}$ is what's known in statistical modeling as an "interaction effect." The interaction term corresponds to the `age:gendermale = 0.014` in the final row of the regression table in Table 7.3.

We say there is an interaction effect if the associated effect of one variable *depends on the value of another variable*, in other words the two variables are "interacting." In our case, the associated effect of the variable age *depends* on the value of another variable, gender. This was evidenced by the difference in slopes for age of +0.014 of male instructors relative to female instructors.

Another way of thinking of interaction effects is as follows. For a given instructor at the UT Austin, there might be an associated effect of their age on their teaching scores, there might be an associated effect of the gender on their teaching scores, but when put together, there might be an *additional effect due to the intersection* of their age and their gender.

7.1.3 Parallel slopes model

When creating regression models with one numerical and one categorical explanatory variable, we are not just limited to interaction models as we just saw. Another type of model we can use is known as the "parallel slopes" model. Unlike with interaction models where the regression line can have both different intercepts and different slopes, parallel slopes models still allow for different intercepts but *force* all lines to have the same slope. The resulting regression lines are thus parallel. Let's visualize the best fitting parallel slopes model to our `evals_ch7` data.

Unfortunately, the `ggplot2` package does not have a convenient way to plot a parallel slopes model. We therefore created our own function `gg_parallel_slopes()` and included it in the `moderndive` package:

```
gg_parallel_slopes(y = "score", num_x = "age", cat_x = "gender",
                     data = evals_ch7)
```

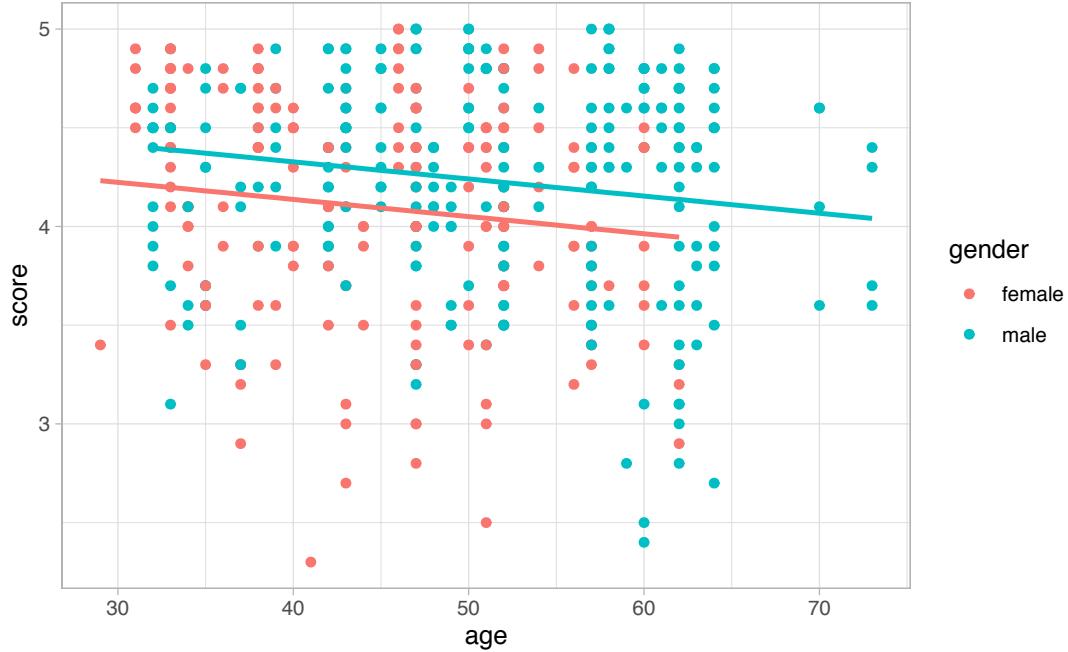


FIGURE 7.2: Parallel slopes model of relationship of score with age and gender.

Note the arguments i.e. inputs to this function: the outcome variable `y = "score"`, the numerical explanatory variable `num_x = "age"`, the categorical explanatory variable `cat_x = "gender"`, and the data frame that includes this `data = evals_ch7`. Be careful to include the quotation marks when specifying all variables, something you don't have to do when creating a visualization with `ggplot()`.

Observe in Figure 7.2 that we now have parallel red and blue lines corresponding to the female and male instructors respectively, in other words they have the same negative slope. In other words, as instructors age, so also do they tend to receive lower teaching evaluation scores from students. However these two lines have different intercepts as evidenced by the fact that the blue line corresponding to the male instructors is higher than the red line corresponding to the female instructors.

In order to obtain the precise numerical values of the intercepts and the common slope, we once again first “fit” the model using the `lm()` “linear model” function and then we apply the `get_regression_table()` function. However, unlike the interaction model which had a model formula of form `y ~ x1 * x2`, our

model formula is now of form $y \sim x_1 + x_2$. In other words our two explanatory variables x_1 and x_2 are separated by a $+$ sign:

```
# Fit regression model:
score_model_parallel_slopes <- lm(score ~ age + gender, data = evals_ch7)
# Get regression table:
get_regression_table(score_model_parallel_slopes)
```

TABLE 7.5: Regression table for parallel slopes model.

term	estimate	std_error	statistic	p_value	lower_ci	upper_ci
intercept	4.484	0.125	35.79	0.000	4.238	4.730
age	-0.009	0.003	-3.28	0.001	-0.014	-0.003
gendermale	0.191	0.052	3.63	0.000	0.087	0.294

Similarly to the regression table for the interaction model from our earlier Table 7.3, we have an `intercept` term corresponding to the intercept for the “baseline for comparison” female instructor group and a `gendermale` term corresponding to the *offset* (or difference) in intercept for the male instructors relative to female instructors. In other words in Figure 7.2 the red regression line corresponding to the female instructors has an intercept of 4.484 while the blue regression line corresponding to the male instructors has an intercept of $4.484 + 0.191 = 4.67$. Once again, since there aren’t any instructors of age 0, the intercepts only have a mathematical interpretation but no practical one.

Unlike in Table 7.3 we now only have a single term relating to the slope for age as we’ve forced both the female and male instructors to have a common slope for age of -0.009. In other words, for every increase of 1 year in instructor age, we observe an associated decrease of on average 0.009 units in teaching for *both* the female and male instructor.

Let’s now write the equation for our regression lines, which we can use to compute our fitted values $\hat{y} = \widehat{\text{score}}$.

$$\begin{aligned}\hat{y} &= \widehat{\text{score}} = b_0 + b_{\text{age}} \cdot \text{age} + b_{\text{male}} \cdot \mathbb{1}_{\text{is male}}(x) \\ &= 4.484 - 0.009 \cdot \text{age} + 0.191 \cdot \mathbb{1}_{\text{is male}}(x)\end{aligned}$$

Let’s put this all together and compute the fitted value $\hat{y} = \widehat{\text{score}}$ for female instructors. Since for female instructors $\mathbb{1}_{\text{is male}}(x) = 0$, the above equation becomes

$$\begin{aligned}
 \hat{y} &= \widehat{\text{score}} = b_0 + b_{\text{age}} \cdot \text{age} + b_{\text{male}} \cdot \mathbb{1}_{\text{is male}}(x) \\
 &= 4.484 - 0.009 \cdot \text{age} + 0.191 \cdot \mathbb{1}_{\text{is male}}(x) \\
 &= 4.484 - 0.009 \cdot \text{age} + 0.191 \cdot 0 \\
 &= 4.484 - 0.009 \cdot \text{age}
 \end{aligned}$$

which is the equation of the red regression line in Figure 7.2 corresponding to the female instructors. Correspondingly, since for male instructors $\mathbb{1}_{\text{is male}}(x) = 1$, the above equation becomes

$$\begin{aligned}
 \hat{y} &= \widehat{\text{score}} = b_0 + b_{\text{age}} \cdot \text{age} + b_{\text{male}} \cdot \mathbb{1}_{\text{is male}}(x) \\
 &= 4.484 - 0.009 \cdot \text{age} + 0.191 \cdot \mathbb{1}_{\text{is male}}(x) \\
 &= 4.484 - 0.009 \cdot \text{age} + 0.191 \cdot 1 \\
 &= (4.484 + 0.191) - 0.009 \cdot \text{age} \\
 &= 4.67 - 0.009 \cdot \text{age}
 \end{aligned}$$

which is the equation of the blue regression line in Figure 7.2 corresponding to the male instructors.

Great! We've considered both an interaction model and a parallel slopes model for our data. Let's compare the visualizations for both models side-by-side in Figure 7.3

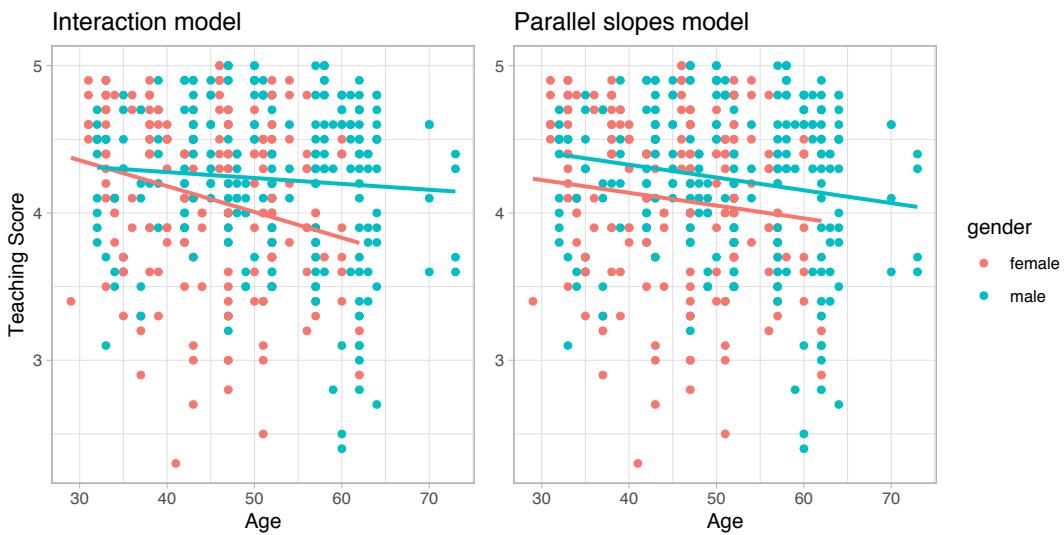


FIGURE 7.3: Comparison of interaction and parallel slopes models.

At this point, you might be asking yourself: “Why would we ever use an parallel slopes model?” Looking at the left-hand plot in Figure 7.3, the two lines definitely do not appear to be parallel, so why would we *force* them to

be parallel as in the right-hand plot?" For this data, we agree! It can easily be argued that the interaction model is more appropriate. However, in Section 7.3.1 below on model selection, we'll present an example where it can be argued that the case for a parallel slopes model might be stronger.

7.1.4 Observed/fitted values and residuals

For brevity's sake, in this section we'll only compute the observed values, fitted values, and residuals for the interaction model which we saved in `score_model_interaction`. You'll have an opportunity to study these values for our parallel slopes model in the upcoming Learning Check.

Say you have a professor who is female and is 36 years old? What fitted value $\hat{y} = \widehat{\text{score}}$ would our model yield? Say you have another professor who is male and is 59 years old? What would their fitted value \hat{y} be? We answer this question visually by finding the intersection of the red regression line and a vertical line at $x = \text{age} = 36$; we mark this value with a large red dot in Figure 7.4. Similarly we can identify the fitted value $\hat{y} = \widehat{\text{score}}$ for the male instructor by finding the intersection of the blue regression line and a vertical line at $x = \text{age} = 59$; we mark this value with a large blue dot in Figure 7.4.



FIGURE 7.4: Fitted values for two new professors.

However, what are these values precisely? We can use the equations of the

two regression lines we computed in Section 7.1.2, which in turn were based on values from the regression table in Table 7.3:

- For all female instructors: $\hat{y} = \widehat{\text{score}} = 4.883 - 0.018 \cdot \text{age}$
- For all male instructors: $\hat{y} = \widehat{\text{score}} = 4.437 - 0.004 \cdot \text{age}$

So our fitted values would be: $4.883 - 0.018 \cdot 36 = 4.25$ and $4.437 - 0.004 \cdot 59 = 4.20$ respectively. What if however we wanted the fitted values not just for these two instructors, but the instructors for all 463 courses? Doing this by hand would be long and tedious! This is where the `get_regression_points()` function from the `moderndive` package can help: it will quickly automate this for all 463 courses. We present the results in Table 7.6.

```
regression_points <- get_regression_points(score_model_interaction)
regression_points
```

TABLE 7.6: Regression points (First 10 out of 463 courses)

ID	score	age	gender	score_hat	residual
1	4.7	36	female	4.25	0.448
2	4.1	36	female	4.25	-0.152
3	3.9	36	female	4.25	-0.352
4	4.8	36	female	4.25	0.548
5	4.6	59	male	4.20	0.399
6	4.3	59	male	4.20	0.099
7	2.8	59	male	4.20	-1.401
8	4.1	51	male	4.23	-0.133
9	3.4	51	male	4.23	-0.833
10	4.5	40	female	4.18	0.318

In fact, it turns out that the female instructor of age 36 taught the first four courses while the male instructor taught the next 3. The resulting $\hat{y} = \widehat{\text{score}}$ fitted values are in the `score_hat` column. Furthermore, `get_regression_points()` function also returns the residuals $y - \hat{y}$. Notice for example the first and fourth courses the female instructor of age 36 taught had positive residuals, indicating that the actual teaching score they received from students was less than their fitted score of 4.25. On the other hand the second and third course this instructor taught had negative residuals, indicating that the actual teaching score they received from students was more than their fitted score of 4.25.

Learning check

(LC7.1) Compute the observed values, fitted values, and residuals not for the interaction model as we just did, but rather for the parallel slopes model we saved in `score_model_interaction`.

7.2 Two numerical explanatory variables

Let's now switch gears and consider multiple regression models where instead of one numerical and one categorical explanatory variable, we have two numerical explanatory variables! The dataset we'll use is from An Introduction to Statistical Learning with Applications in R (ISLR)², an intermediate-level textbook on statistical and machine learning. It's accompanying `ISLR` R package contains datasets that the authors apply various machine learning methods to.

One frequently used dataset in this book `credit` dataset, where the outcome variable of interest is the credit card debt, in other words credit card debt, of 400 individuals. Other variables like income, credit limit, credit rating, and age are included as well. Note that the `credit` data is not based on real individuals' financial information, but rather is a simulated dataset used for educational purposes.

In this section, we'll fit a regression model where we have

1. A numerical outcome variable y , the cardholder's credit card debt
2. Two explanatory variables:
 1. One numerical explanatory variable x_1 , the cardholder's credit limit
 2. Another numerical explanatory variable x_2 , the cardholder's income (in thousands of dollars).

In the forthcoming Learning Checks, we'll consider a different regression model

1. The same numerical outcome variable y , the cardholder's credit card debt
2. Two different explanatory variables:

²<http://www-bcf.usc.edu/~gareth/ISL/>

1. One numerical explanatory variable x_1 , the cardholder's credit rating
2. Another numerical explanatory variable x_2 , the cardholder's age.

7.2.1 Exploratory data analysis

Let's load the `credit` data but to keep things simple to keep things simple, let's `select()` only the subset of the variables we'll consider in this chapter, and save this data in a new data frame called `credit_ch7`. Notice our slightly different use of the `select()` verb here: we'll select the `Balance` variable from `credit` for example, but we'll save it with a new variable name `debt` since this name is a little easier to understand.

```
library(ISLR)
credit_ch7 <- Credit %>%
  as_tibble() %>%
  select(ID, debt = Balance, credit_limit = Limit,
         income = Income, credit_rating = Rating, age = Age)
```

You can observe the effect of our different use of the `select()` verb in the first common step of an EDA: looking at the raw values either in RStudio's spreadsheet viewer or by using the `glimpse()`

```
glimpse(credit_ch7)

Observations: 400
Variables: 6
 $ ID           <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11...
 $ debt         <int> 333, 903, 580, 964, 331, 1151, 20...
 $ credit_limit <int> 3606, 6645, 7075, 9504, 4897, 804...
 $ income        <dbl> 14.9, 106.0, 104.6, 148.9, 55.9, ...
 $ credit_rating <int> 283, 483, 514, 681, 357, 569, 259...
 $ age          <int> 34, 82, 71, 36, 68, 77, 37, 87, 6...
```

Furthermore, let's look at a random sample of five out of the 400 credit card holders in Table 7.7. Note due to the random nature of the sampling, you will likely end up with a different subset of five rows.

```
credit_ch7 %>%
  sample_n(size = 5)
```

TABLE 7.7: Random sample of 5 credit card holders.

ID	debt	credit_limit	income	credit_rating	age
272	436	4866	45.0	347	30
239	52	2910	26.5	236	58
87	815	6340	55.4	448	33
108	0	3189	39.1	263	72
149	0	2420	15.2	192	69

Now that we've looked at the raw values in our `credit_ch7` data frame and obtained a sense of the data, let's move on to next common step in an exploratory data analysis: computing summary statistics. As you're probably used to now, let's use the `skim()` function from the `skimr` package, being sure to only `select()` the columns of interest for our model:

Let's look at some summary statistics, again using the `skim()` function from the `skimr` package:

```
credit_ch7 %>%
  select(debt, credit_limit, income) %>%
  skim()

Skim summary statistics
n obs: 400
n variables: 3

— Variable type:integer —
variable missing complete n mean sd p0 p25 p50 p75 p100
credit_limit 0 400 400 4735.6 2308.2 855 3088 4622.5 5872.75 13913
debt 0 400 400 520.01 459.76 0 68.75 459.5 863 1999

— Variable type:numeric —
variable missing complete n mean sd p0 p25 p50 p75 p100
income 0 400 400 45.22 35.24 10.35 21.01 33.12 57.47 186.63
```

Observe for example:

1. The mean and median credit card debt are \$520.01 and \$459.50 respectively.
2. 25% of card holders had debts of \$68.75 or less.
3. The mean and median credit card limit are \$4735.6 and \$4622.50 respectively.
4. 75% of these card holders had incomes of \$57,470 or less.

Since our outcome variable `debt` and the explanatory variables `credit_limit` and `income` are numerical, we can compute the correlation coefficient between pairs of these variables. First, we could run the `get_correlation()` command as seen in Subsection 6.1.1 twice, once for each explanatory variable:

```
credit_ch7 %>%
  get_correlation(debt ~ credit_limit)
credit_ch7 %>%
  get_correlation(debt ~ income)
```

Or we can simultaneously compute them by returning a *correlation matrix* which we display in Table 7.8. We can read off the correlation coefficient for any pair of variables by looking them up in the appropriate row/column combination.

```
credit_ch7 %>%
  select(debt, credit_limit, income) %>%
  cor()
```

TABLE 7.8: Correlation coefficients between credit card debt, credit limit, and income.

	debt	credit_limit	income
debt	1.000	0.862	0.464
credit_limit	0.862	1.000	0.792
income	0.464	0.792	1.000

For example, the correlation coefficient of:

1. `debt` with itself is 1 as we would expect based on the definition of the correlation coefficient.
2. `debt` with `credit_limit` is 0.862. This indicates a strong positive linear relationship, which makes sense as only individuals with large credit limits can accrue large credit card debts.
3. `debt` with `income` is 0.464. This is suggestive of another positive linear relationship, although not as strong as the relationship between `debt` and `credit_limit`.
4. As an added bonus, we can read off the correlation coefficient between the two explanatory variables, `credit_limit` and `income` of 0.792.

Let's visualize the relationship of the outcome variable with each of the two explanatory variables in two separate plots:

```
ggplot(credit_ch7, aes(x = credit_limit, y = debt)) +
  geom_point() +
  labs(x = "Credit limit (in $)", y = "Credit card debt (in $)",
       title = "Debt and credit limit") +
  geom_smooth(method = "lm", se = FALSE)

ggplot(credit_ch7, aes(x = income, y = debt)) +
  geom_point() +
  labs(x = "Income (in $1000)", y = "Credit card debt (in $)",
       title = "Debt and income") +
  geom_smooth(method = "lm", se = FALSE)
```

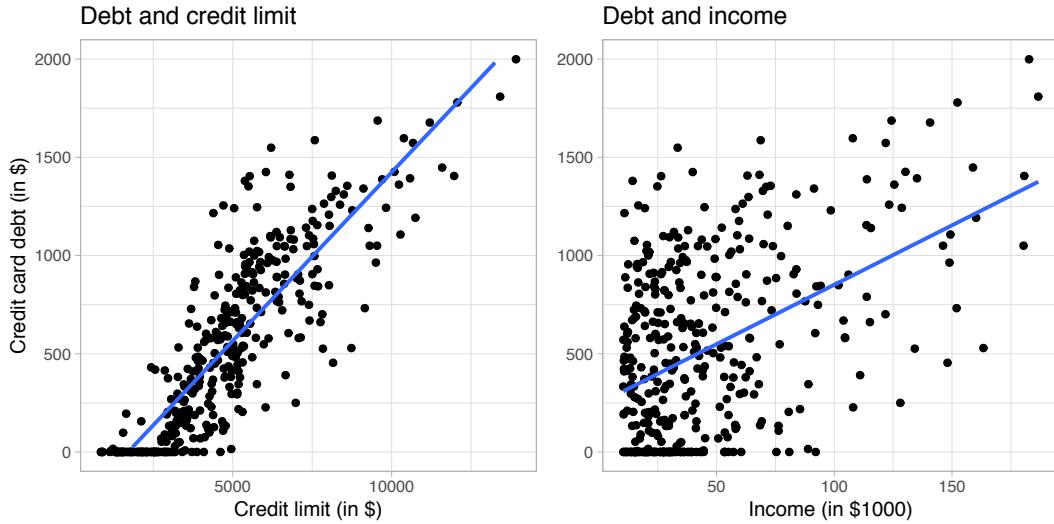


FIGURE 7.5: Relationship between credit card debt and credit limit/income.

Observe there is a positive relationship between credit limit and credit card debt: as credit limit increases so also does credit card debt. This is consistent with the strongly positive correlation coefficient of 0.862 we computed earlier. In the case of income, the positive relationship doesn't appear as strong, given the weakly positive correlation coefficient of 0.464.

However the two plots in Figure 7.5 only focus on the relationship of the outcome variable with each of the two explanatory variables separately. To get a sense of the *joint* relationship of all three variables simultaneously through

a visualization, we need a 3-dimensional (3D) scatterplot where for all 400 points we have

1. The numerical outcome variable y `debt` is on the z-axis (the vertical axis)
2. The two numerical explanatory variables form the axes on the bottom:
 1. The first numerical explanatory variable x_1 `income`
 2. The second numerical explanatory variable x_2 `credit_limit`

Furthermore, we also include a *regression plane*. In the case of regression models with a single numerical explanatory variable, we've seen in Section 6.3.2 that the regression line is "best fitting" in that of all possible lines we can draw through a cloud of points, it minimizes the sum of squared residuals. This concept now extends to when we have two numerical explanatory variables, only now we have a "best fitting" plane that cuts through the cloud of points that similarly minimizes the sum of squared residuals. If in the webpage version of the book, click here³ to open an interactive version of this plot in your browser.

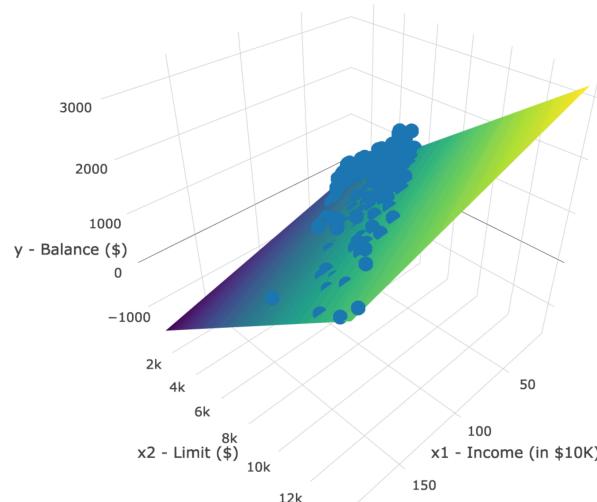


FIGURE 7.6: 3D scatterplot and regression plane.

Learning check

(LC7.2) Conduct a new exploratory data analysis with the same outcome

³<https://beta.rstudioconnect.com/connect/#/apps/3214/>

variable y being `debt` but with `credit_rating` and `age` as the new explanatory variables x_1 and x_2 . Remember, this involves three things:

1. Most crucially: Looking at the raw data values.
2. Computing summary statistics, like means, medians, and interquartile ranges.
3. Creating data visualizations.

What can you say about the relationship between a credit card holder's debt and their credit rating and age?

7.2.2 Regression plane

Let's now fit a regression model and get the regression table corresponding to the regression plane above. For simplicity's sake, we won't consider the two numerical explanatory variable analogue of the interaction model from Section 7.1.2 which we fit with a model formula of the form $y \sim x_1 * x_2$, but rather only regression models with model formula of the form $y \sim x_1 + x_2$. Somewhat confusing however, since we now have a regression plane instead of multiple lines, the label "parallel slopes model" doesn't apply when you have two numerical explanatory variables.

Just as we have done multiple times throughout Chapters 6 and this chapter, let's obtain the regression table for this model using our two-step process and display the results in Table 7.9

1. We first "fit" the linear regression model using the `lm(y ~ x1 + x2, data)` function and save it in `debt_model`.
2. We get the regression table by applying the `get_regression_table()` from the `moderndive` package to `debt_model`.

```
# Fit regression model:
debt_model <- lm(debt ~ credit_limit + income, data = credit_ch7)
# Get regression table:
get_regression_table(debt_model)
```

TABLE 7.9: Multiple regression table

term	estimate	std_error	statistic	p_value	lower_ci	upper_ci
intercept	-385.179	19.465	-19.8	0	-423.446	-346.912
credit_limit	0.264	0.006	45.0	0	0.253	0.276
income	-7.663	0.385	-19.9	0	-8.420	-6.906

How do we interpret the three values in the `estimate` column?

- `intercept` = -\$385.18 (rounded to two decimal points). The intercept in our case represents the credit card debt for an individual who has `credit_limit` of \$0 and `income` of \$0. In our data however, the intercept has limited practical interpretation since no individuals had `credit_limit` or `income` values of \$0. Rather, the intercept is used to situate the regression plane in 3D space.
- `credit_limit` = \$0.26. Taking into account all other the explanatory variables in our model, for every increase of one dollar in `credit_limit`, there is an associated increase of on average \$0.26 in credit card debt. Note:
 - Just as we did in Subsection 6.1.2, we are cautious not to make a causal statement by merely stating there was an *associated* increase.
 - We preface our interpretation with the statement “taking into account all other the explanatory variables in our model”, here `income`, to emphasize that we are now jointly interpreting the associated effect of multiple explanatory variables in the same model at once.
- `income` = -\$7.66. Taking into account all other the explanatory variables in our model, for every increase of one unit in the variable `income`, in other words \$1000 in actual income, there is an associated decrease of on average \$7.66 in credit card debt.

Putting these results together, the equation of the regression plane that gives us fitted values $\hat{y} = \text{debt}$.

$$\begin{aligned}\hat{y} &= b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 \\ \widehat{\text{debt}} &= b_0 + b_{\text{limit}} \cdot \text{limit} + b_{\text{income}} \cdot \text{income} \\ &= -387.179 + 0.263 \cdot \text{limit} - 7.663 \cdot \text{income}\end{aligned}$$

Recall in the right-hand plot of Figure 7.5 that when plotting the relationship between `debt` and `income` in isolation, there appeared to be a positive relationship. In the above multiple regression however, when jointly modeling the relationship between `debt`, `credit_limit`, and `income`, there appears to be a negative relationship of `debt` and `income` as evidenced by the negative slope for `income` of -\$7.66. What explains these contradictory results? A phenomenon known as Simpson's Paradox whereby overall trends that exist in aggregate

either disappear or reverse when the data are broken down into groups. In Subsection 7.3.3 we elaborate on this by looking at the relationship between `credit_limit` and credit card `debt`, but split by different income brackets.

Learning check

(LC7.3) Fit a new simple linear regression using `lm(debt ~ credit_rating + age, data = credit_ch7)` where `credit_rating` and `age` are the new numerical explanatory variables x_1 and x_2 . Get information about the “best-fitting” regression plane from the regression table by applying the `get_regression_table()` function. How do the regression results match up with the results from your exploratory data analysis above?

7.2.3 Observed/fitted values and residuals

Let’s also compute all fitted values and residuals for our regression model using the `get_regression_points()` function and present only the first 10 rows of output in Table 7.10. Remember that the (x, y, z) coordinates of each of the blue points in our 3D scatterplot can be found in the `income`, `credit_limit`, and `debt` columns. The fitted values on the regression plane are found in the `debt_hat` column and are computed using our equation for the regression plane in the previous section:

$$\hat{y} = \widehat{\text{debt}} = -387.179 + 0.263 \cdot \text{limit} - 7.663 \cdot \text{income}$$

```
regression_points <- get_regression_points(debt_model)
regression_points
```

TABLE 7.10: Regression points (First 10 card holders of 400)

ID	debt	credit_limit	income	debt_hat	residual
1	333	3606	14.9	454	-120.8
2	903	6645	106.0	559	344.3
3	580	7075	104.6	683	-103.4
4	964	9504	148.9	986	-21.7
5	331	4897	55.9	481	-150.0
6	1151	8047	80.2	1127	23.6
7	203	3388	21.0	349	-146.4
8	872	7114	71.4	948	-76.0
9	279	3300	15.1	371	-92.2
10	1350	6819	71.1	873	477.3

7.3 Related topics

7.3.1 Model selection

When do we use an interaction model versus a parallel slopes model? Recall in Sections 7.1.2 and 7.1.3 we fit both interaction and parallel slopes models for the outcome variable y teaching score using a numerical explanatory variable x_1 age and a categorical explanatory variable x_2 gender. We compared these models in Figure 7.3, which we display again below.

A lot of you might have asked yourselves: “Why would I force the lines to have parallel slopes (as seen in the right-hand plot) when they clearly have different slopes (as seen in the left-hand plot).”

The answer lies in a philosophical principle known as “Occam’s Razor” which states that “all other things being equal, simpler solutions are more likely to be correct than complex ones.” When viewed in a modeling framework, Occam’s Razor can be recast as “all other things being equal, simpler models are to be preferred over complex ones.” In other words, we should only favor the more complex model if the additional complexity is warranted.

Let’s revisit the equations for the regression line for both the interaction and parallel slopes model:

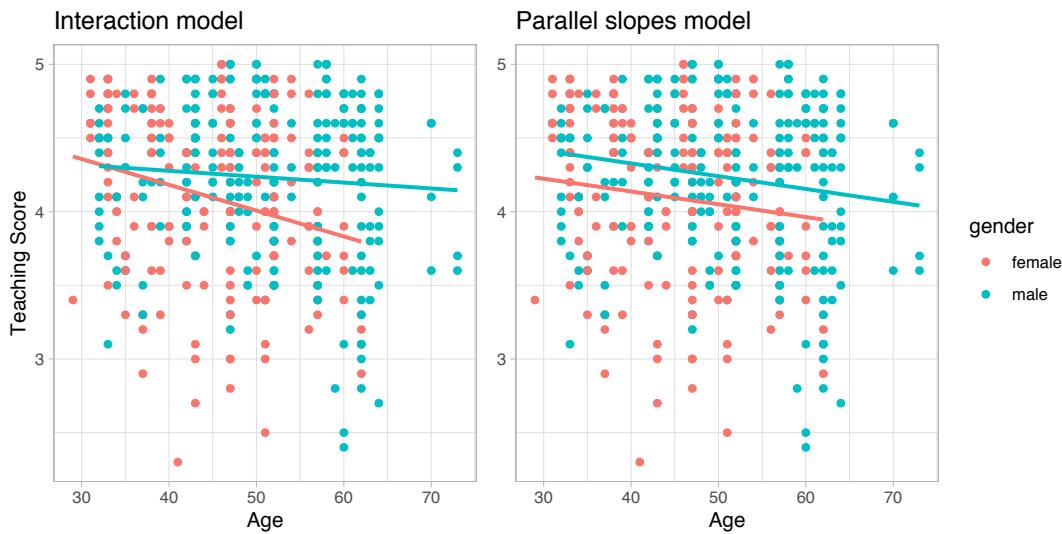


FIGURE 7.7: Previously seen comparison of interaction and parallel slopes models.

$$\text{Interaction : } \hat{y} = \widehat{\text{score}} = b_0 + b_{\text{age}} \cdot \text{age} + b_{\text{male}} \cdot \mathbb{1}_{\text{is male}}(x) + b_{\text{age,male}} \cdot \text{age} \cdot \mathbb{1}_{\text{is male}}$$

$$\text{Parallel slopes : } \hat{y} = \widehat{\text{score}} = b_0 + b_{\text{age}} \cdot \text{age} + b_{\text{male}} \cdot \mathbb{1}_{\text{is male}}(x)$$

The interaction model is “more complex” in that there is an additional $b_{\text{age,male}} \cdot \text{age} \cdot \mathbb{1}_{\text{is male}}$ element to the equation not present for the parallel slopes model. Or viewed alternatively, the regression table for the interaction model in Table 7.3 has *four* rows, whereas the regression table for the parallel slopes model in Table 7.5 has *three* rows. The question becomes: “Is this additional complexity warranted?” In this case, it can be argued that it is.

However, let’s consider an example where it might not be. Let’s consider the `MA_schools` data which contains 2017 data on Massachusetts public high schools provided by Massachusetts Department of Education; read the help file for this data by running `?MA_schools` if you would like more details. Let’s model

1. A numerical outcome variable y , average SAT math score for that high school
2. Two explanatory variables:
 1. A numerical explanatory variable x_1 , the percentage of that high school’s student body that are economically disadvantaged
 2. A categorical explanatory variable x_2 , the school size as measured by enrollment: small (13-341 students), medium (342-541 students), and large (542-4264 students)

Let's create visualizations of both the interaction and parallel slopes model once again and display the output in Figure 7.8.

```
# Interaction model
ggplot(MA_schools, aes(x = perc_disadvan, y = average_sat_math, color = size)) +
  geom_point(alpha = 0.25) +
  geom_smooth(method = "lm", se = FALSE ) +
  labs(x = "Percent economically disadvantaged", y = "Math SAT Score",
       color = "School size", title = "Interaction model")

# Parallel slopes model
gg_parallel_slopes(y = "average_sat_math", num_x = "perc_disadvan",
                     cat_x = "size", data = MA_schools, alpha = 0.25) +
  labs(x = "Percent economically disadvantaged", y = "Math SAT Score",
       color = "School size", title = "Parallel slopes model")
```

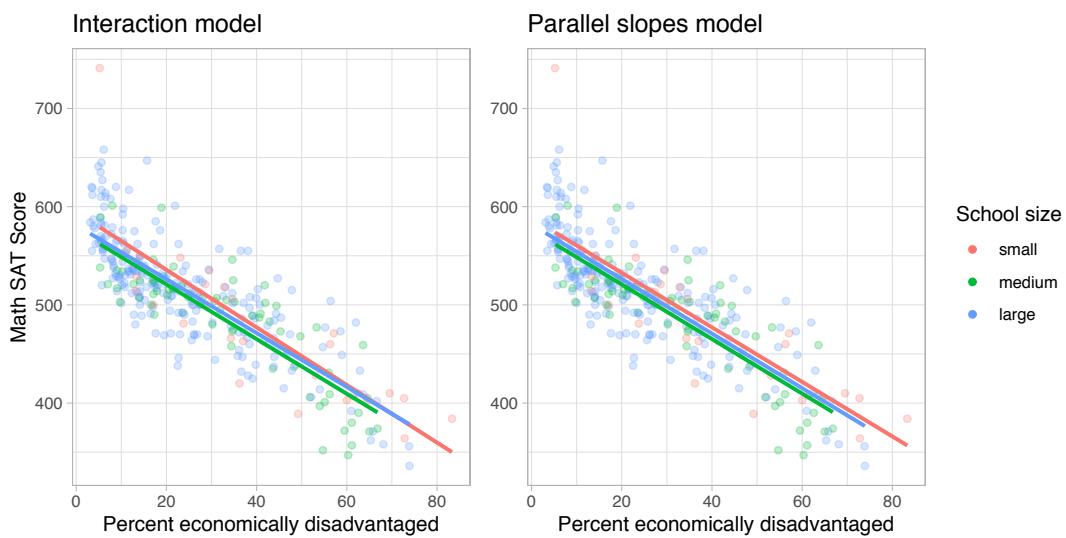


FIGURE 7.8: Comparison of interaction and parallel slopes models.

Looking closely at the left-hand plot of Figure 7.8, while the slopes are indeed different they are not different *by much*, in other words they are near identical. Comparing the left-hand plot with the right-hand plot, they don't appear all that different at all. In this case, it can be argued that the additional complexity of the interaction model is *not warranted* and thus by Occam's Razor the simpler parallel slopes model is to be preferred.

This additional complexity is apparent when comparing the corresponding regression tables in Tables 7.11 and 7.12; the regression table for the interaction model has 2 more rows. Furthermore, the *offsets* in slopes for percent-

age of students that are disadvantaged `perc_disadvan:sizemedium` = 0.146 and `perc_disadvan:sizelarge` = 0.189 are very small relative to the slope for the baseline group of small schools.

```
model_2_interaction <- lm(average_sat_math ~ perc_disadvan * size,
                           data = MA_schools)
get_regression_table(model_2_interaction)
```

TABLE 7.11: Interaction model regression table

term	estimate	std_error	statistic	p_value	lower_ci	upper_ci
intercept	594.327	13.288	44.726	0.000	568.186	620.469
perc_disadvan	-2.932	0.294	-9.961	0.000	-3.511	-2.353
sizemedium	-17.764	15.827	-1.122	0.263	-48.899	13.371
sizelarge	-13.293	13.813	-0.962	0.337	-40.466	13.880
perc_disadvan:sizemedium	0.146	0.371	0.393	0.694	-0.585	0.877
perc_disadvan:sizelarge	0.189	0.323	0.586	0.559	-0.446	0.824

```
model_2_parallel_slopes <- lm(average_sat_math ~ perc_disadvan + size,
                               data = MA_schools)
get_regression_table(model_2_parallel_slopes)
```

TABLE 7.12: Parallel slopes regression table

term	estimate	std_error	statistic	p_value	lower_ci	upper_ci
intercept	588.19	7.607	77.325	0.000	573.23	603.15
perc_disadvan	-2.78	0.106	-26.120	0.000	-2.99	-2.57
sizemedium	-11.91	7.535	-1.581	0.115	-26.74	2.91
sizelarge	-6.36	6.923	-0.919	0.359	-19.98	7.26

These results are suggesting that irrespective of school size, the relationship between average math SAT scores and the percent of the student body that is economically disadvantaged is alas very negative.

What you have just performed is a rudimentary *model selection*: choosing which model fits data best among a set of candidate models. While the model selection you just performed was somewhat qualitative fashion, more statistically rigorous methods exist. If you're curious, take a course on multiple regression!

7.3.2 Correlation coefficient

Recall from Table 7.8 that the correlation coefficient between `income` in thousands of dollars and credit card `debt` was 0.464. What if instead we looked at the correlation coefficient between `income` and credit card `debt`, but where `income` was in dollars and not thousands of dollars? This can be done by multiplying `income` by 1000.

```
library(ISLR)
credit_ch7 %>%
  select(debt, income) %>%
  mutate(income = income * 1000) %>%
  cor()
```

TABLE 7.13: Correlation between income (in dollars) and credit card debt

	debt	income
debt	1.000	0.464
income	0.464	1.000

We see it is the same! We say that the correlation coefficient is invariant to linear transformations! In other words, the correlation between x and y will be the same as the correlation between $a \times x + b$ and y for any numerical values a and b .

7.3.3 Simpson's Paradox

Recall in Section 7.2, we saw the two following seemingly contradictory results when studying the relationship between credit card debt, credit limit, and income. On the one hand, the right hand plot of Figure 7.5 suggested that credit card debt and income were positively related:

On the other hand, the multiple regression in Table 7.9, suggested that when modeling credit card debt as a function of *both* `credit_limit` and `income` at the same time, credit limit has a negative relationship with credit card debt as evidenced by the slope of -7.66. How can this be?

First, let's dive a little deeper into the explanatory variable `credit_limit`. Figure 7.10 shows a histogram of all 400 values of `credit_limit`, along with vertical red lines that cut up the data into quartiles, meaning:

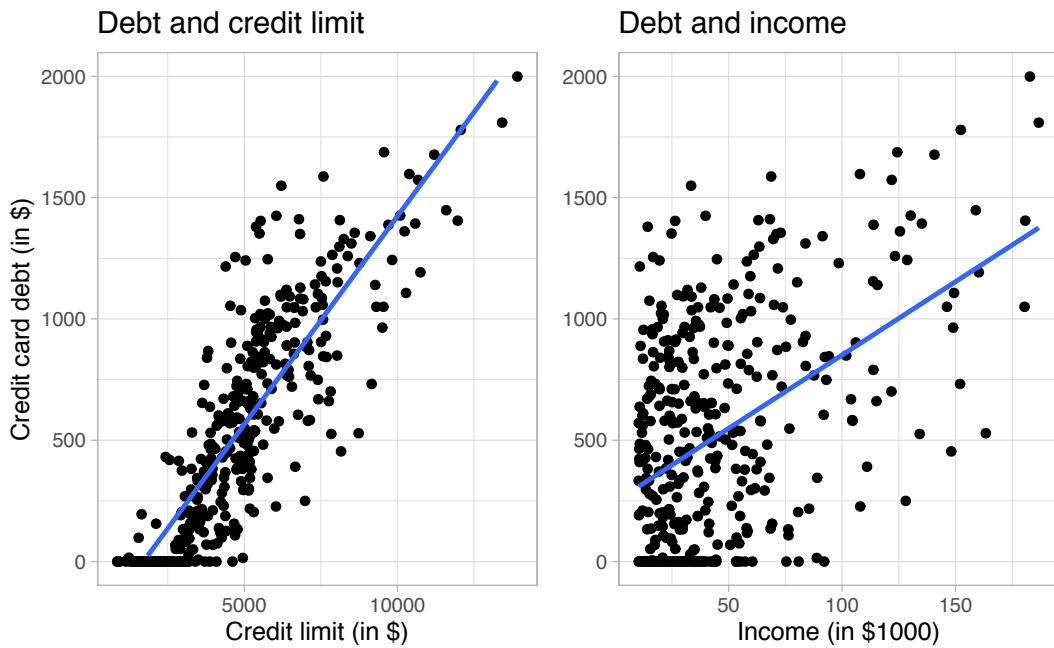


FIGURE 7.9: Relationship between credit card debt and credit limit/income.

1. 25% of credit limits were between \$0 and \$3088. Let's call this the `low` credit limit group.
2. 25% of credit limits were between \$3088 and \$4622. Let's call this the `medium-low` credit limit group.
3. 25% of credit limits were between \$4622 and \$5873. Let's call this the `medium-high` credit limit group.
4. 25% of credit limits were over \$5873. Let's call this the `high` credit limit group.

In Figure 7.11 let's display

1. In the left-hand plot: The scatterplot showing the relationship between credit card `debt` and `credit_limit` from earlier.
2. In the right-hand plot: The same exact same scatterplot both now with color indicating

The left-hand plot focuses of the relationship between debt and income in *aggregate*, which suggests a positive relationship between income and credit card debt. However, the right-hand plot focuses on the relationship between debt and income *broken down by credit limit group*, where we observe that the `low` (red points), `medium-low` (green points), and `medium-high` (blue points) income groups, the strong positive relationship between credit card debt and income

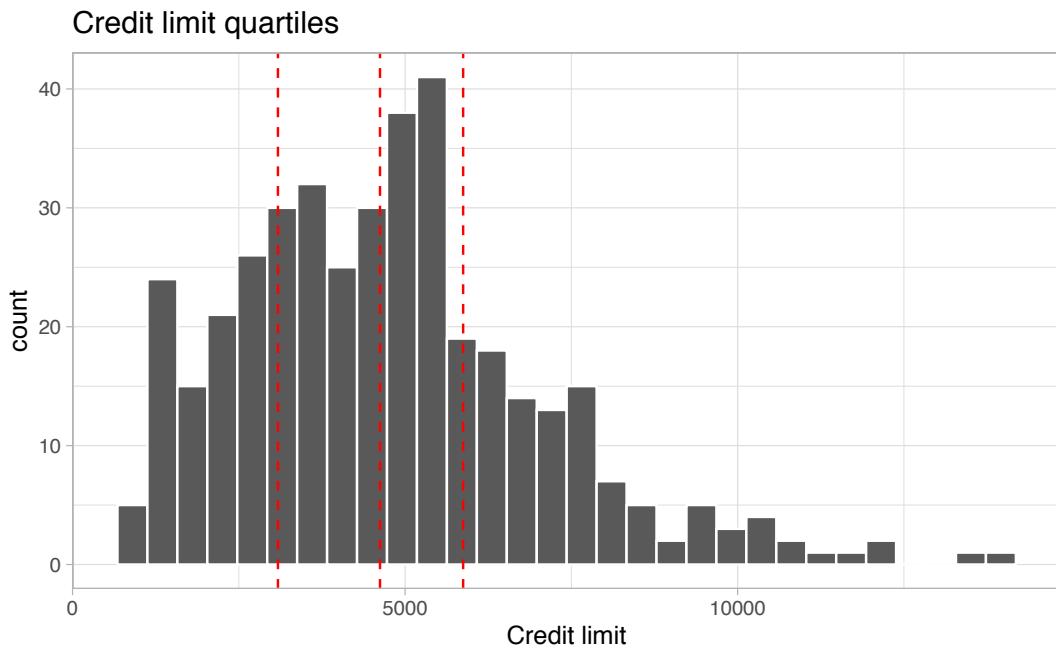


FIGURE 7.10: Histogram of credit limits and quartiles.

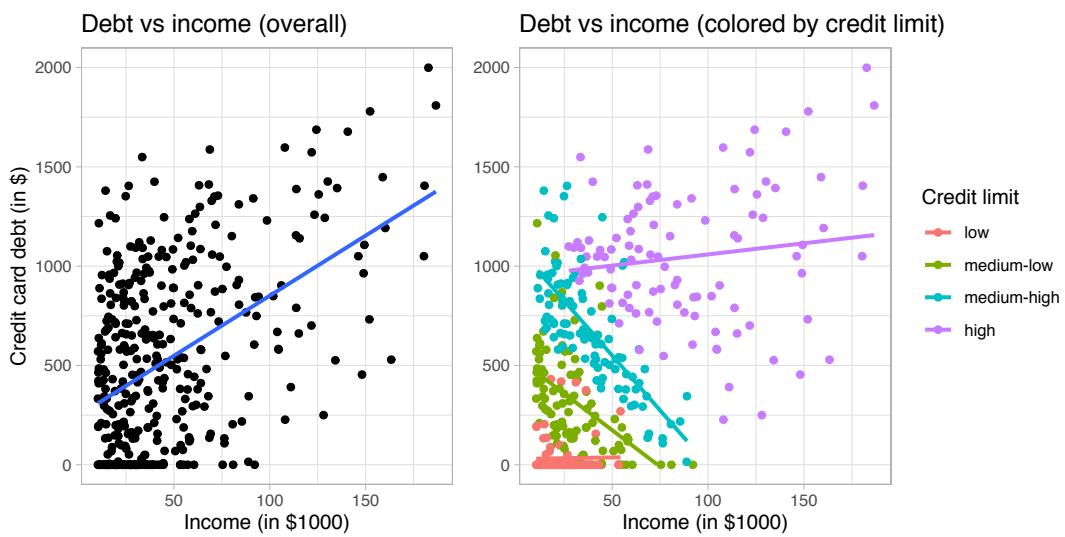


FIGURE 7.11: Relationship between credit card debt and income for different credit limit groups.

disappears! Only for the high bracket does the relationship stay somewhat positive. In this example, credit limit is a *confounding variable* for credit card debt and income. This is a phenomenon known as Simpson's Paradox⁴ whereby overall trends that exist in aggregate either disappear or reverse when the data are broken down into groups.

7.4 Conclusion

7.4.1 Additional resources

An R script file of all R code used in this chapter is available here⁵.

7.4.2 What's to come?

Congratulations! We've completed the "Data Modeling via moderndive" portion of this book! We're ready to proceed to the third and final portion of this book: "Statistical Inference via infer." Statistical inference is the science of inferring about some unknown quantity using sampling. Among the most well-known example of sampling are polls. Because asking an entire population about their opinions would be a long and arduous task, pollsters often take a smaller sample that is hopefully representative of the population. Based on the results of the sample, pollsters hope to make claims about the greater population.

Once we've covered Chapters 8 on sampling, 9 on confidence intervals, and 10 on hypothesis testing, in Chapter 11 on inference for regression we'll revisit the regression models we studied in Chapter 6 and 7. So far we've only studied the `estimate` column of all our regression tables. The next 4 chapters focus on what the remaining columns mean: `std_error` standard error, `statistic` test statistic, `p_value` p-value, `lower_ci` lower 95% confidence interval bound, and `upper_ci` upper 95% confidence interval bound.

Furthermore, we'll talk about the importance of residuals $y - \hat{y}$ play in interpreting the results of a regression. We'll perform what is known as *residual analyses* of the `residual` variable of all `get_regression_points()` output to verify what are known as the "conditions for inference for regression." On to the next one!

⁴https://en.wikipedia.org/wiki/Simpson%27s_paradox

⁵[scripts/07-multiple-regression.R](#)

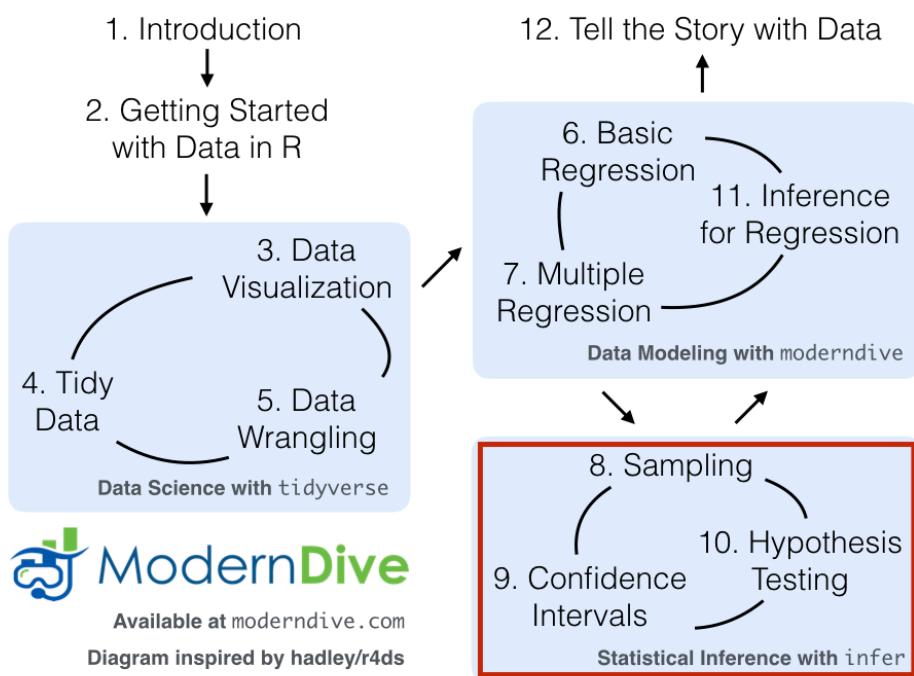


FIGURE 7.12: ModernDive flowchart - On to Part III!

Part III

Statistical Inference via infer

8

Sampling

In this chapter, we kick off the third segment of this book, statistical inference, by learning about **sampling**. The concepts behind sampling form the basis of confidence intervals and hypothesis testing, which we'll cover in Chapters 9 and 10 respectively. We will see that the tools that you learned in the data science segment of this book, in particular, data visualization and data wrangling, will also play an important role here in the development of your understanding. As mentioned before, the concepts throughout this text all build into a culmination allowing you to “think with data.”

Needed packages

Let's load all the packages needed for this chapter (this assumes you've already installed them). Recall from our discussion in Section 5.4.1 that loading the `tidyverse` package by running `library(tidyverse)` loads the following commonly used data science packages all at once:

- `ggplot2` for data visualization
- `dplyr` for data wrangling
- `tidyR` for converting data to “tidy” format
- `readr` for importing spreadsheet data into R
- As well as the more advanced `purrr`, `tibble`, `stringr`, and `forcats` packages

If needed, read Section 2.3 for information on how to install and load R packages.

```
library(tidyverse)
library(moderndive)
```

8.1 Sampling bowl activity

Let's start with a hands-on activity.

8.1.1 What proportion of this bowl's balls are red?

Take a look at the bowl in Figure 8.1. It has a certain number of red and a certain number of white balls all of equal size. Furthermore, it appears the bowl has been mixed beforehand as there does not seem to be any particular pattern to the spatial distribution of red and white balls.

Let's now ask ourselves, what proportion of this bowl's balls are red?

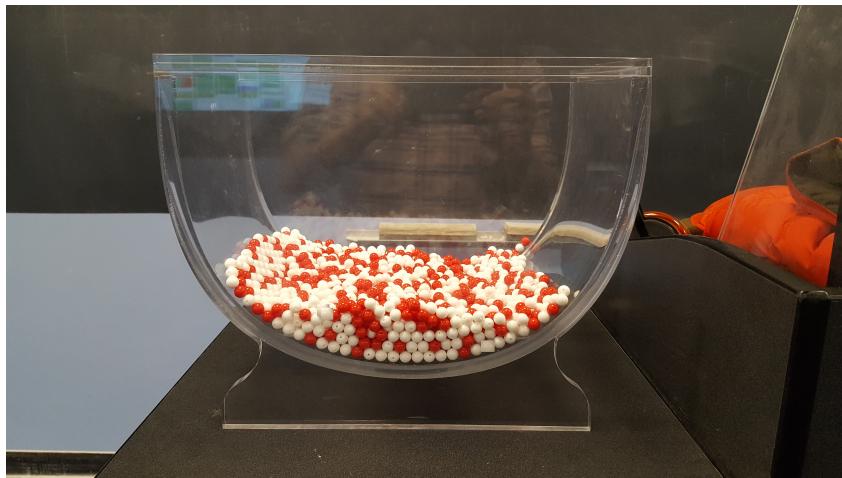


FIGURE 8.1: A bowl with red and white balls.

One way to answer this question would be to perform an exhaustive count: remove each ball individually, count the number of red balls and the number of white balls, and divide the number of red balls by the total number of balls. However, this would be a long and tedious process.

8.1.2 Using the shovel once

Instead of performing an exhaustive count, let's insert a shovel into the bowl as seen in Figure 8.2.

Using the shovel, we remove a number of balls as seen in Figure 8.3.

Observe that 17 of the balls are red and there are a total of $5 \times 10 = 50$ balls and thus $0.34 = 34\%$ of the shovel's balls are red. We can view the proportion of balls that are red *in this shovel* as a guess of the proportion of balls that are red *in the entire bowl*. While not as exact as doing an exhaustive count, our guess of 34% took much less time and energy to obtain.

However, say, we started this activity over from the beginning. In other words, we replace the 50 balls back into the bowl and start over. Would we remove



FIGURE 8.2: Inserting a shovel into the bowl.

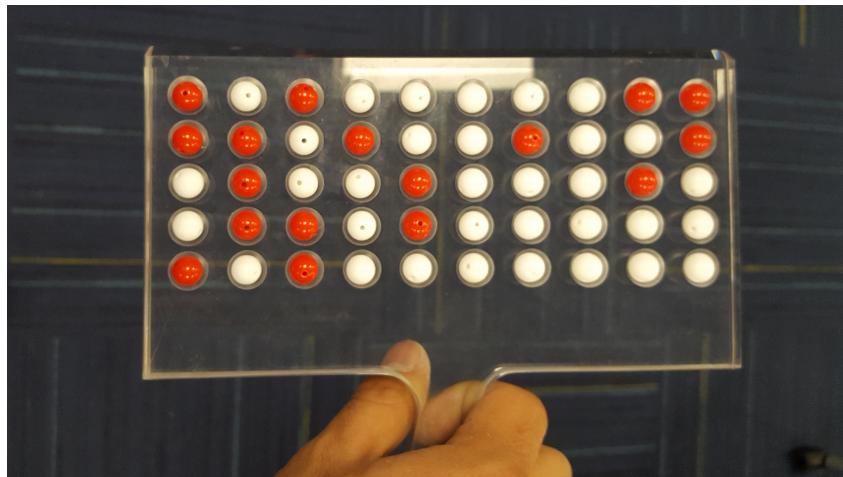


FIGURE 8.3: Fifty balls from the bowl.

exactly 17 red balls again? In other words, would our guess at the proportion of the bowl's balls that are red be exactly 34% again? Maybe?

What if we repeated this exercise several times? Would we obtain exactly 17 red balls each time? In other words, would our guess at the proportion of the bowl's balls that are red be exactly 34% every time? Surely not. Let's actually do and observe the results with the help of 33 of our friends.

8.1.3 Using the shovel 33 times

Each of our 33 friends will do the following:

- use the shovel to remove 50 balls each,
- count the number of red balls,
- use this number to compute the proportion of the 50 balls they removed that are red,
- return the balls into the bowl, and
- mix the contents of the bowl a little to not let a previous group's results influence the next group's set of results.



FIGURE 8.4: Repeating sampling activity 33 times.

However, before returning the balls into the bowl, they are going to mark the proportion of the 50 balls they removed that are red in a histogram as seen in Figure 8.5.

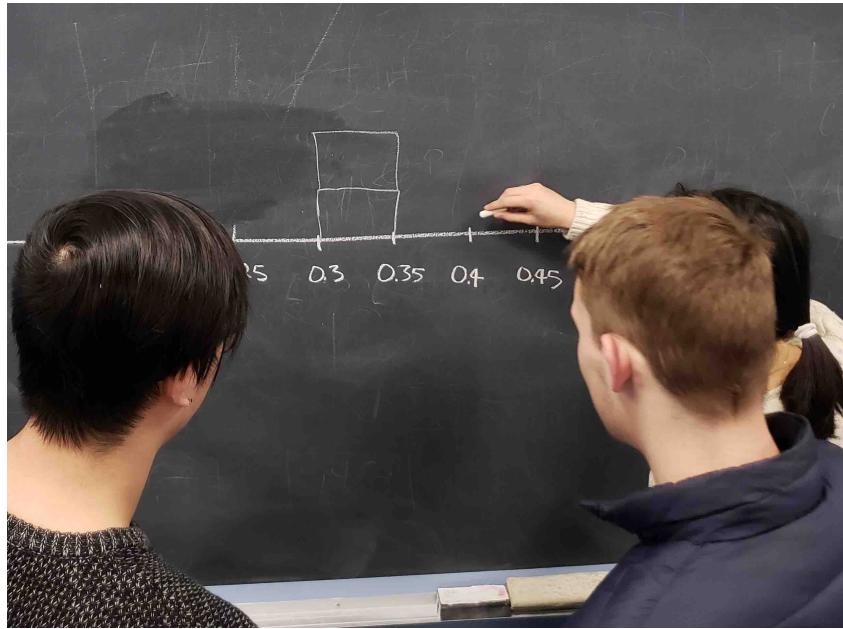


FIGURE 8.5: Constructing a histogram of proportions.

Recall from Section 3.5 that histograms allow us to visualize the *distribution*

of a numerical variable: where the values center and in particular how they vary. The resulting hand-drawn histogram can be seen in Figure 8.6.

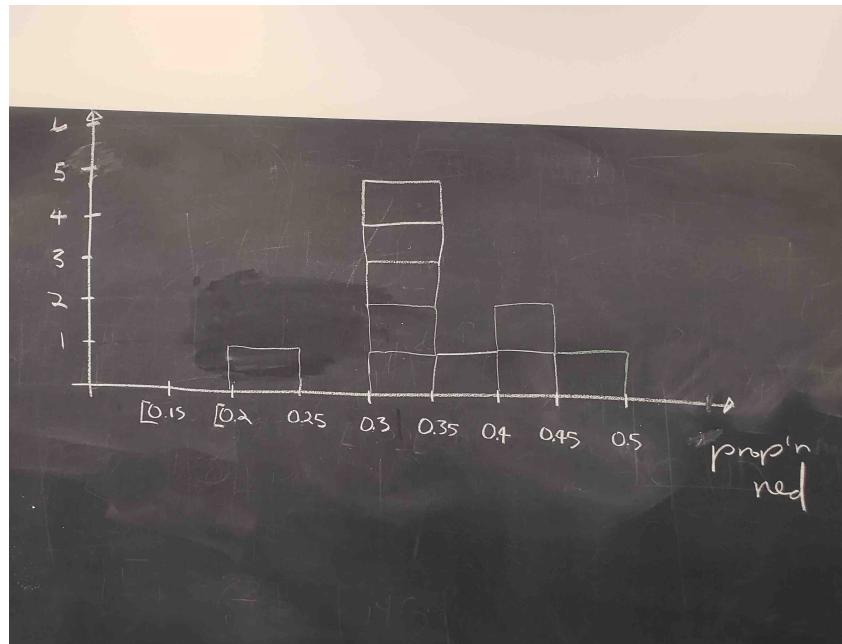


FIGURE 8.6: Hand-drawn histogram of 33 proportions.

Observe the following about the histogram in Figure 8.6:

- At the low end, one group removed 50 balls from the bowl with proportion between $0.20 = 20\%$ and $0.25 = 25\%$.
- At the high end, another group removed 50 balls from the bowl with proportion between $0.45 = 45\%$ and $0.5 = 50\%$ red.
- However the most frequently occurring proportions were between $0.30 = 30\%$ and $0.35 = 35\%$ red, right in the middle of the distribution.
- The shape of this distribution is somewhat bell-shaped.

Let's construct this same hand-drawn histogram in R using your data visualization skills that you honed in Chapter 3. We saved our 33 group of friends' proportion red in a data frame `tactile_prop_red` which is included in the `moderndive` package you loaded earlier.

```
tactile_prop_red  
View(tactile_prop_red)
```

Let's display only the first 10 out of 33 rows of `tactile_prop_red`'s contents in Table 8.1.

TABLE 8.1: First 10 out of 33 groups' proportion of 50 balls that are red.

group	replicate	red_balls	prop_red
Ilyas, Yohan	1	21	0.42
Morgan, Terrance	2	17	0.34
Martin, Thomas	3	21	0.42
Clark, Frank	4	21	0.42
Riddhi, Karina	5	18	0.36
Andrew, Tyler	6	19	0.38
Julia	7	19	0.38
Rachel, Lauren	8	11	0.22
Daniel, Caroline	9	15	0.30
Josh, Maeve	10	17	0.34

Observe for each `group` we have their names, the number of `red_balls` they obtained, and the corresponding proportion out of 50 balls that were red named `prop_red`. Observe, we also have a variable `replicate` enumerating each of the 33 groups; we chose this name because each row can be viewed as one instance of a replicated activity: using the shovel to remove 50 balls and computing the proportion of those balls that are red.

We visualize the distribution of these 33 proportions using a `geom_histogram()` with `binwidth = 0.05` in Figure 8.7, which is appropriate since the variable `prop_red` is numerical. This computer-generated histogram matches our hand-drawn histogram from the earlier Figure 8.6.

```
ggplot(tactile_prop_red, aes(x = prop_red)) +
  geom_histogram(binwidth = 0.05, boundary = 0.4, color = "white") +
  labs(x = "Proportion of 50 balls that were red",
       title = "Distribution of 33 proportions red")
```

8.1.4 What did we just do?

What we just demonstrated in this activity is the statistical concept of *sampling*. We would like to know the proportion of the bowl's balls that are red, but because the bowl has a very large number of balls performing an exhaustive count of the number of red and white balls in the bowl would be very costly in terms of both time and energy. We, therefore, extract a sample of 50 balls using the shovel. Using this sample of 50 balls, we estimate the proportion of the bowl's balls that are red using the proportion of the shovel's balls that are red. This estimate in our earlier example was 17 red balls out of 50 balls = 34%. Moreover, because we mixed the balls before each use of the

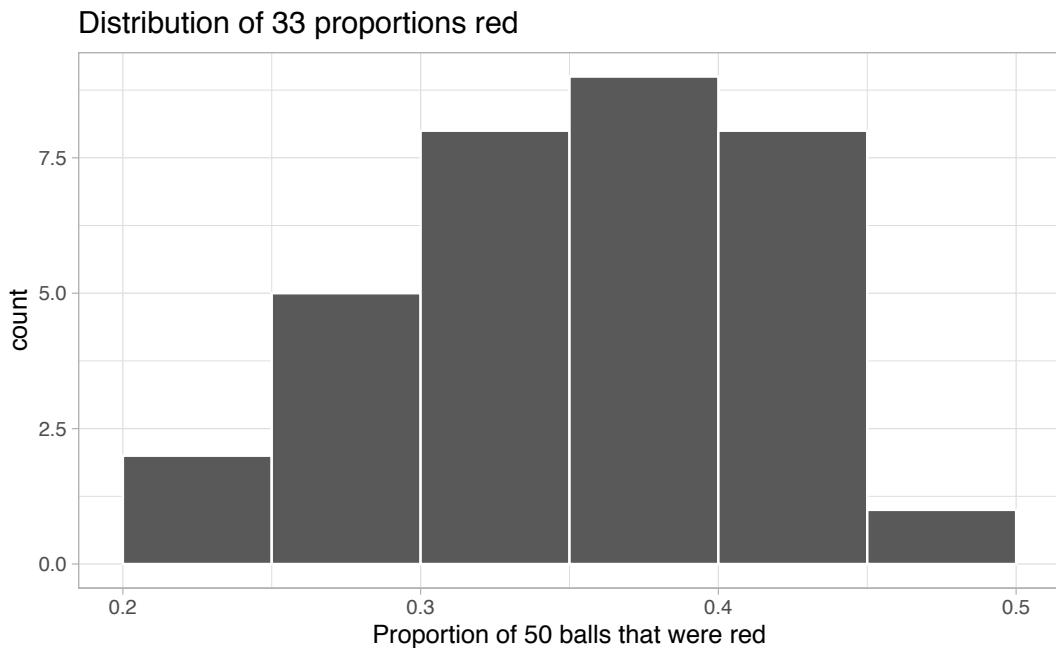


FIGURE 8.7: Distribution of 33 proportions based on 33 samples of size 50.

shovel, the samples were randomly drawn. Because each sample was drawn at random, the samples were different from each other. Because the samples were different from each other, we obtained the different proportions red observed in Table 8.1. This is known as the concept of *sampling variation*.

In Section 8.2 we'll mimic the hands-on sampling activity we just performed in a *computer simulation*; using a computer will allow us to repeat the above sampling activity much more than 33 times. Using a computer, not only will be able to repeat the hands-on activity a very large number of times, but we will also be able to repeat it using different sized shovels.

The purpose of these simulations is to develop an understanding of two key concepts relating to sampling: understanding the concept of sampling variation and the role that sample size plays in this variation. To this end, we'll present you with definitions, terminology, and notation related to sampling in Section 8.3. As with many disciplines, there are definitions, terminology, and notation that seem very inaccessible and even confusing at first. However, as with many difficult topics, if you truly understand the underlying concepts and practice, practice, practice, you'll be able to master these topics.

To tie the contents of this chapter to the real-word, we'll present an example of one of the most recognizable uses of sampling: polls. In Section 8.4 we'll look at a particular case study: a 2013 poll on then President Obama's popu-

larity among young Americans, conducted by the Harvard Kennedy School's Institute of Politics.

We'll close this chapter by generalizing the above sampling from the bowl activity to other scenarios, distinguishing between *random sampling* and *random assignment*, presenting the theoretical result underpinning all our results, and presenting a few mathematical formulas that relate to the concepts and ideas explored in this chapter.

Learning check

(LC8.1) Why was it important to mix the bowl before we sampled the balls?

(LC8.2) Why is it that our 33 groups of friends did not all have the same numbers of balls that were red out of 50, and hence different proportions red?

8.2 Computer simulation of sampling

What we performed in Section 8.1 is a *simulation* of sampling. In other words, we were not in a real-life sampling scenario in order to answer a real-life question, but rather we were mimicking such a scenario with our bowl and shovel. The crowd-sourced Wikipedia definition of simulation states: “A simulation is an approximate imitation of the operation of a process or system.” One example of simulations in practice is flight simulators: before pilots in training are allowed to fly an actual plane, they first practice on a computer that attempts to mimic the reality of flying an actual plane as best as possible.

Now you might be thinking that simulations must necessarily take place on a computer. However, this is not necessarily true. Take for example crash test dummies: before cars are made available to the market, automobile engineers test their safety by mimicking the reality for passengers of being in an automobile crash. To distinguish between these two simulation types, we'll term a simulation performed in real-life as a “tactile” simulation done with your hands and to the touch as opposed to a “virtual” simulation performed on a computer.

So while in Section 8.1 we performed a “tactile” simulation of sampling using an actual bowl and an actual shovel with our hands, in this section we'll

perform a “virtual” simulation using a “virtual” bowl and a “virtual” shovel with our computers.

8.2.1 Using the virtual shovel once

Let’s start by performing the virtual analogue of the tactile sampling simulation we performed in 8.1. We first need a virtual analogue of the bowl seen in Figure 8.1. To this end, we included a data frame `bowl` in the `moderndive` package whose rows correspond exactly with the contents of the actual bowl.

```
bowl
```

```
# A tibble: 2,400 × 2
  ball_ID color
  <int> <chr>
1      1 white
2      2 white
3      3 white
4      4 red
5      5 white
6      6 white
7      7 red
8      8 white
9      9 red
10     10 white
# ... with 2,390 more rows
```

Observe in the output that `bowl` has 2400 rows, telling us that the bowl contains 2400 equally-sized balls. The first variable `ball_ID` is used merely as an “identification variable” for this data frame as discussed in Subsection 2.4.4; none of the balls in the actual bowl are marked with numbers. The second variable `color` indicates whether a particular virtual ball is red or white. View the contents of the bowl in RStudio’s data viewer and scroll through the contents to convince yourselves that `bowl` is indeed a virtual version of the actual bowl in Figure 8.1.

Now that we have a virtual analogue of our bowl, we now need a virtual analogue for the shovel seen in Figure 8.2; we’ll use this virtual shovel to generate our virtual random samples of 50 balls. We’re going to use the `rep_sample_n()` function included in the `moderndive` package. This function allows us to take repeated, or `replicated`, `samples` of size `n`. Run the following and explore `virtual_shovel`’s contents in the RStudio viewer.

```
virtual_shovel <- bowl %>%
  rep_sample_n(size = 50)
View(virtual_shovel)
```

Let's display only the first 10 out of 50 rows of `virtual_shovel`'s contents in Table 8.2.

TABLE 8.2: First 10 sampled balls of 50 in virtual sample

replicate	ball_ID	color
1	1970	white
1	842	red
1	2287	white
1	599	white
1	108	white
1	846	red
1	390	red
1	344	white
1	910	white
1	1485	white

The `ball_ID` variable identifies which of the balls from `bowl` are included in our sample of 50 balls and `color` denotes its color. However what does the `replicate` variable indicate? In `virtual_shovel`'s case, `replicate` is equal to 1 for all 50 rows. This is telling us that these 50 rows correspond to a first repeated/replicated use of the shovel, in our case our first sample. We'll see below when we "virtually" take 33 samples, `replicate` will take values between 1 and 33. Before we do this, let's compute the proportion of balls in our virtual sample of size 50 that are red using the `dplyr` data wrangling verbs you learned in Chapter 4. Let's breakdown the steps individually:

First, for each of our 50 sampled balls, identify if it is red using a test for equality using `==`. For every row where `color == "red"`, the Boolean `TRUE` is returned and for every row where `color` is not equal to "red", the Boolean `FALSE` is returned. Let's create a new Boolean variable `is_red` using the `mutate()` function from Section 4.5:

```
virtual_shovel %>%
  mutate(is_red = (color == "red"))
```

```
# A tibble: 50 x 4
# Groups:   replicate [1]
```

```

replicate ball_ID color is_red
  <int>  <int> <chr> <lgl>
1       1    1970 white FALSE
2       1     842 red   TRUE
3       1    2287 white FALSE
4       1     599 white FALSE
5       1     108 white FALSE
6       1     846 red   TRUE
7       1     390 red   TRUE
8       1     344 white FALSE
9       1     910 white FALSE
10      1    1485 white FALSE
# ... with 40 more rows

```

Second, we compute the number of balls out of 50 that are red using the `summarize()` function. Recall from Section 4.3 that `summarize()` takes a data frame with many rows and returns a data frame with a single row containing summary statistics that you specify, like `mean()` and `median()`. In this case we use the `sum()`:

```

virtual_shovel %>%
  mutate(is_red = (color == "red")) %>%
  summarize(num_red = sum(is_red))

```

```

# A tibble: 1 x 2
  replicate num_red
  <int>    <int>
1       1      12

```

Why does this work? Because R treats `TRUE` like the number 1 and `FALSE` like the number 0. So summing the number of `TRUE`'s and `FALSE`'s is equivalent to summing 1's and 0's, which in the end counts the number of balls where `color` is `red`. In our case, 17 of the 50 balls were red.

Third and last, we compute the proportion of the 50 sampled balls that are red by dividing `num_red` by 50:

```

virtual_shovel %>%
  mutate(is_red = color == "red") %>%
  summarize(num_red = sum(is_red)) %>%
  mutate(prop_red = num_red / 50)

```

```
# A tibble: 1 x 3
```

```
replicate num_red prop_red
<int> <int> <dbl>
1       1      12     0.24
```

In other words, this “virtual” sample’s balls were 34% red. Let’s make the above code a little more compact and succinct by combining the first `mutate()` and the `summarize()` as follows:

```
virtual_shovel %>%
  summarize(num_red = sum(color == "red")) %>%
  mutate(prop_red = num_red / 50)
```

```
# A tibble: 1 x 3
  replicate num_red prop_red
    <int> <int> <dbl>
1       1      12     0.24
```

Great! 34% of `virtual_shovel`’s 50 balls were red! So based on this particular sample, our guess at the proportion of the `bowl`’s balls that are red is 34%. But remember from our earlier tactile sampling activity that if we repeated this sampling, we would not necessarily obtain a sample of 50 balls with 34% of them being red again; there will likely be some variation. In fact in Table 8.2 we displayed 33 such proportions based on 33 tactile samples and then in Figure 8.6 we visualized the distribution of the 33 proportions in a histogram. Let’s now perform the virtual analogue of having 33 groups of students use the sampling shovel!

8.2.2 Using the virtual shovel 33 times

Recall that in our tactile sampling exercise in Section 8.1 we had 33 groups of students each use the shovel, yielding 33 samples of size 50 balls, which we then used to compute 33 proportions. In other words we repeated/replicated using the shovel 33 times. We can perform this repeated/replicated sampling virtually by once again using our virtual shovel function `rep_sample_n()`, but by adding the `reps = 33` argument, indicating we want to repeat the sampling 33 times. Be sure to scroll through the contents of `virtual_samples` in RStudio’s viewer.

```
virtual_samples <- bowl %>%
  rep_sample_n(size = 50, reps = 33)
View(virtual_samples)
```

Observe that while the first 50 rows of `replicate` are equal to 1, the next 50 rows of `replicate` are equal to 2. This is telling us that the first 50 rows correspond to the first sample of 50 balls while the next 50 correspond to the second sample of 50 balls. This pattern continues for all `reps = 33` replicates and thus `virtual_samples` has $33 \times 50 = 1650$ rows.

Let's now take the data frame `virtual_samples` with $33 \times 50 = 1650$ rows corresponding to 33 samples of size 50 balls and compute the resulting 33 proportions red. We'll use the same `dplyr` verbs as we did in the previous section, but this time with an additional `group_by()` of the `replicate` variable. Recall from Section 4.4 that by assigning the grouping variable “meta-data” before `summarizing()`, we'll obtain 33 different proportions red:

```
virtual_prop_red <- virtual_samples %>%
  group_by(replicate) %>%
  summarize(red = sum(color == "red")) %>%
  mutate(prop_red = red / 50)
View(virtual_prop_red)
```

Let's display only the first 10 out of 33 rows of `virtual_prop_red`'s contents in Table 8.3. As one would expect, there is variation in the resulting `prop_red` proportions red for the first 10 out 33 repeated/replicated samples.

TABLE 8.3: First 10 out of 33 virtual proportion of 50 balls that are red.

replicate	red	prop_red
1	23	0.46
2	19	0.38
3	18	0.36
4	19	0.38
5	15	0.30
6	21	0.42
7	21	0.42
8	16	0.32
9	24	0.48
10	14	0.28

Let's visualize the distribution of these 33 proportions red based on 33 virtual samples using a histogram with `binwidth = 0.05` in Figure 8.8.

```
ggplot(virtual_prop_red, aes(x = prop_red)) +
  geom_histogram(binwidth = 0.05, boundary = 0.4, color = "white") +
```

```
labs(x = "Proportion of 50 balls that were red",
     title = "Distribution of 33 proportions red")
```

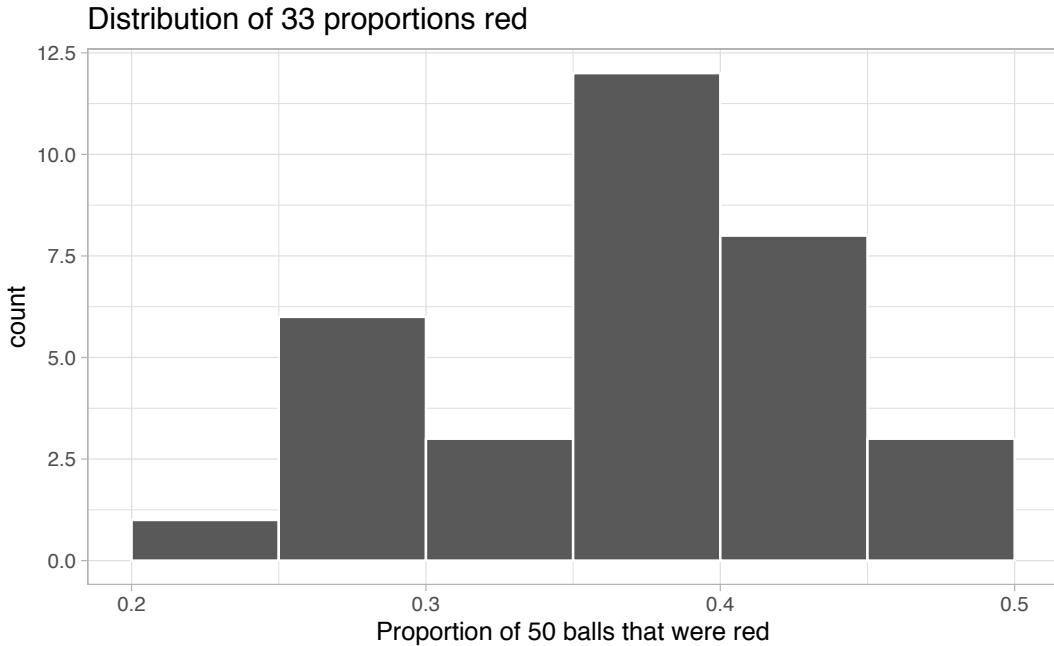


FIGURE 8.8: Distribution of 33 proportions based on 33 samples of size 50.

Observe that occasionally we obtained proportions red that is less than $0.3 = 30\%$, while on the other hand, occasionally we obtained proportions that are greater than $0.45 = 45\%$. However, the most frequently occurring proportions red out of 50 balls were between 35% and 40% (for 11 out of 33 samples). Why do we have these differences in proportions red? Because of sampling variation.

Let's now compare our virtual results with our tactile results from the previous section in Figure 8.9. We see that both histograms, in other words, the distribution of the 33 proportions red, are *somewhat* similar in their center and spread although not identical. These slight differences are again due to random variation. Furthermore, both distributions are *somewhat* bell-shaped.

Learning check

(LC8.3) Why couldn't we study the effects of sampling variation when we used the virtual shovel only once? Why did we need to take more than one virtual sample (in our case 33 virtual samples)?

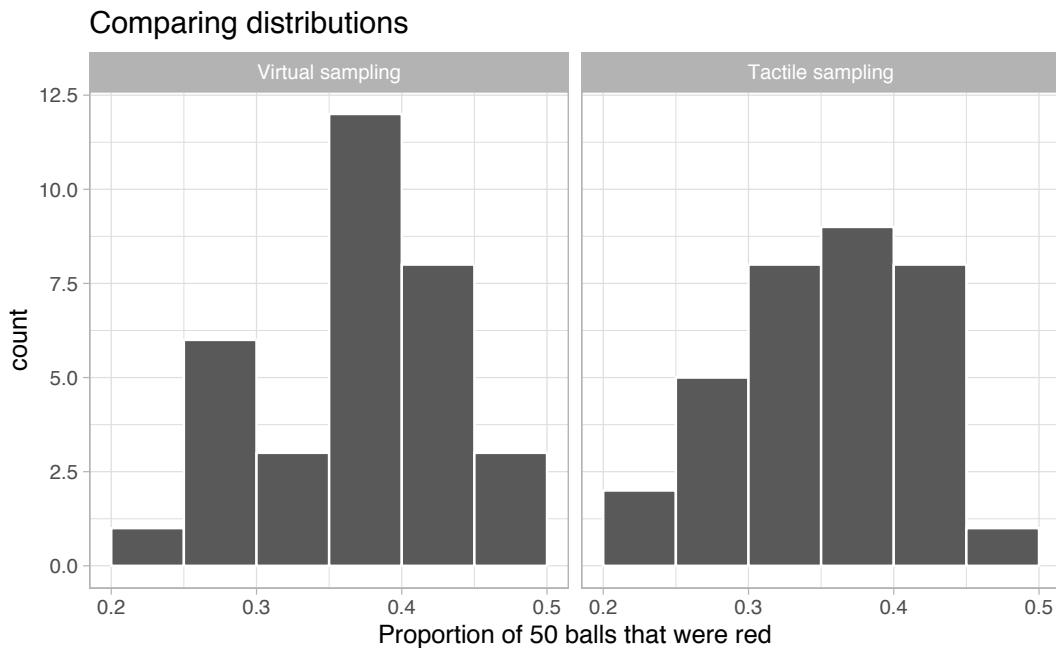


FIGURE 8.9: Comparing 33 virtual and 33 tactile proportions red.

8.2.3 Using the virtual shovel 1000 times

Now say we want to study the variation in proportions red not based on 33 repeated/replicated samples, but rather a very large number of samples say 1000 samples. We have two choices at this point. We could have our students manually take 1000 samples of 50 balls and compute the corresponding 1000 proportion red out 50 balls. This would be cruel and unusual, however, as this would be very tedious and time-consuming. This is where computers excel: automating long and repetitive tasks while performing them very quickly. Therefore, at this point, we will abandon tactile sampling in favor of only virtual sampling. Let's once again use the `rep_sample_n()` function with sample size set to 50 once again, but this time with the number of replicates `reps = 1000`. Be sure to scroll through the contents of `virtual_samples` in RStudio's viewer.

```
virtual_samples <- bowl %>%
  rep_sample_n(size = 50, reps = 1000)
View(virtual_samples)
```

Observe that now `virtual_samples` has $1000 \times 50 = 50,000$ rows, instead of the $33 \times 50 = 1650$ rows from earlier. Using the same code as earlier, let's take the data frame `virtual_samples` with $1000 \times 50 = 50,000$ and compute the resulting 1000 proportions red.

```
virtual_prop_red <- virtual_samples %>%
  group_by(replicate) %>%
  summarize(red = sum(color == "red")) %>%
  mutate(prop_red = red / 50)
View(virtual_prop_red)
```

Observe that we now have 1000 replicates of `prop_red`, the proportion of 50 balls that are red. Using the same code as earlier, let's now visualize the distribution of these 1000 replicates of `prop_red` in a histogram in Figure 8.10.

```
ggplot(virtual_prop_red, aes(x = prop_red)) +
  geom_histogram(binwidth = 0.05, boundary = 0.4, color = "white") +
  labs(x = "Proportion of 50 balls that were red",
       title = "Distribution of 1000 proportions red")
```

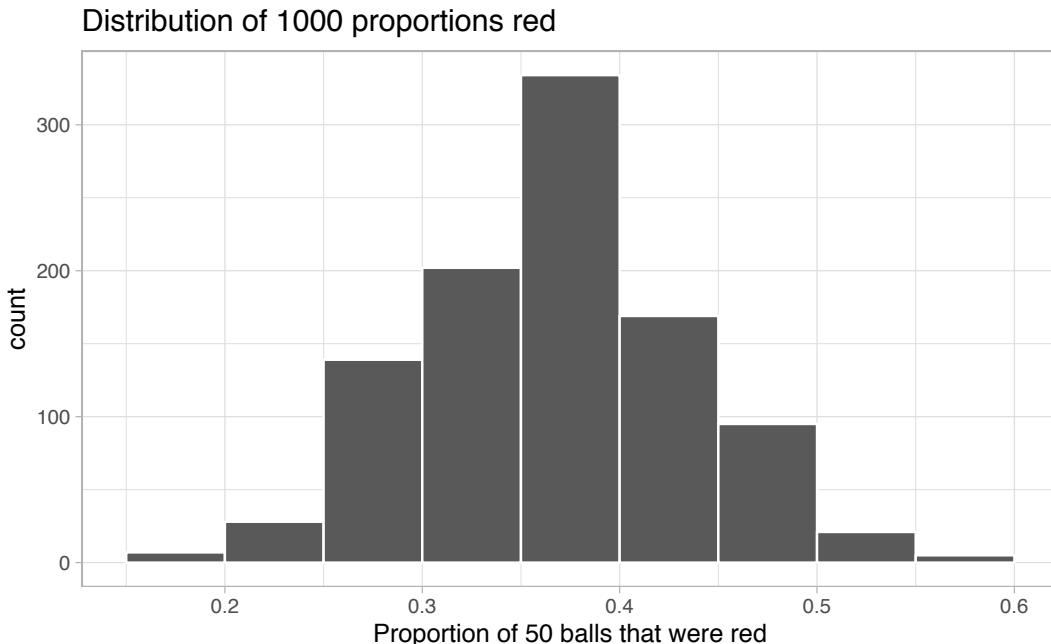


FIGURE 8.10: Distribution of 1000 proportions based on 33 samples of size 50.

Once again, the most frequently occurring proportions red occur between 35%

and 40%. Every now and then, we obtain proportions as low as between 20% and 25%, and others as high as between 55% and 60%. These are rare, however. Furthermore, observe that we now have a much more symmetric and smoother bell-shaped distribution. This distribution is, in fact, a Normal distribution; see the upcoming Section 8.5.3 for a brief discussion on properties of the Normal distribution.

Learning check

(LC8.4) Why did we not take 1000 “tactile” samples of 50 balls by hand?

(LC8.5) Looking at Figure 8.10 above, would you say that sampling 50 balls where 30% of them were red is likely or not? What about sampling 50 balls where 10% of them were red?

(LC8.6)

8.2.4 Using different shovels

Now say instead of just one shovel, you had three choices of shovels to extract a sample of balls with.

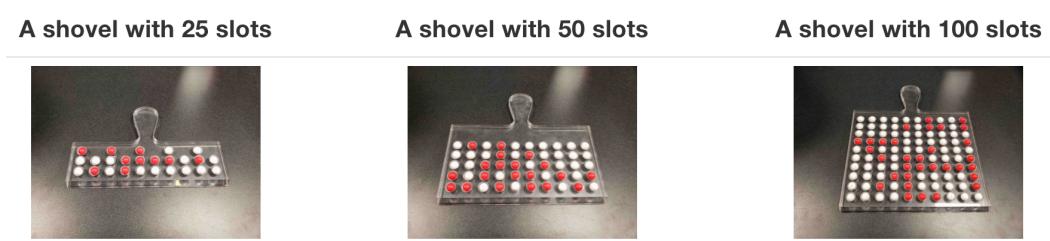


FIGURE 8.11: Three shovels to extract three different sample sizes.

If your goal was still to estimate the proportion of the bowl’s balls that were red, which shovel would you choose? In our experience, most people would choose the shovel with 100 slots since it has the biggest sample size and hence would yield the “best” guess of the proportion of the bowl’s 2400 balls that are red. Using our newly developed tools for virtual sampling simulations, let’s unpack the effect of having different sample sizes! In other words, let’s use `rep_sample_n()` with `size = 25`, `size = 50`, and `size = 100`, while keeping the number of repeated/replicated samples at 1000:

1. Virtually use the appropriate shovel to generate 1000 samples with `size` balls.
2. Compute the resulting 1000 replicates of the proportion of the shovel's balls that are red.
3. Visualize the distribution of these 1000 proportion red using a histogram.

Run each of the following code segments individually and then compare the three resulting histograms.

```
# Segment 1: sample size = 25 -----
# 1.a) Virtually use shovel 1000 times
virtual_samples_25 <- bowl %>%
  rep_sample_n(size = 25, reps = 1000)

# 1.b) Compute resulting 1000 replicates of proportion red
virtual_prop_red_25 <- virtual_samples_25 %>%
  group_by(replicate) %>%
  summarize(red = sum(color == "red")) %>%
  mutate(prop_red = red / 25)

# 1.c) Plot distribution via a histogram
ggplot(virtual_prop_red_25, aes(x = prop_red)) +
  geom_histogram(binwidth = 0.05, boundary = 0.4, color = "white") +
  labs(x = "Proportion of 25 balls that were red", title = "25")
```

```
# Segment 2: sample size = 50 -----
# 2.a) Virtually use shovel 1000 times
virtual_samples_50 <- bowl %>%
  rep_sample_n(size = 50, reps = 1000)

# 2.b) Compute resulting 1000 replicates of proportion red
virtual_prop_red_50 <- virtual_samples_50 %>%
  group_by(replicate) %>%
  summarize(red = sum(color == "red")) %>%
  mutate(prop_red = red / 50)

# 2.c) Plot distribution via a histogram
ggplot(virtual_prop_red_50, aes(x = prop_red)) +
  geom_histogram(binwidth = 0.05, boundary = 0.4, color = "white") +
  labs(x = "Proportion of 50 balls that were red", title = "50")
```

```

# Segment 3: sample size = 100 -----
# 3.a) Virtually using shovel with 100 slots 1000 times
virtual_samples_100 <- bowl %>%
  rep_sample_n(size = 100, reps = 1000)

# 3.b) Compute resulting 1000 replicates of proportion red
virtual_prop_red_100 <- virtual_samples_100 %>%
  group_by(replicate) %>%
  summarize(red = sum(color == "red")) %>%
  mutate(prop_red = red / 100)

# 3.c) Plot distribution via a histogram
ggplot(virtual_prop_red_100, aes(x = prop_red)) +
  geom_histogram(binwidth = 0.05, boundary = 0.4, color = "white") +
  labs(x = "Proportion of 100 balls that were red", title = "100")

```

For easy comparison, we present the three resulting histograms in a single row with matching x and y axes in Figure 8.12. What do you observe?

Comparing distributions of proportions red for 3 different shovels.

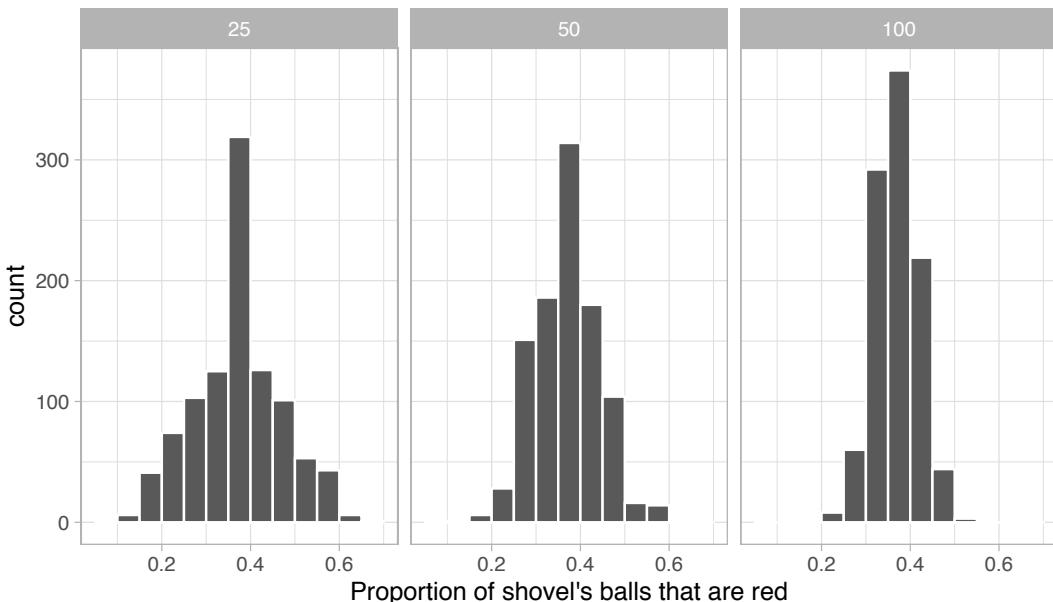


FIGURE 8.12: Comparing the distributions of proportion red for different sample sizes.

Observe that as the sample size increases, the spread of the 1000 replicates of the proportion of red decreases. In other words, as the sample size increases, there are fewer differences due to sampling variation and the distribution centers more tightly around the same value. Eyeballing Figure 8.12, things appear to center tightly around roughly 40%.

We can be numerically explicit about the amount of spread in our 3 sets of 1000 values of `prop_red` using the *standard deviation*: a summary statistic that measures the amount of spread and variation within a numerical variable; see Appendix A for a brief discussion on properties of the standard deviation. For all three sample sizes, let's compute the standard deviation of the 1000 proportions red by running the following data wrangling code that uses the `sd()` summary function.

```
# n = 25
virtual_prop_red_25 %>%
  summarize(sd = sd(prop_red))

# n = 50
virtual_prop_red_50 %>%
  summarize(sd = sd(prop_red))

# n = 100
virtual_prop_red_100 %>%
  summarize(sd = sd(prop_red))
```

Let's compare these three measures of spread of the distributions in Table 8.4.

TABLE 8.4: Comparing standard deviations of proportions red for 3 different shovels.

Number of slots in shovel	Standard deviation of proportions red
25	0.099
50	0.071
100	0.048

As we observed visually in Figure 8.12, as the sample size increases our numerical measure of spread decreases; there is less variation in our proportions red. In other words, as the sample size increases, our guesses at the true proportion of the bowl's balls that are red get more consistent and precise.

Learning check

(LC8.7) In Figure 8.12, we used shovels to take 1000 samples each, computed the resulting 1000 proportions of the shovel’s balls that were red, and then visualized the distribution of these 1000 proportions in a histogram. We did this for shovels with 25, 50, and 100 slots in them. As the size of the shovels increased, the histograms got narrower. In other words, as the size of the shovels increased from 25 to 50 to 100, did the 1000 proportions

1. Vary less
2. Vary by the same amount
3. Vary more

(LC8.8) What summary statistic did we use to quantify how much the 1000 proportions red varied?

1. The inter-quartile range
2. The standard deviation
3. The range: the largest value minus the smallest

8.3 Sampling framework

In both our “hands-on” tactile simulations and our “virtual” simulations using a computer, we used sampling for the purpose of estimation: we extract samples in order to estimate the proportion of the bowl’s balls that are red. We used sampling as a cheaper and less time consuming approach than to do a full census of all the balls. Our virtual simulations all built up to the results shown in Figure 8.12 and Table 8.4, comparing 1000 proportions red based on samples of size 25, 50, and 100. This was our first attempt at understanding two key concepts relating to sampling for estimation:

1. The effect of sampling variation on our estimates.
2. The effect of sample size on sampling variation.

Let’s now introduce some terminologies and notation as well as statistical

definitions related to sampling. Given the number of new words to learn, you will likely have to read these next three subsections multiple times. Keep in mind, however, that none of the concepts underlying these terminology, notation, and definitions are any different than the concepts underlying our simulations in Sections 8.1 and 8.2; it will simply take time and practice to master them.

8.3.1 Terminology & notation

Here is a list of terminology and mathematical notation relating to sampling. For each item, we'll be sure to tie them to our simulations in Sections 8.1 and 8.2.

1. **(Study) Population:** A (study) population is a collection of individuals or observations about which we are interested in. We mathematically denote the population's size using upper case N . In our simulations the (study) population was the collection of $N = 2400$ identically sized red and white balls contained in the bowl.
2. **Population parameter:** A population parameter is a numerical summary quantity about the population that is unknown, but you wish you knew. For example, when this quantity is a mean, the population parameter of interest is the *population mean* which is mathematically denoted with the Greek letter μ (pronounced "mu"). In our simulations however since we were interested in the proportion of the bowl's balls that were red, the population parameter is the *population proportion* which is mathematically denoted with the letter p .
3. **Census:** An exhaustive enumeration or counting of all N individuals or observations in the population in order to compute the population parameter's value *exactly*. In our simulations, this would correspond to manually going over all $N = 2400$ balls in the bowl and counting the number that is red and computing the population proportion p of the balls that are red *exactly*. When the number N of individuals or observations in our population is large, as was the case with our bowl, a census can be very expensive in terms of time, energy, and money.
4. **Sampling:** Sampling is the act of collecting a sample from the population when we don't have the means to perform a census. We mathematically denote the sample's size using lower case n , as opposed to upper case N which denotes the population's size. Typically the sample size n is much smaller than the population size N , thereby making sampling a much cheaper procedure than a census. In our

simulations, we used shovels with 25, 50, and 100 slots to extract a sample of size $n = 25$, $n = 50$, and $n = 100$ balls.

5. **Point estimate (AKA sample statistic):** A summary statistic computed from the sample that *estimates* the unknown population parameter. In our simulations, recall that the unknown population parameter was the population proportion and that this is mathematically denoted with p . Our point estimate is the *sample proportion*: the proportion of the shovel's balls that are red. In other words, it is our guess of the proportion of the bowl's balls that are red. We mathematically denote the sample proportion using \hat{p} ; the “hat” on top of the p indicates that it is an estimate of the unknown population proportion p .
6. **Representative sampling:** A sample is said to be a *representative sample* if it is representative of the population. In other words, are the sample's characteristics a good representation of the population's characteristics? In our simulations, are the samples of n balls extracted using our shovels representative of the bowl's $N=2400$ balls?
7. **Generalizability:** We say a sample is *generalizable* if any results based on the sample can generalize to the population. In other words, can the value of the point estimate be generalized to estimate the value of the population parameter well? In our simulations, can we generalize the values of the sample proportions red of our shovels to the population proportion red of the bowl? Using mathematical notation, is \hat{p} a “good guess” of p ?
8. **Bias:** In a statistical sense, we say *bias* occurs if certain individuals or observations in a population have a higher chance of being included in a sample than others. We say a sampling procedure is *unbiased* if every observation in a population had an equal chance of being sampled. In our simulations, since each ball had the same size and hence an equal chance of being a sample in our shovels, our samples were unbiased.
9. **Random sampling:** We say a sampling procedure is *random* if we sample randomly from the population in an unbiased fashion. In our simulations, this would correspond to sufficiently mixing the bowl before each use of the shovel.

Phew, that's a lot of new terminology and notation to learn! Let's put them all together to describe the paradigm of sampling:

-
- If the sampling of a sample of size n is done at **random**, then

- the sample is **unbiased** and **representative** of the population of size N , thus
 - any result based on the sample can **generalize** to the population, thus
 - the point estimate is a “**good guess**” of the unknown population parameter, thus
 - instead of performing a census, we can **infer** about the population using sampling.
-

Restricting consideration to a shovel with 50 slots from our simulations,

- If we extract a sample of $n = 50$ balls at **random**, in other words, we mix the equally-sized balls before using the shovel, then
 - the contents of the shovel are an **unbiased representation** of the contents of the bowl’s 2400 balls, thus
 - any result based on the sample of balls can **generalize** to the bowl, thus
 - the sample proportion \hat{p} of the $n = 50$ balls in the shovel that are red is a “**good guess**” of the population proportion p of the $N=2400$ balls that are red, thus
 - instead of manually going over all the balls in the bowl, we can **infer** about the bowl using the shovel.
-

Note that last word we wrote in bold: **infer**. The act of “inferring” is to deduce or conclude (information) from evidence and reasoning. In our simulations, we wanted to infer about the proportion of the bowl’s balls that are red. *Statistical inference* is the theory, methods, and practice of forming judgments about the parameters of a population and the reliability of statistical relationships, typically on the basis of random sampling (Wikipedia). In other words, statistical inference is the act of inference via sampling. In the upcoming Chapter 9 on confidence intervals, we’ll introduce the `infer` package, which makes statistical inference “tidy” and transparent. It is why this third portion of the book is called “Statistical inference via `infer`”.

Learning check

(LC8.9) In the case of our bowl activity, what is the *population parameter*? Do we know its value?

(LC8.10) What would performing a census in our bowl activity correspond to? Why did we not perform a census?

(LC8.11) What purpose do *point estimates* serve in general? What is the

name of the point estimate specific to our bowl activity? What is its mathematical notation?

(LC8.12) How did we ensure that our tactile samples using the shovel were random?

(LC8.13) Why is it important that sampling be done *at random*?

(LC8.14) What are we *inferring* about the bowl based the samples using the shovel?

8.3.2 Statistical definitions

Now for some important statistical definitions related to sampling. As a refresher of our 1000 repeated/replicated virtual samples of size $n = 25$, $n = 50$, and $n = 100$ in Section 8.2, let's display Figure 8.12 again below.

Comparing distributions of proportions red for 3 different shovels.

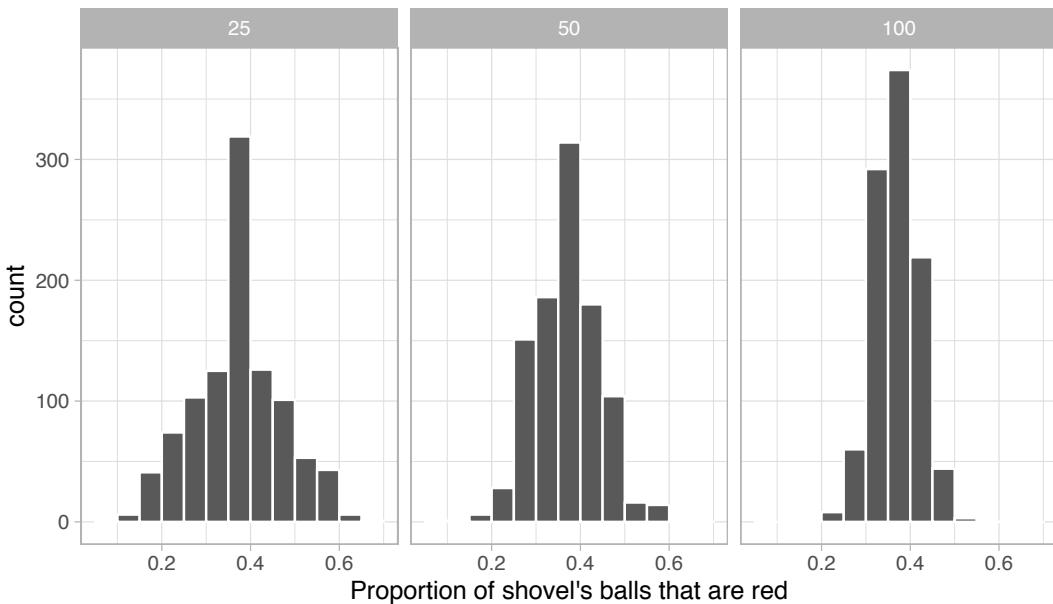


FIGURE 8.13: Previously seen three sampling distributions of the sample proportion \hat{p} .

These types of distributions have a special name: **sampling distributions**; their visualization displays the effect of sampling variation on the distribution of any point estimate, in this case, the sample proportion \hat{p} . Using these sampling distributions, for a given sample size n , we can make statements about

what values we can typically expect. For example, observe the centers of all three sampling distributions: they are all roughly centered around $0.4 = 40\%$. Furthermore, observe that while we are somewhat likely to observe sample proportions red of $0.2 = 20\%$ when using the shovel with 25 slots, we will almost never observe this sample proportion when using the shovel with 100 slots. Observe also the effect of sample size on the sampling variation. As the sample size n increases from 25 to 50 to 100, the spread/variation of the sampling distribution decreases and thus the values cluster more and more tightly around the same center of around 40%. We quantified this spread/variation using the standard deviation of our proportions in Table 8.4, which we display again below:

TABLE 8.5: Previously seen comparing standard deviations of proportions red for 3 different shovels.

Number of slots in shovel	Standard deviation of proportions red
25	0.099
50	0.071
100	0.048

So as the number of slots in the shovel increased, this standard deviation decreased. These types of standard deviations have another special name: **standard errors**; they quantify the effect of sampling variation induced on our estimates. In other words, they are quantifying how much we can expect different proportions of a shovel's balls that are red to vary from a random sample to a random sample.

Unfortunately, many new statistics practitioners get confused by these names. For example, it's common for people new to statistical inference to call the "sampling distribution" the "sample distribution". Another additional source of confusion is the name "standard deviation" and "standard error". Remember that a standard error is merely a *kind* of standard deviation: the standard deviation of any point estimate from a sampling scenario. In other words, all standard errors are standard deviations, but not all standard deviations are a standard error.

To help reinforce these concepts, let's re-display Figure 8.12 but using our new terminology, notation, and definitions relating to sampling in Figure 8.14.

Furthermore, let's re-display Table 8.4 but using our new terminology, notation, and definitions relating to sampling in Table 8.6.

Sampling distributions of the sample proportion \hat{p} based on $n = 25, 50, 100$

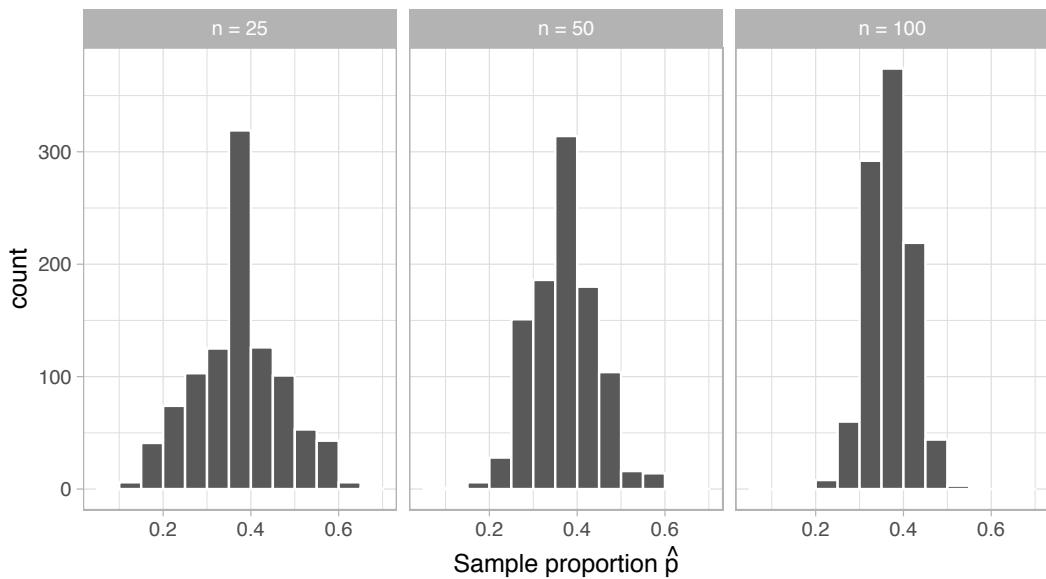


FIGURE 8.14: Three sampling distributions of the sample proportion \hat{p} .

TABLE 8.6: Three standard errors of the sample proportion based on $n = 25, 50, 100$.

Sample size (n)	Standard error of \widehat{p}
$n = 25$	0.099
$n = 50$	0.071
$n = 100$	0.048

Remember the key message of this last table: that as the sample size n goes up, the “typical” error of your point estimate as quantified by the standard error will go down.

Learning check

(LC8.15) What purpose did the *sampling distributions* serve?

(LC8.16) What does the *standard error* of the sample proportion \hat{p} quantify?

8.3.3 The moral of the story

Let's recap this section so far. We've seen that if a sample is generated at random, then the resulting point estimate is a "good guess" of the true unknown population parameter. In our simulations, since we made sure to mix the balls first before extracting a sample with the shovel, the resulting sample proportion \hat{p} of the shovel's balls that were red was a "good guess" of the population proportion p of the bowl's balls that were red.

However, what do we mean by our point estimate being a "good guess"? While sometimes we'll obtain a point estimate less than the true value of the unknown population parameter, other times we'll obtain a point estimate greater than the true value of the unknown population parameter, this is because of sampling variation. However, despite this sampling variation, our point estimates will "on average" be correct. In our simulations, sometimes our sample proportion \hat{p} was less than the true population proportion p , other times the sample proportion \hat{p} was greater than the true population proportion p . This was due to the sampling variability induced by the mixing. However despite this sampling variation, our sample proportions \hat{p} were always centered around the true population proportion. This is also known as having an **accurate** estimate.

What was the value of the population proportion p of the $N = 2400$ balls in the actual bowl? There were 900 red balls, for a proportion red of $900/2400 = 0.375 = 37.5\%$? How do we know this? Did the authors do an exhaustive count of all the balls? No! They were listed in the contents of the box that the bowl came in. Hence we made the contents of the virtual bowl match the tactile bowl:

```
bowl %>%
  summarize(sum_red = sum(color == "red"),
           sum_not_red = sum(color != "red"))

# A tibble: 1 × 2
  sum_red sum_not_red
  <int>      <int>
1     900        1500
```

Let's re-display our sampling distributions from Figures 8.12 and 8.14, but now

with a vertical red line marking the true population proportion p of balls that are red = 37.5% in Figure 8.15. We see that while there is a certain amount of error in the sample proportions \hat{p} for all three sampling distributions, on average the \hat{p} are centered at the true population proportion red p .

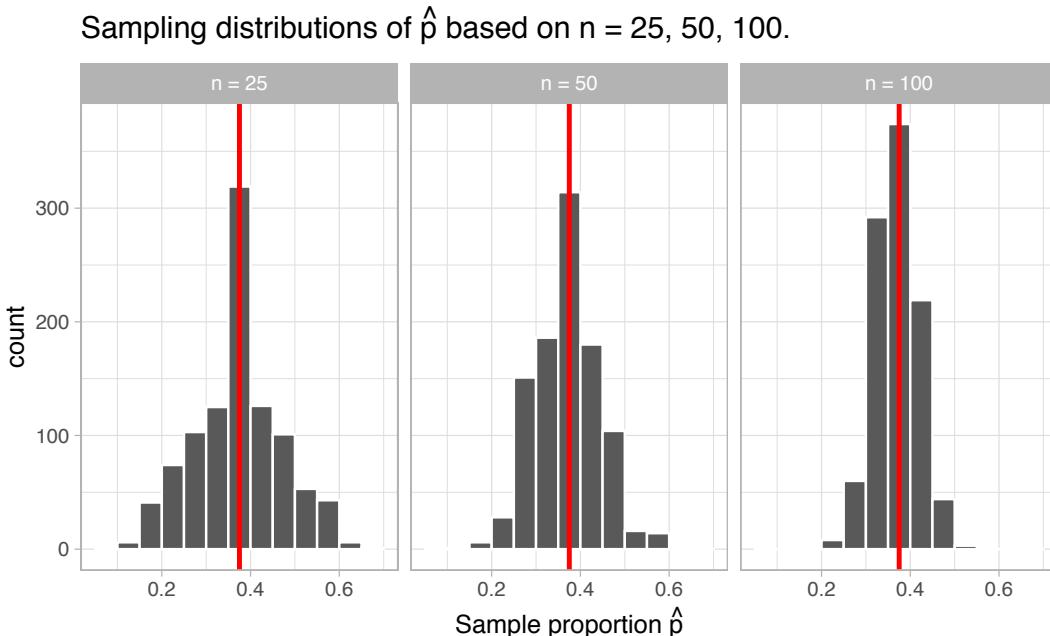


FIGURE 8.15: Three sampling distributions with population proportion p marked in red.

We also saw in this section that as your sample size n increases, your point estimates will vary less and less and be more and more concentrated around the true population parameter; this is quantified by the decreasing standard error. In other words, the typical error of your point estimates will decrease. In our simulations, as the sample size increases, the spread/variation of our sample proportions \hat{p} around the true population proportion p decreases. You can observe this behavior as well in Figure 8.15. This is also known as having a more **precise** estimate.

So random sampling ensures our point estimates are accurate while having a large sample size ensures our point estimates are precise. While accuracy and precision may sound like the same concept, they are actually not. Accuracy relates to how “on target” our estimates are whereas precision relates to how “consistent” our estimates are. Figure 8.16 illustrates the difference.

As this point, you might be asking yourself: “If you already knew the true proportion of the bowl’s balls that are red was 37.5%, then why did do any sampling?” You might also be asking: “Why did we take 1000 repeated samples

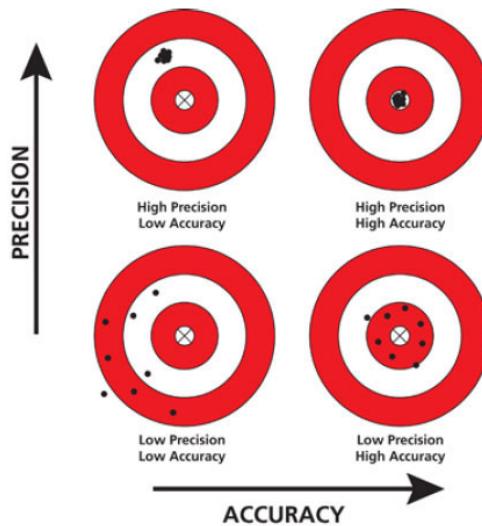


FIGURE 8.16: Comparing accuracy and precision.

of size $n = 25, 50$, and 100 ? Shouldn't we be taking only *one* sample that's as large as possible?"

Recall our definition of a simulation from Section 8.2: an approximate imitation of the operation of a process or system. We performed these simulations to study:

1. The effect of sampling variation on our estimates.
2. The effect of sample size on sampling variation.

In a real-life scenario, we won't know what the true value of the population parameter is and furthermore we won't take repeated/replicated samples, but rather a single sample that's as large as we can afford. The ideas developed in our repeated sampling activity only help inform pollsters about the quality of their polls. In particular, how accurate and how precise they are as we saw in Figure 8.16 with the targets.

In the next section, we present a case study of a real-life sampling scenario: polling.

Learning check

(LC8.17) The table below is a version of Table 8.6 matching sample sizes n to different *standard errors* of the sample proportion \hat{p} , but with the rows

randomly re-ordered and the sample sizes removed. Fill in the table below by matching the correct sample sizes to the correct standard errors.

TABLE 8.7: Three standard errors of the sample proportion based on $n = 25, 50, 100$.

Sample size	Standard error of \hat{p}
$n =$	0.099
$n =$	0.048
$n =$	0.071

For the following learning checks, let the *estimate* be the sample proportion \hat{p} : the proportion of a shovel's balls that were red. It estimated the population proportion p : the proportion of the bowl's balls that were red.

(LC8.18) What is the difference between an *accurate* estimate and a *precise* estimate?

(LC8.19) How do we ensure that an estimate is *accurate*? How do we ensure that an estimate is *precise*?

(LC8.20) In a real-life situation, we would not take 1000 different samples to infer about a population, but rather only one. Then what was the purpose of our exercises about where we took 1000 different samples?

(LC8.21) Figure 8.16 with the targets shows four combinations of “accurate versus precise” estimates. Draw four corresponding *sampling distributions* of the sample proportion \hat{p} , like the one in the left-most plot in Figure 8.15.

8.4 Case study: Polls

Let's now switch gears to a more realistic sampling scenario than our bowl activity: a poll. In practice, pollsters do not take 1000 repeated samples as we did in our sampling activities from above, but rather take only a *single sample* that's as large as possible.

On December 4, 2013, National Public Radio in the US reported on a poll of President Obama's approval rating among young Americans aged 18-29 in an

article “Poll: Support For Obama Among Young Americans Eroding”¹. The poll was conducted by the Harvard University Institute of Politics. A quote from the article:

After voting for him in large numbers in 2008 and 2012, young Americans are souring on President Obama.

According to a new Harvard University Institute of Politics poll, just 41 percent of millennials — adults ages 18-29 — approve of Obama’s job performance, his lowest-ever standing among the group and an 11-point drop from April.

Let’s tie elements of the real-life poll in this new article with our “tactile” and “virtual” simulations from Sections 8.1 and 8.2 using the terminology, notations, and definitions we learned in Section 8.3. You see that our sampling activity with the bowl is an idealized version of what pollsters are trying to do in real-life.

1. **(Study) Population:** Who is the population of N individuals or observations of interest?
 - Simulation: $N = 2400$ identically-sized red and white balls
 - Obama poll: $N = ?$ young Americans aged 18-29
2. **Population parameter:** What is the population parameter?
 - Simulation: The population proportion p of ALL the balls in the bowl that are red.
 - Obama poll: The population proportion p of ALL young Americans who approve of Obama’s job performance.
3. **Census:** What would a census look like?
 - Simulation: Manually going over all $N = 2400$ balls and exactly computing the population proportion p of the balls that are red, a time-consuming task.
 - Obama poll: Locating all $N = ?$ young Americans and asking them all if they approve of Obama’s job performance, an expensive task.
4. **Sampling:** How do you collect the sample of size n individuals or observations?
 - Simulation: Using a shovel with n slots.

¹<https://www.npr.org/sections/itsallpolitics/2013/12/04/248793753/poll-support-for-obama-among-young-americans-eroding>

- Obama poll: One method is to get a list of phone numbers of all young Americans and pick out n phone numbers. In this poll's case, the sample size of this poll was $n = 2089$ young Americans.
5. **Point estimate (AKA sample statistic):** What is your estimate of the unknown population parameter?
- Simulation: The sample proportion \hat{p} of the balls in the shovel that were red.
 - Obama poll: The sample proportion \hat{p} of young Americans in the sample that approve of Obama's job performance. In this poll's case, $\hat{p} = 0.41 = 41\%$, the quoted percentage in the second paragraph of the article.
6. **Representative sampling:** Is the sampling procedure *representative*?
- Simulation: Are the contents of the shovel representative of the contents of the bowl? Because we mixed the bowl before sampling, we can feel confident that they are.
 - Obama poll: Is the sample of $n = 2089$ young Americans representative of all young Americans aged 18-29? This depends on whether the sampling was random.
7. **Generalizability:** Are the samples *generalizable* to the greater population?
- Simulation: Is the sample proportion \hat{p} of the shovel's balls that are red a "good guess" of the population proportion p of the bowl's balls that are red? Given that the sample was representative, the answer is yes.
 - Obama poll: Is the sample proportion $\hat{p} = 0.41$ of the sample of young Americans who support Obama a "good guess" of the population proportion p of all young Americans who support Obama? In other words, can we confidently say that roughly 41% of *all* young Americans approve of Obama? Again, this depends on whether the sampling was random.
8. **Bias:** Is the sampling procedure unbiased? In other words, do all observations have an equal chance of being included in the sample?
- Simulation: Since each ball was equally sized, each ball had an equal chance of being included in a shovel's sample, and hence the sampling was unbiased.
 - Obama poll: Did all young Americans have an equal chance at being represented in this poll? Again, this depends on whether the sampling was random.
9. **Random sampling:** Was the sampling random?
- Simulation: As long as you mixed the bowl sufficiently before sampling, your samples would be random.

- Obama poll: Was the sample conducted at random? We can't answer this question without knowing about the *sampling methodology* used by the Harvard University Institute of Politics. We'll discuss this more at the end of this section.

Once again, let's revisit the sampling paradigm from Section 8.3.1:

- If the sampling of a sample of size n is done at **random**, then
 - the sample is **unbiased** and **representative** of the population of size N , thus
 - any result based on the sample can **generalize** to the population, thus
 - the point estimate is a "**good guess**" of the unknown population parameter, thus
 - instead of performing a census, we can **infer** about the population using sampling.
-

In our simulations using the shovel with 50 slots:

- If we extract a sample of $n = 50$ balls at **random**, in other words, we mix the equally-sized balls before using the shovel, then
 - the contents of the shovel are an **unbiased representation** of the contents of the bowl's 2400 balls, thus
 - any result based on the sample of balls can **generalize** to the bowl, thus
 - the sample proportion \hat{p} of the $n = 50$ balls in the shovel that are red is a "**good guess**" of the population proportion p of the $N = 2400$ balls that are red, thus
 - instead of manually going over all the balls in the bowl, we can **infer** about the bowl using the shovel.
-

In the in-real-life Obama poll:

- If we had a way of contacting a **randomly** chosen sample of 2089 young Americans and poll their approval of Obama, then
- these 2089 young Americans would be an **unbiased** and **representative** sample of *all* young Americans, thus
- any results based on this sample of 2089 young Americans can **generalize** to the entire population of all young Americans, thus
- the reported sample approval rating of 41% of these 2089 young Americans is a **good guess** of the true approval rating among all young Americans, thus

- instead of performing a highly costly census of all young Americans, we can **infer** about all young Americans using polling.
-

So as you can see, it was critical for the Harvard University Institute of Politics sample to be truly random in order to infer about *all* young Americans' opinions about Obama. Was their sample truly random? It's hard to answer such questions without knowing about the *sampling methodology* used. For example, if this poll was conducted using only mobile phone numbers, people without mobile phones would be left out and therefore not represented in the sample. What about if the Harvard University Institute of Politics conducted this poll on an internet news site? Then people who don't read this internet news site would be left out. Ensuring that our samples were random was easy to do in our sampling bowl exercises, however in a real-life situation like the Obama poll, this is much harder to do.

Learning check

Comment on the representativeness of the following *sampling methodologies*:

(LC8.22) The Royal Air Force wants to study how resistant all their airplanes are to bullets. They study the bullet holes on all the airplanes on the tarmac after an air battle against the Luftwaffe (German Air Force).

(LC8.23) Imagine it is 1993, a time when almost all households had landlines. You want to know the average number of people in each household in your city. You randomly pick out 500 phone numbers from the phone book and conduct a phone survey.

(LC8.24) You want to know the prevalence of illegal downloading of TV shows among students at a local college. You get the emails of 100 randomly chosen students and ask them "How many times did you download a pirated TV show last week?"

(LC8.25) A local college administrator wants to know the average income of all graduates in the last 10 years. So they get the records of 5 randomly chosen graduates, contact them, and obtain their answers.

8.5 Conclusion

8.5.1 Sampling scenarios

In this chapter, we performed both tactile and virtual simulations of sampling to infer about an unknown proportion. We also presented a case study of sampling in real life situation: polls. In both cases, we used the sample proportion \hat{p} to estimate the population proportion p . However, we are not just limited to scenarios related to statistical inference for proportions. In other words, we can consider other population parameter and point estimate scenarios than just the population proportion p and sample proportion \hat{p} scenarios we studied in this chapter. We present 5 more such scenarios in Table 8.8.

TABLE 8.8: Scenarios of sampling for inference

Scenario	Population parameter	Notation	Point estimate	Notation.
1	Population proportion	p	Sample proportion	\hat{p}
2	Population mean	μ	Sample mean	\bar{x} or $\hat{\mu}$
3	Difference in population proportions	$p_1 - p_2$	Difference in sample proportions	$\hat{p}_1 - \hat{p}_2$
4	Difference in population means	$\mu_1 - \mu_2$	Difference in sample means	$\bar{x}_1 - \bar{x}_2$
5	Population regression slope	β_1	Fitted regression slope	b_1 or $\hat{\beta}_1$
6	Population regression intercept	β_0	Fitted regression intercept	b_0 or $\hat{\beta}_0$

We'll cover all the remaining scenarios as follows, using the terminology, notation, and definitions related to sampling you saw in Section 8.3:

- In Chapter 9, we'll cover examples of statistical inference for
 - Scenario 2: The mean age μ of all pennies in circulation in the US.
 - Scenario 3: The difference $p_1 - p_2$ in the proportion of people who yawn

when seeing someone else yawn and the proportion of people who yawn without seeing someone else yawn. This is an example of *two-sample* inference.

- In Chapter 10, we'll cover an example of statistical inference for
 - Scenario 4: The difference $\mu_1 - \mu_2$ in average IMDB ratings for action and romance movies. This is another example of *two-sample* inference.
- In Chapter 11, we'll cover an example of statistical inference for the relationship between teaching score and various instructor demographic variables you saw in Chapter 6 on basic regression and Chapter 7 on multiple regression. Specifically
 - Scenario 5: The intercept β_0 of some population regression line.
 - Scenario 6: The slope β_1 of some population regression line.

8.5.2 Central Limit Theorem

What you visualized in Figure 8.12 and summarized in Table 8.4) was a demonstration of a very famous theorem, or mathematically proven truth, called the *Central Limit Theorem*. It loosely states that when sample means are based on larger and larger sample sizes, the sampling distribution of these sample means both

1. Becomes more and more normally shaped, or in other words
2. Becomes more and more narrow, or in other words the standard error decreases.

In other words, their sampling distribution increasingly follows a *normal distribution* and the spread/variation of these sampling distributions, as quantified by their standard errors, gets smaller.

Shuyi Chiou, Casey Dunn, and Pathikrit Bhattacharyya created a 3 minute and 38 second video at <https://youtu.be/jvoxEYmQHNM> explaining this crucial statistical theorem using the average weight of wild bunny rabbits and the average wing span of dragons as examples. Figure 8.17 shows a preview of this video.

8.5.3 Normal distributions

In the previous Subsection on the Central Limit Theorem, we introduced the notion of a normal distribution. Such distributions are defined by two values: 1) the mean μ , which locates the center of the distribution, and 2) the standard deviation σ , which determines the spread of the distribution. In Figure 8.18, we plot three normal distributions where:

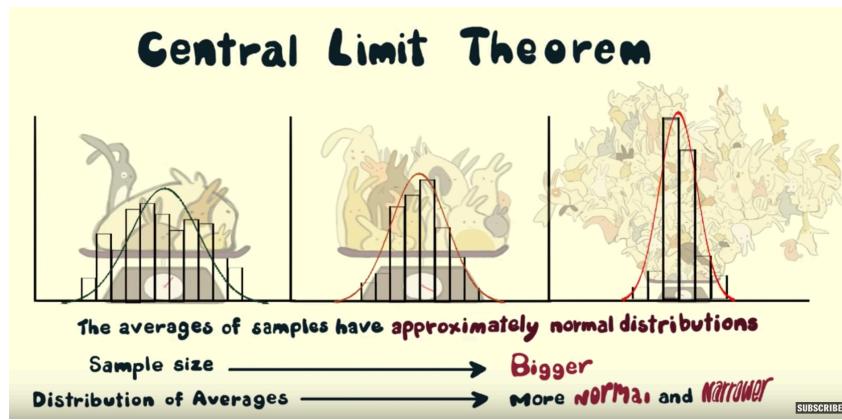


FIGURE 8.17: Preview of Central Limit Theorem video.

1. The solid line normal curve has mean $\mu = 5$ and standard deviation $\sigma = 2$.
2. The dashed line normal curve has mean $\mu = 5$ and standard deviation $\sigma = 5$.
3. The dotted line normal curve has mean $\mu = 14$ and standard deviation $\sigma = 2$.

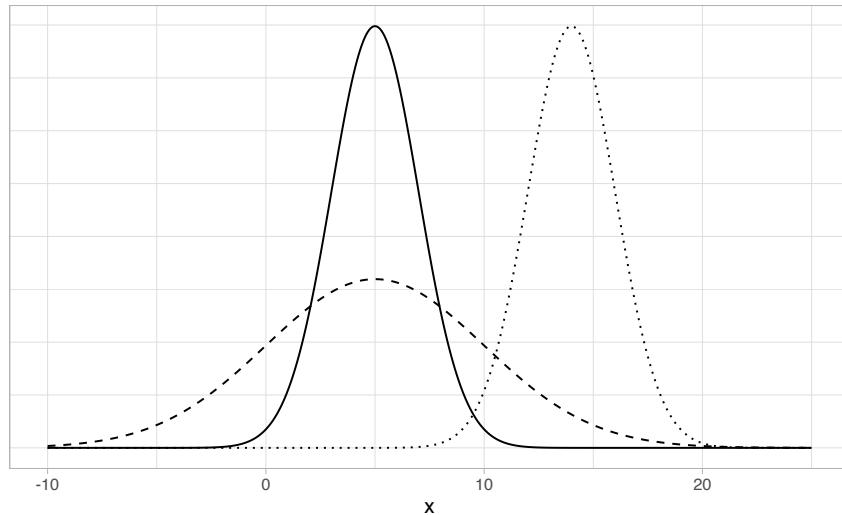


FIGURE 8.18: Three normal distributions

Notice how the solid and dashed line normal curves have the same center owing to their common mean $\mu = 5$. However the dashed line normal curve is wider owing to its larger standard deviation of $\sigma = 5$. On the other hand, the solid and dotted line normal curves have the same spread owing to their common standard deviation $\sigma = 2$. However they are centered at different locations.

When the mean $\mu = 0$ and the standard deviation $\sigma = 1$, the normal distribution has a special name: the standard normal distribution or the z -curve.

Furthermore, if a variable follows a normal curve, there are three rules of thumb we can use:

1. 68% of values will lie within ± 1 standard deviations of the mean.
2. 95% of values will lie within $\pm 1.96 \approx 2$ standard deviations of the mean.
3. 99.7% of values will lie within ± 3 standard deviations of the mean.

Let's illustrate this on a standard normal curve in Figure 8.19. The dashed lines are at -3, -1.96, -1, 0, 1, 1.96, and 3 and cut up the x-axis into 8 segments. The areas under the normal curve for each of the 8 segments are marked and add up to 100%.

1. The middle two segments represent the interval -1 to 1. The shaded area above this interval represents $34\% + 34\% = 68\%$ of the area under the curve. In other words 68% of values.
2. The middle four segments represent the interval -1.96 to 1.96. The shaded area above this interval represents $13.5\% + 34\% + 34\% + 13.5\% = 95\%$ of the area under the curve. In other words 95% of values.
3. The middle six segments represent the interval -3 to 3. The shaded area above this interval represents $2.35\% + 13.5\% + 34\% + 34\% + 13.5\% + 2.35\% = 99.7\%$ of the area under the curve. In other words 99.7% of values.

Learning check

Say you have a normal distribution with mean $\mu = 6$ and standard deviation $\sigma = 3$.

(LC8.26) What proportion of the area under the normal curve is less than 3? Greater than 12? Between 0 and 12?

(LC8.27) What is the 2.5th percentile of the area under the normal curve? The 95th percentile? The 100th percentile.

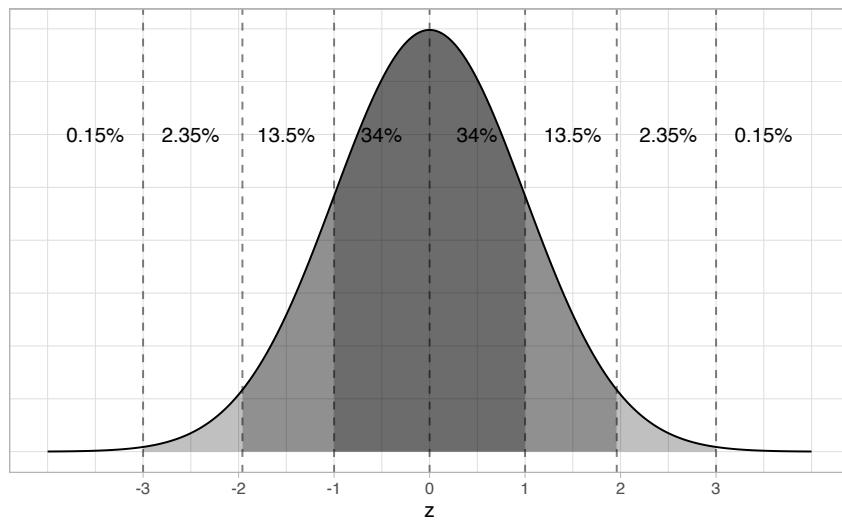


FIGURE 8.19: Rules of thumb about areas under normal curves

8.5.4 Additional resources

An R script file of all R code used in this chapter is available here².

8.5.5 What's to come?

Recall in our Obama poll case study in Section 8.4 that based on this particular sample, the Harvard University Institute of Politics' best guess of Obama's approval rating among all young Americans was 41%. However, this isn't the end of the story. If you read further in the article, it states:

The online survey of 2,089 adults was conducted from Oct. 30 to Nov. 11, just weeks after the federal government shutdown ended and the problems surrounding the implementation of the Affordable Care Act began to take center stage. The poll's margin of error was plus or minus 2.1 percentage points.

Note the term *margin of error*, which here is plus or minus 2.1 percentage points. What this is saying is that most polls won't get it perfectly right; there will always be a certain amount of error caused by *sampling variation*.

²[scripts/08-sampling.R](#)

The margin of error of plus or minus 2.1 percentage points is saying that a typical range of errors for polls of this type is about $\pm 2.1\%$, in words from about 2.1% too small to about 2.1% too big for an interval of $[41\% - 2.1\%, 41\% + 2.1\%] = [37.9\%, 43.1\%]$. Remember that this notation corresponds to 37.9% and 43.1% is included as well as all numbers between the two of them. We'll see in the next chapter that such intervals are known as *confidence intervals*.

9

Bootstrapping and Confidence Intervals

In Chapter 8, we studied sampling. We started with a “tactile” exercise where we wanted to know the proportion of balls in the sampling bowl in Figure 8.1 that are red. While we could have performed an exhaustive count, this would have been a tedious process. So instead we used a shovel to extract a sample of 50 balls and used the resulting proportion that were red as an estimate of the proportion of the bowl’s balls that are red. Furthermore, we made sure to mix the bowl’s contents before every use of the shovel. Because of the randomness induced by the mixing, different uses of the shovel yielded different proportions red and hence different estimates of the proportion of the bowl’s balls that are red.

We then mimicked this “tactile” exercise with an equivalent “virtual” exercise performed on the computer. Using our computers’ random number generator, we could very quickly mimic the above sampling procedure a large number of times. In Section 8.2.4, we quickly repeated the above sampling procedure 1000 times using three different “virtual” shovels with 25, 50, and 100 slots. We compared the variation of these three sets of 1000 estimates of the proportion of the bowl’s balls that are red in the three histograms in Figure 8.15.

What we did there was construct *sampling distributions*. The motivation for taking 1000 repeated samples and visualizing the resulting estimates was to study how these estimates varied from one sample to another; in other words we wanted to study the effect of *sampling variation*. We quantified the variation of these estimates using their standard deviation which has a special name: the *standard error*. In particular, we saw that as the sample size increased from 25 to 50 to 100, the standard error decreased and thus the sampling distributions narrowed. In other words, larger sample sizes lead to more *precise* estimates.

We also described the above sampling exercises using the terminology and mathematical notation related to sampling we introduced in Section 8.3.1. Our *study population* was the large bowl with $N = 2400$ balls, while the *population parameter*, the unknown quantity of interest, here was the population proportion p of the bowl’s balls that are red. Since performing a *census* would be very expensive in terms of time and energy, we instead extracted a *sample*

of size $n = 50$. The *point estimate*, also known as a *sample statistic*, used to estimate p was the sample proportion \hat{p} of these 50 sampled balls that were red. Furthermore, since the sample was obtained at *random*, it can be considered as *unbiased* and *representative* of the population. Thus any results based on the sample could be *generalized* to the population. In other words, the sample proportion \hat{p} of the shovel's $n = 50$ balls that were red was a “good guess” of the true population proportion p of the bowl's $N = 2400$ balls that are red. In other words, we used the sample to *infer* about the population.

However as described in Section 8.2, both the tactile and virtual sampling exercises are not what one would do in real life; they were merely *simulations* used to study the effects of sampling variation. In a real life situation, we would not take 1000 samples of size n , but rather take a *single* representative sample of as large a size as possible. Additionally, we knew what the true value of the population parameter here was: the true population proportion of the bowl's balls that are red. In a real life situation we will not know what this value is. Because if we did, then why would we take a sample to estimate it?

An example of a realistic sampling situation would be a poll, like the one described in the Obama poll¹ article you saw in Section 8.4. Pollsters did not know the true proportion of **all** young Americans who supported President Obama and thus took a single sample of size $n = 2089$ young Americans to estimate the true unknown value.

So how does one study the effects of sampling variation when you only have a single sample to work with? There is no sample-to-sample variation in estimates when you only have one sample. One common method is known as *bootstrapping resampling*, which will be the focus of the earlier sections of this chapter.

Furthermore, what if we would like not only a single estimate of the unknown population parameter, we would like a *range of highly plausible* values? Going back to the Obama poll article, it stated that the pollsters' estimate of the proportion of all young Americans who supported President Obama was 41%, but in addition it stated that the poll's “margin of error was plus or minus 2.1 percentage points.” In other words this “plausible range” was $[41\% - 2.1\%, 41\% + 2.1\%] = [37.9\%, 43.1\%]$. This range of plausible values is known as a *confidence interval* and will be the focus of the later sections of this chapter.

¹<https://www.npr.org/sections/itsallpolitics/2013/12/04/248793753/poll-support-for-obama-among-young-americans-eroding>

Needed packages

Let's load all the packages needed for this chapter (this assumes you've already installed them). Recall from our discussion in Section 5.4.1 that loading the `tidyverse` package by running `library(tidyverse)` loads the following commonly used data science packages all at once:

- `ggplot2` for data visualization
- `dplyr` for data wrangling
- `tidyverse` for converting data to “tidy” format
- `readr` for importing spreadsheet data into R
- As well as the more advanced `purrr`, `tibble`, `stringr`, and `forcats` packages

If needed, read Section 2.3 for information on how to install and load R packages.

```
library(tidyverse)
library(moderndive)
library(infer)
```

9.1 Pennies activity

As we did in Chapter 8, we'll begin with a hands-on tactile activity.

9.1.1 What is the average year on US pennies in 2019?

Try to imagine all pennies being used in the United States in 2019. That's a lot of pennies! Now say we're interested in the average year of minting of *all* these pennies. One way to compute this value would be to gather up all pennies being used in the US, record the year, and compute the average. This would be near impossible! So instead, let's collect a sample of 50 pennies collected from a local bank in downtown Northampton, Massachusetts, the USA seen in Figure 9.1.

An image of these 50 pennies can be seen in Figure 9.2.

For each of the 50 pennies let's assign an “ID” label and mark the year of minting, starting in the top left and ending in the bottom right progressing row by row, as seen in Figure 9.3.



FIGURE 9.1: Collecting a sample of 50 US pennies from a local bank.



FIGURE 9.2: 50 US pennies.

The `modernvive` package contains this data on our 50 sampled pennies. Let's explore this sample data first:

```
pennies_sample
```

```
# A tibble: 50 x 2
  ID    year
  <int> <dbl>
1     1  2002
2     2  1986
3     3  2017
4     4  1988
5     5  2008
6     6  1983
```



FIGURE 9.3: 50 US pennies labelled.

```

7      7  2008
8      8  1996
9      9  2004
10     10 2000
# ... with 40 more rows

```

The `pennies_sample` data frame has 50 rows corresponding to each penny with two variables. The first variable `ID` corresponds to the ID labels in Figure 9.3 whereas the second variable `year` corresponds to the year of minting saved as an integer.

Based on these 50 sampled pennies, what can we say about *all* US pennies in 2019? Let's study some properties of our sample by performing an exploratory data analysis. Let's first visualize the distribution of the year of these 50 pennies using our data visualization tools from Chapter 3. Since `year` is a numerical variable, we use a histogram in Figure 9.4.

```

ggplot(pennies_sample, aes(x = year)) +
  geom_histogram(binwidth = 10, color = "white")

```

We observe a slightly left-skewed distribution since most values fall in between the 1980s through 2010s with only a few older than 1970. What is the average year for the 50 sampled pennies? Eyeballing the histogram it appears to be around 1990. Let's now compute this value exactly using our data wrangling tools from Chapter 4.

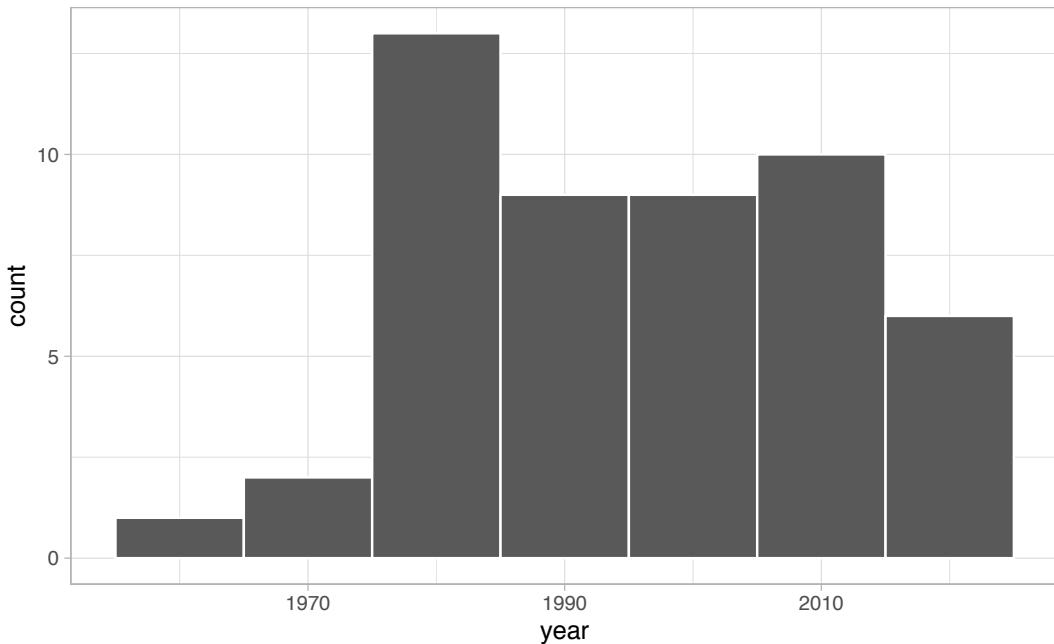


FIGURE 9.4: Distribution of year on 50 US pennies.

```
pennies_sample %>%
  summarize(mean_year = mean(year))
```

```
# A tibble: 1 x 1
  mean_year
  <dbl>
1 1995.44
```

Thus assuming `pennies_sample` is a representative sample from the population of all US pennies, a “good guess” of the average year of minting of all US pennies would be 1995.44, the average year of minting of our 50 sampled pennies. This should all start sounding similar to what we did previously in Chapter 8!

In Chapter 8 our study population was the bowl of $N = 2400$ balls. Our population parameter of interest was the population proportion of these balls that were red, denoted mathematically by p . In order to estimate p , we extracted a sample of 50 balls using the shovel and computed the relevant point estimate: the sample proportion of these 50 balls that were red, denoted mathematically by \hat{p} .

Here our study population is $N =$ whatever the number of pennies are being used in the US, a value which we don’t know and probably never will. The population parameter of interest is now the *population mean* year of all these

pennies, a value denoted mathematically by the Greek letter μ pronounced “mu”. In order to estimate μ , we went to the bank and obtained a sample of 50 pennies and computed the relevant point estimate: the *sample mean* year of these 50 pennies, denoted mathematically by \bar{x} pronounced “x-bar”. An alternative and more intuitive notation for the sample mean is $\hat{\mu}$. However this is unfortunately not as commonly used, so in this text, we’ll always denote the sample mean as \bar{x} .

We summarize the correspondence between the sampling bowl exercise in Chapter 8 and our pennies exercise in Table 9.1, which are the first two rows of the previously seen Table 8.8 of the various sampling scenarios we’ll cover in this text.

TABLE 9.1: Scenarios of sampling for inference

Scenario	Population parameter	Notation	Point estimate	Notation.
1	Population proportion	p	Sample proportion	\hat{p}
2	Population mean	μ	Sample mean	\bar{x} or $\hat{\mu}$

Going back to our 50 sampled pennies in Figure 9.3, the point estimate of interest is the sample mean \bar{x} of 1995.44. This quantity is an estimate of the population mean year of all US pennies μ .

Recall that we also saw in Chapter 8 that such estimates are prone to sampling variation. For example, in this particular sample in Figure 9.3, we observed three pennies with the year of 1999. If we obtained other samples of size 50 would we always observe exactly three pennies with the year of 1999? More than likely not. We might observe none, or one, or two, or maybe even all 50! The same can be said for the other 26 unique years that are represented in our sample of 50 pennies.

To study this sampling variation as we did in Chapter 8, we need more than one sample. In our case with pennies, how would we obtain another sample? We would go to the bank and get another roll of 50 pennies! However, in real-life sampling one doesn’t obtain many samples as we did in Chapter 8; those were merely simulations. So what how can we study sample-to-sample variation when we have only a single sample as in our case?

Just as different uses of the shovel in the bowl led to the different sample proportions red, different samples of 50 pennies will lead to different sample mean years. However, how can we study the effect of sampling variation using

only our *single sample* seen in Figure 9.3? We will do so using a technique known as “bootstrap resampling with replacement”, which we now illustrate.

9.1.2 Resampling once

Step 1: Let’s print out identically-sized slips of paper representing the 50 pennies in Figure 9.3.



FIGURE 9.5: 50 slips of paper representing 50 US pennies.

Step 2: Put the 50 small pieces of paper into a hat or tuque.



FIGURE 9.6: Putting 50 slips of paper in a hat.

Step 3: Mix the hat’s contents and draw one slip of paper at random. Record the year somewhere.

Step 4: Put the slip of paper back in the hat! In other words, replace it!

Step 5: Repeat Steps 3 and 4 49 more times, resulting in 50 recorded years.

What we just performed was a **resampling** of the original sample of 50 pennies. We are not sampling 50 pennies from the population of all US pennies

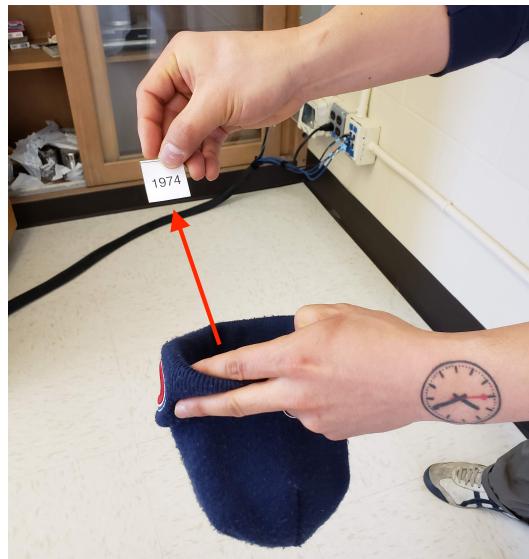


FIGURE 9.7: Drawing one slip of paper.



FIGURE 9.8: Replacing slip of paper.

as we did in our trip to the bank. Instead, we are mimicking this act by “re”-sampling 50 pennies from our originally sampled 50 pennies. However, why did we replace our resampled slip of paper back into the hat in Step 4? Because if we left the slip of paper out of the hat each time we performed Step 4, we would obtain the same 50 pennies, in the end, each time! In other words, replacing the slips of paper induces variation.

Being more precise with our terminology, we just performed a **resampling with replacement** of the original sample of 50 pennies. Had we left the slip of paper out of the hat each time we performed Step 4, this would be “resampling without replacement”.

Let’s study our 50 resampled pennies via an exploratory data analysis. First, let’s load the data into R by manually creating a data frame `pennies_resample` of our 50 resampled values. We’ll do this using the `tibble()` command from the `dplyr` package. Note that the 50 values you obtained will almost certainly not be the same as ours.

```
pennies_resample <- tibble(
  year = c(1976, 1962, 1976, 1983, 2017, 2015, 2015, 1962, 2016, 1976,
          2006, 1997, 1988, 2015, 2015, 1988, 2016, 1978, 1979, 1997,
          1974, 2013, 1978, 2015, 2008, 1982, 1986, 1979, 1981, 2004,
          2000, 1995, 1999, 2006, 1979, 2015, 1979, 1998, 1981, 2015,
          2000, 1999, 1988, 2017, 1992, 1997, 1990, 1988, 2006, 2000)
)
```

The 50 values of `year` in `pennies_resample` represent the resample of size 50 from the original sample of 50 pennies from the bank. We display the 50 resampled pennies in Figure 9.9.

Let’s compare the distribution of the numerical variable `year` of our 50 resampled pennies with the distribution of the numerical variable `year` of our original sample of 50 pennies from the bank in Figure 9.10.

```
ggplot(pennies_resample, aes(x = year)) +
  geom_histogram(binwidth = 10, color = "white") +
  labs(title = "Resample of 50 pennies")
ggplot(pennies_sample, aes(x = year)) +
  geom_histogram(binwidth = 10, color = "white") +
  labs(title = "Original sample of 50 pennies")
```

<ScaleContinuousPosition>

Range:



FIGURE 9.9: 50 resampled US pennies labelled.

```
Limits: 0 -- 15
```

```
<ScaleContinuousPosition>
```

```
Range:
```

```
Limits: 0 -- 15
```

Observe that while the general shape of the distribution of `year` is roughly similar, they are not identical. This is due to the variation induced by replacing the slips of paper each time we pull one out and recorded the year.

Recall from the previous section that the sample mean of the original sample of 50 pennies from the bank was 1995.44. What about for the `year` variable in `pennies_resample`? Any guesses? Let's have `dplyr` help us out as before:

```
pennies_resample %>%
  summarize(mean_year = mean(year))
```

```
# A tibble: 1 × 1
  mean_year
  <dbl>
1 1994.82
```

We obtained a different mean year of 1994.82. Again, this variation is induced by the “with replacement” from the “resampling with replacement” terminology we defined earlier.

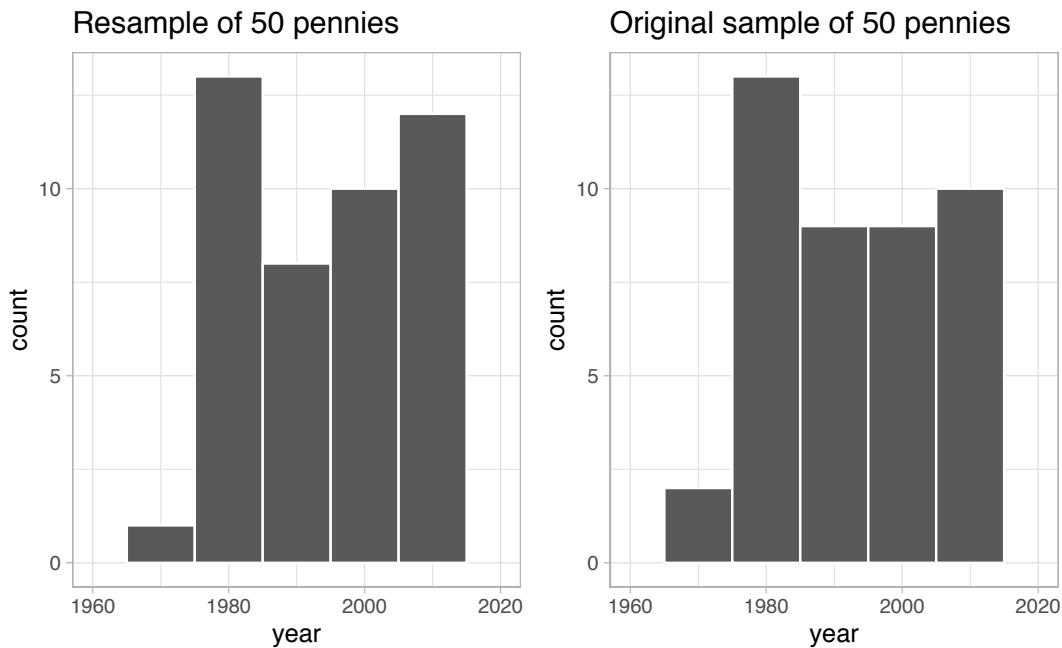


FIGURE 9.10: Comparing `year` in the resample `pennies_resample` with the original sample `pennies_sample`.

What if we repeated several times this resampling exercise many times? Would we obtain the same sample mean `year` value each time? In other words, would our guess at the mean year of all pennies in the US in 2019 be exactly 1994.82 every time? Just as we did in Chapter 8, let's perform this resampling activity with the help of 35 of our friends.

9.1.3 Resampling 35 times

Each of our 35 friends will repeat the same 5 steps above:

1. Start with 50 identically-sized slips of paper representing the 50 pennies.
2. Put the 50 small pieces of paper into a hat or tuque.
3. Mix the hat's contents and draw one slip of paper at random. Record the year somewhere.
4. Replace the slip of paper back in the hat!
5. Repeat Steps 3 and 4 49 more times, resulting in 50 recorded years.

Since we had 35 of our friends perform this task, we ended up with $35 \times 50 =$

1750 values. We recorded these values in a shared spreadsheet² with 50 rows (plus a header row) and 35 columns; we display a snapshot of the first 10 rows and 5 columns in Figure 9.11

A	B	C	D	E
1988	2018	2016	2002	2015
2002	1988	1971	1997	1976
2015	1999	1986	2002	2015
1998	2015	2002	2013	1981
1979	1962	1992	1997	1988
1971	2004	1976	1979	1985
1971	2018	2015	2018	1979
2015	1988	1985	1971	1971
1988	2013	1976	1998	1978
1979	1988	1999	1996	1979
1982	2008	2013	1999	1986
2004	1983	1997	1983	1974

FIGURE 9.11: Snapshot of shared spreadsheet of resampled pennies.

For your convenience, we've taken these $35 \times 50 = 1750$ values and saved them in `virtual_resamples`, a “tidy” data frame included in the `modernive` package:

```
pennies_resamples
```

```
# A tibble: 1,750 x 3
  replicate name   year
  <int> <chr> <dbl>
1       1 A     1988
2       1 A     2002
3       1 A     2015
4       1 A     1998
5       1 A     1979
6       1 A     1971
7       1 A     1971
8       1 A     2015
9       1 A     1988
10      1 A     1979
# ... with 1,740 more rows
```

What did each of our 35 friends obtain as the mean year? `dplyr` to the rescue once more! After grouping the rows by `name`, we summarize each group of rows with their mean `year`:

²https://docs.google.com/spreadsheets/d/1y3kOsU_wDrDd5eiJbEtLeHT9L5SvpZb_TrzwFBsouk0/

```
resampled_means <- pennies_resamples %>%
  group_by(name) %>%
  summarize(mean_year = mean(year))
resampled_means
```

```
# A tibble: 35 x 2
  name   mean_year
  <chr>     <dbl>
1 A         1992.5
2 AA        1995.86
3 B         1996.42
4 BB        1992.4
5 C         1996.32
6 CC        1995.88
7 D         1996.9
8 DD        1997.46
9 E         1991.22
10 EE        1998.44
# ... with 25 more rows
```

Observe that `resampled_means` has 35 rows corresponding to the 35 resample means based the 35 resamples performed by our friends. Furthermore, observe the variation in the 35 values in the variable `mean_year`. This variation exists because by “resampling with replacement”, our 35 friends obtained different resamples of 50 pennies, and thus obtained different resample mean year.

Since the variable `mean_year` is numerical, let’s visualize its distribution using a histogram in Figure 9.12. Note that adding the argument `boundary = 1990` to `geom_histogram()` sets the binning structure of the histogram so that one of the boundaries between bins is at the year 1990 exactly.

```
ggplot(resampled_means, aes(x = mean_year)) +
  geom_histogram(binwidth = 1, color = "white", boundary = 1990) +
  labs(x = "Resampled mean year")
```

Observe the following about the histogram in Figure 9.12:

- The distribution looks roughly normal.
- We rarely observe sample mean years less than in 1992.
- On the other side of the distribution, we rarely observe sample mean years greater than in 2000.
- The most frequently occurring values occur between roughly 1992 and 1998.

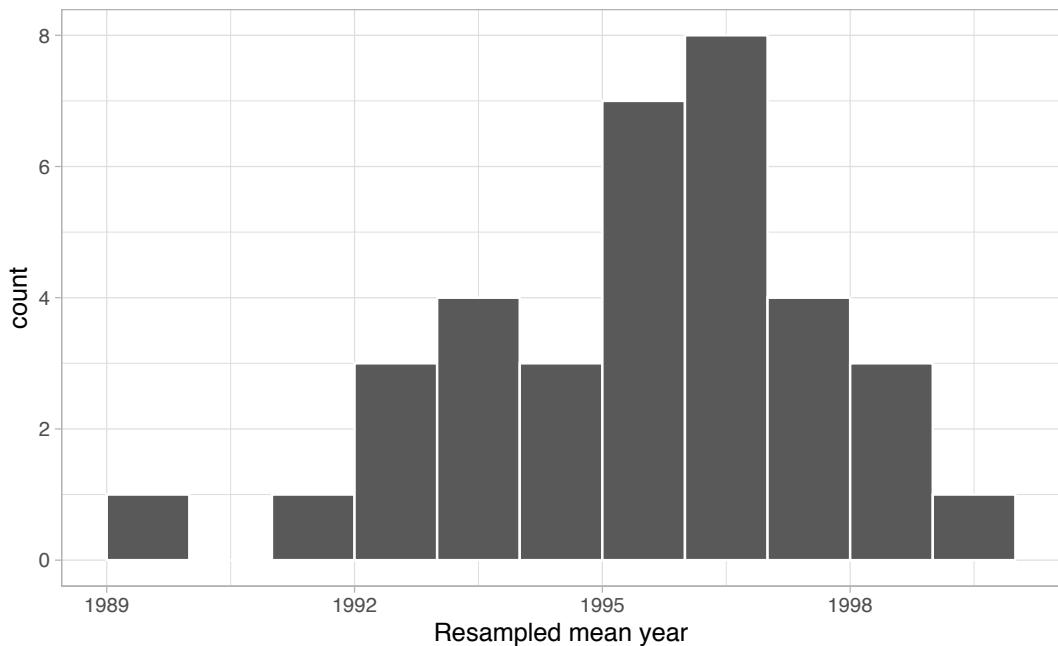


FIGURE 9.12: Distribution of 35 sample means from 35 resamples.

- The distribution of these 35 sample means based on 35 resamples is roughly centered at 1995, which is the sample mean of 1995.44 of the original sample of 50 pennies from the bank.

9.1.4 What did we just do?

What we just demonstrated in this activity is the statistical procedure known as *bootstrap resampling with replacement*. We used *resampling* to mimic the sampling variation we observe from sample-to-sample as we did in Chapter 8 on sampling, but this time using a *single* sample from the population.

In fact, the histogram of sample means from 35 resamples in Figure 9.12 is called the *bootstrap distribution* of the sample mean and it is an approximation of the *sampling distribution* of the same mean, a concept we introduced in Chapter 8. In Section 9.7 we'll show you that the *bootstrap distribution* is an approximation to the *sampling distribution*. Using this bootstrap distribution, we can study the effect of sampling variation on our estimates, in particular study the typical “error” of our estimates, known as the *standard error*.

In Section 9.2 we'll mimic our tactile resampling activity virtually on the computer. We can use a computer to do the resampling many more times than our 35 friends could possibly do. This will allow us to better understand the bootstrap distribution. In Section 9.3 we'll explicitly articulate our goals

for this chapter: understanding resampling variation, defining the statistical concept of a *confidence interval* by building on our pennies example, and discussing the interpretation of confidence intervals.

Following this framework on confidence intervals, we'll discuss the `dplyr` and `infer` package code needed to complete the process of *bootstrapping*, which is another name for this resampling approach that is most commonly found in developing confidence intervals. We've used one of the functions in the `infer` package already with `rep_sample_n()`, but there's a lot more to this package than just that. We'll introduce the tidy statistical inference framework that was the motivation for the `infer` package pipeline that will be the driving package throughout the rest of this book.

As we did in Chapter 8, we'll tie all these ideas together with a real-life case study in Section 9.6 involving data from an experiment about yawning from the US television show Mythbusters. The chapter concludes with a comparison of the sampling distribution and a bootstrap distribution using the balls data from Chapter 8 on sampling.

9.2 Computer simulation of resampling

Let's now mimic our tactile resampling activity virtually by using our computer.

9.2.1 Virtually resampling once

First, let's perform the virtual analog of resampling once. Recall that the `pennies_sample` data frame included in the `modernive` package contains the years of our original sample of 50 pennies from the bank. Furthermore, recall in Chapter 8 on sampling that we used the `rep_sample_n()` function as a virtual shovel to sample balls from our virtual bowl of 2400 balls.

```
virtual_shovel <- bowl %>%
  rep_sample_n(size = 50)
```

Let's combine these two elements to virtually mimic our resampling with replacement exercise involving the slips of paper representing our 50 pennies in Figure 9.3:

```
virtual_resample <- pennies_sample %>%
  rep_sample_n(size = 50, replace = TRUE)
```

Observe how we explicitly set the `replace` argument to `TRUE` in order to tell `rep_sample_n()` that we would like to sample pennies *with* replacement. Had we not set `replace = TRUE`, the function would've assumed the default value of `FALSE`. Additionally, since we didn't specify the number of replicates via the `reps` argument, the function assumes the default of one replicate `reps = 1`. Note also that the `size` argument is set to match the original sample size of 50 pennies. So what does `virtual_resample` look like?

```
View(virtual_resample)
```

We'll display only the first 10 out of 50 rows of `virtual_resample`'s contents in Table 8.2.

TABLE 9.2: First 10 resampled rows of 50 in virtual sample

replicate	ID	year
1	37	1962
1	1	2002
1	45	1997
1	28	2006
1	50	2017
1	10	2000
1	16	2015
1	47	1982
1	23	1998
1	44	2015

The `replicate` variable only takes on the value of 1 corresponding to us only having `reps = 1`, the `ID` variable indexes which of the 50 pennies from `pennies_sample` was resampled, and `year` denotes the year of minting.

Let's compute the mean `year` in our virtual resample of size 50 using data wrangling functions included in the `dplyr` package:

```
virtual_resample %>%
  summarize(resample_mean = mean(year))
```

```
# A tibble: 1 × 2
```

```
replicate resample_mean
<int>      <dbl>
1        1       1996
```

As when we did our tactile resampling, the resulting mean year is different than that mean year of our 50 originally sampled pennies of 1995.44.

9.2.2 Virtually resampling 35 times

Let's now have 35 virtual friends perform our virtual resampling exercise. Using these results, we'll be able to study the variability in the sample means from 35 resamples of size 50. Let's first add a `reps = 35` argument to `rep_sample_n()` to indicate we would like 35 replicates, or in other words, repeat the resampling with the replacement of 50 pennies 35 times.

```
virtual_resamples <- pennies_sample %>%
  rep_sample_n(size = 50, replace = TRUE, reps = 35)
virtual_resamples
```

```
# A tibble: 1,750 x 3
# Groups:   replicate [35]
  replicate     ID   year
  <int> <int> <dbl>
1       1     21  1981
2       1     34  1985
3       1      4  1988
4       1     11  1994
5       1     26  1979
6       1      8  1996
7       1     19  1983
8       1     21  1981
9       1     49  2006
10      1      2  1986
# ... with 1,740 more rows
```

The resulting `virtual_resamples` data frame has $35 \times 50 = 1750$ rows corresponding to 35 resamples of 50 pennies. What did each of our 35 virtual friends obtain as the mean year? We'll use the same `dplyr` verbs as we did in the previous section, but computing the mean for each of our virtual friends separately by adding a `group_by(replicate)`:

```
virtual_resampled_means <- virtual_resamples %>%
  group_by(replicate) %>%
  summarize(mean_year = mean(year))
virtual_resampled_means
```

```
# A tibble: 35 x 2
  replicate mean_year
  <int>     <dbl>
1       1    1995.58
2       2    1999.74
3       3    1993.7 
4       4    1997.1 
5       5    1999.42
6       6    1995.12
7       7    1994.94
8       8    1997.78
9       9    1991.26
10      10   1996.88
# ... with 25 more rows
```

Observe that `virtual_resampled_means` has 35 rows corresponding to the 35 resampled means and that the values of `mean_year` vary. Let's visualize the distribution of the numerical variable `mean_year` using a histogram in Figure 9.13.

```
ggplot(virtual_resampled_means, aes(x = mean_year)) +
  geom_histogram(binwidth = 1, color = "white", boundary = 1990) +
  labs(x = "Resampled mean year")
```

To convince ourselves that our virtual resampling indeed mimics the resampling done by our 35 friends, let's compare the bootstrap distribution we just virtually constructed with the bootstrap distribution our 35 friends constructed via tactile resampling in the previous section.

Recall that in the “resampling with replacement” scenario we are illustrating here both the above histograms have a special name: the *bootstrap distribution of the sample mean*. Furthermore, they are an approximation to the *sampling distribution* of the sample mean, a concept you saw in Chapter 8 on sampling. These distributions allow us to study the effect of sampling variation on our estimates of the true population mean, in this case the true mean year for *all* US pennies. However, unlike in Chapter 8 where we simulated the act of taking multiple samples, something one would never do in practice, bootstrap

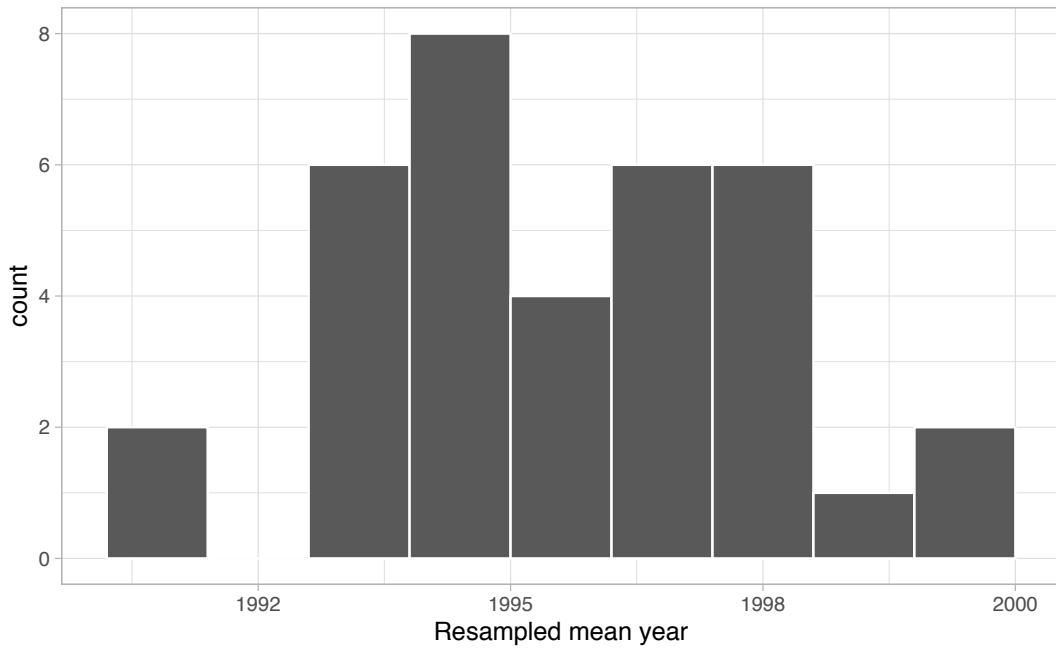


FIGURE 9.13: Distribution of 35 sample means from 35 resamples.

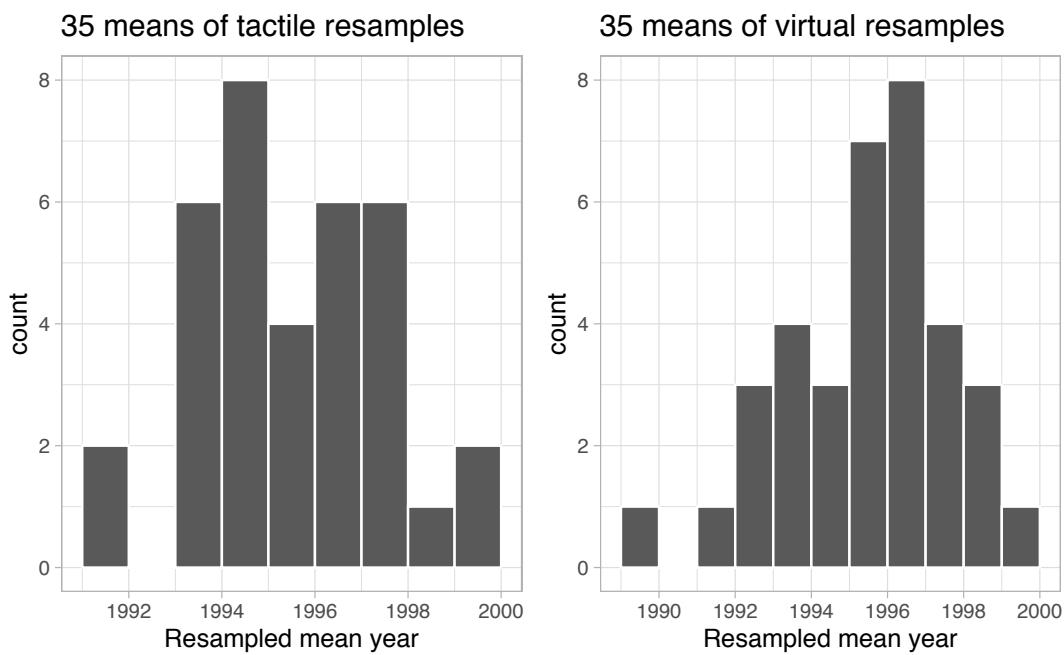


FIGURE 9.14: Comparing distributions of means from resamples.

distributions are constructed from a *single* sample, in this case the 50 original pennies from the bank.

Learning check

(LC9.1) Ask learners to compare the distributions since we did something similar in Chapter 8 and they should be well versed on this by now.

9.2.3 Virtually resampling 1000 times

Remember that one of the goals of resampling with replacement is to construct the bootstrap distribution, which is an approximation of the sampling distribution of the point estimate of interest, here the sample mean year. However, the bootstrap distribution of in Figure 9.13 is based only on 35 resamples and hence looks a little coarse. Let's increase the number of resamples to 1000 to better observe the shape and the variability from one resample to the next.

```
# Repeat resampling 1000 times
virtual_resamples <- pennies_sample %>%
  rep_sample_n(size = 50, replace = TRUE, reps = 1000)

# Compute 1000 sample means
virtual_resampled_means <- virtual_resamples %>%
  group_by(replicate) %>%
  summarize(mean_year = mean(year))
```

However, in the interest of brevity, going forward let's combine the above two operations into a single chain of %>% pipe operators:

```
virtual_resampled_means <- pennies_sample %>%
  rep_sample_n(size = 50, replace = TRUE, reps = 1000) %>%
  group_by(replicate) %>%
  summarize(mean_year = mean(year))
virtual_resampled_means

# A tibble: 1,000 x 2
  replicate mean_year
  <int>     <dbl>
```

```

1      1  1992.6
2      2  1994.78
3      3  1994.74
4      4  1997.88
5      5  1990
6      6  1999.48
7      7  1990.26
8      8  1993.2
9      9  1994.88
10     10 1996.3
# ... with 990 more rows

```

Let's visualize the bootstrap distribution of these 1000 sample means from 1000 virtual resamples looks like in Figure 9.15:

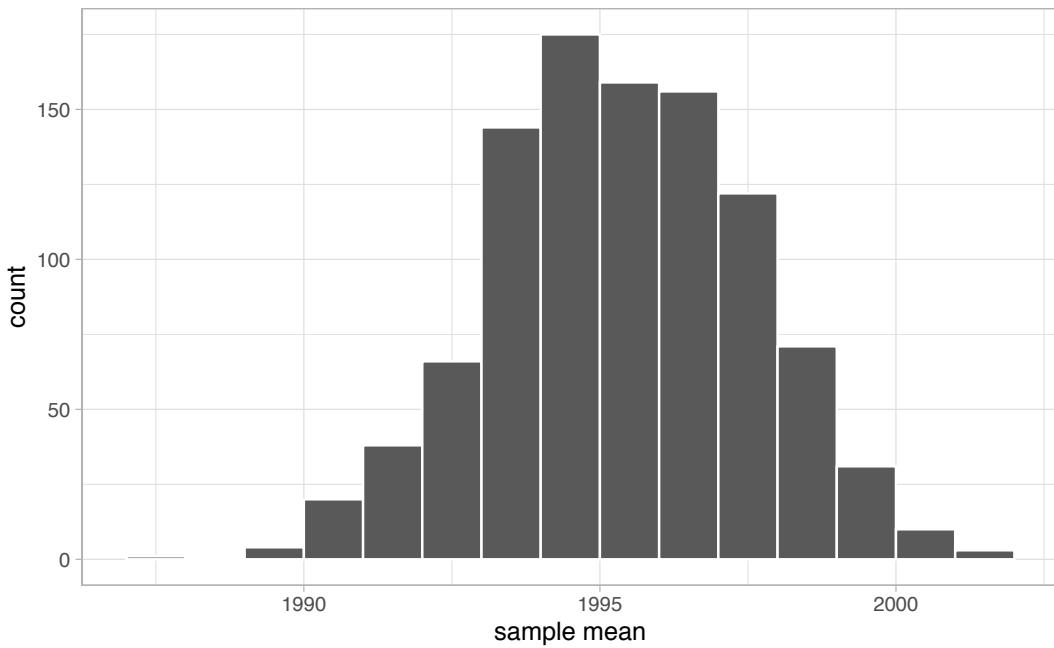


FIGURE 9.15: Bootstrap resampling distribution based on 1000 resamples.

Note here the bell shape starting to become more apparent. We now have a general sense for the range of values that the sample mean may take on in these resamples from this histogram of the bootstrap distribution. Do you have a guess as to where this histogram is centered? With it being close to symmetric, either the mean or the median would serve as a good estimate for the center here. Let's compute the mean:

```
virtual_resampled_means %>%
  summarize(mean_of_means = mean(mean_year))
```

```
# A tibble: 1 × 1
  mean_of_means
  <dbl>
1 1995.36
```

The mean of the 1000 means from 1000 resamples is 1995.365. Note that this is quite close to the mean of our original sample of 50 pennies from the bank: 1995.44. This is the case since each of the 1000 resamples are based on the original sample of 50 pennies.

Learning check

(LC9.2) What is the difference between a bootstrap distribution and a sampling distribution?

(LC9.3) Ask learners to summarize important features of the plot as was done in Chapter 8.

9.3 Understanding confidence intervals

Let's start this section with an analogy involving fishing. Say you are trying to catch a fish. On the one hand, you could use a spear, while on the other you could use a net. Using the net will probably yield better results! Bringing things back to the pennies: you are trying to estimate the true population mean year μ of all US pennies. Think of the value of μ as the fish.

On the one hand, we could use the appropriate point estimate/sample statistic to estimate μ , which we saw in Table 9.1 is the sample mean \bar{x} . Based on our sample of 50 pennies from the bank, the sample mean was 1995.44. Think of this value as fishing with a spear.

On the other hand, let's use our results from the previous section to construct a range of highly probable values for μ . Looking at the bootstrap distribution in Figure 9.15, between which two years would you say that "most" sample

means lie? While this question is somewhat subjective, saying that most sample means lie in the interval 1992 to 2000 would not be unreasonable. Think of this interval as fishing with a net.

What we've just illustrated is the concept of a *confidence interval*, which we'll abbreviate with "CI" throughout this book. So as opposed to a point estimate/sample statistic that estimates the value of an unknown population parameter with a single value, a *confidence interval* gives a range of plausible values. Going back to our analogy, point estimates/sample statistics can be thought of as spears, whereas confidence intervals can be thought of as nets.

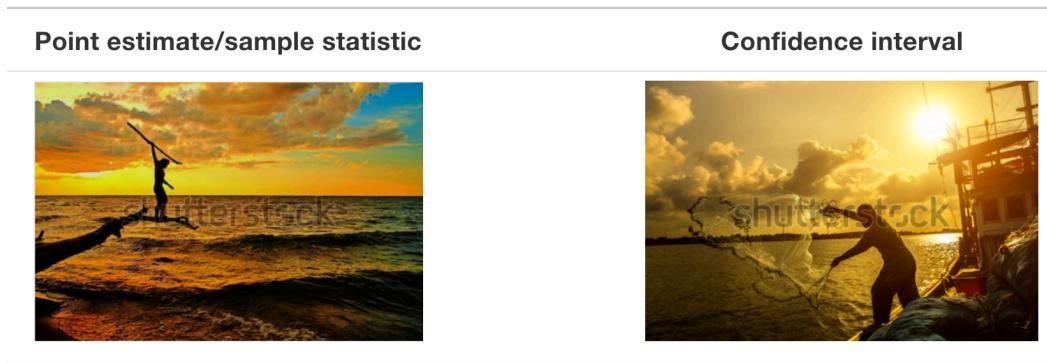


FIGURE 9.16: Analogy of difference between point estimates and confidence intervals.

Our proposed interval of highly probable values for μ of 1992 to 2000 was constructed by eye and is thus somewhat subjective. We now introduce two methods for constructing such intervals in a more principled fashion: the percentile method and the standard error method.

Both methods for confidence interval construction share some commonalities. First, they are both constructed from the bootstrap distribution, an example of which you created using 1000 bootstrap resamples with replacement in Subsection 9.2.3 and saved in the `virtual_resampled_means` data frame.

Second, they both require you to specify the *confidence level*. All other things being equal, higher confidence levels correspond to wider confidence intervals and lower confidence levels corresponding to narrower confidence intervals. Commonly used confidence levels include 90%, 95%, and 99%; we'll be mostly using 95% and hence constructing "95% confidence intervals for μ ".

9.3.1 Percentile method

Recall that the actual population mean year μ for all pennies in circulation in the US is unknown to us. The only way to know this value exactly would be to conduct a census of all pennies, a near impossible task. Instead, by constructing a confidence interval we'll obtain a range of plausible values for this unknown μ .

One method to construct this range is to use the middle 95% of the 1000 sample means we computed using bootstrap resampling with replacement. We can do this by computing the 2.5th and 97.5th percentiles, which are 1991.059 and 1999.283 respectively. For now, let's focus on the concepts behind a percentile method constructed confidence interval; we'll show you the code to compute these values in the next section.

We can mark these percentiles on the bootstrap distribution with red vertical lines in Figure 9.17. You can see that 95% of the values in the `mean_year` variable in `virtual_resampled_means` fall between the two endpoints, with 2.5% to the left of the left-most red line and 2.5% to the right of the right-most red line.

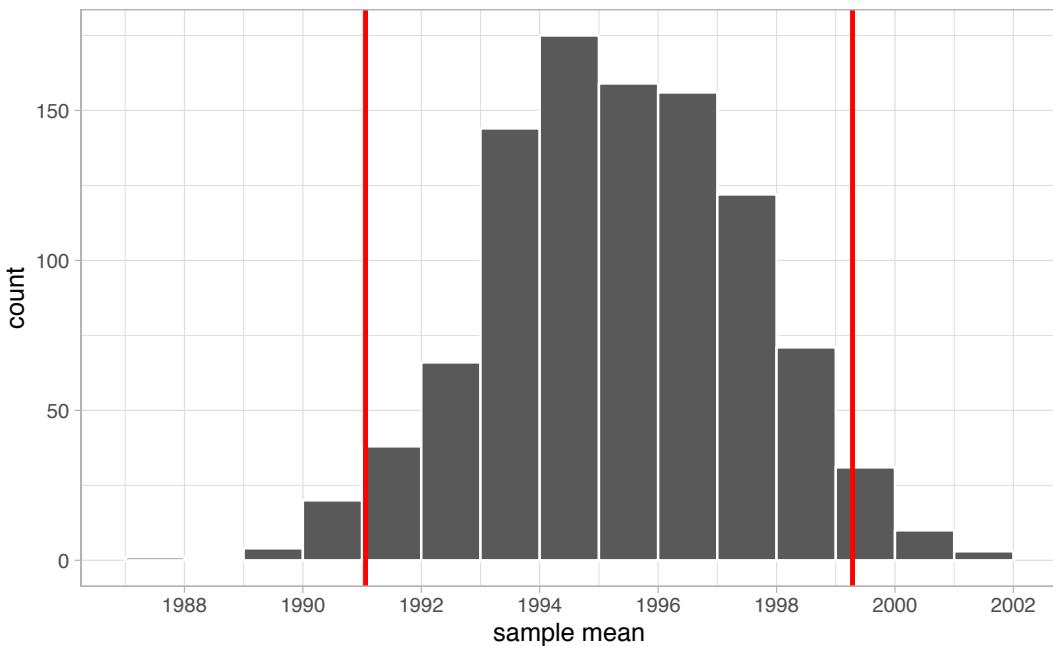


FIGURE 9.17: Percentile method 95 percent confidence interval.

9.3.2 Standard error method

Recall in Subsection 8.5.3, we saw that if a numerical variable follows a normal distribution, or in other words, the histogram of this variable is bell-shaped, then roughly 95% of values fall between ± 1.96 standard deviations of the mean. Given that our bootstrap distribution based on 1000 resamples with replacement in Figure 9.15 is normally shaped, let's use the above fact about normal distributions to construct a confidence interval in a different way.

First, the bootstrap distribution has a mean equal to \bar{x} : the sample mean of our original 50 pennies of 1995.44. In other words, the bootstrap distribution is centered at 1995.44. Second, let's compute the standard deviation of the bootstrap distribution

```
virtual_resampled_means %>%
  summarize(SE = sd(mean_year))

# A tibble: 1 × 1
  SE
  <dbl>
1 2.15466
```

What is this value? Recall that the bootstrap distribution is an approximation to the sampling distribution and that the standard deviation of the sampling distribution has a special name: the *standard error*. So in other words, 2.15 is an approximation of the standard error of \bar{x} .

Thus using our 95% rule of thumb about normal distributions from Subsection 8.5.3, we can use the following formula to determine the lower and upper endpoints of the 95% confidence interval for μ :

$$\begin{aligned}\bar{x} \pm 1.96 \cdot SE &= (\bar{x} - 1.96 \cdot SE, \bar{x} + 1.96 \cdot SE) \\ &= (1995.44 - 1.96 \cdot 2.15, 1995.44 + 1.96 \cdot 2.15) \\ &= (1991.15, 1999.73)\end{aligned}$$

Let's add the SE method confidence interval (in blue) to our previously constructed percentile method confidence (in red) in Figure 9.18.

We see that both methods produce nearly identical confidence intervals with the percentile method yielding (1991.06, 1999.28) while the standard error method being (1991.22, 1999.66). However, recall that we can only use the standard error rule when the bootstrap distribution is roughly normally-shaped.

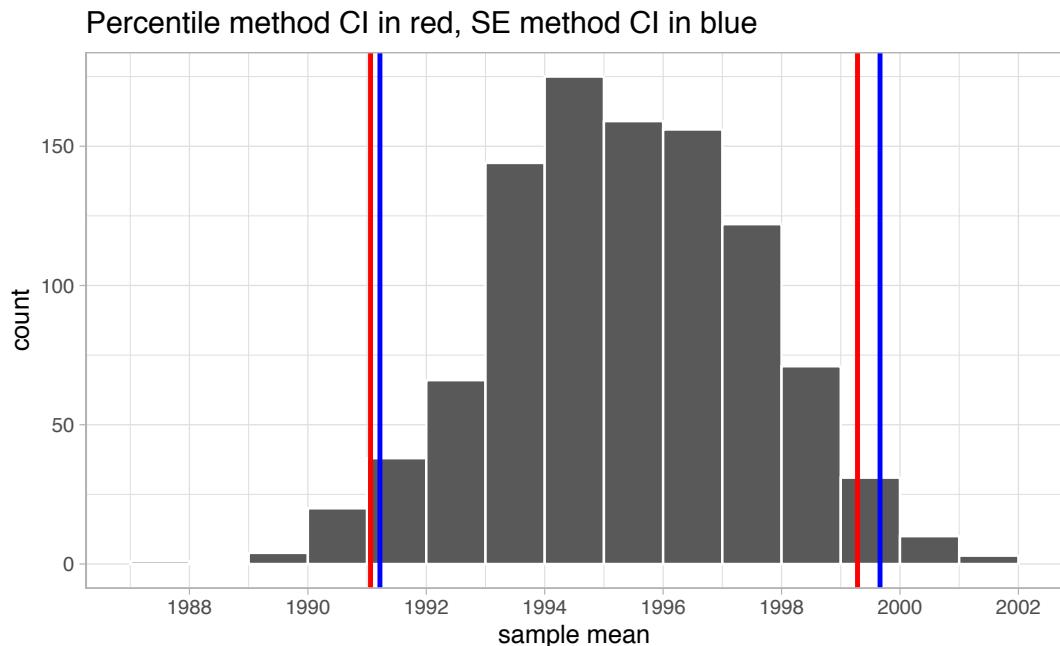


FIGURE 9.18: Comparing 95 percent confidence interval methods.

Now that we've introduced the concept of confidence intervals and laid out the intuition behind two methods for constructing them, let's explore the code that allows us to construct them.

Learning check

(LC9.4) What condition about the bootstrap distribution must be met for us to be able to construct confidence intervals using the standard error method?

(LC9.5) Say we wanted to construct a 68% confidence interval instead of a 95% confidence interval for μ ?

9.4 Constructing confidence intervals

Recall that the process of resampling with a replacement we performed by hand in Section 9.1 and virtually in Section 9.2 is known as *bootstrapping*. The term bootstrapping originates in the expression of “pulling oneself up by

their bootstraps”: to “succeed only by one’s own efforts or abilities.”³ From a statistical perspective, it alludes to succeeding in being able to study the effects of sampling variation on estimates from the “effort” of a single sample. Or more precisely, constructing an approximation to the sampling distribution using only one sample.

To perform this resampling with replacement virtually in Section 9.2, we used the `rep_sample_n()` function, making sure that the size of the resamples matched the original sample size. In this section, we’ll build off these ideas to construct confidence intervals using a new package: the `infer` package for “tidy” and transparent statistical inference.

9.4.1 Original workflow

Recall that in Section 9.2, we virtually performed bootstrap resampling with replacement to build the bootstrap distribution, which in turn is an approximation to the sampling distribution we saw in Chapter 8 but using only a single sample. Let’s revisit the flow using the `%>%` pipe operator:

First, we used the `rep_sample_n()` function to sample `size = 50` pennies with replacement from the original sample of 50 pennies in `pennies_sample` by setting `replace = TRUE`. Furthermore, we repeated this resampling 1000 times by setting `reps = 1000`:

```
pennies_sample %>%
  rep_sample_n(size = 50, replace = TRUE, reps = 1000)
```

Second, since for each of our 1000 resamples of size 50, we want to compute a separate sample mean, we used the `dplyr` verb `group_by()` to group observations/rows together by the `replicate` variable...

```
pennies_sample %>%
  rep_sample_n(size = 50, replace = TRUE, reps = 1000) %>%
  group_by(replicate)
```

... followed by using `summarize()` to compute the sample `mean()` year from each `replicate` group:

```
pennies_sample %>%
```

³https://en.wiktionary.org/wiki/pull_oneself_up_by_one%27s_bootstraps

```
rep_sample_n(size = 50, replace = TRUE, reps = 1000) %>%  
  group_by(replicate) %>%  
  summarize(mean_year = mean(year))
```

For this simple case, we can get by with using the `rep_sample_n()` function and a couple of `dplyr` verbs to construct the bootstrap distribution. However, using only `dplyr` verbs only provides us with a limited set of tools. For more complicated situations, we need a little more firepower. Let's repeat the above using the `infer` package.

9.4.2 infer package workflow

Just as `group_by() %>% summarize()` produces a useful workflow in `dplyr`, we can also use `specify() %>% calculate()` to compute summary measures on our original sample data. It's often helpful both in confidence interval calculations, but also in hypothesis testing to identify what the corresponding statistic is in the original data. For our example on penny age, we computed above a value of `x_bar` using the `summarize()` verb in `dplyr`:

```
pennies_sample %>%  
  summarize(stat = mean(year))
```

This can also be done by skipping the `generate()` step in the pipeline feeding `specify()` directly into `calculate()`:

```
pennies_sample %>%  
  specify(response = year) %>%  
  calculate(stat = "mean")
```

This shortcut will be particularly useful when the calculation of the observed statistic is tricky to do using `dplyr` alone. This is particularly the case when working with more than one explanatory variable as will be seen in Chapter 10.

The `infer` package makes efficient use of the `%>%` pipe operator we saw in Chapter 4 to spell out the sequence of steps necessary to perform statistical inference in a “tidy” and transparent fashion. Much in the same way that the functions in the `dplyr` have intuitive verb-based names, the `infer` package’s functions are also verbs that spell out the computational process of constructing confidence intervals, as well as hypothesis tests as we’ll see in Chapter 10.

Let's illustrate the sequence of verbs to construct a confidence interval for μ , the population mean year of minting of all pennies in the US.

1. `specify` variables



FIGURE 9.19: Diagram of `specify()` variables.

The `specify()` function is used to choose which variables in a data frame will be the focus of the statistical inference. We do this by specifying the `response` argument. For example, in our `pennies_sample` data frame of the 50 pennies sampled from the bank, the variable of interest is `year`:

```
pennies_sample %>%
  specify(response = year)
```

```
Response: year (numeric)
# A tibble: 50 x 1
  year
  <dbl>
1 2002
2 1986
3 2017
4 1988
5 2008
6 1983
7 2008
8 1996
9 2004
10 2000
# ... with 40 more rows
```

Notice how the data itself doesn't change, but the `Response: year (numeric)`

meta-data does. This is similar to how the `group_by()` verb from `dplyr` doesn't change the data, but only adds "grouping" meta-data as we saw in Section 4.4.

We can also specify which variables will be the focus of the statistical inference using a `formula = y ~ x` argument, where `y` is the response variable, `x` is the explanatory variable, with both separated by a "tilde" `~`. Recall that you used this same formula notation within the `lm()` function in Chapters 6 and 7 when fitting regression models. The following use of `specify()` with the `formula` argument yields the same result as above:

```
pennies_sample %>%
  specify(formula = year ~ NULL)
```

Since in the case of pennies we only have a response variable and no explanatory variable of interest, we set the `x` on the right-hand side of the `~` to `NULL`.

While in the case of the pennies either specification works just fine, we'll see examples later on where we have no choice but to use the `formula` specification, in particular in the upcoming Section 9.6 on comparing two proportions and Chapter 10 on hypothesis testing.

2. generate replicates

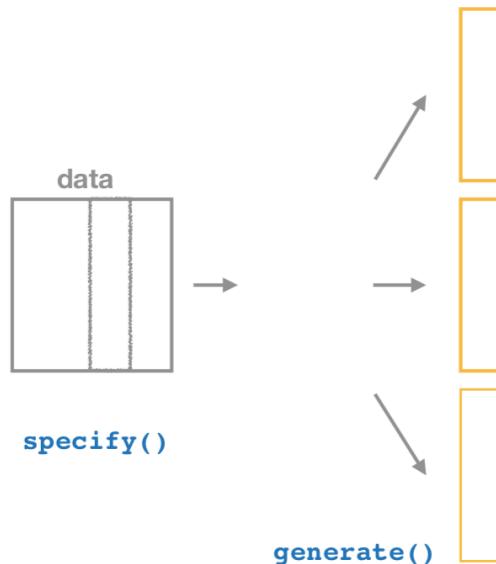


FIGURE 9.20: Diagram of `generate()` replicates.

After we `specify()` the variables of interest, we pipe the results into the `gener-`

`ate()` function to generate resampling replicates, or in other words, repeat the resampling process a large number of times. The `generate()` function's first argument is `reps`, which is used to give how many different repetitions one would like to perform. The second argument `type` determines the type of resampling we'd like to perform.

In our case, since we want to resample the 50 pennies in `pennies_sample` with replacement 1000 times, we set `reps = 1000` and `type = "bootstrap"` indicating that we want to perform bootstrap resampling.

```
pennies_sample %>%
  specify(response = year) %>%
  generate(reps = 1000, type = "bootstrap")
```

```
Response: year (numeric)
# A tibble: 50,000 × 2
# Groups:   replicate [1,000]
  replicate year
  <int> <dbl>
1       1 1996
2       1 1988
3       1 1979
4       1 1978
5       1 1983
6       1 1981
7       1 1993
8       1 1996
9       1 1992
10      1 1978
# ... with 49,990 more rows
```

Note the the resulting data frame has 50,000 rows. This is because we performed resampling of 50 pennies with replacement 1000 times and thus $50,000 = 1000 \times 50$. Accordingly, the variable `replicate`, indicating which resample each row belongs to, has the value 1 50 times, the value 2 50 times, all the way through to the value 1000 50 times.

The default value of the `type` argument is `"bootstrap"`, so if the last line above were written as `generate(reps = 1000)`, we'd obtain the same results.

Comparing with original workflow: Note that the steps up of the `infer` workflow so far produce the same results as the original workflow using the `rep_sample_n()` function we saw earlier. In other words, the following two code chunks produce similar results:

```
# infer workflow:           # Original workflow:
pennies_sample %>%      pennies_sample %>%
  specify(response = year) %>%   rep_sample_n(size = 50, replace = TRUE,
  generate(reps = 1000)          reps = 1000)
```

3. calculate summary statistics

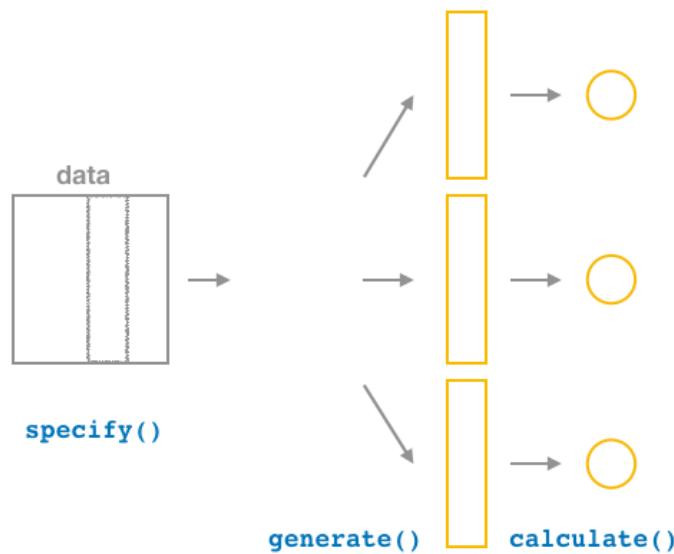


FIGURE 9.21: Diagram of calculate() summary statistics.

After we generate() many replicates of bootstrap resampling with replacement, we next want to condense each of 1000 resamples of size 50 to a single statistic value. As seen in the diagram, the calculate() function does this.

In our case as we did earlier, we want to calculate the mean `year` for each bootstrap resample of size 50. To do so, we set the `stat` argument to "mean". You can also set the `stat` argument to a variety of other common summary statistics, like "median", "sum", "sd" (standard deviation), and "prop" (proportion); we'll see examples of their use throughout the remaining chapters. Let's save the result in a data frame called `bootstrap_distribution`:

```
bootstrap_distribution <- pennies_sample %>%
  specify(response = year) %>%
  generate(reps = 1000) %>%
```

```
calculate(stat = "mean")
bootstrap_distribution
```

```
# A tibble: 1,000 x 2
  replicate   stat
  <int>   <dbl>
1       1 1993.48
2       2 1993.8
3       3 1996.88
4       4 1995.34
5       5 1996.98
6       6 1995.72
7       7 1995.36
8       8 1992.6
9       9 1994.24
10      10 1993.16
# ... with 990 more rows
```

We see that the resulting data frame has 1000 rows and 2 columns corresponding to the 1000 replicates and the mean `year` for each bootstrap resample saved in the variable `stat`.

Comparing with original workflow: You may have recognized at this point that the `calculate()` step in the `infer` workflow produces the same output as the `group_by() %>% summarize()` steps in the original workflow:

<pre><code># infer workflow: pennies_sample %>% specify(response = year) %>% generate(reps = 1000) %>% calculate(stat = "mean")</code></pre>	<pre><code># Original workflow: pennies_sample %>% rep_sample_n(size = 50, replace = TRUE, reps = 1000) %>% group_by(replicate) %>% summarize(mean_year = mean(year))</code></pre>
---	--

4. `visualize` the results

The `visualize()` verb provides a quick way to visualize the bootstrap distribution as a histogram of the numerical `stat` variable's values.

```
visualize(bootstrap_distribution)
```

Comparing with original workflow: In fact, `visualize()` is a *wrapper function* for the `ggplot2::ggplot` function.

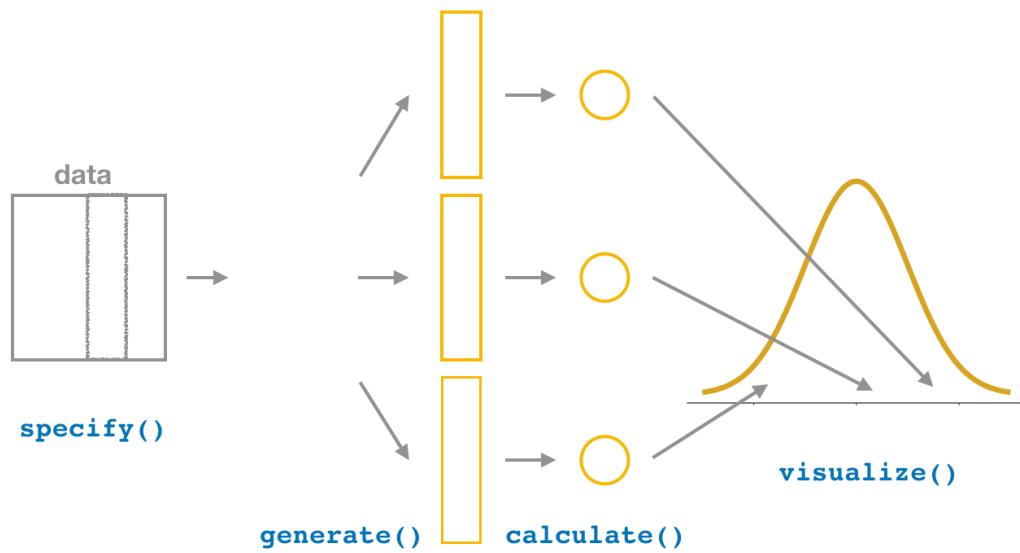


FIGURE 9.22: Diagram of `visualize()` results.

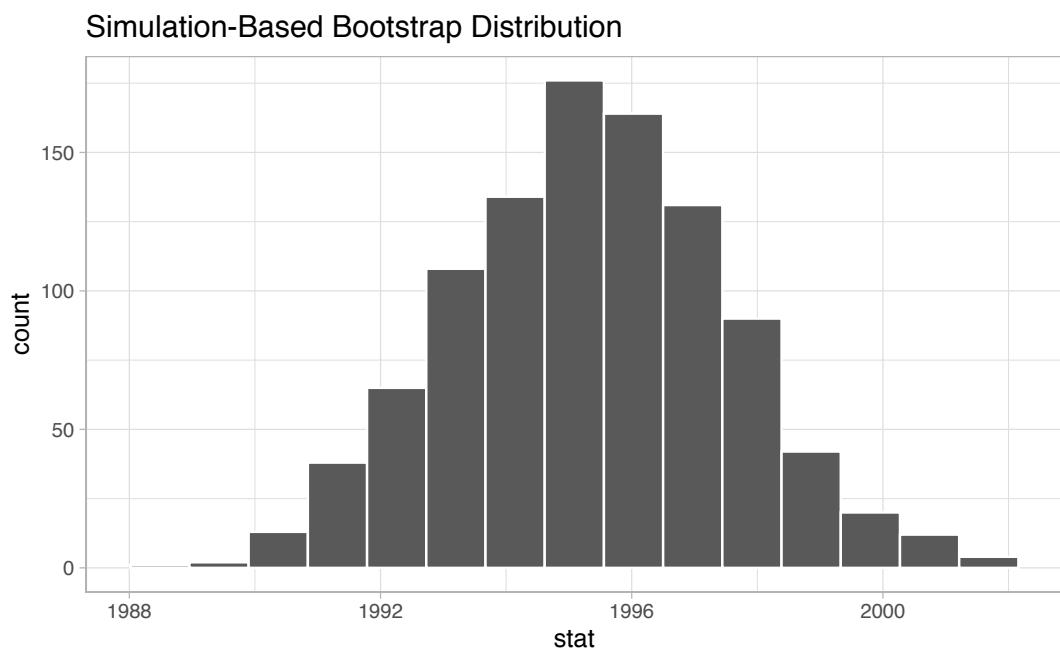


FIGURE 9.23: Bootstrap distribution.

tion for the `ggplot()` function that uses a `geom_histogram()` layer. Recall that we illustrated the concept of a wrapper function in Figure 6.5 in Section 6.1.2.

```
# infer workflow:          # Original workflow:
visualize(bootstrap_distribution) ggplot(bootstrap_distribution,
                                         aes(x = stat)) +
                                         geom_histogram()
```

The `visualize()` function can take many other arguments which we'll see momentarily to customize the plot further. It also works with helper functions to do the shading of the histogram values corresponding to the confidence interval values.

Let's recap the steps of the `infer` workflow for creating a visualization of the bootstrap distribution.

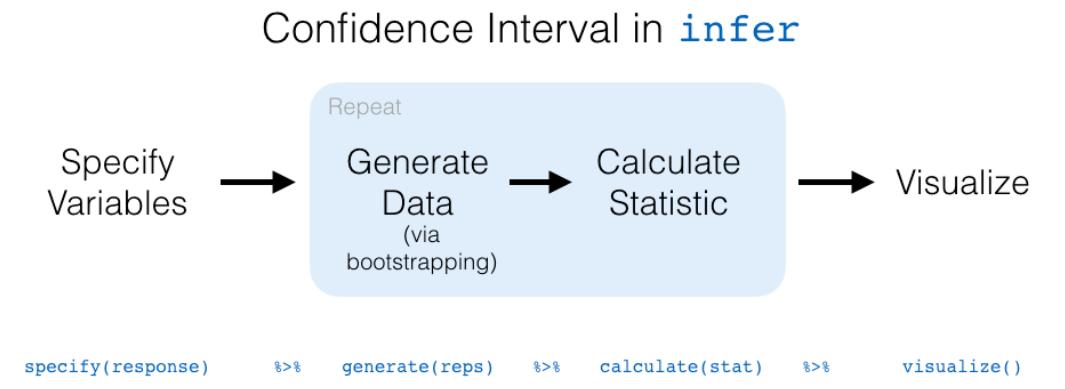


FIGURE 9.24: `infer` package workflow for confidence intervals.

Recall how we introduced two different methods for constructing 95% confidence intervals for an unknown population parameter in Section 9. Let's now check out the `infer` package code to explicitly construct these. There are also some additional neat functions to visualize the resulting confidence intervals built-in!

9.4.3 Percentile method with `infer`

Recall the percentile method for constructing 95% confidence intervals we introduced in Section 9.3.1. This method sets the lower endpoint at the 2.5th percentile of the bootstrap distribution and similarly sets the upper-endpoint at the 97.5th percentile. The resulting interval captures the middle 95% of the values of the sample mean in the bootstrap distribution.

We can compute the 95% confidence interval by piping the `bootstrap_distribution` data frame we created above into the `get_confidence_interval()` function from the `infer` package, with the `confidence_level` set to 0.95 and the confidence interval `type` to be percentile. Let's save the results in `percentile_ci`.

```
percentile_ci <- bootstrap_distribution %>%
  get_confidence_interval(level = 0.95, type = "percentile")
percentile_ci
```

```
# A tibble: 1 × 2
`2.5%` `97.5%
<dbl>   <dbl>
1 1991.16 1999.58
```

If we would like to visualize the interval (1991.16, 1999.58), we can pipe the `bootstrap_distribution` data frame into the `visualize()` function and add a `shade_confidence_interval()` layer to our plot with the `endpoints` argument to be `percentile_ci`:

```
visualize(bootstrap_distribution) +
  shade_confidence_interval(endpoints = c(1991.28, 1999.76))
```

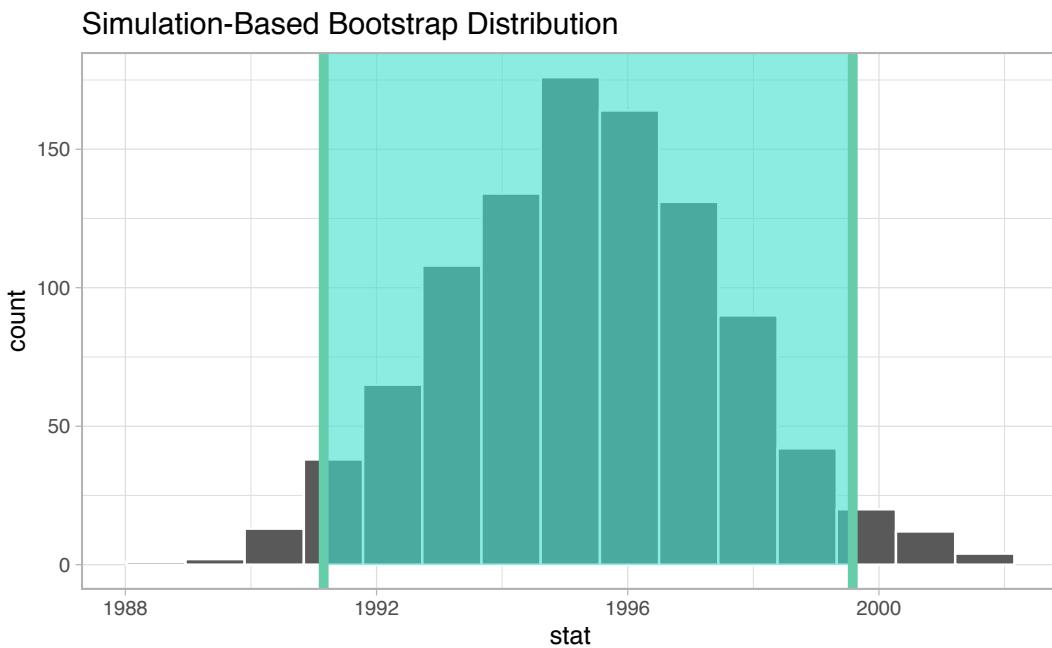


FIGURE 9.25: Percentile method 95 percent confidence interval.

Observe that 95% of the sample means stored in the `stat` variable in `bootstrap_distribution` falls between the two endpoints marked with the darker lines, with 2.5% of the sample means to the left of the shaded area and 2.5% of the sample means to the right. You also have the option to change the colors of the shading using the `color` and `fill` arguments. There's also the alias `shade_ci()` for folks that don't want to type out all of `confidence_interval` and prefer `ci` instead.

```
visualize(bootstrap_distribution) +
  shade_ci(endpoints = percentile_ci, color = "hotpink", fill = "khaki")
```

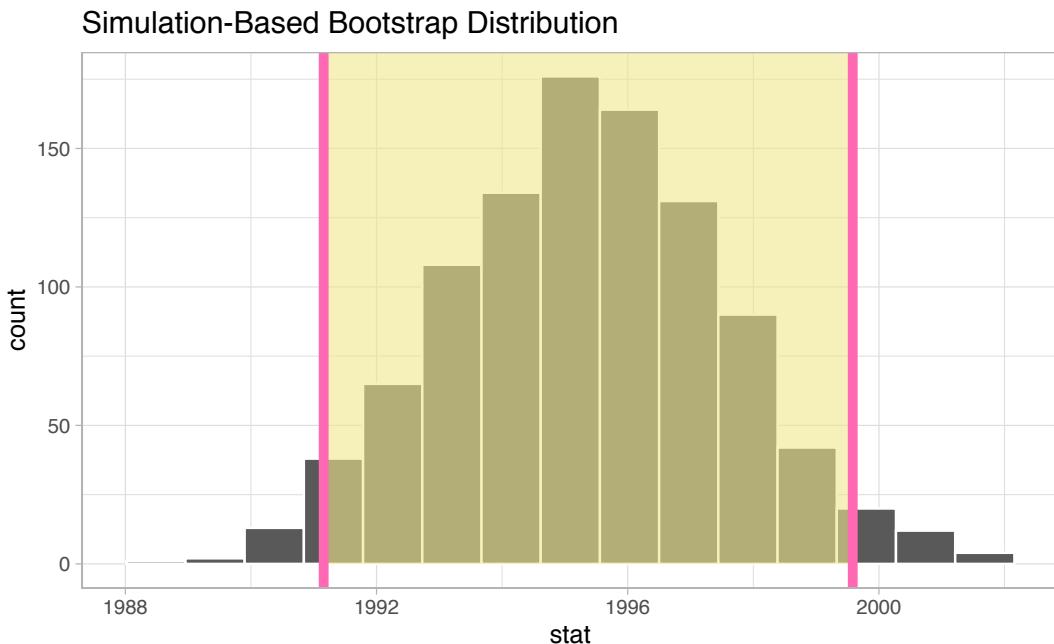


FIGURE 9.26: Alternate display of percentile method 95 percent confidence interval.

9.4.4 Standard error method with `infer`

Recall the standard error method for constructing 95% confidence intervals we introduced in Section 9.3.2. For any distribution that is normally shaped, roughly 95% of values lie within two standard deviations of the mean. In the case of the bootstrap distribution, the standard deviation has a special name: the standard error. So using our rule of thumb about normally shaped distributions, a 95% confidence interval is $\bar{x} \pm 1.96 \cdot SE = (\bar{x} - 1.96 \cdot SE, \bar{x} + 1.96 \cdot SE)$.

We can compute the 95% confidence interval by piping the `bootstrap_distribution` data frame we created above into the `get_confidence_interval()` function. First, we set the `type` argument set to be "se". Second, we must specify the `point_estimate` argument in order to set the center of the confidence interval: we set this to be sample mean of the original sample of 50 pennies of 1995.44.

```
standard_error_ci <- bootstrap_distribution %>%
  get_confidence_interval(type = "se", point_estimate = 1995.44)
standard_error_ci

# A tibble: 1 × 2
  lower    upper
  <dbl>    <dbl>
1 1991.16 1999.72
```

If we would like to visualize the interval (1991.16, 1999.72), we can pipe the `bootstrap_distribution` data frame into the `visualize()` function and add a `shade_confidence_interval()` layer to our plot with the `endpoints` argument to be `standard_error_ci`:

```
visualize(bootstrap_distribution) +
  shade_confidence_interval(endpoints = standard_error_ci)
```

As noted in Section 9.3, both methods produce similar confidence intervals:

- Percentile method: (1991.16, 1999.58)
- Standard error method: (1991.16, 1999.72)

9.5 Interpreting confidence intervals

Now that we've shown you how to construct confidence intervals using a sample drawn from the population, let's now focus on how to interpret them. In order to interpret a confidence interval thoroughly however, we need to know the true value of the population parameter in question.

In the case of our pennies example, we don't know the value of the population parameter of interest: the population mean year of minting μ of all pennies in the US. Furthermore, we probably never will know this value given the near impossibility of performing a census.

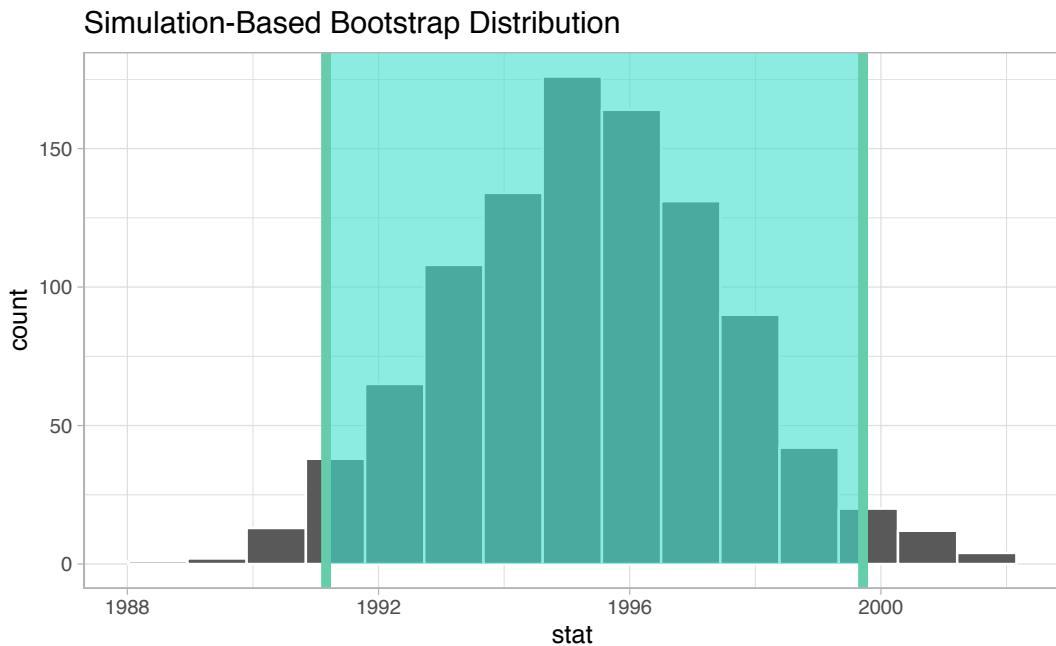


FIGURE 9.27: Standard error method 95 percent confidence interval.

In the case of our sampling bowl from Chapter 8 however, we can compute the value of the population parameter of interest: the population proportion p of the $N = 2400$ balls that are red. Recall the `bowl` data frame in the `moderndive` package contains our population of interest. We can calculate the proportion of red balls in this population to get the value of p :

```
bowl %>%
  summarize(p_red = mean(color == "red"))
```

```
# A tibble: 1 × 1
  p_red
  <dbl>
1 0.375
```

At this point you might be asking yourself: If we already know that $p = 0.375 = 37.5\%$ of the bowl's balls are red, then why are we sampling to estimate this value, to begin with? Your skepticism is merited! Recall from Section 8.2, that this was virtual “simulation” used to study sampling. In any real-world setting, however, we would not know the value of the population proportion p and hence we have no choice but use sampling to estimate it.

Bringing the discussion back to confidence intervals constructed based on samples from the bowl, we have to ask ourselves: Does the interval include $p =$

$0.375 = 37.5\%$ or not? Alternatively, going back to our “fishing with a spear” versus “fishing with a net” analogy from Section 9.3, we have to ask ourselves: Did our net capture the fish or not?

9.5.1 Did the net capture the fish?

Recall from in Table 8.1 that we had 33 groups of friends repeat this sampling simulation, each taking samples of size 50 from the bowl and compute the sample proportion of red \hat{p} , resulting in 33 such estimates of p . Let’s focus on Ilyas and Yohan’s sample in Table 8.1 where they observed 21 red balls out of the 50 in their shovel, in other words, their sample proportion $\hat{p} = 21/50 = 0.42 = 42\%$. This data is stored in the `bowl_sample_1` data frame in the `moderndive` package:

```
bowl_sample_1
```

```
# A tibble: 50 × 1
  color
  <chr>
  1 white
  2 white
  3 red
  4 red
  5 white
  6 white
  7 red
  8 white
  9 white
 10 white
# ... with 40 more rows
```

Let’s follow the `infer` package workflow from Section 9.4.2 to create a percentile method 95% confidence interval for p using sample data in `bowl_sample_1`.

1. specify variables

First, we `specify()` the response variable of interest `color`:

```
bowl_sample_1 %>%
  specify(response = color)
```

Error: A level of the response variable `color` needs to be specified for the `success` argument in `specify()`.

Whoops! We need to define which event is of interest! red or white balls? Since we are interested in proportions red, let's set success to be "red":

```
bowl_sample_1 %>%
  specify(response = color, success = "red")
```

```
Response: color (factor)
# A tibble: 50 × 1
  color
  <fct>
  1 white
  2 white
  3 red
  4 red
  5 white
  6 white
  7 red
  8 white
  9 white
 10 white
# ... with 40 more rows
```

2. generate replicates

Second, we `generate()` 1000 replicates via bootstrap with replacement of our original sample of 50 balls in `bowl_sample_1` by setting `reps = 1000` and `type = "bootstrap"`.

```
bowl_sample_1 %>%
  specify(response = color, success = "red") %>%
  generate(reps = 1000, type = "bootstrap")
```

Note the resulting data frame has 50,000 rows. This is because we performed resampling of 50 balls with replacement 1000 times and thus $50,000 = 1000 \times 50$. Accordingly, the variable `replicate`, indicating which resample each row belongs to, has the value 1 50 times, the value 2 50 times, all the way through to the value 1000 50 times. Recall generating 1000 replicates means we are repeating the resampling 1000 times so that we can study the sampling variation from resampling to resample!

3. calculate summary statistics

Third, we summarize each of 1000 resamples of size 50 with their proportion of “successes”, in other words, the proportion of the balls that are "red". Let's save the result in a data frame called `sample_1_bootstrap`:

```
sample_1_bootstrap <- bowl_sample_1 %>%
  specify(response = color, success = "red") %>%
  generate(reps = 1000, type = "bootstrap") %>%
  calculate(stat = "prop")
sample_1_bootstrap
```

```
# A tibble: 1,000 x 2
  replicate   stat
  <int>   <dbl>
1       1  0.36
2       2  0.42
3       3  0.52
4       4  0.38
5       5  0.38
6       6  0.38
7       7  0.46
8       8  0.3
9       9  0.5
10      10  0.46
# ... with 990 more rows
```

4. visualize the results

Fourth and lastly, let's compute the resulting 95% confidence interval.

```
percentile_ci_1 <- sample_1_bootstrap %>%
  get_confidence_interval(level = 0.95, type = "percentile")
percentile_ci_1
```

```
# A tibble: 1 x 2
  `2.5%`  `97.5%`
  <dbl>    <dbl>
1 0.28  0.540500
```

Furthermore, let's visualize the bootstrap distribution where we've adjusted the number of bins to better see the resulting shape. Furthermore, we add a

vertical red line at $\hat{p} = 21/50 = 0.42 = 42\%$ using `geom_vline()` by setting the `xintercept` argument.

```
sample_1_bootstrap %>%
  visualize(bins = 15) +
  shade_confidence_interval(endpoints = c(0.28, 0.56)) +
  geom_vline(xintercept = 0.375, col = "red")
```

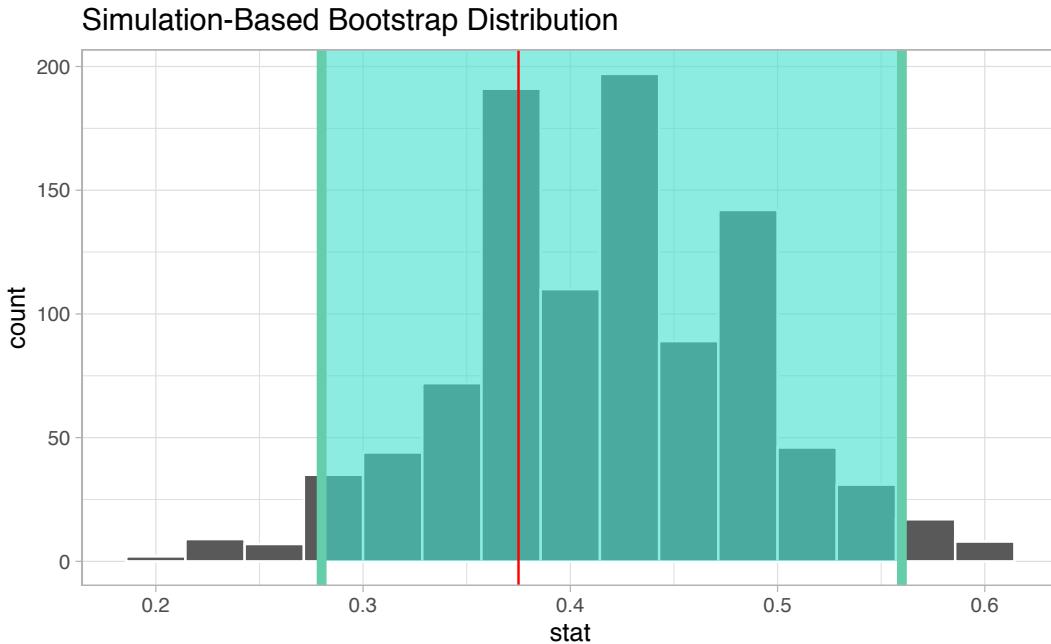


FIGURE 9.28: Bootstrap distribution.

Did Ilyas and Yohan's net capture the fish? In other words, did the 95% confidence interval for p based on the 50 balls they sampled contain the true value of p , the population proportion of the bowl's balls that are red? Yes! 0.375 is between the endpoints of our confidence interval (0.28, 0.54).

However, will *every* 95% confidence interval for p capture this value? In other words, if we had a different sample of size 50 and constructed a confidence interval using the same method, would we be guaranteed that it contained the population proportion p as well? Let's study the effect of sampling variation by randomly sampling *another* 50 balls from our virtual `bowl` using our virtual shovel:

```
bowl_sample_2 <- bowl %>%
```

```
rep_sample_n(size = 50)
bowl_sample_2
```

```
# A tibble: 50 × 3
# Groups:   replicate [1]
  replicate ball_ID color
  <int>     <int> <chr>
1       1      1665 red
2       1      1312 red
3       1      2105 red
4       1       810 white
5       1      189 white
6       1     1429 white
7       1     2294 red
8       1     1233 white
9       1     1951 white
10      1     2061 white
# ... with 40 more rows
```

Let's perform the same steps of the `infer` pipeline we did on `bowl_sample_1` to generate *another* 95% confidence interval for p based on the new sample of 50 balls in `bowl_sample_2`. First we create the bootstrap distribution of the sample proportion \hat{p} and save the results in `sample_2_bootstrap`:

```
sample_2_bootstrap <- bowl_sample_2 %>%
  specify(response = color, success = "red") %>%
  generate(reps = 1000, type = "bootstrap") %>%
  calculate(stat = "prop")
sample_2_bootstrap
```

```
# A tibble: 1,000 × 2
  replicate  stat
  <int>  <dbl>
1       1  0.36
2       2  0.38
3       3  0.42
4       4  0.26
5       5  0.5 
6       6  0.32
7       7  0.4 
8       8  0.32
```

```

9          9  0.5
10         10  0.44
# ... with 990 more rows

```

We once again compute the percentile-based confidence interval.

```

percentile_ci_2 <- sample_2_bootstrap %>%
  get_confidence_interval(level = 0.95, type = "percentile")
percentile_ci_2

```

```

# A tibble: 1 × 2
`2.5%` `97.5%`
<dbl>   <dbl>
1     0.22      0.5

```

Does this new net capture the fish? In other words, did the 95% confidence interval for p based on the 50 newly sampled balls contain the true value of p , the population proportion of the bowl's balls that are red? Yes! 0.375 is between the endpoints of our confidence interval (0.22, 0.5).

Let's now repeat this process 100 more times, leaving us with 100 different 95% confidence intervals p derived from 100 different samples of size 50 balls from the population `bowl`. Let's visualize the results in Figure 9.29 where:

1. We mark the true population proportion red $p = 0.375 = 0.375\%$ with a vertical red line.
2. We mark each of the 100 95% confidence intervals for p with horizontal lines. These are the “nets.”
3. The horizontal line is colored orange if the confidence interval “captures” the true value of p in red and the line is blue otherwise.
4. We also mark each line with a dot indicated the value of the point estimate: the sample proportion \hat{p} . These are the “spears.”

Of the 100 confidence intervals based on 100 samples of size $n = 50$, 96 of them captured the population mean $p = 0.375$, whereas 4 of them didn't. In other words, 96 of our nets caught the fish whereas 4 of our nets didn't.

This is where the 95% confidence level we defined in Section 9.3 comes into play: for every 100 confidence intervals based on 100 different random examples, we *expect* that 95 of them will capture p and 5 won't. Note that “expect” is a probabilistic statement that averages over the sampling variation. In other words, for every 100 confidence intervals, we will observe about 5 confidence intervals that fail to capture p . In Figure 9.29 for example, 4 of the confidence intervals failed to capture p .

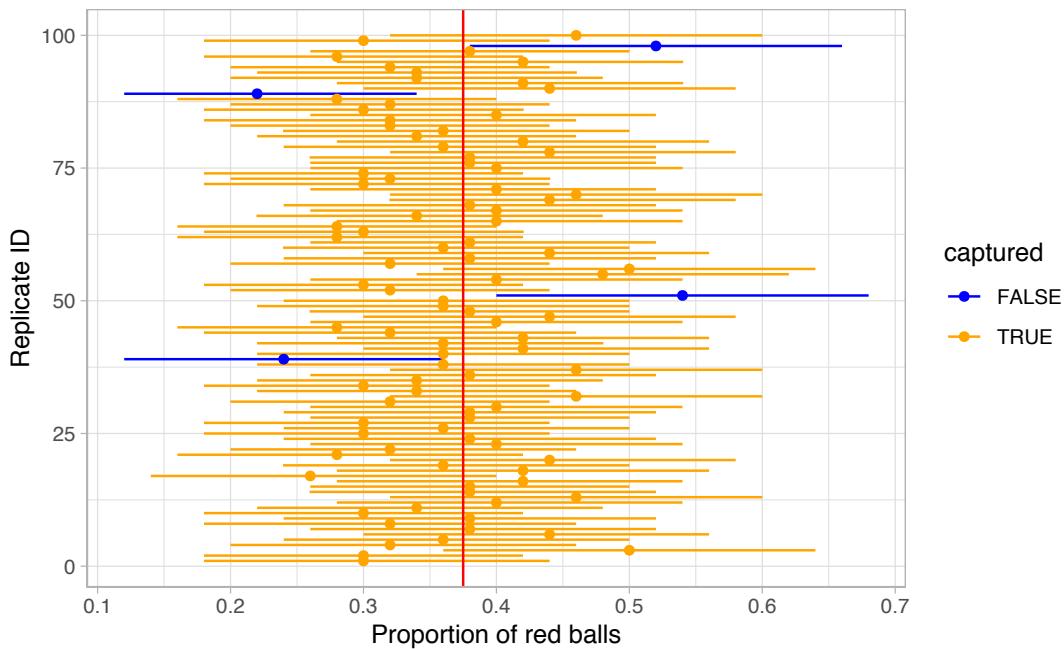


FIGURE 9.29: 100 SE-based 95 percent confidence intervals for p .

To further accentuate this point, let's perform a similar procedure using 85% standard-error based confidence intervals instead. Let's visualize the results in Figure 9.30. Observe how the widths of the 80% confidence intervals are narrower than the 95% confidence intervals; we'll explore other determinants of the widths of confidence intervals in the next section.

Of the 100 confidence intervals based on 100 samples of size $n = 50$, 86 of them captured the population proportion $p = 0.375$, whereas 14 of them did not. Note that since we lowered the confidence level from 95% to 80%, we now have a much larger number of confidence intervals that failed to “catch the fish.”

9.5.2 Precise & shorthand interpretation

Let's return our attention to our 95% confidence intervals. The precise and mathematically correct interpretation of a 95% confidence interval is a little long-winded:

Precise interpretation: If we repeated our sampling procedure a large number of times, we expect about 95% of the resulting confidence intervals to capture the value of the population parameter.

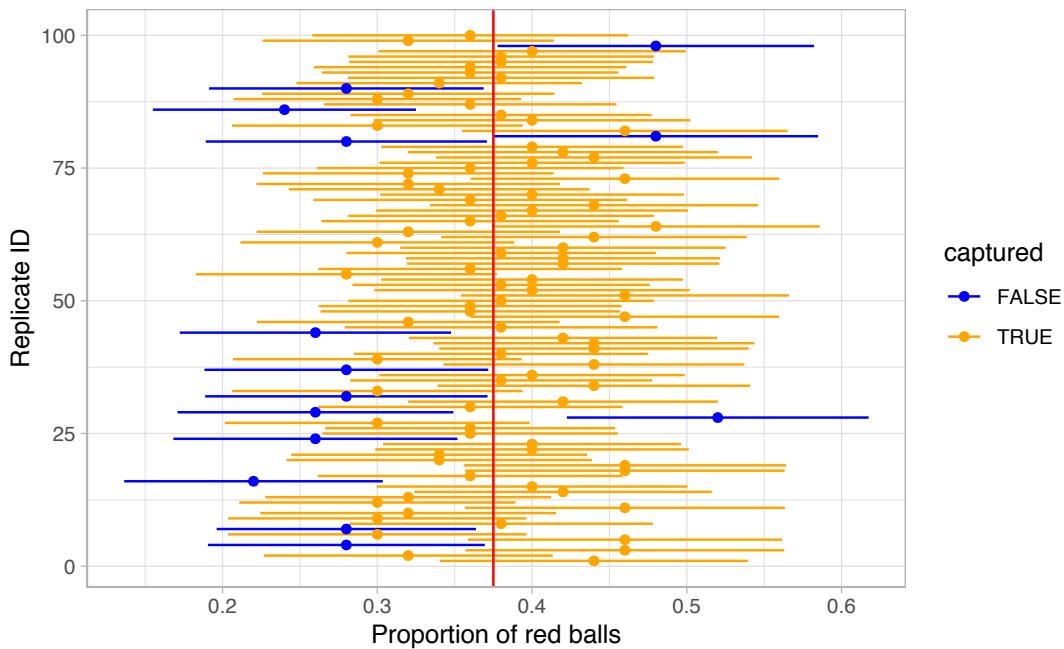


FIGURE 9.30: 100 SE-based 85 percent confidence intervals for p

This is what we observed in Figure 9.29: that our confidence interval construction procedure is 95% “reliable”. In other words, we can expect our confidence intervals to include the true population parameter 95% of the time.

A common but incorrect interpretation is: “There is a 95% probability that the confidence interval contains p .” Because looking at Figure 9.29, each of the confidence intervals either does or doesn’t contain p , in other words, the probability is either 1 or 0.

So if the 95% confidence level only relates to the reliability of the confidence interval construction procedure, what insight can be derived from one particular confidence interval? For example, going back to the pennies example, we found that the percentile method 95% confidence interval for μ was (1991.16, 1999.58) whereas the standard error method 95% confidence interval was (1991.16, 1999.72).

Loosely speaking, we can think of these intervals as our “best guess” of a plausible range of values for the mean year of minting of all US pennies. Furthermore, for the rest of this text, we’ll use the following shorthand to summarize the precise interpretation.

Short-hand interpretation: We are 95% “confident” that a 95% confidence interval captures the value of the population parameter.

We use quotation marks around “confident” to emphasize that while 95% relates to the reliability of our confidence interval construction procedure, ultimately the resulting interval is our best guess of a range of values that contain the population parameter.

So returning to our pennies example and focusing on the percentile-method, we are 95% “confident” that the true mean year of pennies in circulation in 2019 is somewhere between 1991.16 and 1999.58.

9.5.3 Width of confidence intervals

Now that we know how to interpret confidence intervals, let’s go over some factors that determine their width.

Impact of confidence level

One factor that determines the confidence interval width is the pre-specified confidence level. For example in Figures 9.29 and 9.30, we compared the widths of 95% and 80% confidence intervals. Recall that the 95% confidence intervals were wider. The quantification of the confidence level should match what you expect of the word “confident.” In order to be more confident in our best guess of a range of values, we need to widen the range of values.

To elaborate on this, imagine we want to guess the forecasted high temperature in Seoul, South Korea on August 15th. Given Seoul’s temperate climate with 4 distinct seasons, we could say somewhat confidently that the high temperature would be between 50°F - 95°F (10°C - 35°C). However, if we wanted a temperature range we were *absolutely* confident about, would we need to widen or narrow this range? We’d need to widen it! We need this wider range since it is possible to have a freak cold spell or heat wave. So a range of temperatures we could be near certain about would be between 32°F - 110°F (0°C - 43°C). On the other hand, if we wanted a range we could tolerate being a little less confident, we could narrow this range to between 70°F - 85°F (21°C - 30°C).

Going back to our sampling bowl, let’s compare confidence intervals for p

based on three different confidence levels: 80%, 95%, and 99%. Specifically, we'll first take 30 different random samples of $n = 50$ balls from the bowl. We'll then construct 10 percentile-based confidence intervals based on each of the three different confidence levels and compare the widths of these intervals. We visualize the resulting 30 confidence intervals in Figure 9.31 along with a vertical red line marking the true value of $p = 0.375 = 0.375\%$, the population proportion of the bowl's balls that are red.

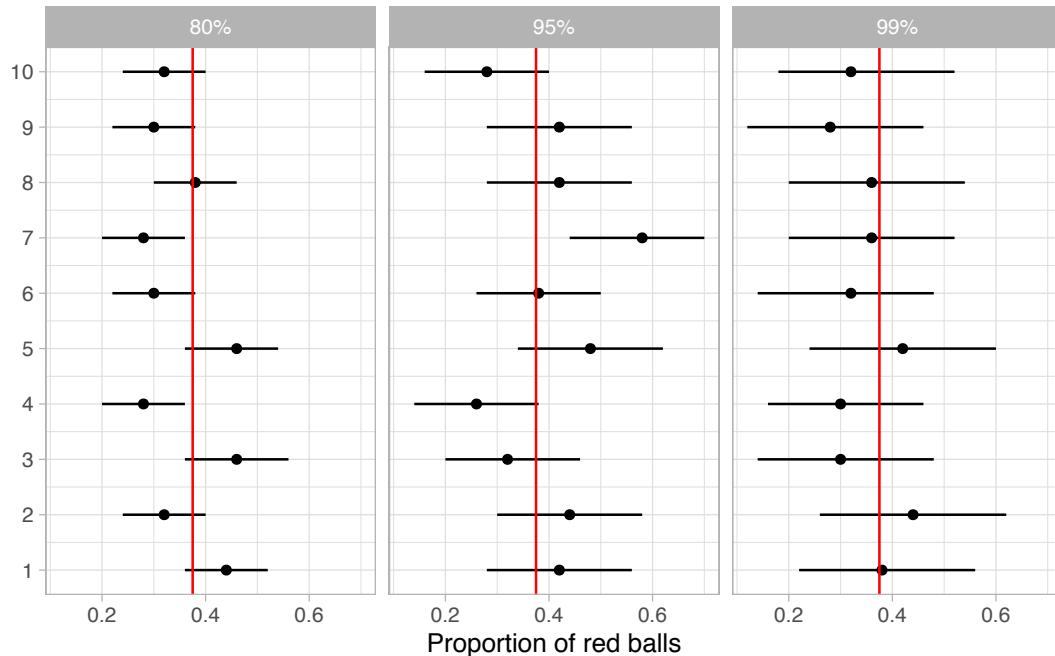


FIGURE 9.31: Ten 80, 95, and 99 percent confidence intervals for p based on $n = 50$.

Observe that as the confidence level increase from 80% to 95% to 99%, in general, the confidence intervals get wider. Let's compare the average widths in Table 9.3.

TABLE 9.3: Mean width of 80, 95, and 99 percent confidence intervals.

Confidence level	Mean width
80%	0.166
95%	0.264
99%	0.338

So in order to have a higher confidence interval, our “plausible range of values” must be wider. Ideally we would have both high confidence level and narrower

confidence intervals; however, we cannot have it both ways. If we want to “be more confident”, we need to allow for wider intervals. Conversely, if we would like a narrow and tight interval, we must sacrifice confidence level.

The moral of the story is: **Higher confidence levels tend to produce wider confidence intervals.** However, it is important to keep in mind in our example that we kept the sample size fixed at $n = 50$. In other words, all 30 random samples from `bowl` used to construct the 30 confidence intervals had the same sample size. What happens if, instead, we take samples of different sizes? Recall that we did this in Section 8.2.4 where did this using virtual shovels with 25, 50, and 100 slots. We delve into this next.

Impact of sample size

This time, let’s fix the confidence level at 95%, but consider three different sample sizes: n equals 25, 50, and 100. Specifically, we’ll first take 10 different random samples of size 25, 10 different random samples of size 50, and 10 different random samples of size 100. We’ll then construct 95% percentile-based confidence intervals. We visualize the resulting 30 confidence intervals in Figure 9.32 along with a vertical red line marking the true value of $p = 0.375 = 0.375\%$, the population proportion of the bowl’s balls that are red.

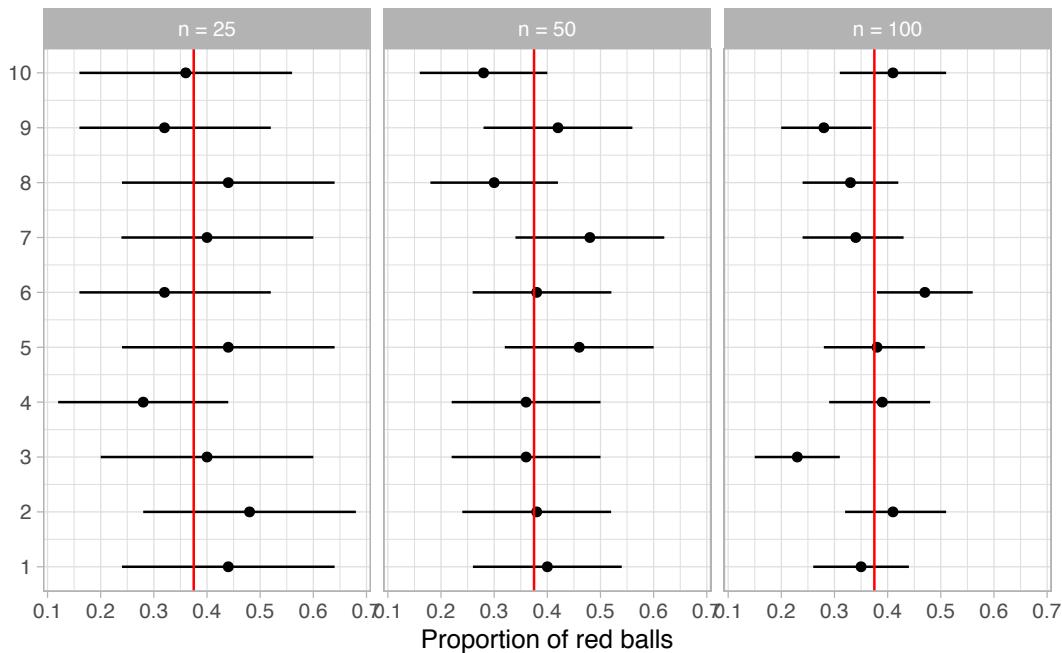


FIGURE 9.32: Ten 95 percent confidence intervals for p based on $n = 25$, 50, and 100.

Observe that as our confidence intervals are based on larger and larger sample sizes, in general, the confidence intervals get narrower. Let's compare the average widths in Table 9.4.

TABLE 9.4: Mean width of 95 percent confidence intervals based on $n = 25$, 50, and 100.

Sample size	Mean width
$n = 25$	0.380
$n = 50$	0.270
$n = 100$	0.183

The moral of the story is: **Larger sample sizes tend to produce narrow confidence intervals.** Recall that this was a key message in Section 8.3.3. As we used the larger and larger shovels to draw our samples, our sample proportions red \hat{p} tended to vary less. In other words, our estimates got more and more precise.

We visualized these results in Figure 8.15 where we compared the *sampling distributions* for \hat{p} based on samples of size n equal 25, 50, 100. Furthermore, we could quantify the spread of these sampling distributions using their standard deviation, which as a special name: the *standard error*. As the sample size increases, the standard error decreases.

In fact, the standard error is a related factor in confidence interval width, which we explore in Subsection 9.7.2 when we discuss theoretically constructed confidence intervals using mathematical formulas.

9.6 Case study: Is yawning contagious?

Let's apply our knowledge of confidence intervals to answer the question: "Is yawning contagious?" If you see someone else yawn, are you more likely to yawn? In an episode⁴ of the US show *Mythbusters*, the hosts conducted an experiment to answer this question. The episode is available to view in the United States on the Discovery Network website here⁵ and more information about the episode is also available on IMDb⁶.

⁴<http://www.discovery.com/tv-shows/mythbusters/mythbusters-database/yawning-contagious/>

⁵<https://www.discovery.com/tv-shows/mythbusters/videos/is-yawning-contagious>

⁶<https://www.imdb.com/title/tt0768479/>

9.6.1 Mythbusters study data

Fifty adult participants who thought they were being considered for an appearance on the show were interviewed by a show recruiter who either yawned or did not. Participants then sat by themselves in a large van and were asked to wait. While in the van, the Mythbusters watched via hidden camera to see if the participants yawned. The data frame containing the results is available in the `mythbusters_yawn` data frame included in the `moderndive` package:

```
mythbusters_yawn
```

```
# A tibble: 50 × 3
  subj   group    yawn
  <int> <chr>    <chr>
1     1 seed     yes
2     2 control yes
3     3 seed     no
4     4 seed     yes
5     5 seed     no
6     6 control no
7     7 seed     yes
8     8 control no
9     9 control no
10    10 seed    no
# ... with 40 more rows
```

The variables are:

- `subj`: The participant ID with values 1 through 50.
- `group`: A binary categorical variable of whether the participant was exposed to yawning, where "seed" indicates the participant was exposed to yawning and "control" indicates the participant was not.
- `yawn`: A "yes" vs "no" binary categorical variable indicating whether the participant responded by yawning.

Let's use some data wrangling to obtain counts of the four possible outcomes:

```
mythbusters_yawn %>%
  group_by(group, yawn) %>%
  summarize(count = n())

# A tibble: 4 × 3
# Groups:   group [2]
```

```

group    yawn   count
<chr>    <chr> <int>
1 control no      12
2 control yes     4
3 seed    no      24
4 seed    yes     10

```

So 12 participants who were not exposed to yawning did not yawn, while 4 participants who were not exposed to yawning did yawn. So out of the 16 people who were not exposed to yawning, $4/16 = 0.25 = 25\%$ did yawn. On the other hand, 24 participants who were exposed to yawning did not yawn, while 10 participants who were exposed to yawning did yawn. So out of the 34 people who were exposed to yawning, $10/34 = 0.294 = 29.4\%$ did yawn.

Putting these two values together, the participants who were exposed to yawning yawned $29.4\% - 25\% = 4.4\%$ more often than those who were not exposed to yawning.

9.6.2 Sampling scenario

In Chapter 8 our study population was the bowl of $N = 2400$ balls. Our population parameter of interest was the population proportion of these balls that were red, denoted mathematically by p . In order to estimate p , we extracted a sample of 50 balls using the shovel and computed the relevant point estimate: the sample proportion of these 50 balls that were red, denoted mathematically by \hat{p} .

Who is the study population here? All humans? All the people who watch the show Mythbusters? It's hard to say! This question can only be answered if we know how the show's hosts recruited participants! We alas don't have this information. Only for the purposes of this case study, however, we'll assume that the 50 participants are a representative sample of all Americans, and thus any results of this experiment will generalize to all $N = 327$ million Americans (2018 population).

Just like with our sampling bowl, the population parameter of interest will involve proportions, but this time it will be the difference in population proportions $p_{seed} - p_{control}$, where p_{seed} is the population proportion of people exposed to yawning who yawn and $p_{control}$ is the population proportion of people not exposed to yawning who yawn. Correspondingly, the point estimate/sample statistic based on sampled data will be the difference in sample proportions $\hat{p}_{seed} - \hat{p}_{control}$. Let's extend Table 8.8 of scenarios of sampling for inference to include our latest scenario.

TABLE 9.5: Scenarios of sampling for inference

Scenario	Population parameter	Notation	Point estimate	Notation.
1	Population proportion	p	Sample proportion	\hat{p}
2	Population mean	μ	Sample mean	\bar{x} or $\hat{\mu}$
3	Difference in population proportions	$p_1 - p_2$	Difference in sample proportions	$\hat{p}_1 - \hat{p}_2$

This is known as a situation of *two-sample* inference since we have two separate samples, in this case, those who were exposed to yawning and those who were not. So in our case, based on two separate samples of size $n_{seed} = 34$ and $n_{control} = 16$,

$$\hat{p}_{seed} - \hat{p}_{control} = \frac{24}{34} - \frac{12}{16} = 0.04411765 \approx 4.4\%$$

However, say we had sampled 50 different people, 34 to be exposed to yawning and 16 not, and repeated this experiment. Would we obtain the exact same estimated difference of 4.4%? Probably not, because of sampling variation. How does this sampling variation affect our estimate of 4.4%? In other words, what would be a plausible range of values for this difference that accounts for this sampling variation? We can answer this question with confidence intervals! Furthermore, since we only have one single sample of 50 participants, we can construct the 95% confidence interval for $p_{seed} - p_{control}$ using bootstrap resampling with replacement.

9.6.3 Constructing the confidence interval

As we did in Section 9.4.2, let's spell out the steps of the `infer` workflow to construct the 95% confidence interval for $p_{seed} - p_{control}$. However, since the difference in proportions is a new scenario for inference, we'll need to identify some new arguments to include in the `infer` functions along the way.

1. `specify` variables

We take our `mythbusters_yawn` data frame with the data and `specify()` which variables are of interest using the formula interface where

- Our response variable is `yawn`: whether or not a participant yawned.
- The explanatory variable is `group`: whether or not a participant was exposed to yawning.

```
mythbusters_yawn %>%
  specify(formula = yawn ~ group)
```

Error: A level of the response variable `yawn` needs to be specified for the `success` argument in `specify()`.

Alas, we got an error message. `infer` is telling us that one of the levels of the categorical variable `yawn` needs to be defined as the `success`, or the event of interest we are trying to count and compute proportions. Are we interested in those participants who "yes" yawned or are we interested in those participants who "no" didn't yawn? This isn't clear. So we set the `success` argument to "yes" as follows:

```
mythbusters_yawn %>%
  specify(formula = yawn ~ group, success = "yes")
```

```
Response: yawn (factor)
Explanatory: group (factor)
# A tibble: 50 x 2
  yawn   group
  <fct> <fct>
1 yes    seed
2 yes    control
3 no     seed
4 yes    seed
5 no     seed
6 no     control
7 yes    seed
8 no     control
9 no     control
10 no    seed
# ... with 40 more rows
```

2. generate replicates

Our next step in building a confidence interval is to create a bootstrap distribution of statistics (differences in proportions of successes). We saw how it works with both a single variable in computing bootstrap means in Subsection

9.4 and in computing bootstrap proportions in Section 9.5, but we haven't yet worked with bootstrapping involving multiple variables though.

In the `infer` package, bootstrapping with multiple variables means that each `row` is potentially resampled. Let's investigate this by looking at the first few rows of `mythbusters_yawn`:

```
head(mythbusters_yawn)
```

```
# A tibble: 6 x 3
  subj group  yawn
  <int> <chr>  <chr>
1     1 seed    yes
2     2 control yes
3     3 seed    no 
4     4 seed    yes
5     5 seed    no 
6     6 control no
```

When we bootstrap this data, we are potentially pulling the subject's readings multiple times. Thus, we could see the entries of "seed" for `group` and "no" for `yawn` together in a new row in a bootstrap sample. This is further seen by exploring the `sample_n()` function in `dplyr` on this smaller 6-row data frame comprised of `head(mythbusters_yawn)`. The `sample_n()` function can perform this bootstrapping procedure and is similar to the `rep_sample_n()` function in `infer`, except that it is not repeated but rather only performs one sample with or without replacement.

```
head(mythbusters_yawn) %>%
  sample_n(size = 6, replace = TRUE)
```

```
# A tibble: 6 x 3
  subj group  yawn
  <int> <chr>  <chr>
1     1 seed    yes
2     6 control no 
3     1 seed    yes
4     5 seed    no 
5     4 seed    yes
6     4 seed    yes
```

We can see that in this bootstrap sample generated from the first six rows of `mythbusters_yawn`, we have some rows repeated. The same is true when we

perform the `generate()` step in `infer` as done below. Next, we generate 1000 replicates, or in other words, we bootstrap resample the 50 participants with replacement 1000 times. This is what will inject sampling variation into our results.

```
mythbusters_yawn %>%
  specify(formula = yawn ~ group, success = "yes") %>%
  generate(reps = 1000, type = "bootstrap")
```

```
Response: yawn (factor)
Explanatory: group (factor)
# A tibble: 50,000 x 3
# Groups:   replicate [1,000]
  replicate yawn   group
  <int> <fct> <fct>
1       1 no     seed
2       1 no     seed
3       1 yes    control
4       1 yes    seed
5       1 no     control
6       1 yes    seed
7       1 no     control
8       1 no     seed
9       1 no     seed
10      1 no    seed
# ... with 49,990 more rows
```

Note the resulting data frame has 50,000 rows. This is because we performed resampling of 50 participants with replacement 1000 times and thus $50,000 = 1000 \times 50$. Accordingly, the variable `replicate`, indicating which resample each row belongs to, has the value 1 50 times, the value 2 50 times, all the way through to the value 1000 50 times.

3. calculate summary statistics

After we `generate()` many replicates of bootstrap resampling with replacement, we next want to summarize of the bootstrap resamples of size 50 with a single summary statistic, the difference in proportions. I do this by setting the `stat` argument to "diff in props":

```
mythbusters_yawn %>%
  specify(formula = yawn ~ group, success = "yes") %>%
  generate(reps = 1000, type = "bootstrap") %>%
  calculate(stat = "diff in props")
```

Error: Statistic is based on a difference; specify the `order` in which to subtract the levels of the explanatory variable.

We see another error here. We need to specify the order of the subtraction. Is it $\hat{p}_{seed} - \hat{p}_{control}$ or $\hat{p}_{control} - \hat{p}_{seed}$. We specify it to be $\hat{p}_{seed} - \hat{p}_{control}$ by setting `order = c("seed", "control")`. You can also set `order = c("control", "seed")`; it makes no difference in the analysis. However, whatever order you choose, it is important to stay consistent throughout your analysis.

Let's save the output in a data frame `bootstrap_distribution_yawning`:

```
bootstrap_distribution_yawning <- mythbusters_yawn %>%
  specify(formula = yawn ~ group, success = "yes") %>%
  generate(reps = 1000, type = "bootstrap") %>%
  calculate(stat = "diff in props", order = c("seed", "control"))
bootstrap_distribution_yawning
```

```
# A tibble: 1,000 x 2
  replicate      stat
  <int>     <dbl>
1       1 -0.0213904
2       2  0.0459770
3       3  0
4       4 -0.0129870
5       5  0.326765
6       6  0.122807
7       7  0.293718
8       8  0.0761905
9       9  0.0679117
10      10 -0.0231729
# ... with 990 more rows
```

We see that the resulting data frame has 1000 rows and 2 columns corresponding to the 1000 replicates and the difference in proportions for each bootstrap resample saved in the variable `stat`.

4. visualize the results

In Figure 9.33 we `visualize()` the resulting bootstrap resampling distribution.

```
visualize(bootstrap_distribution_yawning)
```

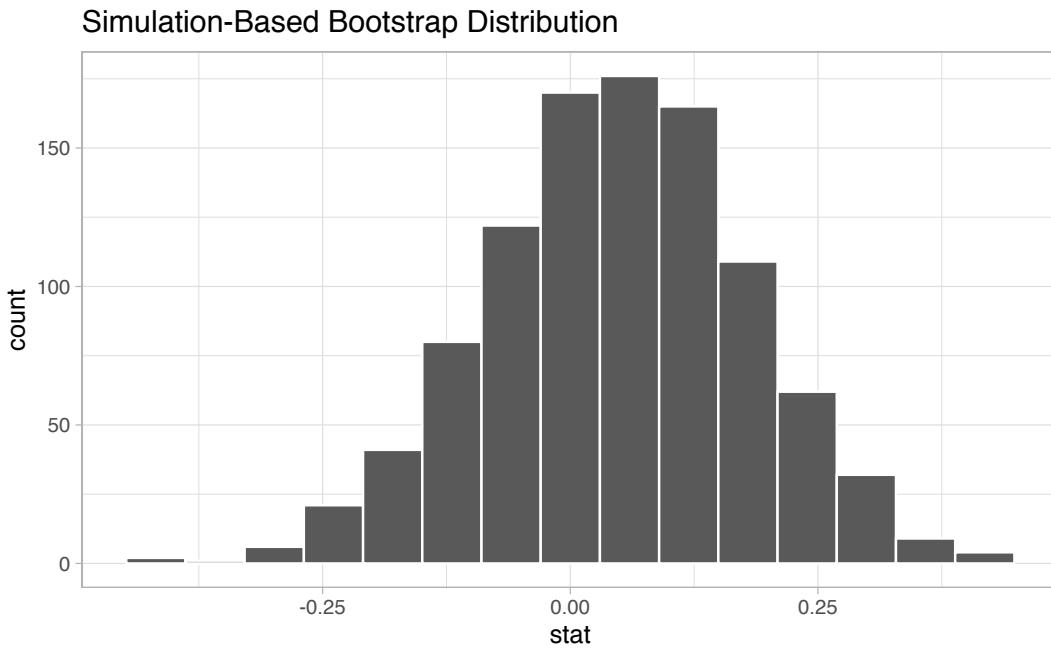


FIGURE 9.33: Bootstrap distribution.

First, let's compute the 95% confidence interval for $p_{seed} - p_{control}$ using the percentile method, in other words by identifying the 2.5th and 97.5th percentiles which include the middle 95% of values. Recall that this method does not require the bootstrap distribution to be normally shaped.

```
bootstrap_distribution_yawning %>%
  get_confidence_interval(type = "percentile", level = 0.95)
```

```
# A tibble: 1 × 2
`2.5%` `97.5%`
<dbl>    <dbl>
1 -0.218313 0.304763
```

Second, since the bootstrap distribution is roughly bell-shaped, it is reasonable to assume that it is normally shaped, and thus we can construct a confidence interval using the standard error method. Recall to construct a standard error method, we need to specify the center of the interval using the `point_estimate`

argument. We set it to be the observed difference in sample proportions of 4.4% we computed earlier.

However, we can also use the `infer` workflow to let R compute this value by excluding the `generate()` 1000 bootstrap replicates step. In other words, do not compute the difference in proportions for 1000 bootstrap samples with replacement, rather focus only on the observed sample data. We can achieve this by commenting out the `generate()` line, telling R to ignore it:

```
mythbusters_yawn %>%
  specify(formula = yawn ~ group, success = "yes") %>%
  # generate(reps = 1000, type = "bootstrap") %>%
  calculate(stat = "diff in props", order = c("seed", "control"))
```

```
# A tibble: 1 × 1
  stat
  <dbl>
1 0.0441176
```

We thus plug this value as the `point_estimate` argument.

```
bootstrap_distribution_yawning %>%
  get_confidence_interval(type = "se", point_estimate = 0.0441176)
```

```
# A tibble: 1 × 2
  lower     upper
  <dbl>     <dbl>
1 -0.213435 0.301670
```

Let's visualize both confidence intervals in Figure 9.34, with the percentile method interval in red and the standard error method interval in blue. Observe that they are both similar.

9.6.4 Interpreting the confidence interval

Given that both confidence intervals are quite similar, let's focus our interpretation to only the percentile method confidence interval of (-0.22, 0.3). Recall that the correct statistical interpretation of a 95% confidence interval is: if repeated this procedure 100 times, then we'd expect 95 of the confidence intervals to capture the true population difference in proportions $p_{seed} - p_{control}$. In other words, if we gathered 100 samples of $n = 50$ participants from a similar pool of people and constructed 100 confidence intervals, about 95 of them

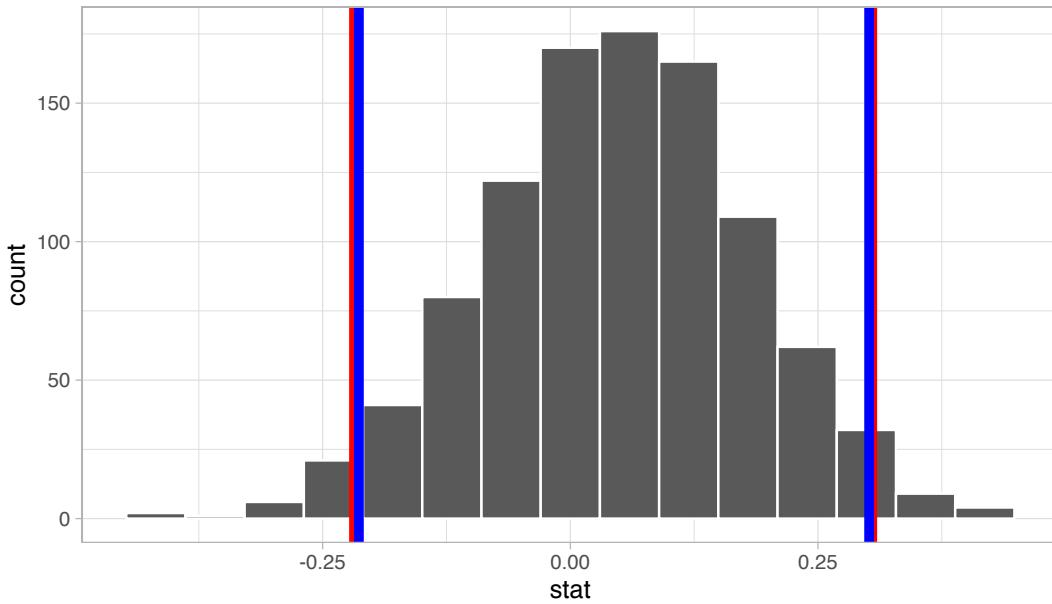


FIGURE 9.34: Two 95 percent confidence intervals: percentile method in red, standard error method in blue.

will contain the true value of $p_{seed} - p_{control}$ while about 5 won't. Given that this is a little long winded, we use the shorthand: we're 95% "confident" that the true difference in proportions $p_{seed} - p_{control}$ is between (-0.22, 0.3).

There is one value of particular interest that this interval contains: zero. If $p_{seed} - p_{control} = 0$, then there would be no difference in proportion yawning, suggesting that there is no associated effect of being exposed to yawning. Since the 95% confidence interval includes 0, we cannot conclusively say if either proportion is larger. Of our 1000 bootstrap resamples with replacement, sometimes \hat{p}_{seed} was higher and thus those exposed to yawning yawned themselves more often, and other times the reverse. Say on the other hand the 95% confidence interval was entirely above zero. This would be suggestive that $p_{seed} - p_{control} > 0$ and thus those exposed to yawning yawned more often.

Furthermore, if the 50 participants were randomly allocated to the "seed" and "control" groups, then this would be suggestive that being exposed to yawning doesn't not *cause* yawning. In other words, yawning is not contagious. However, no information on how participants were assigned to be exposed to yawning or not could be found, so we cannot make such a causal statement.

9.7 Conclusion

9.7.1 Comparing bootstrap and sampling distributions

In Section 8.2.3, we took 1000 virtual samples from the `bowl` using a virtual shovel, then computed 1000 values of the sample proportion red \hat{p} , and visualized their distribution in a histogram. Recall that this distribution is called the *sampling distribution of \hat{p}* and furthermore the standard deviation of this sampling distribution has a special name: the *standard error*.

However, this exercise was not what one would typically do in real-life, rather it was a simulation to study the effects of sampling variation on the value of \hat{p} . In real-life, one would take only one sample that's as large as possible. But how can we get a sense of the effect of sample-to-sample variation if we only have one sample? Don't we need many samples?

The solution to this was to perform bootstrap resampling with replacement from the original sample. We did this in the resampling activity in Section 9.1 where in this case, we were focused on the mean year of minting of pennies rather than the proportion of the bowl's balls that were red. We used pieces of paper representing the original sample of 50 pennies from the bank and resampled them with replacement from the hat. We had 35 of our friends perform this activity and visualized their resulting sample means \bar{x} .

This distribution was called the *bootstrap distribution* of \bar{x} and it is an *approximation* to the sampling distribution of \bar{x} . Thus the variation of the bootstrap distribution of \bar{x} could be used as an approximation to the variation of the sampling distribution of \bar{x} .

Let's now compare the sampling distribution of \hat{p} with the bootstrap distribution of \hat{p} and see how well they line up. Recall that we computed the sampling distribution of \hat{p} in Section 8.2.3 based on 1000 virtual samples of size $n = 50$ from the `bowl`. We'll compare this to the bootstrap distribution of \hat{p} from Section 9.5.1 based on 1000 resamples with replacement from Ilyas and Yohan's sample of 50 balls saved in `bowl_sample_1`.

Sampling distribution

Here is the code you previously saw in Section 8.2.3 to construct the sampling distribution of \hat{p} , with some small changes to incorporate the statistical terminology relating to sampling you learned in Section 8.3.1.

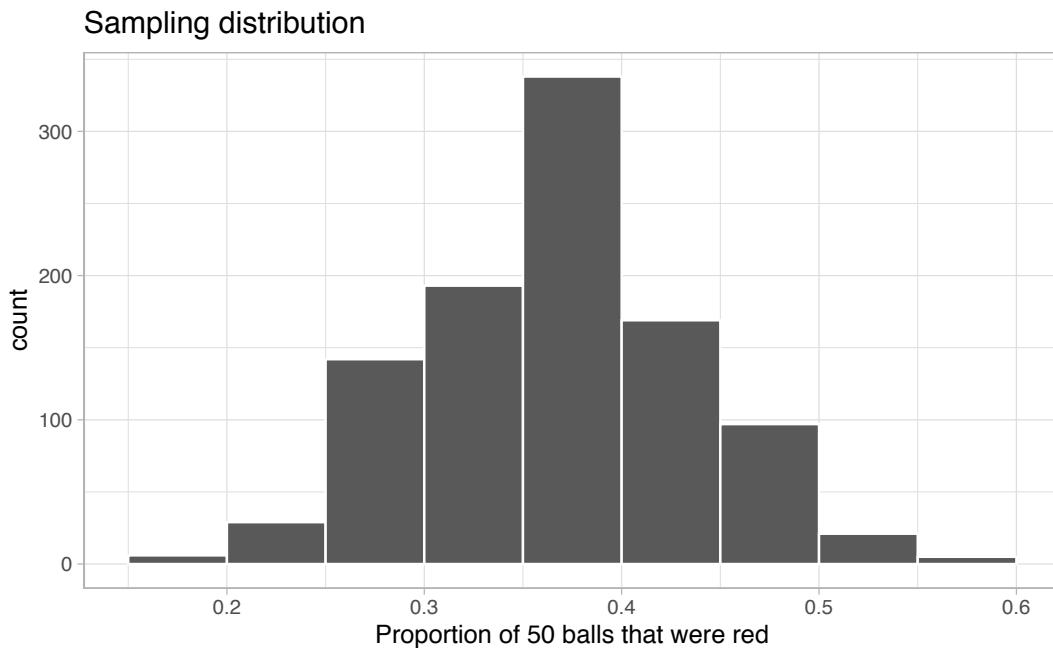


FIGURE 9.35: Previously seen sampling distribution of sample proportion red for $n = 1000$.

An important thing to keep in mind is the default value for `replace` is `FALSE` when using `rep_sample_n()`. This is because when sampling 50 balls with a shovel, we are extracting 50 balls one-by-one *without* replacing them. This is in contrast to bootstrap resampling *with* replacement, where we resample a ball and put it back, and repeat this process 50 times.

We can also examine the variability in this sampling distribution by calculating the standard deviation of the `prop_red` variable representing 1000 values of the sample proportion \hat{p} . Remember that the standard deviation of the sampling distribution is the **standard error**, frequently denoted as `se`.

```
sampling_distribution %>%
  summarize(se = sd(prop_red))
```

```
# A tibble: 1 × 1
  se
  <dbl>
1 0.0673987
```

Bootstrap distribution

Here is the code you previously saw in Section 9.5.1 to construct the bootstrap distribution of \hat{p} based on Ilyas and Yohan's original sample of 50 balls saved in `bowl_sample_1`.

```
# Compute the bootstrap distribution using infer workflow:  
bootstrap_distribution <- bowl_sample_1 %>%  
  specify(response = color, success = "red") %>%  
  generate(reps = 1000, type = "bootstrap") %>%  
  calculate(stat = "prop")
```

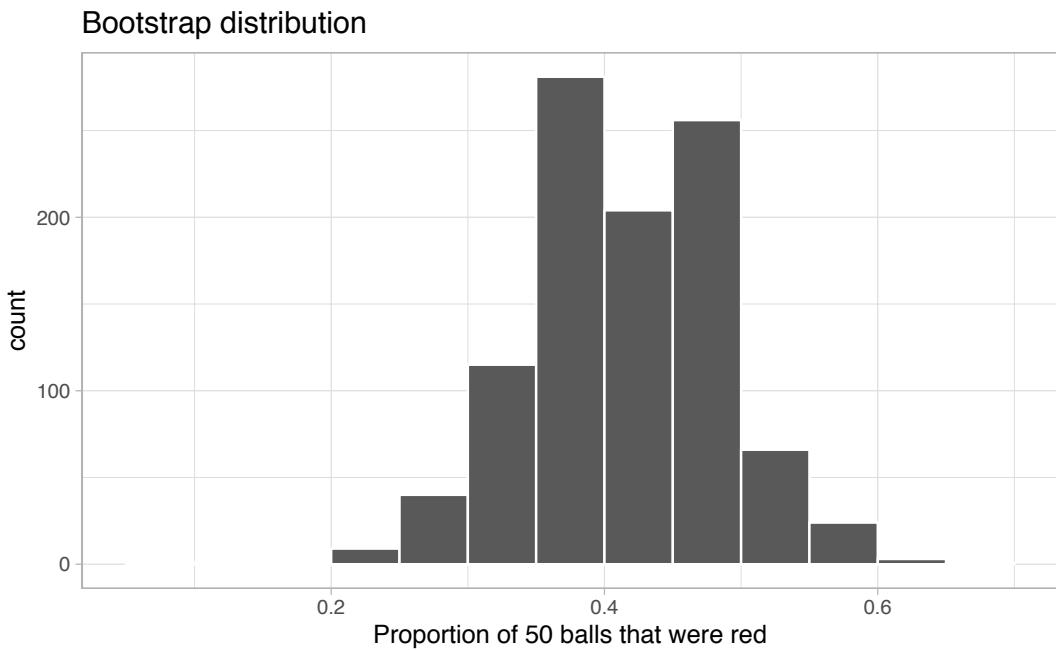


FIGURE 9.36: Bootstrap distribution of sample proportion red for $n = 1000$.

```
bootstrap_distribution %>%  
  summarize(se = sd(stat))
```

```
# A tibble: 1 × 1  
      se  
    <dbl>  
1 0.0693340
```

Comparison

Now that we have computed both the sampling distribution and the bootstrap distributions, let's visualize them side-by-side in Figure 9.37. We'll make both histograms have the same scale on the x and y-axes to make them more comparable and add vertical red lines that denote the proportion of the bowl's balls that are red $p = 0.375$. Furthermore, we'll add a vertical dashed line at Ilyas and Yohan's value of the sample proportion $\hat{p} = 21/50 = 0.42 = 42\%$.

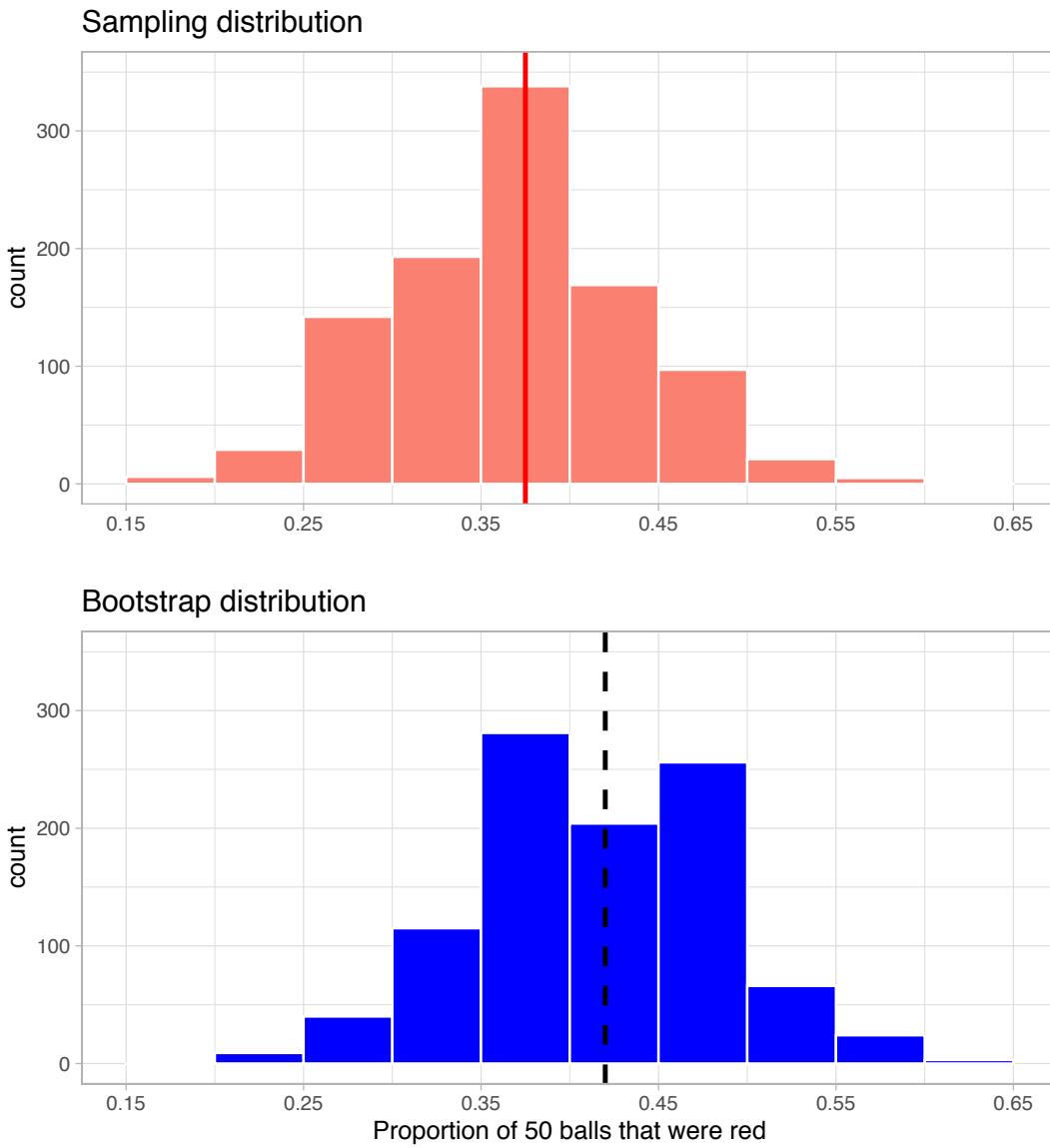


FIGURE 9.37: Comparing the sampling and bootstrap distributions of \hat{p}

There is a lot going on in this comparison, so let's pick them apart slowly.

First, notice how the sampling distribution in salmon is centered at $p = 0.375$. This is because the sample is done at random in an unbiased fashion, so the estimates \hat{p} are centered at the true value of p . This is actually an artifact of the Central Limit Theorem introduced in Chapter 8. The center of the sampling distribution is expected to be the value of the population parameter, in this case p .

However, this is not the case with the bootstrap distribution in blue. The bootstrap distribution is centered at 0.42, which is the proportion red of Ilyas and Yohan's 50 sampled balls. This is because we are resampling from the sample and not from the population. Since the bootstrap distribution is centered at the original sample proportion, it doesn't necessarily provide a good estimate of the overall population proportion p , which we calculated to be 0.375. This leads us to our first lesson about bootstrapping:

The bootstrap distribution will likely not have the same center as the sampling distribution. In other words, bootstrapping cannot improve the quality of an estimate resulting from a single sample.

However, let's now visually compare the spread of the two distributions; they are somewhat similar. In fact, we computed the standard deviations of both distributions earlier. Let's compare them in Table 9.6

TABLE 9.6: Comparing standard errors

Distribution type	Standard error
Sampling distribution	0.067
Bootstrap distribution	0.069

Notice that the bootstrap distribution's standard error is a good approximation to the sampling distribution's standard error. This leads us to our second lesson about bootstrapping:

Even if the bootstrap distribution might not have the same center as the sampling distribution, it will likely have a very similar spread. In other words, bootstrapping will give you a good estimate of the standard error using only a single sample.

Using the fact that the bootstrap distribution and sampling distributions have similar spreads, we can build confidence intervals using bootstrapping as we've done all throughout this chapter!

9.7.2 Theory-based confidence intervals

So far in this chapter, we've constructed confidence intervals using two methods: the percentile method and the standard error method. Recall from Section 9.3.2 however that we can only use the standard-error method if the bootstrap distribution is bell-shaped i.e. normally distributed. If this is the case, however, there is another method for constructing confidence intervals that do not involve using your computer to do resampling with replacement. There actually is a theory-based method involving a mathematical formula!

The formula uses the rule of thumb we saw in Subsection 8.5.3 that 95% of values in a normal distribution are within ± 1.96 standard deviations of the mean. In the case of sampling and bootstrap distributions, recall that the standard deviation has a special name: the *standard error*. Furthermore, there is in many cases a formula that approximates the standard error! In the case of our `bowl` where we used the sample proportion red \hat{p} to estimate the proportion of the bowl's balls that are red, the formula that approximates the standard error is:

$$\text{SE}_{\hat{p}} \approx \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$

For example, recall from Table 8.1 that Yohan and Ilyas sampled $n = 50$ balls and observed a sample proportion \hat{p} of $21/50 = 0.42 = 42\%$. So using the approximation of the standard error is

$$\text{SE}_{\hat{p}} \approx \sqrt{\frac{0.42(1 - 0.42)}{50}} = \sqrt{0.004872} = 0.0698$$

The key observation to make here is that there is an n in the denominator. In other words, as the sample size n increases, the standard error decreases. We've demonstrated this fact this using virtual simulation in Section 8.3.3. If you don't recall this demonstration, we highly recommend you go back and read that section.

So going back to Yohan and Ilyas' sample proportion of \hat{p} of $21/50 = 0.42 =$

42%, say this were based on a sample of size $n=100$ instead of 50. Then the standard error would be:

$$\text{SE}_{\hat{p}} \approx \sqrt{\frac{0.42(1 - 0.42)}{100}} = \sqrt{0.002436} = 0.0494$$

Observe that the standard error has gone down from 0.0698 to 0.0494. In other words, this particular estimate is more *precise*. Recall that we illustrated the difference between accuracy and precision of estimates in Figure 8.16.

Why is this formula true? Unfortunately, we don't have the tools at this point to prove this; you'll need to take a more advanced course in probability and statistics.

Theory-based method for constructing confidence intervals

We now present a third method for constructing a 95% confidence interval for p that does not involve bootstrap resampling with replacement, but rather mathematical formulas. It is critical to remember that this method holds only if the sampling distribution is normally shaped.

1. Collect a single representative sample of size n that's as large as possible.
2. Compute the point estimate: the sample proportion \hat{p} . Think of this as the center of your net.
3. Compute the approximation to the standard error

$$\text{SE}_{\hat{p}} \approx \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$

1. Compute a quantity known as the *margin of error* (more later):

$$\text{MoE}_{\hat{p}} = 1.96 \cdot \text{SE}_{\hat{p}} = 1.96 \cdot \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$

1. Compute both endpoints of the confidence interval.
 - The lower end-point `lower_ci`. Think of this as the left end-point of the net:
$$\hat{p} - \text{MoE}_{\hat{p}} = \hat{p} - 1.96 \cdot \text{SE}_{\hat{p}} = \hat{p} - 1.96 \cdot \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$
 - The upper endpoint `upper_ci`. Think of this as the right end-point

of the net:

$$\hat{p} + \text{MoE}_{\hat{p}} = \hat{p} + 1.96 \cdot \text{SE}_{\hat{p}} = \hat{p} + 1.96 \cdot \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$

- Alternatively, you can succinctly summarize a 95% confidence interval for p using the \pm symbol:

$$\hat{p} \pm \text{MoE}_{\hat{p}} = \hat{p} \pm 1.96 \cdot \text{SE}_{\hat{p}} = \hat{p} \pm 1.96 \cdot \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$

So going back to Yohan and Ilyas' sample of $n = 50$ balls that had 21 red balls, the 95% confidence interval for p is $0.42 \pm 1.96 \cdot 0.0698 = 0.42 \pm 0.137 = (0.42 - 0.137, 0.42 + 0.137) = (0.283, 0.557)$. In other words, Yohan and Ilyas are 95% “confident” that the true proportion red of the bowl's balls is between 28.3% and 55.7%. Given that the true population proportion p was 37.5%, they successfully captured the fish.

In Step 4 above, we defined the *margin of error*. You can think of this quantity as how much the net extends to the left and to the right of the center of our net. The 1.96 multiplier roots in the 95% rule of thumb we introduced earlier and the fact that we want the confidence level to be 95%. The value of the margin error entirely determines the width of the confidence interval. Recall from Section 9.5.3 that confidence interval widths are determined by an interplay of the confidence level, the sample size n , and the standard error.

Let's study a real-life example by revisiting the poll of President Obama's approval rating among young Americans aged 18-29 in the 2013 National Public Radio article Poll: Support For Obama Among Young Americans Eroding⁷. Pollsters found that based on a representative sample of $n = 2089$ young Americans, $\hat{p} = 0.41 = 41\%$ supported President Obama.

If you look towards the end of the article, it states: “The poll's margin of error was plus or minus 2.1 percentage points.” This is precisely the MoE from above:

$$\begin{aligned} \text{MoE} &= 1.96 \cdot \text{SE} = 1.96 \cdot \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}} = 1.96 \cdot \sqrt{\frac{0.41(1 - 0.41)}{2089}} \\ &= 1.96 \cdot 0.0108 = 0.021 \end{aligned}$$

Thus their poll results are based on a confidence level of 95% and the resulting 95% confidence interval for the proportion of all young Americans who support Obama is: $\hat{p} \pm \text{MoE} = 0.42 \pm 0.021 = (0.339, 0.441) = (33.9\%, 44.1\%)$

⁷<https://www.npr.org/sections/itsallpolitics/2013/12/04/248793753/poll-support-for-obama-among-young-americans-eroding>

Confidence intervals based on 33 tactile samples

Let's revisit our 33 friend's samples from the `bowl` from Section 8.1.3. Recall this data was saved in the `tactile_prop_red` data frame included in the `modern-dive` package. We'll then apply the theory-based procedure for constructing confidence intervals for p using some of the `dplyr` data wrangling tools seen in Chapter 4:

1. Rename `prop_red` to `p_hat`, the statistical name of the sample proportion \hat{p} .
2. Make explicit the sample size `n` of $n = 50$
3. Compute the:
 - Standard error `SE` for \hat{p} using the formula above.
 - Margin of error `MoE` by multiplying the `SE` by 1.96
 - Left endpoint of the confidence interval `lower_ci`
 - Right endpoint of the confidence interval `upper_ci`

```
conf_ints <- tactile_prop_red %>%
  rename(p_hat = prop_red) %>%
  mutate(
    n = 50,
    SE = sqrt(p_hat * (1 - p_hat) / n),
    MoE = 1.96 * SE,
    lower_ci = p_hat - MoE,
    upper_ci = p_hat + MoE
  )
conf_ints
```

```
# A tibble: 33 x 9
  group replicate red_balls p_hat     n       SE       MoE
  <chr>     <int>    <int> <dbl> <dbl>     <dbl>     <dbl>
1 Ilya~       1        21  0.42    50  0.0697997  0.136807
2 Morg~       2        17  0.34    50  0.0669925  0.131305
3 Mart~       3        21  0.42    50  0.0697997  0.136807
4 Clar~       4        21  0.42    50  0.0697997  0.136807
5 Ridd~       5        18  0.36    50  0.0678823  0.133049
6 Andr~       6        19  0.38    50  0.0686440  0.134542
7 Julia       7        19  0.38    50  0.0686440  0.134542
8 Rach~       8        11  0.22    50  0.0585833  0.114823
9 Dani~       9        15  0.3     50  0.0648074  0.127023
10 Josh~      10       17  0.34    50  0.0669925  0.131305
```

```
# ... with 23 more rows, and 2 more variables:
#   lower_ci <dbl>, upper_ci <dbl>
```

Let's plot now plot the 33 confidence intervals for p saved in `conf_ints` along with a vertical red line at $p = 0.375 = 37.5\%$ indicating the true proportion of the bowl's balls that are red in Figure 9.38.

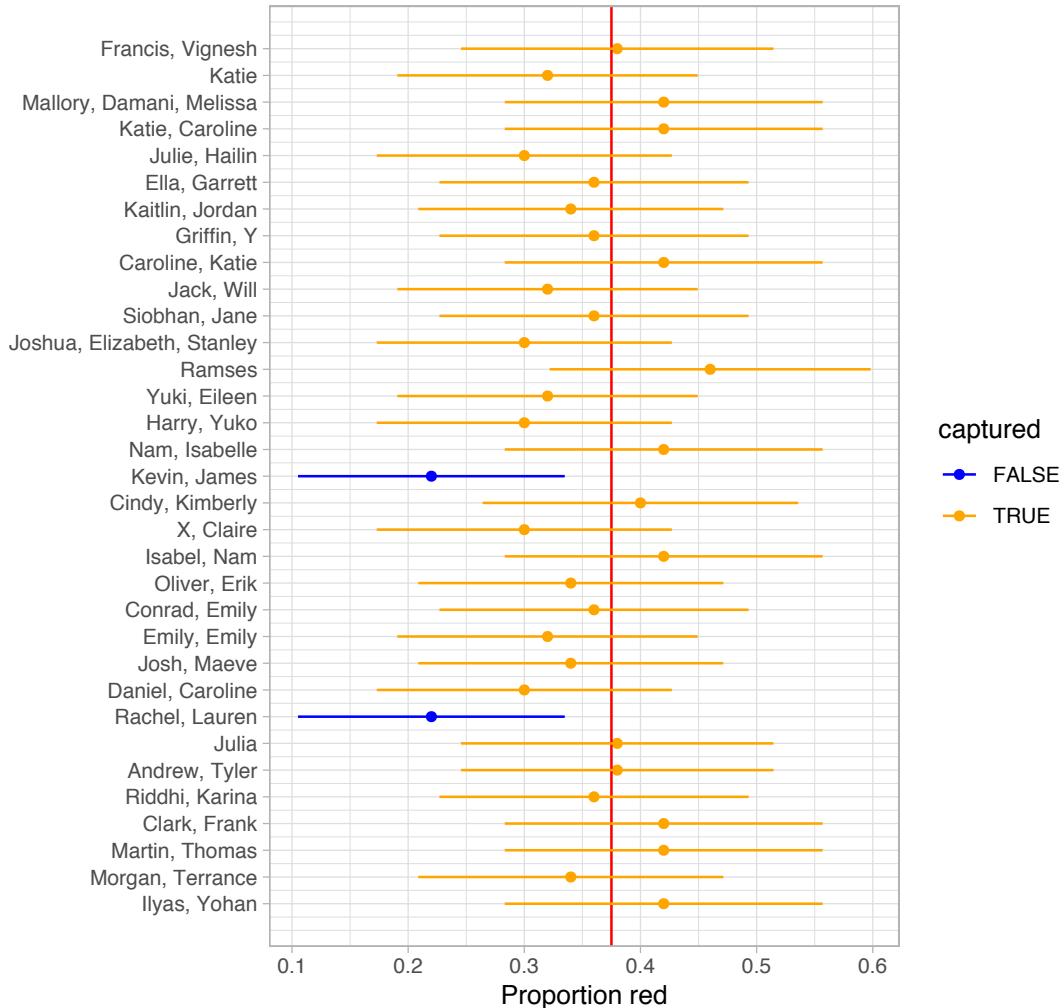


FIGURE 9.38: 33 95 percent confidence intervals based on 33 tactile samples of size $n = 50$.

Observe that 31 of the 33 confidence intervals “captured” the true value of p , for a success rate of $31 / 33 = 93.94\%$. While this is not quite 95%, recall that we *expect* about 95% of such confidence intervals to capture p , the actual observed success rate will vary.

Theoretical-based methods like this have largely been used in the past because we didn't have the computing power to perform simulation-based meth-

ods such as bootstrapping. They are still commonly used though and if the sampling and bootstrap distributions are normally distributed, they can provide a nice option for constructing confidence intervals as well as performing hypothesis tests as we will see in Chapter 10.

9.7.3 Additional resources

An R script file of all R code used in this chapter is available here⁸.

If you want to more examples of the `infer` workflow to construct confidence intervals, we suggest you check out the `infer` package homepage, in particular, a series of example analyses available at <https://infer.netlify.com/articles/>.

9.7.4 What's to come?

Now that we've equipped ourselves with confidence intervals, in Chapter 10 we'll cover the other common tool for statistical inference: hypothesis testing.

⁸ [scripts/09-confidence-intervals.R](#)

10

Hypothesis Testing

Now that we've studied one commonly used method for statistical inference, confidence intervals in the previous Section 9, we'll now study the other commonly used method, hypothesis testing. Hypothesis tests allow us to take a sample of data from a population and infer about the plausibility of competing hypotheses. For example, in the upcoming “promotions” activity in Section 10.1, you'll study the data collected from a psychology study in the 1970's to infer as to whether there exists gender-based discrimination in the banking industry as a whole.

The good news is we've already covered many of the necessary concepts to understand hypothesis testing in Chapters 8 and 9. We will expand further on these ideas here and also provide a general framework for understanding hypothesis tests. By understanding this general framework, you'll be able to adapt it to many different scenarios.

The same can be said for confidence intervals. There was one general framework that applies to all confidence intervals and we elaborated on this using the `infer` package pipeline in Chapter 9. The specifics may change slightly for each variation, but the important idea is to understand the general framework so that you can apply it to more specific problems. We believe that this approach is much better in the long-term than teaching you specific tests and confidence intervals rigorously.

Needed packages

Let's load all the packages needed for this chapter (this assumes you've already installed them). Recall from our discussion in Section 5.4.1 that loading the `tidyverse` package by running `library(tidyverse)` loads the following commonly used data science packages all at once:

- `ggplot2` for data visualization
- `dplyr` for data wrangling
- `tidy` for converting data to “tidy” format
- `readr` for importing spreadsheet data into R
- As well as the more advanced `purrr`, `tibble`, `stringr`, and `forcats` packages

If needed, read Section 2.3 for information on how to install and load R packages.

```
library(tidyverse)
library(infer)
library(moderndive)
library(nycflights13)
library(ggplot2movies)
```

10.1 Promotions activity

Let's start with an activity studying the effect of gender on promotions at a bank.

10.1.1 Does gender affect promotions at bank?

Say you are working at a bank in the 1970's and you are submitting your resume to apply for a promotion. Will your gender affect your chances of getting promoted? To answer this question, we'll focus on a study published in the "Journal of Applied Psychology" in 1974 and previously used in the OpenIntro¹ series of statistics textbooks.

To begin the study, 48 bank supervisors were asked to assume the role of a hypothetical personnel director of a bank with multiple branches. Every one of the bank supervisors was given a resume and asked whether or not the candidate on the resume was fit to be promoted to a new position in one of their branches.

However, each of these 48 resumes were identical in all respects except one: the name of the applicant at the top of the resume. 24 of the supervisors were randomly given resumes with stereotypically "male" names while 24 of the supervisors were randomly given resumes with stereotypically "female" names. Since only (binary) gender varied from resume to resume, researchers could isolate the effect of this variable in promotion rates.

Note: While many people today (including the authors) disagree with such a binary view of gender, it is important to remember that this study was

¹<https://www.openintro.org/>

conducted at a time where more nuanced views of gender were not as prevalent. Despite this imperfection, we decided to still use this example as we feel it still demonstrates relevant insight about the nature of the workplace.

The `moderndive` package contains the data on the 48 applicants in the `promotions` data frame. Let's explore this data first:

```
promotions

# A tibble: 48 × 3
  id decision gender
  <int> <fct>   <fct>
1     1 promoted male
2     2 promoted male
3     3 promoted male
4     4 promoted male
5     5 promoted male
6     6 promoted male
7     7 promoted male
8     8 promoted male
9     9 promoted male
10    10 promoted male
# ... with 38 more rows
```

The variable `id` acts as an identification variable for all 48 rows, the `decision` variable indicates whether the applicant was selected for promotion or not, while the `gender` variable indicates the gender of the name used on the resume. Recall that this data does not pertain to 24 actual men and 24 actual women, but rather 48 identical resumes of which 24 were assigned stereotypically “male” names and 24 were assigned stereotypical “female” names.

Let's perform an exploratory data analysis in Figure 10.1 of the relationship between the two categorical variables `decision` and `gender`. Recall that we saw in Section 3.8.3 that one way we can visualize such a relationship is using a stacked barplot.

```
ggplot(promotions, aes(x = gender, fill = decision)) +
  geom_bar() +
  labs(x = "Gender of name on resume")
```

It appears that resumes with female names were much less likely to be accepted for promotion. Let's quantify these promotions rates by computing the

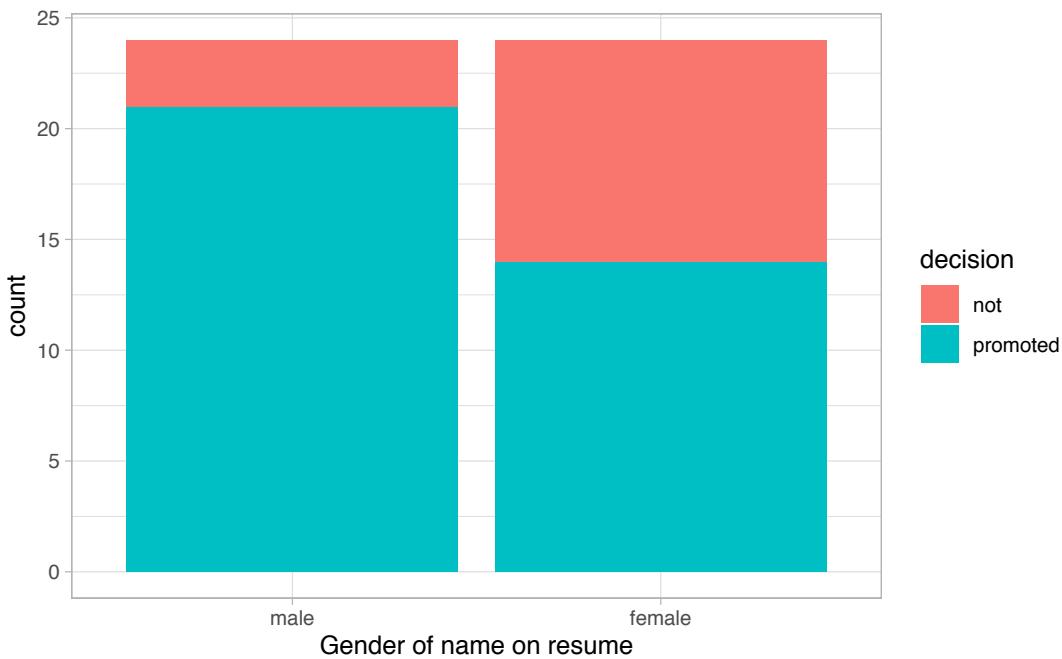


FIGURE 10.1: Barplot of relationship between gender and promotion decision.

proportion of resumes accepted for promotion for each group using the `dplyr` package for data wrangling:

```
promotions %>%
  group_by(gender, decision) %>%
  summarize(n = n())

# A tibble: 4 x 3
# Groups:   gender [2]
  gender decision     n
  <fct>   <fct>   <int>
1 male    not        3
2 male    promoted  21
3 female  not        10
4 female  promoted  14
```

So of the 24 resumes with male names 21 were selected for promotion, for a proportion of $21/24 = 0.875 = 87.5\%$. On the other hand, of the 24 resumes with female names 14 were selected for promotion, for a proportion of $14/24 = 0.583 = 58.3\%$. Comparing these two rates of promotion, it appears that resumes with male names were selected for promotion at a rate $0.875 - 0.583$

$= 0.292 = 29.2\%$ higher than resumes with female names. A clear edge for the “male” applicants.

The question is however, does this provide conclusive evidence that there is some discrimination in terms of gender in promotions at banks? Could a difference in promotion rates of 29.2% still occur by chance, even in a world of no gender-based discrimination? In other words, what is the role of sampling variation in these results? To answer this question, we’ll again rely on simulation to generate results as we did with our sampling bowl in Chapter 8 and our pennies in Chapter 9.

10.1.2 Shuffling once

First, imagine a hypothetical universe with no gender discrimination in promotions, with a big emphasis on the “hypothetical.” In such a hypothetical universe, the gender of an applicant would have no bearing on their chances of promotions. Bringing things back to our `promotions` data frame, the `gender` variable would thus be an irrelevant label. If the `gender` label is irrelevant, then we can randomly “shuffle” this label to no consequence!

To illustrate this idea, let’s narrow our focus to six arbitrarily chosen resumes of the 48 in Table 10.1: three resumes not resulting in a promotion and three resumes resulting in a promotion. The left-hand side of the table displays the original relationship between `decision` and `gender` that was actually observed by researchers.

However, in our hypothesized universe of no gender discrimination, gender is irrelevant and thus it is of no consequence to randomly shuffle the values of `gender`. The right-hand side of the table displays one such possible random shuffling. Observe how the number of male and female remains the same at three each, but they are listed in a different order.

Again, such random shuffling of gender only makes sense in the hypothesized universe of no gender discrimination. How could we extend this shuffling of the gender variable to all 48 resumes by hand? One way would be by using standard deck of 52 playing cards, which we display in Figure 10.2.

Since half the cards are red and the other half are black, by removing 2 red cards and 2 black cards, we would end up with 24 red cards and 24 black cards. After shuffling these 48 cards as seen in Figure 10.3, we can flip the cards over one-by-one, assigning “male” for each red card and “female” for each black card.

We’ve saved one such shuffling/permuation in the `promotions_shuffled` data

TABLE 10.1: Relationship of decision and gender for 6 resumes: original (left) and shuffled (right).

	id	decision	gender
	1	promoted	male
	5	promoted	male
	6	promoted	male
	20	promoted	male
	21	promoted	male
	47	not	female

	id	decision	gender
	1	promoted	male
	5	promoted	male
	6	promoted	male
	20	promoted	female
	21	promoted	male
	47	not	male

*



FIGURE 10.2: Standard deck of 52 playing cards.



FIGURE 10.3: Shuffling deck cards.

frame of the `moderndive` package. If you view both the original `promotions` and the shuffled `promotions_shuffled` data frames and compare them, you'll see that while the `decision` variables are identical, the `gender` variables are indeed different.

```
promotions_shuffled
```

```
# A tibble: 48 × 3
  id decision gender
  <int> <fct>   <fct>
1 1 promoted female
2 2 promoted female
3 3 promoted male 
4 4 promoted female
5 5 promoted male 
6 6 promoted male 
7 7 promoted male 
8 8 promoted female
9 9 promoted male 
10 10 promoted female
```

```
# ... with 38 more rows
```

Let's repeat the same exploratory data analysis we did for the original `promotions` data on our shuffled `promotions_shuffled` data frame. Let's create a barplot visualizing the relationship between `decision` and shuffled `gender` and compare this to the original unshuffled version in Figure 10.4.

```
ggplot(promotions_shuffled, aes(x = gender, fill = decision)) +
  geom_bar() +
  labs(x = "Gender of resume name")
```

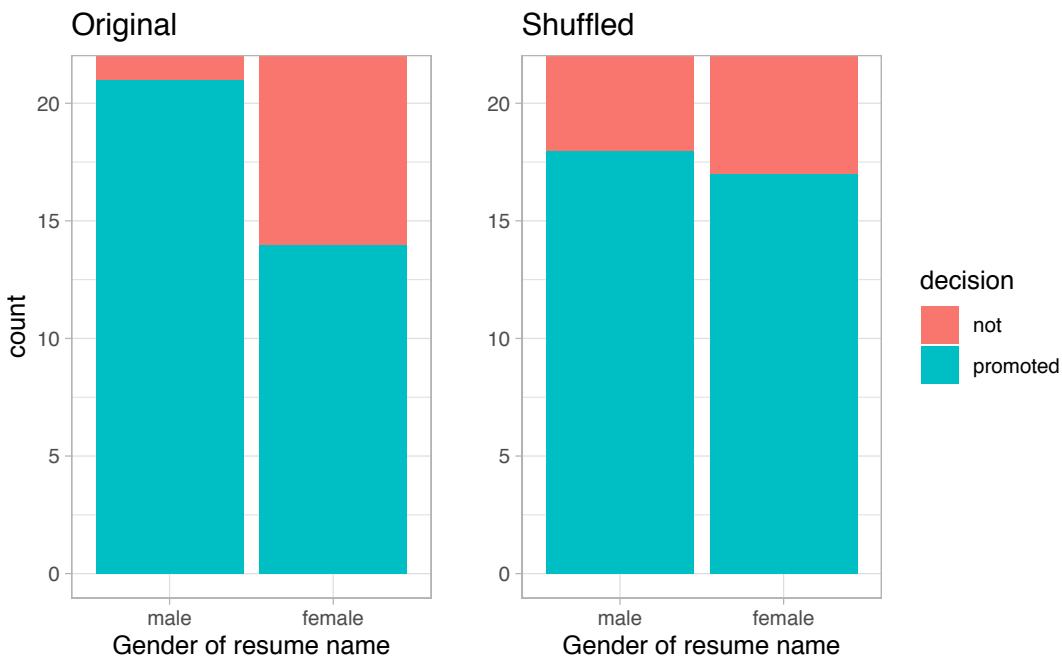


FIGURE 10.4: Barplot of relationship between shuffled gender and promotion decision.

Compared to the barplot in Figure 10.1, it appears the difference in “male” vs “female” promotions rates is now different. Let's also compute the proportion of resumes accepted for promotion for each group:

```
promotions_shuffled %>%
  group_by(gender, decision) %>%
  summarize(n = n())

# A tibble: 4 x 3
# Groups:   gender [2]
```

```

gender decision      n
<fct> <fct> <int>
1 male   not       6
2 male   promoted  18
3 female not       7
4 female promoted 17

```

So in this hypothetical universe of no discrimination, $18/24 = 0.75 = 75\%$ of “male” resumes were selected for promotion. On the other hand, $17/24 = 0.708 = 70.8\%$ of “female” resumes were selected for promotion. Comparing these two values, it appears that resumes with male names were selected for promotion at a different rate of $0.75 - 0.708 = 0.042 = 4.2\%$.

Observe how this difference in rates is different than the difference in rates of $0.292 = 29.2\%$ we originally observed. This is once again due to sampling variation. How can we better understand the effect of this sampling variation? By doing this several times!

10.1.3 Shuffling 16 times

We recruited 16 groups of our friends to repeat the above shuffling/permuting exercise. We recorded these values in a shared spreadsheet²; we display a snapshot of the first 10 rows and 5 columns in Figure 10.5

id	decision	Cassandra, Nox	Priya, Jenny, Eindra	Maddie, Grace, Stephanie	Dahlia, Sarah	Claire, Cindy, Danna
1	not	m	m	m	m	m
2	not	m	m	f	m	m
3	not	m	f	m	m	f
4	not	f	f	f	f	f
5	not	f	m	f	f	f
6	not	m	m	m	f	f
7	not	f	f	m	f	m
8	not	m	f	f	m	f
9	not	m	f	f	m	f
10	not	m	f	m	f	f

FIGURE 10.5: Snapshot of shared spreadsheet of shuffled resumes.

In Figure 10.6, we show the distribution of the 16 “shuffled” differences in promotion rates using a histogram. Remember that the histogram represents the differences in promotion rates in our *hypothesized universe* of no gender discrimination. We also mark the observed difference in promotion rate that happened in real-life of $0.292 = 29.2\%$ with a red line.

²<https://docs.google.com/spreadsheets/d/1Q-ENy3o5IrpJshJ7gn3hJ5A0TOWV2AZrKNHMsshQtIE/edit?usp=sharing>

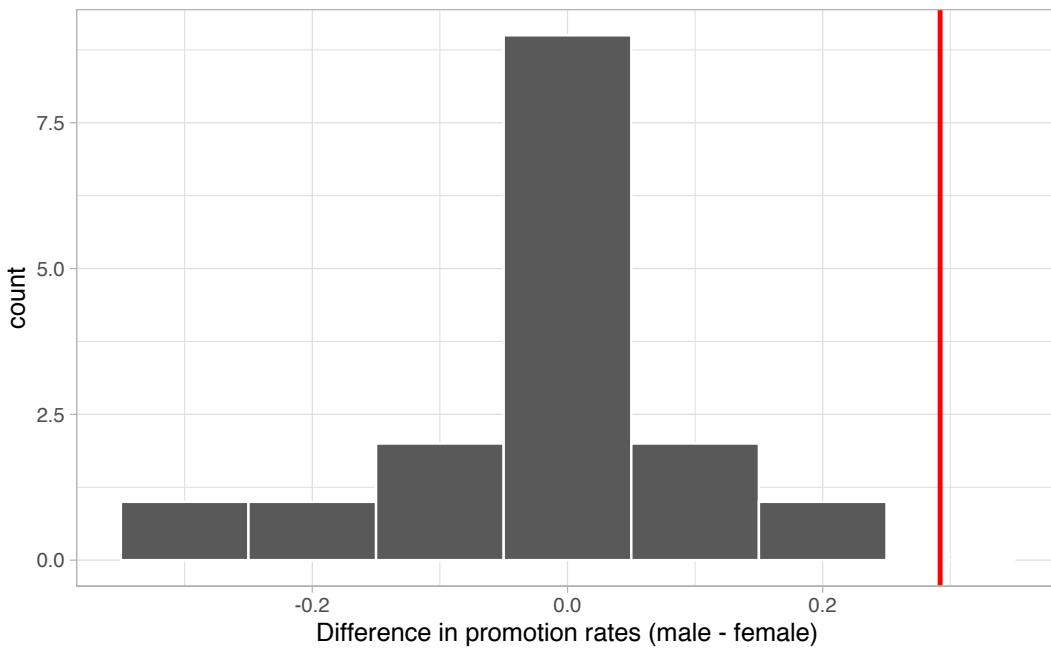


FIGURE 10.6: Distribution of "shuffled" differences in promotions.

Observe first that the histogram is both roughly centered at 0. Saying that the difference in promotion rates is 0 is equivalent to saying that both genders had the same promotion rate. In other words, these 16 values are consistent with what we would expect in our hypothesized universe of no gender discrimination. However, while the values are centered at 0, there is variation about 0. This is because even in a hypothesized universe of no gender discrimination, you still observe small differences in promotion rates because of *sampling variation*. Looking at the histogram, it could even be as extreme as -0.292 or 0.208.

Turning our attention to what we observed in real-life: the difference of 0.292 = 29.2% is marked with a red line. Ask yourself: in a hypothesized world of no gender discrimination, how likely would it be that we observe this difference? In our opinion, not often! Now ask yourself: what does this say about our hypothesized universe of no gender discrimination.

10.1.4 What did we just do?

What we just demonstrated in this activity is the statistical procedure known as *hypothesis testing*, specifically via a *permutation test*. Such procedures allow us to test the validity of hypotheses using sampled data. The term "permutation" is the mathematical term for "shuffling": take a series of values and reorder them randomly, as you did with the playing cards.

In fact, permutations are another form of resampling. While the bootstrap method involves resampling with replacement, permutation methods involve resampling without replacement. Think of our exercise involving the slips of paper representing pennies and the hat in Section 9.1: after sampling a penny, you put it back in the hat. Now think of our deck of cards, after drawing a card, you laid it out in front of you without putting it back in the hat.

In our above example, we tested the validity of the hypothesized universe of no gender discrimination. Since the evidence contained in our observed sample of 48 resumes in the `promotions` data frame was somewhat inconsistent with our hypothesized universe, we would be inclined to *reject* this hypothesized universe and declare that the evidence suggests there is gender discrimination.

Much like with our case study on whether yawning is contagious from Section 9.6, the above example involves inference about an unknown difference of population proportions: $p_m - p_f$, where p_m is the population proportion of “male” resumes being recommended for promotion and p_f is the population proportion of “male” resumes being recommended for promotion.

So based on our sample of $n_m = 24$ “male” applicants and $n_w = 24$ “female” applicants, the point estimate for $p_m - p_f$ is the difference of sample proportions $\hat{p}_m - \hat{p}_f = 0.875 - 0.583 = 0.292 = 29.2\%$. This difference in favor of “male” resumes of 0.292 is greater than 0, suggesting discrimination in favor of men.

However the question we asked ourselves was “is this difference meaningfully different than 0?” In other words, is that difference indicative of true discrimination, or can we just attribute it to sampling variation? Hypothesis testing allows us to make such distinctions.

10.2 Understanding hypothesis tests

Much like the terminology, notation, and definitions relating to sampling you saw in Section 8.3, there is a lot of terminology, notation, and definitions related to hypothesis testing that one must know before being able to conduct hypothesis tests effectively. Learning these may seem like a very daunting task at first. However with practice, practice, and practice, anyone can master them.

First, a **hypothesis** is a statement about the value of an unknown population parameter. In our resume activity, our population parameter is the difference

in population proportions $p_m - p_f$. Hypothesis tests can involve any of the population parameters in Table 8.8 of the 6 inference scenarios we'll cover in this book and more.

Second, a **hypothesis test** consists of a test between two competing hypotheses: 1) a **null hypothesis** H_0 (pronounced “H-naught”) versus 2) an **alternative hypothesis** H_A (also denoted H_1).

Generally the null hypothesis is a claim that there really is “no effect” or “no difference.” In many cases, the null hypothesis represents the status quo or that nothing interesting is happening. Furthermore, generally the alternative hypothesis is the claim the experimenter or researcher wants to establish or find evidence for and is viewed as a “challenger” hypothesis to the null hypothesis H_0 . In our resume activity, an appropriate hypothesis test would be:

$$\begin{aligned} H_0 &: \text{men and women are promoted at the same rate} \\ \text{vs } H_A &: \text{men are promoted at a higher rate than women} \end{aligned}$$

Note some of the choices we have made. First, we set the null hypothesis H_0 to be that there is no difference in promotion rate and the “challenger” alternative hypothesis H_A to be that there is a difference. While it would not be wrong in principle to reverse the two, it is a convention in statistical inference that the null hypothesis is set to reflect a “null” situation where “nothing is going on,” in this case that there is no difference in promotion rates. Furthermore we set H_A to be that men are promoted at a *higher* rate, a subjective choice reflecting a prior suspicion we have that this is the case. We call such alternative hypotheses *one-sided alternatives*. If someone else however does not share such suspicions and only wants to investigate that there is a difference in rate, whether higher or lower, they we set what is known as a *two-sided alternative*.

We can re-express the formulation of our hypothesis test in terms of the notation for our population parameter of interest:

$$\begin{aligned} H_0 &: p_m - p_f = 0 \\ \text{vs } H_A &: p_m - p_f > 0 \end{aligned}$$

Observe how the alternative hypothesis H_A is one-sided $p_m - p_f > 0$. Had we opted for a two-sided alternative, we would have set $p_m - p_f \neq 0$. For the purposes of the illustration of the terminology, notation, and definitions related to hypothesis testing however, we'll stick with the simpler one-sided alternative. We'll present an example of a two-sided alternative in Section 10.5.

Third, a **test statistic** is a point estimate/sample statistic formula used for hypothesis testing, where a sample statistic is merely a summary statistic based on a sample of observations. Recall we saw in Section 4.3 that a summary statistic takes in many values and returns only one. Here, a sample would consist of $n_m = 24$ “male” resumes and $n_f = 24$ “female” resumes. The point estimate of interest is the resulting difference in sample proportions $\hat{p}_m - \hat{p}_f$. This quantity estimates of the unknown population parameter of interest: the difference in population proportions $p_m - p_f$.

Fourth, the **observed test statistic** is the value of the test statistic that we observed in real-life. In our case we computed this value using the data saved in the `promotions` data frame: it was the observed difference of $\hat{p}_m - \hat{p}_f = 0.875 - 0.583 = 0.292 = 29.2\%$.

Fifth, the **null distribution** is the sampling distribution of the test statistic *assuming the null hypothesis H_0 is true*. Ooof! That’s a long one! Let’s unpack it slowly. The key to understanding the null distribution is that the null hypothesis H_0 *assumed* to be true. We’re not saying that it is true at this point, merely assuming it. In our case, this corresponds to our hypothesized universe of no gender discrimination in promotion rates. Assuming the null hypothesis H_0 , also stated as “Under H_0 ”, how does the test statistic vary due to sampling variation? In our case, how will the difference in sample proportions $\hat{p}_m - \hat{p}_f$ vary due to sampling? Recall from Section 8.3.2 that distributions that display how point estimates vary due to sampling variation are called *sampling distributions*. The only additional thing to keep in mind for null distributions is that they are sampling distributions *assuming the null hypothesis H_0 is true*.

In our case, we previously visualized the null distribution in Figure 10.6, which we re-display below in Figure 10.7 using our new notation and terminology. It is the distribution of the 16 different difference in sample proportions our friends computed *assuming* a hypothetical universe of no gender discrimination.

Sixth, the **p-value** is the probability of obtaining a test statistic just as extreme or more extreme than the observed test statistic *assuming the null hypothesis H_0 is true*. Double ooof! Let’s unpack this slowly as well. You can think of the p-value as a quantification of “surprise”: assuming H_0 is true how surprised are we at observing the test statistic we did? Or in our case, in our hypothesized universe of no gender discrimination how surprised are we that we observed a difference in promotion rates of $0.292 = 29.2\%$? Very surprised? Somewhat surprised?

The p-value quantifies this probability, or in the case of Figure 10.7, of our 16 difference in sample proportions, what proportion had a more “extreme”

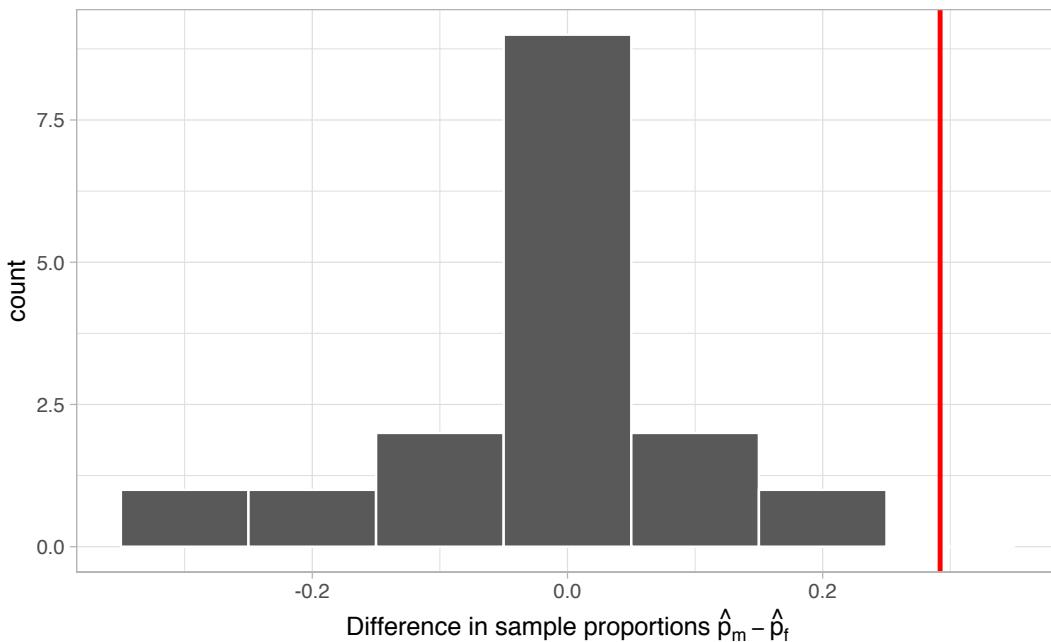


FIGURE 10.7: Null distribution and observed test statistic.

result? Here, extreme is defined in terms of the alternative hypothesis H_A that “male” applicants are promoted at a higher rate than “female” applicants. In other words, how often was the discrimination in favor of men even more pronounced than $0.875 - 0.583 = 0.292 = 29.2\%$?

In this case, only 0 time out of 16 did we obtain a difference in proportion greater than or equal to the observed difference of $0.292 = 29.2\%$. A very rare outcome of only 1 time in 16! Given the rarity of such a pronounced difference in promotion rates in our hypothesized universe of no gender discrimination, we’re inclined to *reject* this hypothesis in favor of the one saying there is discrimination in favor of the “male” applicants. We’ll see later on however, the p-value isn’t quite $1/16$, but rather $(0 + 1)/(16 + 1) = 1/17 = 0.059$ as we need to include the the observed test statistic in our calculation.

Seventh and lastly, in many hypothesis testing procedures, it is common to and recommended to set the **significance level** of the test beforehand. It is denoted by the Greek letter α . This value acts as a cutoff on the p-values, where if the p-value falls below α , we would reject the null hypothesis H_0 . Alternatively, if the p-value does not fall below α , we would fail to reject H_0 . Note this statement is not quite the same as saying we “accept” H_0 . This distinction is rather subtle and not immediately obvious, so we’ll revisit it later in Section 10.4.

While different fields tend to use different values of α , some commonly used

values for α are 0.1, 0.01, and 0.05, with 0.05 being the choice people often make when people don't put much thought into it. We'll talk more about α significance levels in Section 10.4, but first let's fully conduct the hypothesis test corresponding to our promotions activity in the next section.

10.3 Conducting hypothesis tests

In Section 9.4, we showed you how to construct confidence intervals. We first illustrated how to do this using raw `dplyr` data wrangling verbs and the `rep_sample_n()` function which we introduced in Section 8.2.3 when illustrating the use of the virtual shovel. In particular, we constructed confidence intervals by resampling with replacement by setting the `replace = TRUE` argument to the `rep_sample_n()` function.

We then showed you how to perform the same task using the `infer` package workflow. While the end result of both workflows are the same, a bootstrap distribution from which we can construct a confidence interval, the `infer` package workflow emphasizes each of the steps in the overall process in Figure 10.8 using function names that are intuitively named:

1. `specify()` the variables of interest in your data frame
2. `generate()` replicates of bootstrap resamples with replacement
3. `calculate()` the summary statistic of interest
4. `visualize()` the resulting bootstrap distribution and the confidence interval.

In this section, we now show you how to extend and modify the previously seen `infer` pipeline to conduct hypothesis tests. You'll notice that the basic outline of the workflow is almost identical, except for an additional `hypothesize()` step between `specify()` and `generate()`, as can be seen in Figure 10.9.

Furthermore, we'll use a pre-specified significance level $\alpha = 0.001$ for this hypothesis test. Let's leave the justification of this choice of α until later on in Section 10.4.

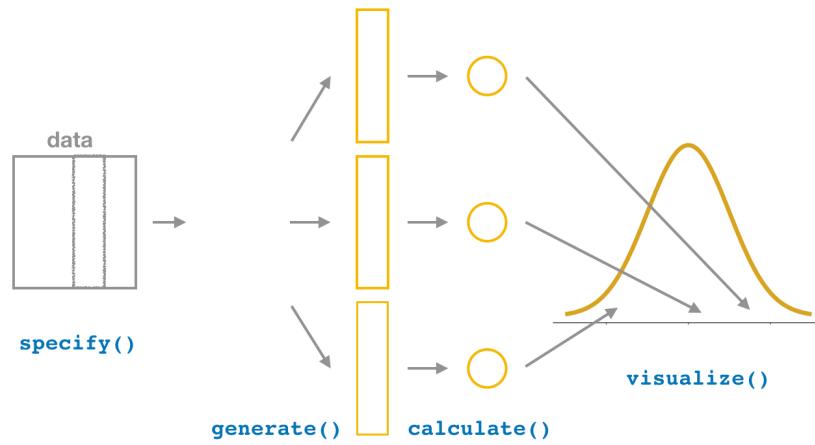


FIGURE 10.8: Confidence intervals via the infer package.

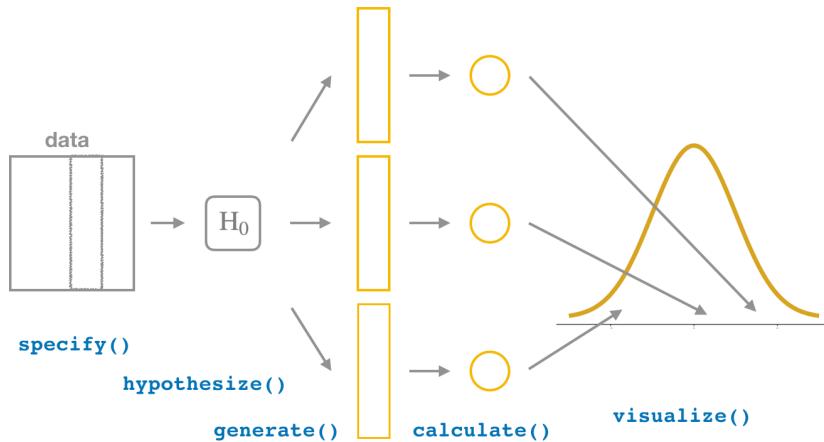


FIGURE 10.9: Hypothesis testing via the infer package.

10.3.1 infer package workflow

1. `specify` variables

Recall that we use the `specify()` verb to denote the response and, if needed, explanatory variables for our study. In this case, since we are interested in any potential effects of gender on promotion decisions, we set `decision` as the response variable and `gender` as the explanatory variable using the `formula` argument using the notation `<response> ~ <explanatory>` where `<response>` is the name of the response variable in the data frame and `<explanatory>` is the name of the explanatory variable. So in our case it is `decision ~ gender`. Lastly, since

we are interested in the proportions of resumes "promoted" and not proportions of resumes not promoted, we set the argument `success = "promoted"`

```
promotions %>%
  specify(formula = decision ~ gender, success = "promoted")
```

```
Response: decision (factor)
Explanatory: gender (factor)
# A tibble: 48 x 2
  decision gender
  <fct>    <fct>
  1 promoted male
  2 promoted male
  3 promoted male
  4 promoted male
  5 promoted male
  6 promoted male
  7 promoted male
  8 promoted male
  9 promoted male
  10 promoted male
# ... with 38 more rows
```

Again, notice how the `promotions` data itself doesn't change, but the `Response: decision (factor)` and `Explanatory: gender (factor)` *meta-data* do. This is similar to how the `group_by()` verb from `dplyr` doesn't change the data, but only adds "grouping" meta-data as we saw in Section 4.4.

2. hypothesize the null

In order to conduct hypothesis tests using the `infer` workflow, we need a new step: `hypothesize()`. Recall from Section 10.2 that our hypothesis test was

$$\begin{aligned} H_0 : p_m - p_f &= 0 \\ \text{vs } H_A : p_m - p_f &> 0 \end{aligned}$$

In other words, the null hypothesis H_0 corresponding to our "hypothesized universe" stated that there was no difference in gender-based discrimination rates. We set this null hypothesis H_0 in our `infer` workflow by setting the `null` argument of the `hypothesize()` function to either:

- "point" for hypotheses involving a single sample or
- "independence" for hypotheses involving two samples

In our case, since we have two samples (the “male” and “female” applicants), we set `null = "independence"`.

```
promotions %>%
  specify(formula = decision ~ gender, success = "promoted") %>%
  hypothesize(null = "independence")
```

```
# A tibble: 48 x 2
  decision gender
  <fct>    <fct>
  1 promoted male
  2 promoted male
  3 promoted male
  4 promoted male
  5 promoted male
  6 promoted male
  7 promoted male
  8 promoted male
  9 promoted male
  10 promoted male
# ... with 38 more rows
```

Again, the data has not changed yet. This will occur at the upcoming `generate()` step; we’re merely setting meta-data for now.

Where do the terms “point” and “independence” come from? These are two technical statistics terms. The term “point” relates from the fact that for a single group of observations, you will test the value of the point. Going back to the pennies example from Chapter 9, say we wanted to test if the mean year of all US pennies was equal to 1993 or not, we would be testing the “point” value μ as follows

$$\begin{aligned} H_0 : \mu &= 1993 \\ \text{vs } H_A : \mu &\neq 1993 \end{aligned}$$

The term “independence” relates to the fact that for two groups of observations, you are testing whether or not the response variable is independent of the explanatory variable that assigns the group. In our case, we are testing whether the `decision` response variable is “independent” of the explanatory variable `gender` that assigns each resume to either one of the two groups.

3. generate replicates

After we have set the null hypothesis, we simulate observations assuming the null hypothesis is true by repeating the shuffling exercise you performed in Section 10.1 several times. Instead of merely doing it 16 times as our groups of friends did, let's use the computer to repeat this 1000 times by setting `reps = 1000` in the `generate()` function. However, unlike with confidence intervals where we generated replicates using `type = "bootstrap"` resampling with replacement, we'll now perform shuffles/permuations by setting `type = "permute"`. Recall that shuffles/permuations are a form of resampling without replacement.

```
promotions %>%
  specify(formula = decision ~ gender, success = "promoted") %>%
  hypothesize(null = "independence") %>%
  generate(reps = 1000, type = "permute")
```

```
Response: decision (factor)
Explanatory: gender (factor)
Null Hypothesis: independence
# A tibble: 48,000 x 3
  decision gender replicate
  <fct>    <fct>     <int>
1 promoted male      1
2 not       male      1
3 promoted male      1
4 promoted female   1
5 promoted female   1
6 promoted female   1
7 promoted female   1
8 promoted female   1
9 promoted female   1
10 not      female   1
# ... with 47,990 more rows
```

Note the the resulting data frame has 48,000 rows. This is because we performed shuffles/permuations of the 48 values of `gender` 1000 times and thus $48,000 = 1000 \times 48$. Accordingly the variable `replicate`, indicating which resample each row belongs to, has the value 1 48 times, the value 2 48 times, all the way through to the value 1000 48 times.

4. calculate summary statistics

Now that we have 1000 replicated “shuffles” assuming the null hypothesis that both “male” and “female” applicants were promoted at the same rate, let’s `calculate()` the appropriate summary statistic for each of our 1000 shuffles. Recall from Section 10.2 that point estimates/summary statistics relating to hypothesis testing have a specific name: *test statistics*. Since the unknown population parameter of interest is the difference in population proportions $p_m - p_f$, the test statistic of interest here is the difference in sample proportions $\hat{p}_m - \hat{p}_f$.

For each of our 1000 shuffles, we can calculate this test statistic by setting `stat = "diff in props"`. Furthermore, since we are interested in $\hat{p}_m - \hat{p}_f$ and not the reverse-ordered $\hat{p}_f - \hat{p}_m$, we set `order = c("male", "female")`. Let’s save the result in a data frame called `null_distribution`:

```
null_distribution <- promotions %>%
  specify(formula = decision ~ gender, success = "promoted") %>%
  hypothesize(null = "independence") %>%
  generate(reps = 1000, type = "permute") %>%
  calculate(stat = "diff in props", order = c("male", "female"))
null_distribution
```

```
# A tibble: 1,000 x 2
  replicate      stat
  <int>     <dbl>
1       1 -0.208333
2       2  0.291667
3       3  0.125
4       4 -0.208333
5       5 -0.125
6       6  0.0416667
7       7 -0.0416667
8       8  0.291667
9       9  0.0416667
10      10  0.125
# ... with 990 more rows
```

Observe that we have 1000 values of `stat`, each representing one “shuffled” instance of $\hat{p}_m - \hat{p}_f$ in a hypothesized world of no gender discrimination. Note as well we chose the name of this data frame carefully: `null_distribution`. Recall once again from Section 10.2 that such sampling distributions when the null hypothesis H_0 is assumed to be true have a special name: the *null distribution*.

But wait! What happened in real-life? What was the observed difference in promotions rates? In other words, what was the *observed test statistic* $\hat{p}_m - \hat{p}_f$? Recall from Section 10.1 that we computed this observed difference by hand to be $0.875 - 0.583 = 0.292 = 29.2\%$. We can also achieve this using the code above but with the `hypothesize()` and `generate()` steps removed. Let's save this in `obs_diff_prop`

```
obs_diff_prop <- promotions %>%
  specify(decision ~ gender, success = "promoted") %>%
  calculate(stat = "diff in props", order = c("male", "female"))
obs_diff_prop
```

```
# A tibble: 1 × 1
  stat
  <dbl>
1 0.291667
```

5. visualize the p-value

The final step is to measure how surprised would we be by a promotion difference of 29.2% in a hypothesized universe of no gender discrimination. If very surprised, then we would be inclined to reject the validity of our hypothesized universe.

We start by visualizing the *null distribution* of our 1000 values of $\hat{p}_m - \hat{p}_f$ using `visualize()` in Figure 10.10. Recall that these are values of the difference in promotion rate assuming H_0 is true, in other words in our hypothesized universe of no gender discrimination.

```
visualize(null_distribution, binwidth = 0.1)
```

Let's now add what happened in real-life to Figure 10.10, the observed difference in promotions rates of $0.875 - 0.583 = 0.292 = 29.2\%$. However, instead of merely adding a vertical line using `geom_vline()`, let's use the `shade_p_value()` function with `obs_stat` set to the observed test statistic value we saved in `obs_diff_prop` and `direction = "right"`:

```
visualize(null_distribution, bins = 10) +
  shade_p_value(obs_stat = obs_diff_prop, direction = "right")
```

In the resulting Figure 10.11, the solid red line marks $0.292 = 29.2\%$. However,

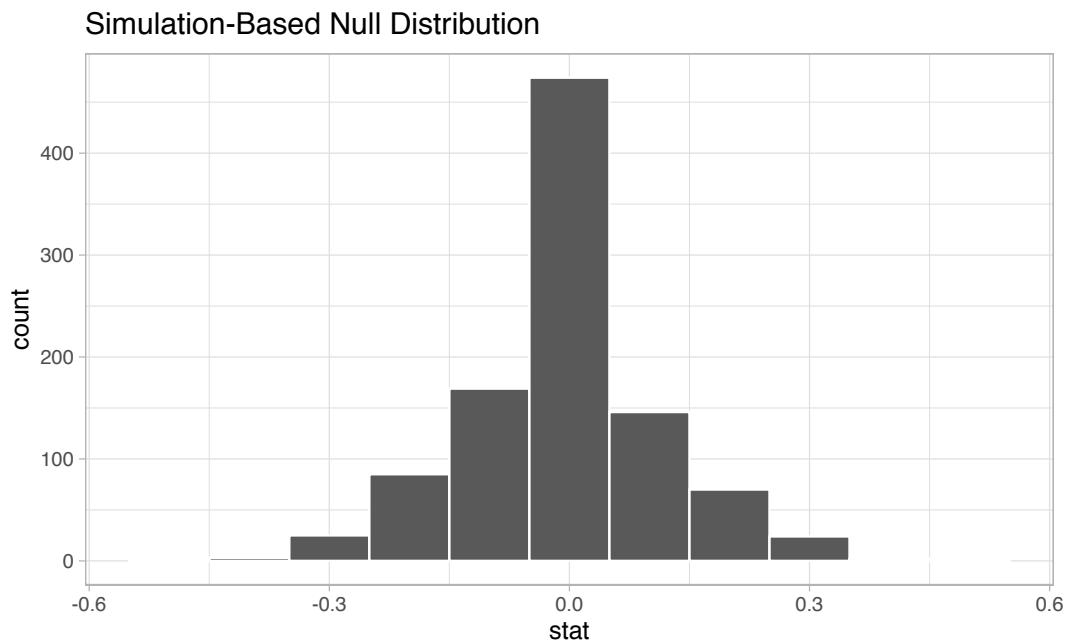


FIGURE 10.10: Bootstrap distribution

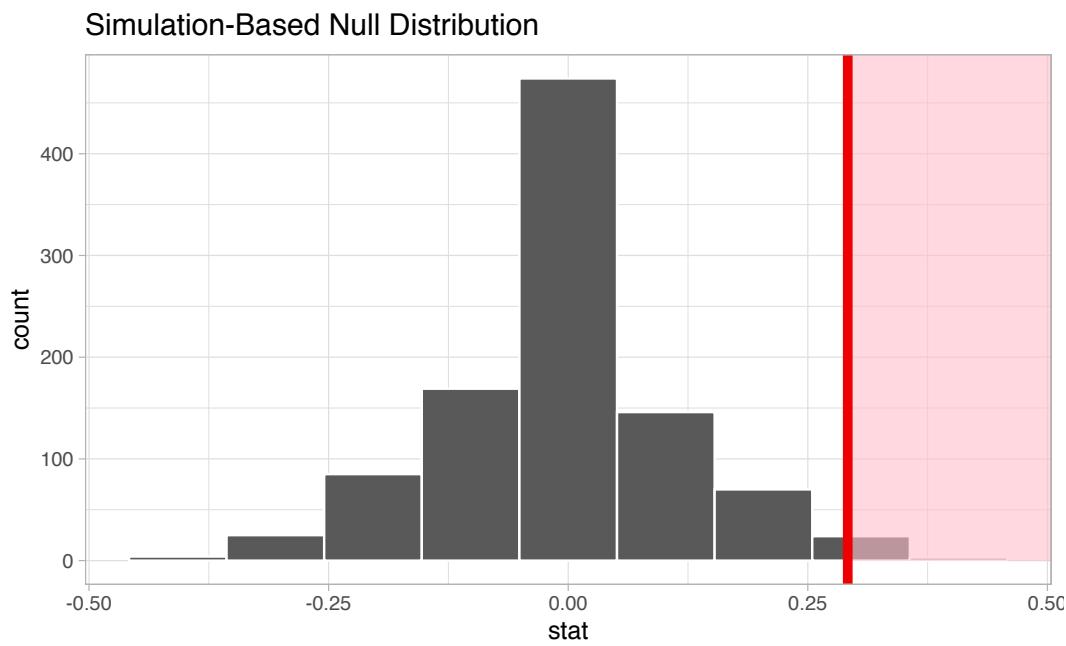


FIGURE 10.11: Shaded histogram to show p-value.

what does the shaded-region correspond to? This is the p-value. Recall the definition of the p-value from Section 10.2:

A p-value is the probability of obtaining a test statistic just as or more extreme than the observed test statistic *assuming the null hypothesis H_0 is true*.

Recall our alternative hypothesis H_A is that $p_m - p_f > 0$ i.e. there is a difference in promotion rates in favor of men. So “more extreme” corresponds to differences that are “bigger” or “more positive” or “more to the right.” Hence we set the `direction` argument of `shade_p_value()` to be `"right"`. Had our alternative hypothesis H_A been the other possible one-sided alternative $p_m - p_f < 0$ suggesting discrimination in favor of “female” applicants, we would’ve set `direction = "left"`. Had our alternative hypothesis H_A been two-sided $p_m - p_f \neq 0$ suggesting discrimination in either direction, we would’ve set `direction = "both"`.

So judging by the shaded region in Figure 10.11, it seems we would somewhat rarely observe differences in promotion rates of $0.292 = 29.2\%$ or more in a hypothesized universe of no gender discrimination. In other words, the p-value is somewhat small. Hence, we would be inclined to reject this hypothesized universe, or in statistical language: reject H_0 .

What fraction of the null distribution is shaded? In other words, what is the exact p-value? We can compute its numerical value using the `get_p_value()` function using the exact same arguments as with the `visualize()` code above:

```
null_distribution %>%
  get_p_value(obs_stat = obs_diff_prop, direction = "right")
```

```
# A tibble: 1 × 1
  p_value
  <dbl>
1 0.027
```

In other words, the probability of observing a difference in promotion rates as large as $0.292 = 29.2\%$ due to sampling variation alone is only $0.027 = 2.7\%$. Since this p-value is greater than our pre-specified significance level $\alpha = 0.001$, we fail to reject the null hypothesis $H_0 : p_m - p_f = 0$. In other words, this p-value wasn’t sufficiently small to reject our hypothesized universe of no gender discrimination.

10.3.2 Comparison with confidence intervals

One of the great things about the `infer` pipeline is that we can jump between hypothesis tests and confidence intervals with minimal changes! Recall from the previous section that to create the null distribution needed to compute the p-value, we ran the following code:

```
null_distribution <- promotions %>%
  specify(formula = decision ~ gender, success = "promoted") %>%
  hypothesize(null = "independence") %>%
  generate(reps = 1000, type = "permute") %>%
  calculate(stat = "diff in props", order = c("male", "female"))
```

To create the corresponding bootstrap distribution needed to construct a 95% confidence interval for $p_m - p_f$, we only need to make two changes. First, we remove the `hypothesize()` step since we are no longer assuming a null hypothesis H_0 is true when we bootstrap. We do this by commenting out the `hypothesize()` line of code. Second, we switch the `type` of resampling in the `generate()` step to be "bootstrap" instead of "permute".

```
bootstrap_distribution <- promotions %>%
  specify(formula = decision ~ gender, success = "promoted") %>%
  # Change 1 - Remove hypothesize():
  # hypothesize(null = "independence") %>%
  # Change 2 - Switch type from "permute" to "bootstrap":
  generate(reps = 1000, type = "bootstrap") %>%
  calculate(stat = "diff in props", order = c("male", "female"))
```

Using `bootstrap_distribution`, we first compute the percentile-based confidence intervals:

```
percentile_ci <- bootstrap_distribution %>%
  get_confidence_interval(level = 0.95, type = "percentile")
percentile_ci

# A tibble: 1 x 2
  `2.5%`  `97.5%
  <dbl>    <dbl>
1 0.0414187 0.522222
```

Using our shorthand interpretation for 95% confidence intervals from Section

9.5.2, we are 95% “confident” that the true difference in population proportions $p_m - p_f$ is between (0.041, 0.522). Let’s visualize `bootstrap_distribution` and this percentile-based 95% confidence interval for $p_m - p_f$ in Figure 10.12.

```
visualize(bootstrap_distribution) +
  shade_confidence_interval(endpoints = percentile_ci)
```

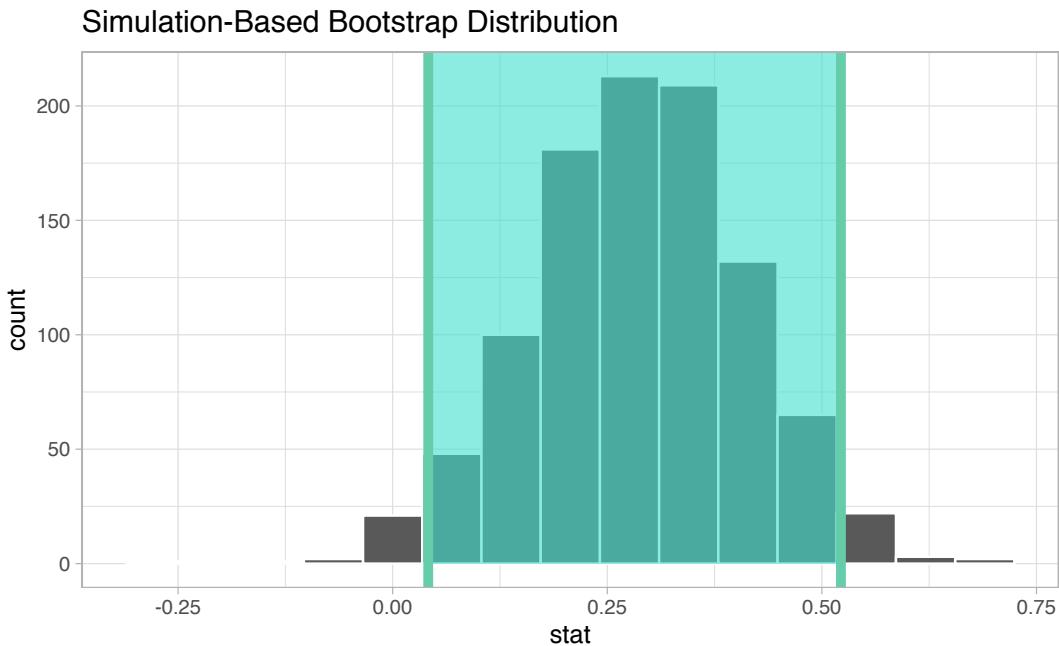


FIGURE 10.12: Percentile-based 95 percent confidence interval.

Notice a key value that is not included in the 95% confidence interval for $p_m - p_f$: 0. In other words, a difference of 0 is not included in our net, suggesting that p_m and p_f are different!

Since the bootstrap distribution appears to be roughly normally shaped, we can also used the standard error based confidence intervals, being sure to specify `point_estimate` as the observed difference in promotion rates $0.292 = 29.2\%$ saved in `obs_diff_prop`:

```
se_ci <- bootstrap_distribution %>%
  get_confidence_interval(level = 0.95, type = "se",
                          point_estimate = obs_diff_prop)
se_ci

# A tibble: 1 x 2
```

```
lower      upper
<dbl>      <dbl>
1 0.0490607 0.534273
```

Let's visualize `bootstrap_distribution` again and now the standard error based 95% confidence interval for $p_m - p_f$ in Figure 10.13. Again, notice how the value 0 is not included in our confidence interval, again suggesting that p_m and p_f are different!

```
visualize(bootstrap_distribution) +
  shade_confidence_interval(endpoints = se_ci)
```

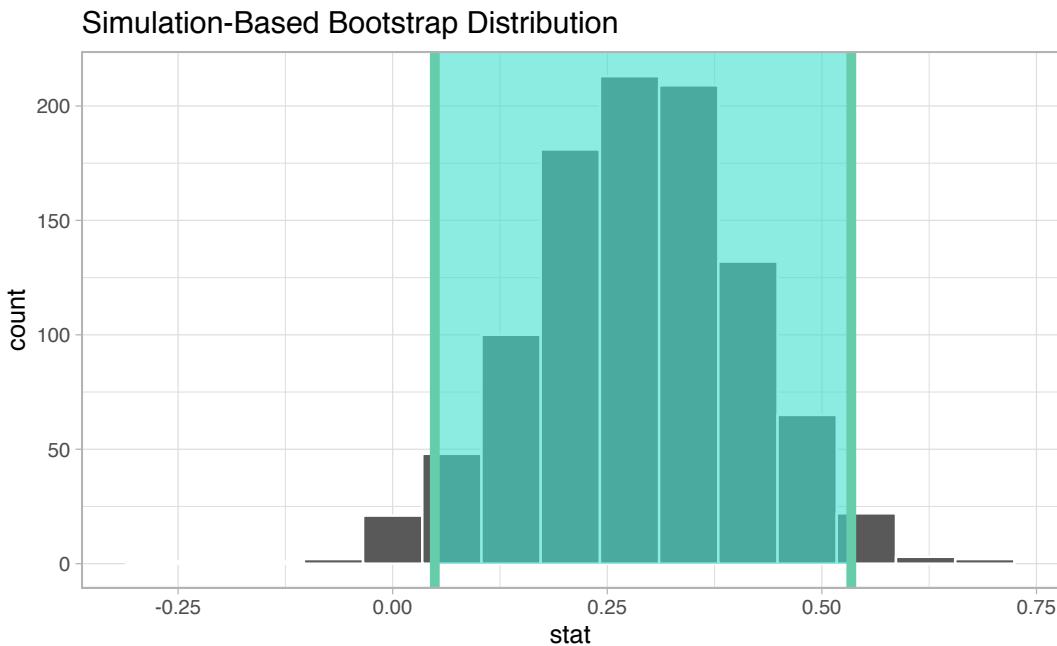


FIGURE 10.13: Standard error-based 95 percent confidence interval.

Learning check

(LC10.1) Conduct the same analysis comparing male and female promotion rates using the median rating instead of the mean rating? What was different and what was the same?

(LC10.2) Describe in a paragraph how we used Allen Downey's diagram to conclude if a statistical difference existed between the promotion rate of males and females using this study.

(LC10.3) Why are we relatively confident that the distributions of the sample

proportions will be good approximations of the population distributions of promotion proportions for the two genders?

(LC10.4) Using the definition of “*p*-value”, write in words what the *p*-value represents for the hypothesis test above comparing the promotion rates for males and females.

(LC10.5) What is the value of the *p*-value for the hypothesis test comparing the mean rating of romance to action movies? How can it be interpreted in the context of the problem?

10.3.3 “There is only one test”

Let’s recap the steps necessary to conduct a hypothesis test using the terminology, notation, and definitions related to sampling you saw in Section 10.2 and the `infer` workflow from Section #ref(infer-workflow-ht):

1. `specify()` the variables of interest in your observed data.
2. `hypothesize()` the null hypothesis H_0 . In other words, set a “model” for the universe assuming H_0 is true.
3. `generate()` shuffles assuming H_0 is true. In other words, *simulate* data assuming H_0 is true.
4. `calculate()` the *test statistic* of interest, both for the observed data and your simulated data.
5. `visualize()` the resulting *null distribution* and compute the *p-value* by comparing the null distribution to the observed test statistic.

While this is a lot to digest, especially the first time you encounter hypothesis testing, the nice is thing is once you understand this framework, then you can understand any hypothesis test. In a famous blog post, computer scientist Allen Downey called this the “There is only one test”³ framework, which he displayed in Figure 10.14.

Notice a similarity with the “hypothesis testing via `infer`” diagram you saw in Figure 10.9? That’s because the `infer` package was explicitly designed to match the “There is only one test” framework. So if you can understand the framework, you can easily generalize these ideals for all hypothesis testing scenarios, whether for population proportions p , population means μ , differences in population proportions $p_1 - p_2$, differences in population means $\mu_1 - \mu_2$, and

³<http://allendowney.blogspot.com/2016/06/there-is-still-only-one-test.html>

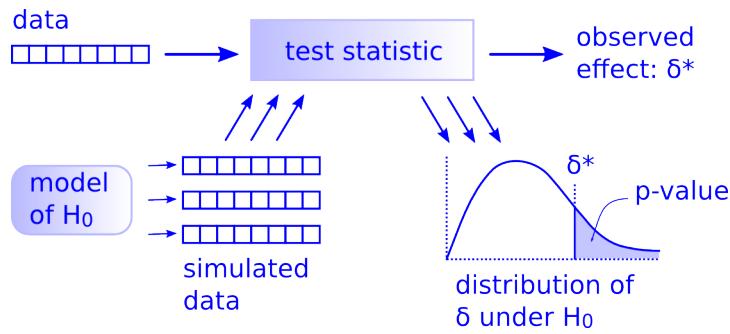


FIGURE 10.14: Hypothesis Testing Framework.

as you'll see in Chapter 11 on inference for regression, population regression intercepts β_0 and population regression slopes β_1 as well.

10.4 Interpreting hypothesis tests

Hypothesis tests are often challenging to understand at first. In this section, we'll focus on ways to help with deciphering the process and address some common misconceptions.

10.4.1 Two possible outcomes

In Section 10.2, we mentioned that given a pre-specified significance level α there are two possible outcomes of a hypothesis test:

- If the p-value is less than α , we *reject* the null hypothesis H_0 in favor of H_A .
- If the p-value is greater than or equal to α , we *fail to reject* the null hypothesis H_0 .

Unfortunately, the latter result is often misinterpreted as “accepting” the null hypothesis H_0 . While at first glance it may seem to be saying the same thing, there is a subtle difference. Saying that we “accept the null hypothesis H_0 ” is equivalent to stating “we think the null hypothesis H_0 is true.” However, saying that we “fail to reject the null hypothesis H_0 ” is saying something else: “ H_0 may still be false, we don’t have any evidence to say so.” In other words, there is an absence of proof. However absence of proof is not proof of absence.

As an analogy to this distinction, let's use the United States criminal justice system as an analogy. A criminal trial in the United States is a similar situation

in which a choice between two contradictory claims must be made about a defendant who is on trial.

1. The defendant is truly either “innocent” or “guilty.”
2. The defendant is presumed “innocent until proven guilty.”
3. The defendant is found guilty only if there is *strong evidence* that the defendant is guilty. The phrase “beyond a reasonable doubt” is often used to set the cutoff value for when enough evidence exists to find the defendant guilty.
4. The defendant is found to be either “not guilty” or “guilty” in the ultimate verdict.

In other words, “not guilty” verdicts are not suggesting the defendant is “innocent”, but instead that “while the defendant may still actually be guilty, there wasn’t enough evidence to prove this fact.” Now let’s make the connections with hypothesis tests.

1. Either the null hypothesis H_0 or the alternative hypothesis H_A is true.
2. Hypothesis tests are always conducted assuming the null hypothesis H_0 is true.
3. We reject the null hypothesis H_0 in favor of H_A only if the evidence found in the sample suggests that H_A is true. The significance level α is used to set the threshold on how strong evidence we require.
4. We ultimately decide to either “fail to reject H_0 ” or “reject H_0 .”

So while gut instinct may suggest “fail to reject H_0 ” and “accept H_0 ” are equivalent statements, they are not. “Accepting H_0 ” is equivalent to finding a defendant innocent. We cannot show that a person is innocent; we can only say that there was not enough substantial evidence to find the person guilty.

So going back to our `promotions` activity, recall in Section 10.3 that our hypothesis test was $H_0 : p_m - p_f = 0$ versus $H_A : p_m - p_f > 0$, we used a pre-specified significance level of $\alpha = 0.001$, and we found a p-value of 0.027. Since the p-value was greater than $\alpha = 0.001$, we fail to reject H_0 . In other words, while H_0 may actually be false, we didn’t find any evidence in this particular sample of 48 to suggest so. We also state this conclusion using non-statistical language: we found no evidence in the data to suggest that there was no gender discrimination.

10.4.2 Types of errors

Unfortunately, there is some chance a jury or a judge can make an incorrect decision in a criminal trial by reaching the wrong verdict. This often stems from the fact that prosecutors don't have all the relevant evidence, but are limited to what evidence the police can find. The same holds for hypothesis testing. We can make incorrect decisions about a population parameter because we only have a sample of data from the population and thus sampling variation can lead us to incorrect conclusions.

There are two possible erroneous conclusions in a criminal trial: either 1) a truly innocent person is found guilty or 2) a truly guilty person is found not guilty. Similarly, there are two possible errors in a hypothesis test: either 1) rejecting H_0 when in fact H_0 is true called a **Type I error** or 2) failing to reject H_0 when in fact H_0 is false called a **Type II error**. Another term used for "Type I error" is "false positive" while another term for "Type II error" include "false negative."

This risk of error is the price researchers pay for basing inference about a population on a sample. The only way we could be absolutely certain about our conclusion is to perform a full census of the population, but as we've seen in our numerous examples and activities so far, this is often very expensive and other impossible. This in any hypothesis test based on a sample, we tolerate the chance that a Type I error will be made and some chance that a Type II error will occur.

To help understand the concepts of Type I error and Type II errors, we apply these terms in our criminal justice analogy in Figure 10.15.

	Truly not guilty	Truly guilty
Verdict		
Not guilty verdict	Correct	Type II error
Guilty verdict	Type I error	Correct

FIGURE 10.15: Type I and Type II errors in criminal trials.

Thus a Type I error corresponds to incorrectly putting a truly innocent person in jail whereas a Type II error corresponds to letting a truly guilty person go free. Let's show the corresponding table for hypothesis tests

	H0 true	HA true
Verdict		
Fail to reject H0	Correct	Type II error
Reject H0	Type I error	Correct

FIGURE 10.16: Type I and Type II errors in hypothesis tests.

10.4.3 How do we choose alpha?

We stated earlier if we are using sample data to make inferences about a population, we run the risk of making mistakes. For confidence intervals, this would be obtaining a constructed confidence interval that doesn't contain the true value of the population parameter. For hypothesis tests, this would be making either a Type I or Type II error. Obviously, we want to minimize the probability of either error; we want a small probability of drawing an incorrect conclusion:

- The probability of a Type I Error occurring is denoted by α and is the **significance level** of the hypothesis test we defined in Section 10.2
- The probability of a Type II Error is denoted by β . $1 - \beta$ is known as the **power** of the hypothesis test.

In other words,

- α corresponds to the probability of incorrectly rejecting H_0 when in fact H_0 is true.
- β corresponds to the probability of incorrectly failing to reject H_0 when in fact H_0 is false.

Ideally, we want $\alpha = 0$ and $\beta = 0$, meaning that the chance of making either error is 0. However, this can never be the case in any situation where we are sampling for inference. We will always have the possibility of at least one error existing when we use sample data. Furthermore, these two error probabilities are inversely related. As the probability of a Type I error goes down, the probability of a Type II error goes up.

What is typically done is we fix the probability of a Type I error by pre-specifying α and try to minimize β . In other words, we will tolerate a certain fraction of incorrect rejections of the null hypothesis H_0 . This is analogous to setting the confidence level of a confidence interval. So for example if we used $\alpha = 0.01$, we would be using a hypothesis testing procedure that in the

long run would incorrectly reject the null hypothesis H_0 one percent of the time.

So what value should you use for α ? Different fields have different conventions, but some commonly used values include 0.10, 0.05, 0.01, and 0.001. However, it is important to keep in mind that if you use a relatively small value of α , then all things being equal p-values will have a harder time being less than α , and thus we would reject the null hypothesis less often. In other words, we would reject the null hypothesis H_0 only if we have *very strong* evidence to do so. This is known as a “conservative” test. On the other hand, if we used a relatively large value of α , then all things being equal p-values will have an easier time being less than α , and thus we would reject the null hypothesis more often. In other words, we would reject the null hypothesis H_0 even if we only have *mild* evidence to do so. This is known as a “liberal” test.

Learning check

(LC10.6) What is wrong about saying “The defendant is innocent.” based on the US system of criminal trials?

(LC10.7) What is the purpose of hypothesis testing?

(LC10.8) What are some flaws with hypothesis testing? How could we alleviate them?

10.5 Case study: Are action or romance movies rated higher?

Let’s apply our knowledge of hypothesis testing to answer the question: “Are action or romance movies rated higher on IMDb?” IMDb⁴ is an internet movie database providing information on movie and television show casts, plot summaries, trivia, and ratings. We’ll investigate if on average action or romance movies get a higher rating on IMDb.

⁴<https://www.imdb.com/>

10.5.1 IMDb ratings data

The `movies` dataset in the `ggplot2movies` package contains information on 58,788 movies that have been rated by users of IMDB.com.

```
movies
```

```
# A tibble: 58,788 x 24
  title    year length budget rating votes     r1     r2     r3
  <chr>   <int>  <int>  <dbl>  <dbl> <int>  <dbl>  <dbl>  <dbl>
1 $       1971    121    NA  6.4    348   4.5   4.5   4.5
2 $1000~  1939     71    NA   6      20    0     14.5  4.5
3 $21 a~  1941      7    NA  8.200    5    0     0     0
4 $40,0~  1996    70    NA  8.200    6   14.5   0     0
5 $50,0~  1975    71    NA  3.4     17   24.5   4.5   0
6 $spent   2000    91    NA  4.3     45   4.5   4.5   4.5
7 $wind~  2002    93    NA  5.3     200  4.5   0     4.5
8 '15'    2002    25    NA  6.7     24   4.5   4.5   4.5
9 '38     1987    97    NA  6.6     18   4.5   4.5   4.5
10 '49-'~ 1917    61    NA   6      51   4.5   0     4.5
# ... with 58,778 more rows, and 15 more variables:
#   r4 <dbl>, r5 <dbl>, r6 <dbl>, r7 <dbl>, r8 <dbl>,
#   r9 <dbl>, r10 <dbl>, mpaa <chr>, Action <int>,
#   Animation <int>, Comedy <int>, Drama <int>,
#   Documentary <int>, Romance <int>, Short <int>
```

We'll focus on a random sample of 68 movies that are classified as either "action" or "romance" movies but not both. We disregard movies that are classified as both so that we can assign all 68 movies into either category. Furthermore, since the original `movies` dataset was a little messy, we provided a pre-wrangled version of our data in the `movies_sample` data frame included in the `moderndive` package (you can look at the code to do this data wrangling here⁵):

```
movies_sample
```

```
# A tibble: 68 x 4
  title                      year rating genre
  <chr>                     <int>  <dbl> <chr>
  1 Underworld                1985   3.1 Action
```

⁵https://github.com/moderndive/moderndive/blob/master/data-raw/process_data_sets.R#L14

```

2 Love Affair           1932   6.3 Romance
3 Junglee               1961   6.8 Romance
4 Eversmile, New Jersey 1989    5   Romance
5 Search and Destroy    1979    4   Action
6 Secreto de Romelia, El 1988   4.9 Romance
7 Amants du Pont-Neuf, Les 1991   7.4 Romance
8 Illicit Dreams        1995   3.5 Action
9 Kabhi Kabhie           1976   7.7 Romance
10 Electric Horseman, The 1979   5.8 Romance
# ... with 58 more rows

```

The variables include the `title` and `year` the movie was filmed. Furthermore, we have a numerical variable `rating`, which is the IMDb rating out of 10 stars, and a binary categorical variable `genre` indicating if the movie was an `Action` or `Romance` movie. We are interested in whether `Action` or `Romance` movies got on average a higher `rating`.

Let's perform an exploratory data analysis of this data. Recall from Section 3.7.1 that a boxplot one visualization we can use to show the relationship between a numerical and a categorical variable. Another option you saw in Section 3.6 would be to use a faceted histogram. However, in the interest of brevity let's visualize just the boxplot in Figure 10.17.

```

ggplot(data = movies_sample, aes(x = genre, y = rating)) +
  geom_boxplot() +
  labs(y = "IMDb rating")

```

Eyeballing Figure 10.17, it appears that romance movies have a higher median rating. Do we have reason to believe however, that there is a *significant* difference between the mean `rating` for action movies compared to romance movies? It's hard to say just based on the plots. The boxplot does show that the median sample rating is higher for romance movies. Let's now calculate the number of movies, the mean rating, and the standard deviation split by the binary variable `genre`. We'll do this using `dplyr` data wrangling verbs, in particular the count of each type of movies using the `n()` summary statistic function.

```

movies_sample %>%
  group_by(genre) %>%
  summarize(n = n(), mean_rating = mean(rating), std_dev = sd(rating))

# A tibble: 2 x 4

```

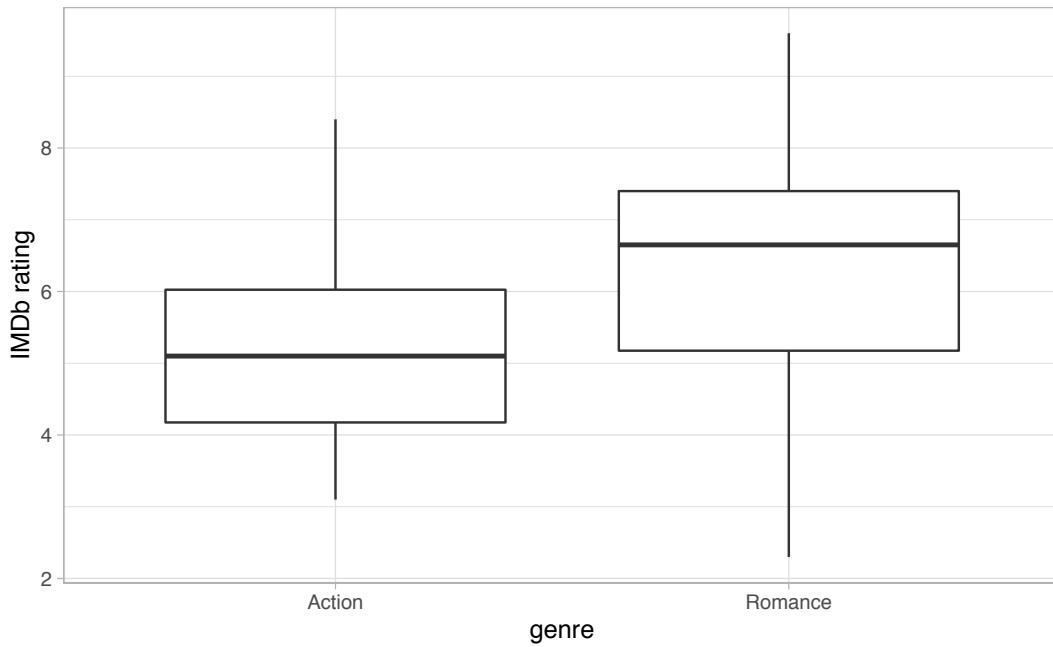


FIGURE 10.17: Boxplot of IMDb rating vs genre.

genre	n	mean_rating	std_dev
		<dbl>	<dbl>
Action	32	5.275	1.36121
Romance	36	6.32222	1.60963

So we have 36 movies with an average rating of 6.32 stars out of 10 and `n_action` movies with a sample mean rating of 5.28 stars out of 10. The difference in these average ratings is thus $6.32 - 5.28 = 1.05$. So there appears to be an edge of 1.05 stars in romance movie ratings. The question is however, are these results indicative of a true difference for all romance and action movies? Or could this difference be attributable to chance and sampling variation?

10.5.2 Sampling scenario

Let's tie things in back with our sampling idea in Chapter 8. Recall our sampling bowl with $N = 2400$ balls. Our population parameter of interest was the population proportion of these balls that were red, denoted mathematically by p . In order to estimate p , we extracted a sample of 50 balls using the shovel and computed the relevant point estimate: the sample proportion of these 50 balls that were red, denoted mathematically by \hat{p} .

What is the study population here? It is all movies in the IMDb database that are either action or romance (but not both). In other words, of all 58,788 in the

`movies` data frame included in the `ggplot2movies` package, our study population are those movie who are either `Action` or `Romance`. What is the sample here? It is the 68 movies included in the `movies_sample` dataset. Since this sample was randomly taken from the population `movies`, it is representative of all romance and action movies, and thus any analysis and results based on `movies_sample` can generalize to the entire population. Recall you studied these ideas in Section 8.3.1.

What are the relevant population parameter and point estimates? We introduce the fourth sampling scenario in Table 10.4.

TABLE 10.4: Scenarios of sampling for inference

Scenario	Population parameter	Notation	Point estimate	Notation.
1	Population proportion	p	Sample proportion	\hat{p}
2	Population mean	μ	Sample mean	\bar{x} or $\hat{\mu}$
3	Difference in population proportions	$p_1 - p_2$	Difference in sample proportions	$\hat{p}_1 - \hat{p}_2$
4	Difference in population means	$\mu_1 - \mu_2$	Difference in sample means	$\bar{x}_1 - \bar{x}_2$

So whereas the sampling bowl exercise in Section 8.1 concerned proportions, the pennies exercise in Section 9.1 concerned means, the case study on whether yawning is contagious in Section 9.6 and the promotions activity in Section 10.1 concerned differences in proportions, we are now concerned with differences in means.

In other words, the population parameter of interest is the difference in population mean ratings $\mu_a - \mu_r$, where μ_a is the mean rating of all action movies on IMDb and similarly μ_r is the mean rating of all romance movies. Thus, the point estimate/sample statistic of interest is the difference in sample means $\bar{x}_a - \bar{x}_r$, where \bar{x}_a is the mean rating of the $n_a = n_{\text{action}}$ movies in our sample and \bar{x}_r is the mean rating of the $n_r = n_{\text{romance}}$ in our sample. Based on our earlier exploratory data analysis, our estimate $\bar{x}_a - \bar{x}_r$ is $5.28 - 6.32 = -1.05$.

So there appears to be a slight difference of -1.05 in favor of romance movies. The question is however, could this difference of -1.05 be merely due to chance and sampling variation? Or are these results indicative of a true difference

in mean ratings for all romance and action movies? To answer this question, we'll use hypothesis testing.

10.5.3 Conducting the hypothesis test

We'll be testing

$$\begin{aligned} H_0 : \mu_a - \mu_r &= 0 \\ \text{vs } H_A : \mu_a - \mu_r &\neq 0 \end{aligned}$$

In other words, the null hypothesis H_0 suggests that both romance and action movies have the same mean rating. This is the “hypothesized universe” we'll *assume* is true. The alternative hypothesis H_A suggests on the other hand that there is a difference. Note that unlike the one-sided alternative we used in the promotions exercise $H_a : p_m - p_f > 0$, we are now considering a two-sided alternative of $H_A : \mu_a - \mu_r \neq 0$.

Furthermore, we'll pre-specify a relatively high significance level $\alpha = 0.2$. By setting this value high, all things being equal there is a higher chance that the p-value will be less than this value, and thus there is a higher chance that we'll reject the null hypothesis H_0 in favor of the alternative hypothesis H_A . In other words, we'll reject the hypothesis that there is no difference in mean ratings for all action and romance movies even if we only have mild evidence.

1. specify variables

We first `specify()` the variables of interest in the `movies_sample` data frame using the `rating ~ genre`. This tells `infer` that the numerical variable `rating` is the outcome variable while the binary categorical variable `genre` is the explanatory variable. Note here however, than unlike when we are interested in proportions, since we are interested in the mean of a numerical variable, we do not need to set the `success` argument.

```
movies_sample %>%
  specify(formula = rating ~ genre)
```

```
Response: rating (numeric)
Explanatory: genre (factor)
# A tibble: 68 x 2
  rating genre
  <dbl> <fct>
1     3.1 Action
```

```

2   6.3 Romance
3   6.8 Romance
4   5   Romance
5   4   Action
6   4.9 Romance
7   7.4 Romance
8   3.5 Action
9   7.7 Romance
10  5.8 Romance
# ... with 58 more rows

```

2. hypothesize the null

We set the null hypothesis $H_0 : \mu_a - \mu_r = 0$ by using the `hypothesize()` function. Since we have two samples, the action and romance movies, we set `null = "independence"` as we described in Section 10.3.

```

movies_sample %>%
  specify(formula = rating ~ genre) %>%
  hypothesize(null = "independence")

```

```

# A tibble: 68 x 2
  rating genre
  <dbl> <fct>
1 3.1   Action
2 6.3   Romance
3 6.8   Romance
4 5     Romance
5 4     Action
6 4.9   Romance
7 7.4   Romance
8 3.5   Action
9 7.7   Romance
10 5.8   Romance
# ... with 58 more rows

```

3. generate replicates

After we have set the null hypothesis, we simulate observations assuming the null hypothesis is true by repeating the shuffling/permuation exercise you performed in Section 10.1. We'll repeat this resampling of `type = "permute"` a total of `reps = 1000` times .

```
movies_sample %>%
  specify(formula = rating ~ genre) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 1000, type = "permute")
```

```
Response: rating (numeric)
Explanatory: genre (factor)
Null Hypothesis: independence
# A tibble: 68,000 × 3
# Groups:   replicate [1,000]
  rating genre   replicate
  <dbl> <fct>     <int>
1 4.4   Action      1
2 5.2   Romance     1
3 7.3   Romance     1
4 4.9   Romance     1
5 4.100 Action      1
6 7.4   Romance     1
7 5     Romance     1
8 5.100 Action      1
9 4.4   Romance     1
10 8    Romance     1
# ... with 67,990 more rows
```

Observe that it is at this point our output differs from the original `movies_sample` data frame. The resulting data frame has 68,000 rows. This is because we performed shuffles/permuations of the 68 values of `genre` 1000 times and thus $68,000 = 1000 \times 68$.

4. calculate summary statistics

Now that we have 1000 replicated “shuffles” assuming the null hypothesis that both `Action` and `Romance` movies on average have the same ratings on IMDb, let’s `calculate()` the appropriate summary statistic for each of our 1000 shuffles. Recall from Section 10.2 that point estimates/summary statistics relating to hypothesis testing have a specific name: *test statistics*. Since the unknown population parameter of interest is the difference in population means $\mu_a - \mu_r$, the test statistic of interest here is the difference in sample means $\bar{x}_a - \bar{x}_r$.

For each of our 1000 shuffles, we can calculate this test statistic by setting `stat = "diff in means"`. Furthermore, since we are interested in $\bar{x}_a - \bar{x}_r$ and not

the reverse-ordered $\bar{x}_r - \bar{x}_a$, we set `order = c("Action", "Romance")`. Let's save the result in a data frame called `null_distribution_movies`:

```
null_distribution_movies <- movies_sample %>%
  specify(formula = rating ~ genre) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 1000, type = "permute") %>%
  calculate(stat = "diff in means", order = c("Action", "Romance"))
null_distribution_movies
```

```
# A tibble: 1,000 x 2
  replicate      stat
  <int>     <dbl>
1       1 -0.923264
2       2  0.363542
3       3  0.404861
4       4  0.463889
5       5 -0.610417
6       6 -0.279861
7       7 -0.262153
8       8 -0.291667
9       9 -0.114583
10      10  0.398958
# ... with 990 more rows
```

Observe that we have 1000 values of `stat`, each representing one “shuffled” instance of $\bar{x}_a - \bar{x}_r$ in a hypothesized world of no difference in movie ratings between romance and action movies.

But wait! What happened in real-life? What was the observed difference in promotions rates? In other words, what was the *observed test statistic* $\bar{x}_a - \bar{x}_r$? Recall our earlier data wrangling from earlier, this observed difference in means was $5.28 - 6.32 = -1.05$. We can also achieve this using the code above but with the `hypothesize()` and `generate()` steps removed. Let's save this in `obs_diff_means`

```
obs_diff_means <- movies_sample %>%
  specify(formula = rating ~ genre) %>%
  calculate(stat = "diff in means", order = c("Action", "Romance"))
obs_diff_means
```

```
# A tibble: 1 x 1
  stat
```

```
<dbl>
1 -1.04722
```

5. visualize the p-value

Lastly to compute the p-value, we have to assess how “extreme” the observed difference in means -1.05 is by comparing it to our null distribution constructed in a hypothesized universe of no difference in movie ratings. Let’s visualize the p-value in Figure 10.18. Unlike our example in Section 10.3.1 involving promotions, since we have a two-sided alternative hypothesis $H_A : \mu_a - \mu_r \neq 0$, we have to allow for both possibilities for “more extreme”, so we set `direction = "both"`.

```
visualize(null_distribution_movies, bins = 10) +
  shade_p_value(obs_stat = obs_diff_means, direction = "both")
```

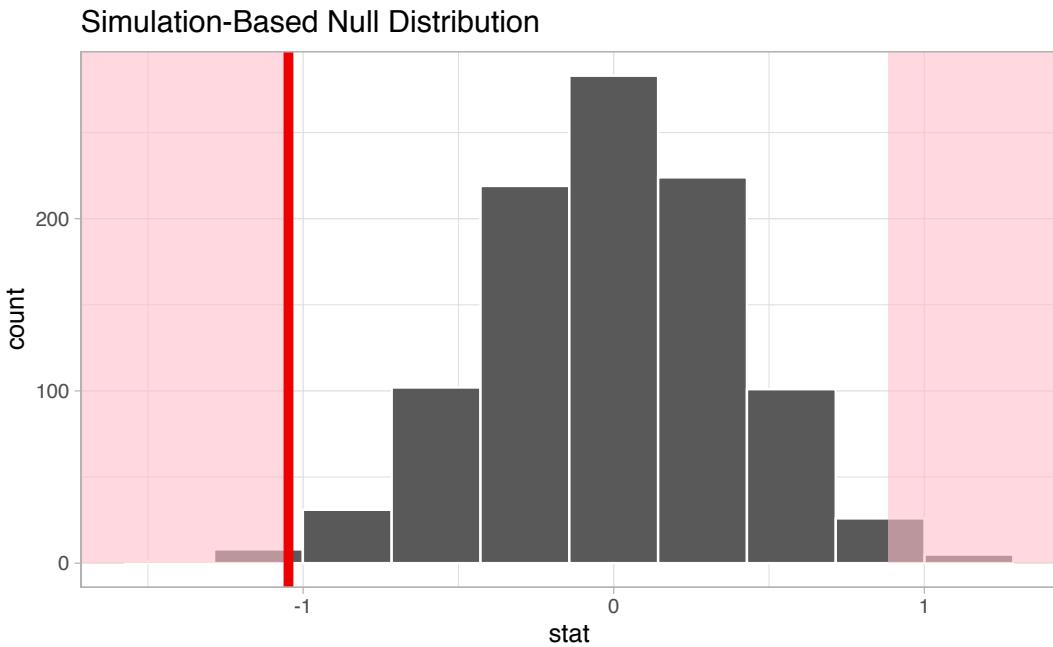


FIGURE 10.18: Null distribution, observed test statistic, and p-value.

Recall the three elements of this plot. First, the histogram is the null distribution, which is the technical term for the sampling distribution of the “shuffled” difference in sample means $\bar{x}_a - \bar{x}_r$ assuming H_0 is true. Second, the solid line is the observed test statistic, or the difference in sample means we observed in real-life of $5.28 - 6.32 = -1.05$. Notice where this solid line is located in the null distribution: it is very plausible to observe such a value. Third, the shaded

area is the p-value, or the probability of obtaining a test statistic just as or more extreme than the observed test statistic *assuming the null hypothesis H_0 is true.*

What proportion of the null distribution is shaded? In other words, what is the numerical value of the p-value? We use the `get_p_value()` function to compute this value:

```
null_distribution_movies %>%  
  get_p_value(obs_stat = obs_diff_means, direction = "both")  
  
# A tibble: 1 × 1  
  p_value  
  <dbl>  
1 0.016
```

This p-value of 0.016 is very small. In other words, there is a small chance that we'd observe a sample with a difference of $5.28 - 6.32 = -1.05$ in a universe where there was truly no difference in ratings. This p-value is in fact much small than our pre-specified α significance level of 0.2, and thus we are very inclined to reject the null hypothesis $H_0 : \mu_a - \mu_r = 0$ in favor of the alternative hypothesis $H_A : \mu_a - \mu_r \neq 0$. In non-statistical language, the conclusion is: the evidence in this sample of data suggests we should reject the hypothesis that there is no difference in mean IMDb ratings between romance and action movies in favor of the hypothesis that there is a difference.

Learning check

(LC10.9) Conduct the same analysis comparing action movies versus romantic movies using the median rating instead of the mean rating. What was different and what was the same?

(LC10.10) What conclusions can you make from viewing the faceted histogram looking at `rating` versus `genre` that you couldn't see when looking at the boxplot?

(LC10.11) Describe in a paragraph how we used Allen Downey's diagram to conclude if a statistical difference existed between mean movie ratings for action and romance movies.

(LC10.12) Why are we relatively confident that the distributions of the sample ratings will be good approximations of the population distributions of ratings for the two genres?

(LC10.13) Using the definition of “*p*-value”, write in words what the *p*-value represents for the hypothesis test above comparing the mean rating of romance to action movies.

(LC10.14) What is the value of the *p*-value for the hypothesis test comparing the mean rating of romance to action movies?

(LC10.15) Do the results of the hypothesis test match up with the original plots we made looking at the population of movies? Why or why not?

10.6 Conclusion

10.6.1 Theory-based hypothesis tests

Much as we did in Section 9.7.2 when we showed you a theory-based method for constructing confidence intervals that involved mathematical formulas, we now present an example of a traditional theory-based method to conduct the hypothesis test to determine if there is a statistically significant difference in the mean rating of Action versus Romance movies. This method relies on probability models, probability distributions, and a few assumptions to construct the null distribution. This is contrast to the approach we used in Section 10.3 where we relied on simulations to construct the null distribution.

These traditional theory-based methods have been used for decades mostly because researchers didn’t have access to computers that could run thousands of simulations quickly and efficiently. Now that computing power is much cheaper and much more accessible, simulation-based methods are much more feasible, however many fields and researchers continue to use theory-based methods. Hence we make it a point to include discussion about them here.

As we’ll show in this section, any theory-based method is ultimately an approximation to the simulation-based method. The theory-based method we’ll focus on is known as the *two-sample t-test* for testing differences in sample means where the test statistic of interest isn’t the difference in sample means $\bar{x}_1 - \bar{x}_2$, but the related two-sample *t*-statistic. The data we’ll use will once again be the `movies_sample` data of action and romance movies where the outcome variable of interest are movies’ IMDb ratings.

Two-sample t-statistic

A common task in statistics is the process of “standardizing a variable.” By standardizing different variables, we can make them more comparable. For example, say you are interested in studying the distribution of temperature recordings from Portland, Oregon, USA with temperature recordings in Montreal, Quebec, Canada. Given that the US temperatures are generally recorded in degrees Fahrenheit and Canadian temperatures are generally recorded in degrees Celsius, how can we make them comparable? On the one hand, we could convert the degrees Fahrenheit into Celsius, or vice versa. On the other, we could convert them both to a common “standardized” scale.

One common method for standardization from probability theory is computing the z -score:

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

where x represent the value of a variable, μ represents the mean of the variable, and σ represents the standard deviation of the variable. You first subtract the mean μ from each value of x and then divide $x - \mu$ by the standard deviation σ . These operations will have the effect of “re-centering” your variable around 0 and “re-scaling” your variable x to have what are known as “standard units.”

Thus, if your variable has 10 elements, each one has a corresponding z -score that gives how many standard deviations away that value is from its mean. z -scores are normally distributed with mean 0 and standard deviation 1. Such a curve is called a “ z -distribution” as well a “standard normal” curve and they have the common, bell-shaped pattern from Figure 10.19.

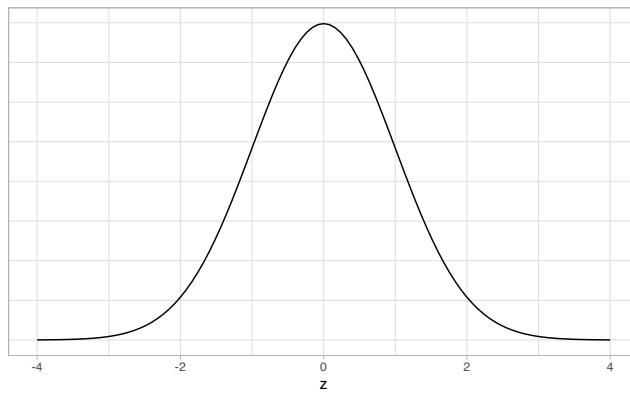


FIGURE 10.19: Standard normal z curve.

Bringing these back to the difference of sample mean ratings $\bar{x}_a - \bar{x}_r$ of action versus romance movies, how would we standardize this variable? By once

again subtracting it's mean and standard deviation. Building on ideas from Section 8.3.3 that: 1) If the sampling was done in representative fashion, then the sampling distribution of $\bar{x}_a - \bar{x}_r$ would be centered at the true population parameter: the difference in population means $\mu_a - \mu_r$. 2) The standard deviation of point estimates like $\bar{x}_a - \bar{x}_r$ have a special name: the standard error

Applying these ideas, we present the two-sample t -statistic:

$$t = \frac{(\bar{x}_a - \bar{x}_r) - (\mu_a - \mu_r)}{\text{SE}_{\bar{x}_a - \bar{x}_r}} = \frac{(\bar{x}_a - \bar{x}_r) - (\mu_a - \mu_r)}{\sqrt{\frac{s_a^2}{n_a} + \frac{s_r^2}{n_r}}}$$

Oofda! There is a lot to try to unpack here! Let's go slowly. In the numerator $\bar{x}_a - \bar{x}_r$ is the difference in sample means while $\mu_a - \mu_r$ is the difference in population means. In the denominator s_a and s_r are the *sample standard deviations* of the action and romance movies in our sample `movies_sample` while n_a and n_r is the sample sizes of the action and romance movies. Putting the above together gives us the standard error $\text{SE}_{\bar{x}_a - \bar{x}_r}$.

Look closely at the formula for $\text{SE}_{\bar{x}_a - \bar{x}_r}$ in the denominator: the sample sizes n_a and n_r are there. So as the sample sizes increase, the standard error goes down. We've seen this concept numerous times now, in particular in our simulations using the three shovels with $n = 25, 50$, and 100 slots in Figure 8.15 and in Section 9.5.3 where we studied the effect of using larger sample sizes on the widths of confidence intervals.

So how can we use the two-sample t -statistic as a test statistic in our hypothesis test? First, assuming the null hypothesis $H_0 : \mu_a - \mu_r = 0$ is true, the right-hand side of the numerator becomes 0. Second, similarly to how the Central Limit Theorem from Section 8.5.2 states that sample means follow a normal distribution, it can be mathematically proven that T follows a t distribution with *degrees of freedom* “roughly equal” to $df = n_a + n_r - 2$. We display three examples of t -distributions in Figure 10.20 along with the standard normal z curve.

Begin by looking at the center of the plot at 0 on the horizontal axis. Note that the bottom curve corresponds to 1 degrees of freedom, the curve above it is for 3 degrees of freedom, the curve above that is for 10 degrees of freedom, and lastly the dashed curve is the standard normal z curve.

Observe that all four curves have a bell shape, are centered at 0, and that as the degrees of freedom increase, the t -distribution resembles the standard normal z curve. The “degrees of freedom” can be thought of measuring how

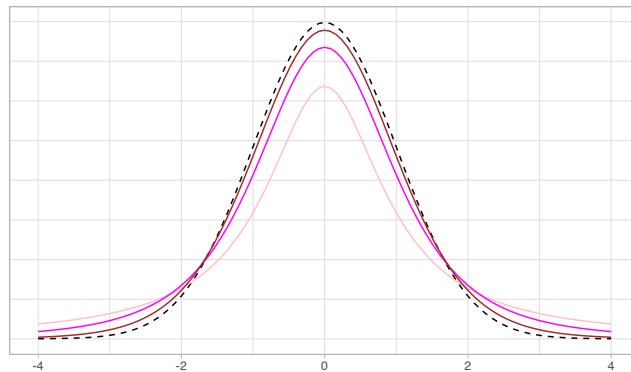


FIGURE 10.20: Examples of t-distributions and the z curve.

different the t distribution will be compared to a normal distribution. The “roughly equal” above indicates that the equation $df = n_a + n_r - 2$ is a “good enough” approximation to the true degrees of freedom. The true formula⁶ is a bit more complicated than this simple expression, but we’ve found the formula to be beyond the scope of this book since it does little to build the intuition of the t -test. The message to remain however is that small sample sizes lead to small degrees of freedom and thus lead to t distributions that are slightly different than the z curve: they have more values in the tails of their distributions. On the other hand, large sample sizes lead to large degrees of freedom and thus lead to t distributions that closely align with the standard normal z -curve.

So, assuming the null hypothesis H_0 is true, our formula for the test statistic simplifies a bit:

$$t = \frac{(\bar{x}_a - \bar{x}_r) - 0}{\sqrt{\frac{s_a^2}{n_a} + \frac{s_r^2}{n_r}}} = \frac{\bar{x}_a - \bar{x}_r}{\sqrt{\frac{s_a^2}{n_a} + \frac{s_r^2}{n_r}}}$$

Recall the summary statistics we computed during our exploratory data analysis in Section 10.5.1.

```
movies_sample %>%
  group_by(genre) %>%
  summarize(n = n(), mean_rating = mean(rating), std_dev = sd(rating))
```

A tibble: 2 x 4

⁶https://en.wikipedia.org/wiki/Student%27s_t-test#Equal_or_unequal_sample_sizes,_unequal_variances

```

genre      n mean_rating std_dev
<chr>    <int>      <dbl>    <dbl>
1 Action     32      5.275   1.36121
2 Romance    36      6.32222  1.60963

```

Using these values, we can show that the observed two-sample t -test statistic is -2.906. Great! How can we compute the p-value using this theory-based test statistic? We need to compare it to a null distribution, which we construct next.

Null distribution

Let's revisit the null distribution for the test statistic $\bar{x}_a - \bar{x}_r$ we constructed in Section 10.5.

```

# Construct null distribution of xbar_a - xbar_r:
null_distribution_movies <- movies_sample %>%
  specify(formula = rating ~ genre) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 1000, type = "permute") %>%
  calculate(stat = "diff in means", order = c("Action", "Romance"))

# Visualize:
visualize(null_distribution_movies, bins = 10)

```

The `infer` package also includes some built-in theory-based test statistics as well. So instead of calculating the test statistic of interest as the "diff in means" $\bar{x}_a - \bar{x}_r$, we can calculate the above defined t -statistic by setting `stat = "t"`:

```

# Construct null distribution of t:
null_distribution_movies_t <- movies_sample %>%
  specify(formula = rating ~ genre) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 1000, type = "permute") %>%
  # Notice we switched stat from "diff in means" to "t"
  calculate(stat = "t", order = c("Action", "Romance"))

# Visualize:
visualize(null_distribution_movies_t, bins = 10)

```

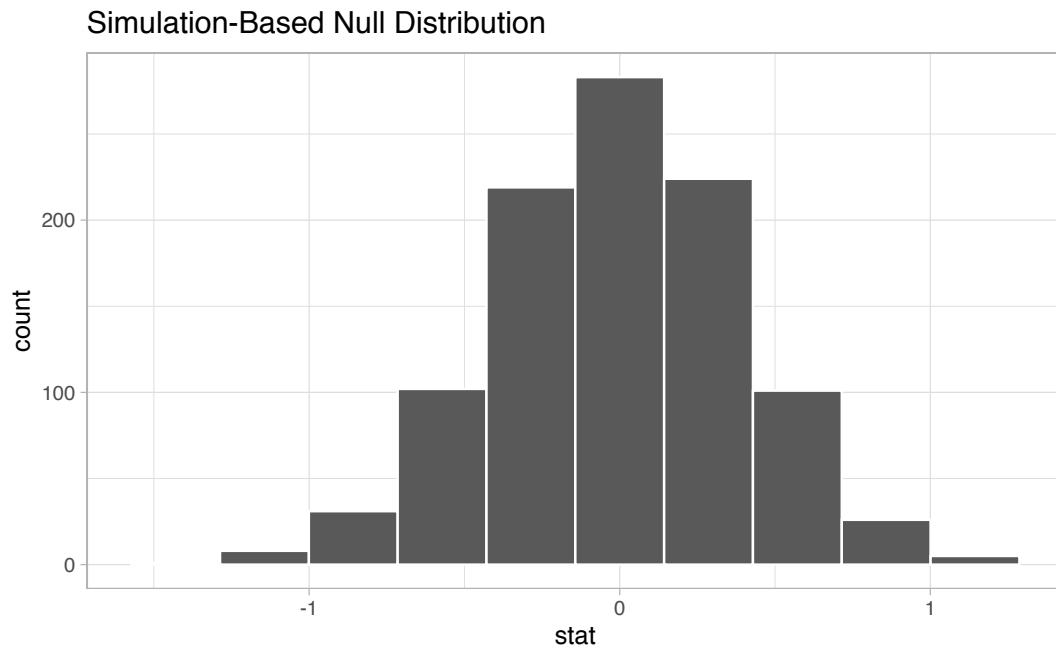


FIGURE 10.21: Null distribution using difference in means.

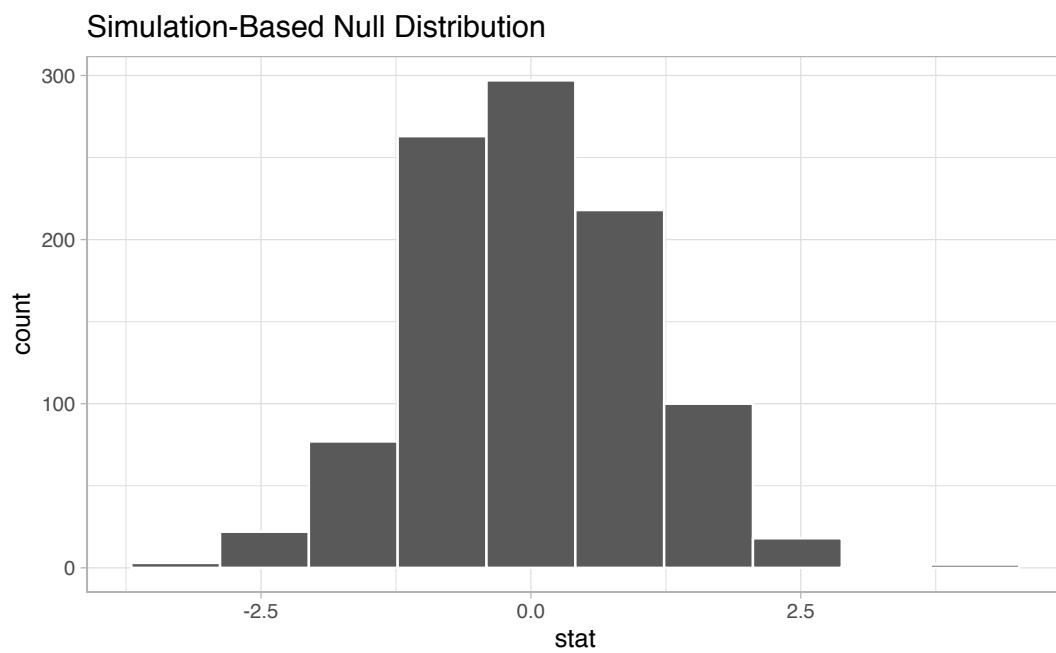


FIGURE 10.22: Null distribution using t-statistic.

Observe that while the shape of the `stat = "diff in means"` null distribution is the similar to `stat = "t"` null distribution, the scale on the x-axis has changed with the t values having less spread than the difference in means. However, a traditional theory-based t -test doesn't look at the simulated histogram in `null_distribution_movies_t`, but instead it looks at the t -distribution curve with degrees of freedom equal to roughly 65.85. This calculation is based on the complicated formula referenced above which we approximated with $df = n_a + n_r - 2 = 32 + 36 - 2 = 66$. We overlay this t -distribution curve over the top of our simulated t -statistics using the `method = "both"` argument in `visualize()`.

```
visualize(null_distribution_movies_t, bins = 10, method = "both")
```

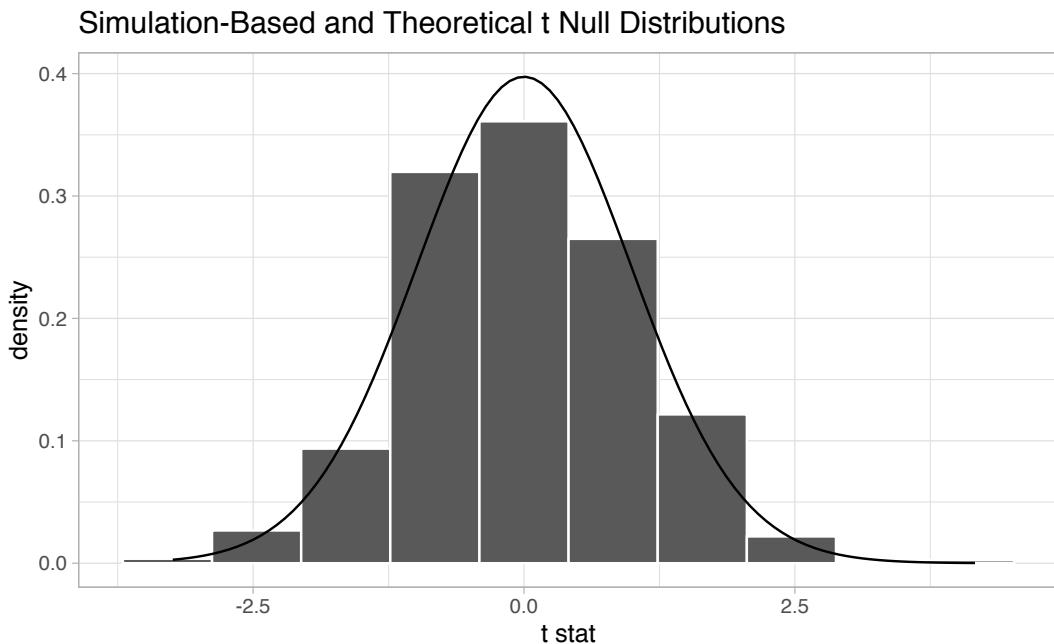


FIGURE 10.23: Null distribution using t-statistic and t-distribution.

Observe that the curve does a good job of approximating the histogram here. To calculate the p -value in this case, we need to figure out how much of the total area under the t -distribution curve is at our observed t -statistic or is “more extreme.” Since our alternative hypothesis $H_A : \mu_a - \mu_r \neq 0$ is a two-sided alternative, we need add up the areas at both tails.

We first compute the observed two-sample t -statistic using `infer` verbs:

```
obs_two_sample_t <- movies_sample %>%
  specify(formula = rating ~ genre) %>%
```

```
calculate(stat = "t", order = c("Action", "Romance"))
obs_two_sample_t
```

```
# A tibble: 1 × 1
  stat
  <dbl>
1 -2.90589
```

So we are interested in finding the percentage of values that are at or above $\text{obs_two_sample_t} = -2.906$ or at or below $-\text{obs_two_sample_t} = 2.906$. We do this using the `shade_p_value()` function with the `direction` argument set to "both":

```
visualize(null_distribution_movies_t, method = "both") +
  shade_p_value(obs_stat = obs_two_sample_t, direction = "both")
```

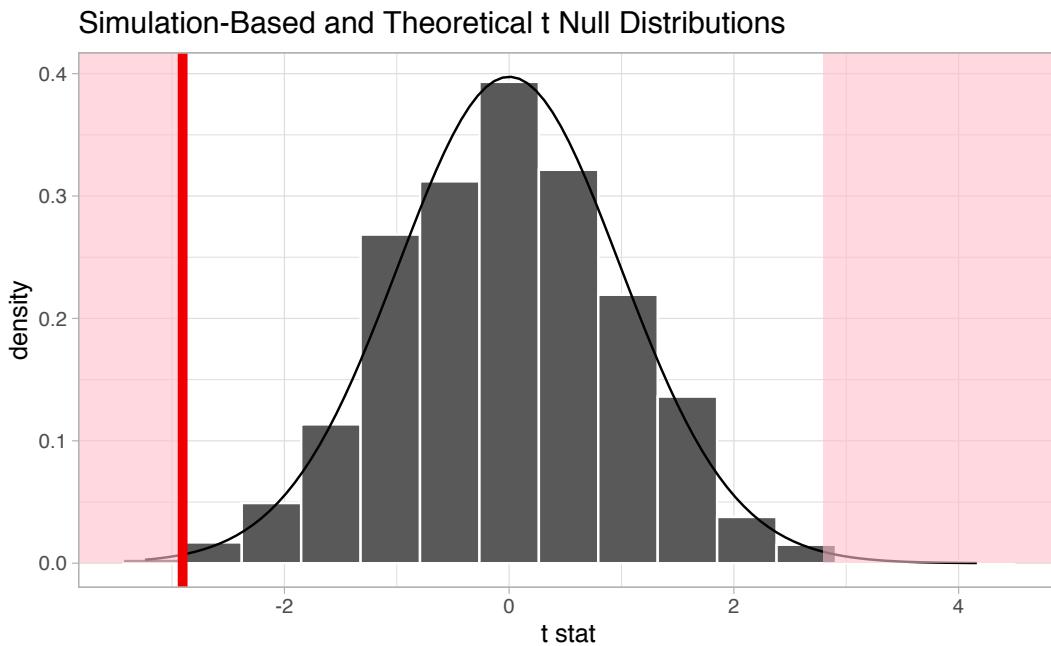


FIGURE 10.24: Null distribution using t-statistic and t-distribution with p-value shaded.

What is the p-value? We apply `get_p_value()` to our null distribution saved in `null_distribution_movies_t`:

```
null_distribution_movies_t %>%
  get_p_value(obs_stat = obs_two_sample_t, direction = "both")
```

```
# A tibble: 1 × 1
  p_value
  <dbl>
1 0.004
```

However, to be able to use the *t*-test and other such theoretical methods, there are always a few conditions to check. The `infer` package does not automatically check these conditions, hence the warning message we received above. These conditions are necessary so that the underlying mathematical theory holds. In order for the results of our two-sample *t*-test to be valid, three conditions must be met:

1. Nearly normal populations or large sample sizes. A general rule of thumb that works in many (but not all) situations is that n should be greater than 30.
2. Both samples are selected independently of each other.
3. All observations are independent from each other.

Let's see if these conditions hold for our `movies_sample` data:

1. This is met since $n_a = 32$ and $n_r = 36$ are both larger than 30, satisfying our rule of thumb.
2. This is met since we sampled the action and romance movies at random and in an unbiased fashion from the database of all IMDb movies.
3. Unfortunately, we don't know how IMDb computes the ratings. For example, if the same person rated multiple movies, then those observations would be related and hence not independent.

Assuming all three conditions are met, we can be reasonably certain that the theory-based *t*-test results are valid. If any of the conditions were not met, we couldn't put as much faith into any conclusions.

10.6.2 When inference is not needed

We've now walked through a several different examples of how to use the `infer` package to perform statistical inference: construct confidence intervals and conduct hypothesis tests. For each of these examples, we made it a point to always perform an exploratory data analysis (EDA) first, specifically using data visualization via `ggplot2` and data wrangling via `dplyr` beforehand. We *highly* encourage you to always do the same. As a beginner to statistics, EDA helps you develop intuition as to what statistical methods like confidence intervals

and hypothesis tests can tell us. Even as a seasoned practitioner of statistics, EDA helps guide your statistical investigations. In particular, is statistical inference even needed?

Let's consider an example. Say we're interested in the following question: Of flights leaving a New York City airport, are Hawaiian Airlines flights in the air for longer than Alaska Airlines flights? Furthermore, let's assume that 2013 flights are a representative sample of all such flights and thus we can use the data in the `flights` data frame in the `nycflights13` package we introduced in Section 2.4. Let's filter this data frame to only consider Hawaiian `HA` and Alaska `AS` using their `carrier` codes as listed in the `airlines` data frame.

```
flights_sample <- flights %>%
  filter(carrier %in% c("HA", "AS"))
```

There are two possible statistical inference methods we could use to answer such questions. First, we could construct a 95% confidence interval for the difference in population means $\mu_{HA} - \mu_{AS}$, where μ_{HA} is the mean air time of all Hawaiian Airlines flights and μ_{AS} is the mean air time of all Alaska Airlines flights. We could then check if the entirety of the interval is greater than 0, suggesting $\mu_{HA} - \mu_{AS} > 0$ i.e. $\mu_{HA} > \mu_{AS}$. Second, we could perform a hypothesis test of the null hypothesis $H_0 : \mu_{HA} - \mu_{AS} = 0$ versus the alternative hypothesis $H_A : \mu_{HA} - \mu_{AS} > 0$.

However, let's first construct an exploratory visualization as we suggest. Since `air_time` is numerical and `carrier` is categorical, a boxplot can display the relationship between these two variables, which we display in Figure 10.25

```
ggplot(data = flights_sample, mapping = aes(x = carrier, y = air_time)) +
  geom_boxplot() +
  labs(x = "Carrier", y = "Air Time")
```

This is what we like to call “you don't need no PhD in statistics” moments. You don't need to be an expert in statistics to know that Alaska Airlines and Hawaiian Airlines have significantly different air times. The two boxes don't even overlap! Constructing a confidence interval or conducting a hypothesis test would frankly not provide much more information.

In our example, why do we observe such a clear cut difference between these two airlines? Let's delve a little deeper using data wrangling. Let's first group by the rows of `flights_sample` by not only `carrier` but also destination `dest`. Subsequently we'll compute two summary statistics: the number of observations using `n()` and the mean airtime:

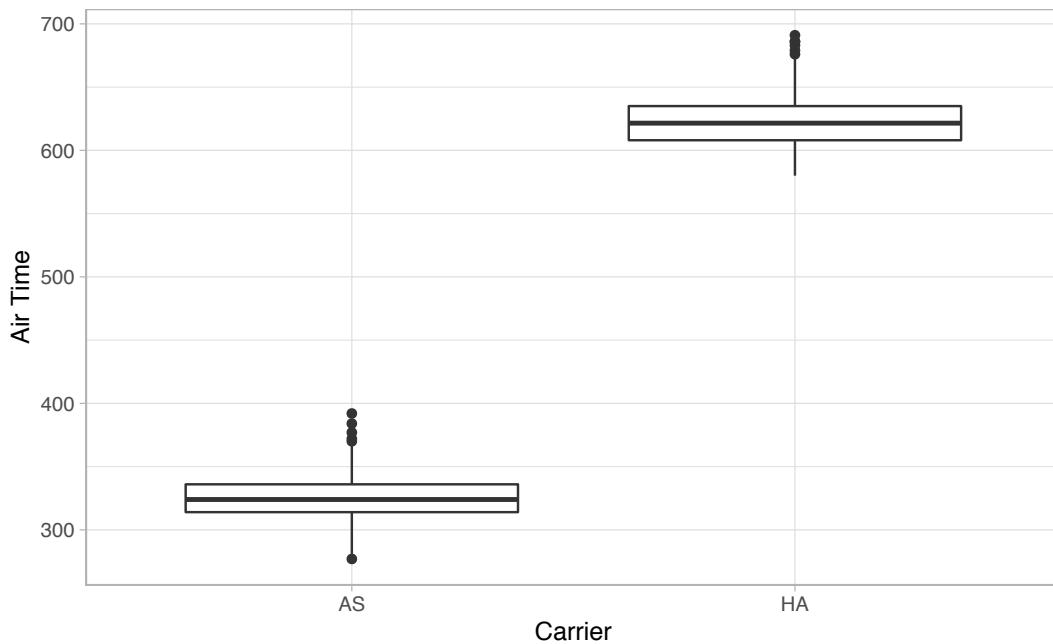


FIGURE 10.25: Air time for Hawaiian and Alaska Airlines flights departing NYC in 2013.

```
flights_sample %>%
  group_by(carrier, dest) %>%
  summarize(n = n(), mean_time = mean(air_time, na.rm =TRUE))
```

```
# A tibble: 2 x 4
# Groups:   carrier [2]
  carrier dest      n  mean_time
  <chr>   <chr> <int>    <dbl>
1 AS      SEA     714    325.618
2 HA      HNL     342    623.088
```

It turns out that Alaska only flies to `SEA` (Seattle) from New York City (NYC) while Hawaiian only flies to `HNL` (Honolulu) from NYC. Given the clear difference in distance from New York City to Seattle versus New York City to Honolulu, it is not surprising that we observe such different air times in flights.

This is a clear example of not needing to do anything more than some simple exploratory data analysis with data visualization and descriptive statistics to get an appropriate inferential conclusion. This is why we highly recommend you perform an EDA of any sample data first before going to the trouble of running statistical methods like confidence intervals and hypothesis testing.

Learning check

(LC10.16) Could we make the same type of immediate conclusion that SFO had a statistically greater `air_time` if, say, its corresponding standard deviation was 200 minutes? What about 100 minutes? Explain.

10.6.3 Problems with p-values

On top of the many common misunderstandings about hypothesis testing and p-values we listed in Section 10.4, another unfortunate consequence of the expanded use of p-values and hypothesis testing is a phenomenon known as “p-hacking.” p-hacking is act of “cherry-picking” only results that are “statistically significant” while dismissing those that aren’t, even if at the expense of the scientific ideas. There are lots of articles and much has been written recently about misunderstandings and the problems with p-values that we encourage readers to check out and to ponder on. Here are just a few:

1. Misunderstandings of *p*-values⁷
2. What a nerdy debate about p-values shows about science - and how to fix it⁸
3. Statisticians issue warning over misuse of *P* values⁹
4. You Can’t Trust What You Read About Nutrition¹⁰
5. A Litany of Problems with p-values¹¹

In fact, the American Statistical Association put out a statement in 2016 titled “The ASA Statement on p-Values: Context, Process, and Purpose”¹² with six principles underlying the proper use and interpretation of the p-value. The ASA released this guidance on p-values to improve the conduct and interpretation of quantitative science and inform the growing emphasis on reproducibility of science research.

We as authors much prefer the use of confidence intervals for statistical in-

⁷https://en.wikipedia.org/wiki/Misunderstandings_of_p-values

⁸<https://www.vox.com/science-and-health/2017/7/31/16021654/p-values-statistical-significance-redefine-0005>

⁹<https://www.nature.com/news/statisticians-issue-warning-over-misuse-of-p-values-1.19503>

¹⁰<https://fivethirtyeight.com/features/you-can-t-trust-what-you-read-about-nutrition/>

¹¹<http://www.fharrell.com/post/pval-litany/>

¹²<https://www.amstat.org/asa/files/pdfs/P-ValueStatement.pdf>

ference, as they are in our opinions much less prone to misinterpretations. However, many fields still use p -values exclusively for statistical inference and thus we still included them in our text. We also encourage to learn more about “p-hacking” and its implication for science.

10.6.4 Additional resources

An R script file of all R code used in this chapter is available here¹³.

If you want to see more examples of the `infer` workflow to construct confidence intervals, we suggest you check out the `infer` package homepage, in particular, a series of example analyses available at <https://infer.netlify.com/articles/>.

10.6.5 What’s to come

We conclude by showing the `infer` pipeline diagram for hypothesis testing.

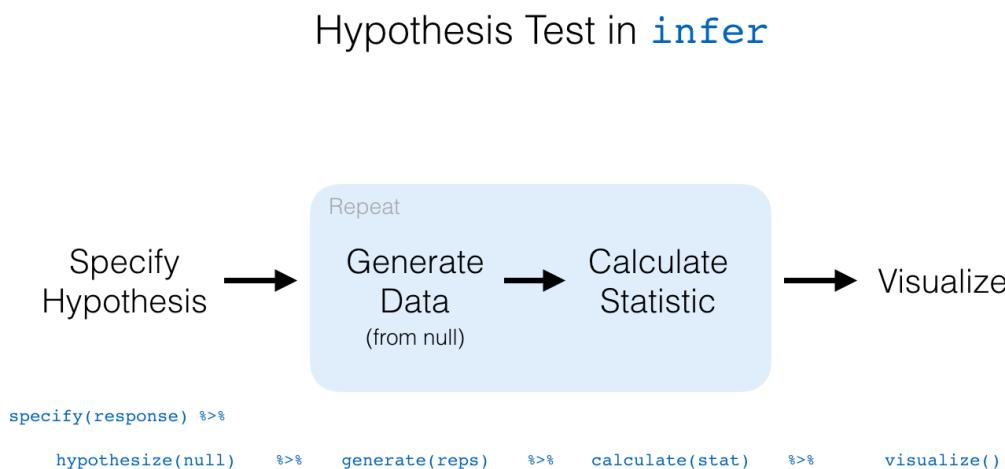


FIGURE 10.26: `infer` package workflow for hypothesis testing.

In Chapter 11, we’ll revisit the regression models we studied in Chapters 6 on basic regression and 7. Armed with our understanding of confidence intervals from Chapter 9 and hypothesis tests from this chapter, we’ll study inference for regression. For example, recall Table 6.2, where we displayed the regression table corresponding to our regression model for teaching score as a function of beauty score.

¹³ [scripts/10-hypothesis-testing.R](#)

```
# Fit regression model:  
score_model <- lm(score ~ bty_avg, data = evals)  
# Get regression table:  
get_regression_table(score_model)
```

TABLE 10.5: Linear regression table

term	estimate	std_error	statistic	p_value	lower_ci	upper_ci
intercept	3.880	0.076	50.96	0	3.731	4.030
bty_avg	0.067	0.016	4.09	0	0.035	0.099

We previously saw in Section 6.1.2 that the values in the `estimate` column are the fitted intercept b_0 and fitted slope for beauty score b_1 . In Chapter 11, we'll unpack the remaining columns: `std_error` which is the standard error, `statistic` which is the observed **standardized** test statistic to compute the `p_value`, and the 95% confidence intervals as given by `lower_ci` and `upper_ci`.

11

Inference for Regression

In our penultimate chapter, we'll revisit regression models we first studied in Chapters 6 and 7. Armed with our knowledge of confidence intervals and hypothesis test from Chapters 9 and 10, we'll be able to apply statistical inference to regression intercepts and slopes.

Needed packages

Let's load all the packages needed for this chapter (this assumes you've already installed them). Recall from our discussion in Section 5.4.1 that loading the `tidyverse` package by running `library(tidyverse)` loads the following commonly used data science packages all at once:

- `ggplot2` for data visualization
- `dplyr` for data wrangling
- `tidyr` for converting data to “tidy” format
- `readr` for importing spreadsheet data into R
- As well as the more advanced `purrr`, `tibble`, `stringr`, and `forcats` packages

If needed, read Section 2.3 for information on how to install and load R packages.

```
library(tidyverse)
library(moderndive)
library(infer)
```

11.1 Regression refresher

Before jumping into inference for regression, let's remind ourselves of the University of Texas student evaluations analysis in Section 6.1.

11.1.1 Teaching evals analysis

Recall using simple linear regression we modeled the relationship between

1. A numerical outcome variable y , the instructor's teaching score and
2. A single numerical explanatory variable x , the instructor's beauty score.

We first created an `evals_ch6` data frame that selected a subset of variables from the `evals` data frame included in the `moderndive` package. This `evals_ch6` data frame contains only the variables of interest for our analysis, in particular the instructor's teaching `score` and the beauty rating `bty_avg`:

```
evals_ch6 <- evals %>%
  select(ID, score, bty_avg, age)
glimpse(evals_ch6)
```

```
Observations: 463
Variables: 4
$ ID      <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ...
$ score   <dbl> 4.7, 4.1, 3.9, 4.8, 4.6, 4.3, 2.8, 4.1, ...
$ bty_avg <dbl> 5.00, 5.00, 5.00, 5.00, 3.00, 3.00, 3.0...
$ age     <int> 36, 36, 36, 36, 59, 59, 51, 51, 40, ...
```

In Section 6.1.1, we performed an exploratory data analysis of the relationship between these two variables. We saw there that there was a weakly positive correlation of 0.187 between the two variables. This was evidenced in Figure 11.1 of the scatterplot along with the “best fitting” regression line that summarizes the linear relationship between the two variables.

```
ggplot(evals_ch6, aes(x = bty_avg, y = score)) +
  geom_point() +
  labs(x = "Beauty Score", y = "Teaching Score",
       title = "Relationship between teaching and beauty scores") +
  geom_smooth(method = "lm", se = FALSE)
```

Looking at this plot again, you might be asking “But is that line really that positive in its slope?” It does increase from left to right as the `bty_avg` variable increases, but by how much? To get to this information, we used the `lm()` function to fit a regression model, which took a formula `y ~ x = score ~ bty_avg` as an input.

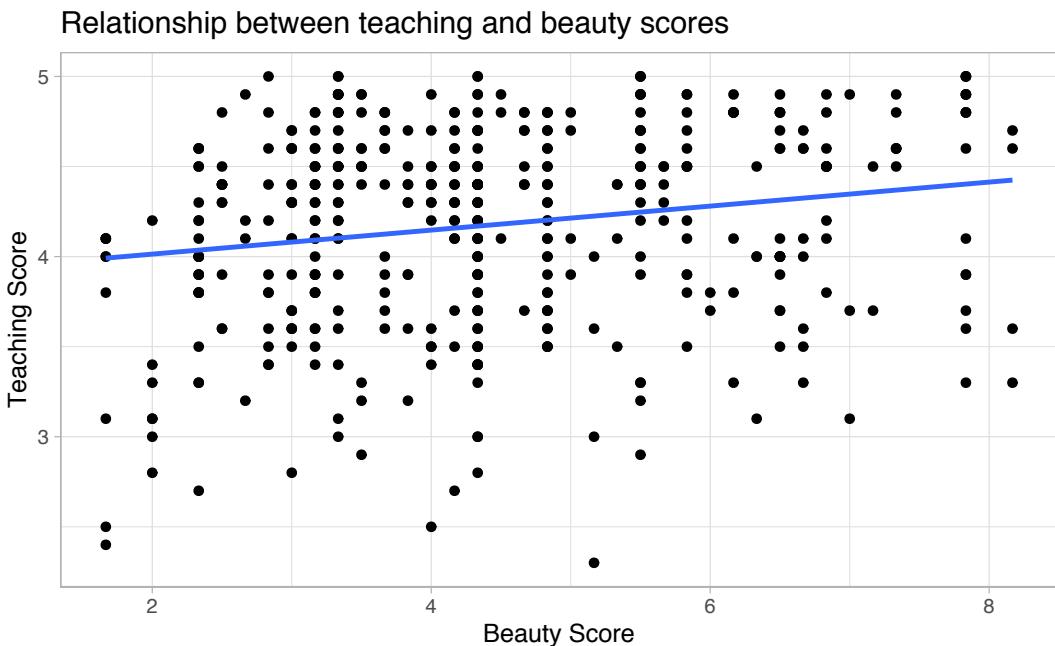


FIGURE 11.1: Relationship with regression line.

1. We first “fit” the linear regression model using the `lm()` function and save it in `score_model`.
2. We get the regression table by applying the `get_regression_table()` from the `moderndive` package to `score_model`.

```
# Fit regression model:
score_model <- lm(score ~ bty_avg, data = evals_ch6)
# Get regression table:
get_regression_table(score_model)
```

TABLE 11.1: Previously seen linear regression table.

term	estimate	std_error	statistic	p_value	lower_ci	upper_ci
intercept	3.880	0.076	50.96	0	3.731	4.030
bty_avg	0.067	0.016	4.09	0	0.035	0.099

Using the values in the `estimate` column of the regression table in Table 11.1, we could then obtain the equation of the “best-fitting” regression line in blue in Figure 11.1 where b_0 is the fitted intercept and b_1 is the fitted slope for `bty_avg`.

$$\begin{aligned}\hat{y} &= b_0 + b_1 \cdot x \\ \widehat{\text{score}} &= b_0 + b_{\text{bty_avg}} \cdot \text{bty_avg} \\ &= 3.880 + 0.067 \cdot \text{bty_avg}\end{aligned}$$

Recall the interpretation of the $b_1 = 0.067$ value of the fitted slope:

For every increase of one unit in beauty rating, there is an associated increase, on average, of 0.067 units of evaluation score.

Thus, the slope value quantifies the relationship between the y variable of `score` and the x variable of `bty_avg`. We also discussed the intercept value of $b_0 = 3.88$ and its lack of practical interpretation since the range of possible beauty scores does not include 0.

11.1.2 Sampling scenario

Now let's view the instructors for these 463 courses in as a representative sample from a greater population as we defined in Section 8.3.1. Perhaps these instructors can be viewed as a representative sample of all instructors in the University of Texas system, not just UT Austin? Or perhaps all Texas university and college level instructors? Or all instructors in the United States? For our purposes, since these data were collected between 2000-2002, we'll view the `evals_ch6` as a representative sample of courses between 1995 and 2002.

Since we are viewing these $n = 463$ courses as a sample, we can view our fitted slope $b_1 = 0.067$ as a point estimate of a population slope β_1 , in other words the slope quantifying the relationship between teaching `score` and beauty average `bty_avg` for *all* instructors in our population. Similarly, we can view our fitted intercept $b_0 = 3.88$ as a point estimate of a population intercept β_0 for *all* instructors in our population. Putting these two together, we can view the equation of the fitted line $\hat{y} = b_0 + b_1 \cdot x = 3.880 + 0.067 \cdot \text{bty_avg}$ as an estimate of some true and unknown population line $y = \beta_0 + \beta_1 \cdot x$.

Thus we can draw parallels between our teaching `evals` analysis and all our previously seen sampling scenarios. In this chapter, we'll focus on the final two scenarios of Table 8.8: regression slopes and regression intercepts.

TABLE 11.2: Scenarios of sampling for inference

Scenario	Population parameter	Notation	Point estimate	Notation.
1	Population proportion	p	Sample proportion	\hat{p}
2	Population mean	μ	Sample mean	\bar{x} or $\hat{\mu}$
3	Difference in population proportions	$p_1 - p_2$	Difference in sample proportions	$\hat{p}_1 - \hat{p}_2$
4	Difference in population means	$\mu_1 - \mu_2$	Difference in sample means	$\bar{x}_1 - \bar{x}_2$
5	Population regression slope	β_1	Fitted regression slope	b_1 or $\hat{\beta}_1$
6	Population regression intercept	β_0	Fitted regression intercept	b_0 or $\hat{\beta}_0$

Since we are now viewing our fitted slope b_1 and fitted intercept b_0 as estimates based on a sample, these estimates will be subject to *sampling variability* as we've seen numerous times throughout this book. In other words, if we collected new sample of data on a different set of $n = 463$ courses and their instructors, the new fitted slope b_1 will very likely differ from 0.067. The same goes for the new fitted intercept b_0 .

But by how much will they differ? In other words, by how much will these estimates *vary*? This information is contained in remaining columns in Table 11.1. Our knowledge about sampling from Chapter 8, confidence intervals from Chapter 9, and hypothesis tests from Chapter 10 will help us interpret these remaining columns.

11.2 Interpreting regression tables

Both in Chapter 6 and in our regression refresher earlier, we focused only on the two left-most columns the regression table in Table 11.1: `term` and `estimate`. Let's now shift to focus on the remaining columns of `std_error`, `statis-`

`tic`, `p_value`, and the two columns related to confidence intervals `lower_ci` and `upper_ci` that were not discussed previously. Given the lack of practical interpretation for the fitted intercept b_0 , we'll focus only on the second row of the table corresponding to the fitted slope b_1 .

TABLE 11.3: Previously seen regression table.

term	estimate	std_error	statistic	p_value	lower_ci	upper_ci
intercept	3.880	0.076	50.96	0	3.731	4.030
bty_avg	0.067	0.016	4.09	0	0.035	0.099

11.2.1 Standard error

The third column of the regression table in Table 11.1 `std_error` corresponds to the *standard error* of our estimates. Recall the definition of **standard error** we saw in Section 8.3.2:

The standard error is the standard deviation of any point estimate from a sampling scenario.

So what does this mean in terms of the fitted slope $b_1 = 0.067$? This value is just one possible value of the fitted slope resulting from this particular sample of $n = 463$ pairs of `score` and `bty_avg`. However, if we collected a different sample of $n = 463$ pairs of `bty_avg` and `score`, we will almost certainly obtain a different fitted slope b_1 . This is due to sampling variability, as we studied in Chapter 8.

Say we hypothetically collected 1000 such samples and plotted the histogram of the 1000 resulting values of the fitted slope b_1 , this would be a visualization of the *sampling distribution of b_1* . Recall we introduced the sampling distribution for the sample proportion \hat{p} of a shovel's balls that are red in Section 8.3.2. Further recall that the standard deviation of the sampling distribution of b_1 has a special name: the standard error.

This value quantifies how much variation in the fitted slope b_1 one would expect from sample-to-sample. So in our case, we can expect about 0.016 units of deviation on the `bty_avg` slope variable. Recall that `estimate` and `std_error` play a key role in helping us to make an educated guess about possible values for the unknown population slope β_1 relating to *all* instructors in our population

11.2.2 Test statistic

The fourth column of the regression table in Table 11.1 **statistic** corresponds to a *test statistic* relating to the following *hypothesis test*:

$$\begin{aligned} H_0 &: \beta_1 = 0 \\ \text{vs } H_A &: \beta_1 \neq 0 \end{aligned}$$

Recall our definitions of hypothesis tests and test statistics we introduced in Section 10.2:

A hypothesis test consists of a test between two competing hypotheses: 1) a **null hypothesis** H_0 versus 2) an **alternative hypothesis** H_A .

A test statistic is point estimate/sample statistic formula used for hypothesis testing.

Here our null hypothesis H_0 assumes that the population slope β_1 of the relationship between teaching and beauty score for *all* instructors in our population is 0. If the population slope β_1 is truly 0, then this is equivalent to saying that there is *no relationship* between teaching and beauty score. In other words, x = beauty score would have no associated effect on y = teaching score. The alternative hypothesis H_A on the other hand assumes that population slope β_1 is not 0, thus meaning it could be either positive or negative. Recall we called such alternative hypotheses two-sided. By convention, all hypothesis testing for regression assumes two-sided alternatives.

So much in the same vein as the “hypothesized universe” of no gender discrimination we assumed in our `promotions` activity in Section 10.1, when conducting this hypothesis test we’ll assume a “hypothesized universe” where there is no relationship between teaching and beauty scores. In other words, we’ll assume the null hypothesis $H_0 : \beta_1 = 0$ is true.

The **statistic** column in the regression table is a tricky one however. It corresponds to a “standardized statistic” as we saw in Subsection 10.6.1 where we computed a *two-sample t statistic* where the null distribution is a *t*-distribution. This is a tricky statistic for individuals new to statistical inference to study, so we’ll jump into interpreting the p-value in the next section. If you’re curious however, we’ve included a discussion of how this value is computed in Section 11.2.5.

11.2.3 p-value

The fifth column of the regression table in Table 11.1 `p-value` corresponds to the *p-value* of the above hypothesis test $H_0 : \beta_1 = 0$ versus $H_A : \beta_1 \neq 0$. Recall our definition of a p-value we introduced in Section 10.2:

A p-value is the probability of obtaining a test statistic just as extreme or more extreme than the observed test statistic *assuming the null hypothesis H_0 is true*

You can intuitively think of the p-value as quantifying how extreme the observed fitted slope of $b_1 = 0.067$ is in a “hypothesized universe” where there is no relationship between teaching and beauty scores. Since in this case the p-value is 0, following the hypothesis testing procedure we outlined in Section 10.4, for any choice of significance level α we would reject H_0 in favor of H_A . Using non-statistical language, this is saying: we reject the hypothesis that there is no relationship between teaching and beauty scores in favor of the hypothesis that there is. In other words, the evidence suggests there is a significant relationship, one that is positive.

More precisely however, the p-value corresponds to how extreme the observed test statistic of 4.09 is when compared to the appropriate null distribution. We’ll perform a simulation using the `infer` package in Section 11.4. An extra caveat here is that this hypothesis test is only valid if certain “conditions for inference for regression” are met, which we’ll introduce shortly in Section 11.3.

11.2.4 Confidence interval

The two right-most columns of the regression table in Table 11.1 `lower_ci` and `upper_ci` correspond to the endpoints of the 95% confidence interval for the population slope β_1 . Recall our analogy of “nets are to fish” what “confidence intervals are to population parameters” from Section 9.3. The resulting 95% confidence interval for β_1 of (0.035, 0.099) is a range of plausible values for the population slope β_1 of the linear relationship between teaching and beauty score.

As we discussed in Section 9.5.2 on the precise and shorthand interpretation of confidence intervals, the statistically precise interpretation of this confidence interval is: “if we repeated this sampling procedure a large number of times, we

expect about 95% of the resulting confidence intervals to capture the value of the population slope β_1 .” However, we’ll summarize this using our shorthand interpretation that “we’re 95%”confident” that the true population slope β_1 lies between 0.035 and 0.099.”

Notice in this case that the resulting 95% confidence interval for β_1 of (0.035, 0.099) does not contain a very particular value: $\beta_1 = 0$. Recall from the earlier Subsection 11.2.2 that if the population regression slope β_1 is 0, this is equivalent to saying there is no relationship between teaching and beauty scores. Since $\beta_1 = 0$ is not in our plausible range of values for β_1 , we are inclined to believe that there is in fact a relationship between teaching and beauty scores.

So in this case, the inferential conclusion about the population slope β_1 obtained from the 95% confidence interval matches the conclusion reached from the hypothesis test above: the evidence suggests there is a meaningful relationship between teaching and beauty scores!

Recall however from Subsection 9.5.3 that the confidence level is one the many factors that determine confidence interval widths. So for example, say we used a higher confidence level of 99%, the resulting confidence intervals would be wider, and thus the resulting 99% confidence interval for β_1 might include 0. The lesson is to remember that any confidence interval based conclusion depends highly on the confidence level used.

What are the calculations that went into computing the two endpoints of the 95% confidence interval for β_1 ? Recall our sampling bowl example from Section 9.7.2. Since the sampling/bootstrap distribution of the sample proportion \hat{p} was bell-shaped, we could use the rule of thumb for bell-shaped distributions to create a 95% confidence interval for p with the following equation:

$$\hat{p} \pm \text{MoE}_{\hat{p}} = \hat{p} \pm 1.96 \cdot \text{SE}_{\hat{p}} = \hat{p} \pm 1.96 \cdot \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$

We can generalize this to other point estimates that have bell-shaped sampling/bootstrap distributions:

$$\text{point estimate} \pm \text{MoE} = \text{point estimate} \pm 1.96 \cdot \text{SE}$$

We’ll show in Section 11.4 that the sampling/bootstrap distribution for the fitted slope b_1 is in fact bell-shaped as well. Thus we can construct a 95% confidence interval for β_1 with the following equation:

$$b_1 \pm \text{MoE}_{b_1} = b_1 \pm 1.96 \cdot \text{SE}_{b_1}$$

What are the values of the standard error SE_{b_1} ? Recall from the earlier Sub-section 11.2.1 that they are in fact in the third column of the regression table. Thus the

$$\begin{aligned} b_1 \pm 1.96 \cdot \text{SE}_{b_1} &= 0.067 \pm 1.96 \cdot 0.016 = 0.067 \pm 0.031 \\ &= (0.036, 0.098) \end{aligned}$$

Much like hypothesis tests however, this confidence interval also only yields valid if the “conditions for inference for regression” discussed in Section 11.3 are met.

11.2.5 How does R compute the table?

Since we didn’t do any bootstrapping or simulation-based procedures to get the values of the standard error, test statistic, p-value, and endpoints of the 95% confidence interval in Table 11.1, you might be wondering how were these values computed? What did R do behind the scenes? Does R run simulations like we’ve been doing using the `rep_sample_n()` function and the `infer` package as we’ve been doing in Chapters 8 on sampling, 9 on confidence intervals, and 10?

The answer is no! Much like the theory-based method for constructing confidence intervals you saw in Section 9.7.2 and the theory-based hypothesis test you saw in Section 10.6.1, there are actually mathematical formulas that allow you to construct confidence intervals and conduct hypothesis test for inference for regression. These formulas were derived in a time when computers didn’t exist, so running extensive simulations as we’ve been doing in this book would’ve been impossible.

In particular there is a formula for the standard error of the fitted slope b_1 :

$$\text{SE}_{b_1} = \frac{\frac{s_y}{s_x} \cdot \sqrt{1 - r^2}}{\sqrt{n - 2}}$$

As with many formulas in statistics, there’s a lot going on here so let’s first break down what each symbol represents: 1) s_x and s_y are the sample standard deviations for the explanatory variable `bty_avg` and the response variable `score` respectively. 2) r is the sample correlation coefficient between `score` and `bty_avg`. This was computed as 0.187 in Chapter 6. 3) n is the number of pairs of points in the `evals_ch6` data frame, here 463.

To put the relationship into words, the standard error of b_1 depends on the

relationship between how the response variable varies and the explanatory variable varies in the s_y/s_x term. Next it looks into the relationship of how the two variables relate to each other in the $\sqrt{1 - r^2}$ term.

However, the most important observation to make in the above formula is that there is a $n - 2$ in the denominator. In other words, as the sample size n increases, the standard error SE_{b_1} decreases. Just as we demonstrated in Section 8.3.3 when we used shovels with $n = 25, 50$, and 100 , the amount of sample-to-sample variation in the fitted slope b_1 will depend on the sample size n . In particular, as the sample size increases, the sampling/bootstrap distribution narrows i.e. the standard error SE_{b_1} goes down. Hence our estimates b_1 of the true population slope β_1 get more and more precise.

R then uses this formula for the standard error of b_1 to fill in the third column of the regression table and subsequently to construct 95% confidence intervals. What about the hypothesis test? Much like in our theory-based hypothesis test in Section 10.6.1, R uses the following t -statistic as the test statistic for hypothesis testing:

$$t = \frac{b_1 - \beta_1}{\text{SE}_{b_1}}$$

And since the null hypothesis $H_0 : \beta_1 = 0$ is assumed during the hypothesis test, the t -statistic becomes

$$t = \frac{b_1 - 0}{\text{SE}_{b_1}} = \frac{b_1}{\text{SE}_{b_1}}$$

What are the values of b_1 and SE_{b_1} ? They are in the second and third column of the regression table in Table 11.1. Thus the value of 4.09 in the table is computed as $0.067/0.016 = 4.188$. Note there is a slight difference due to rounding error.

Lastly, to compute the p-value, you need to compare to observed test statistic of 4.09 to the appropriate null distribution: the sampling distribution of the above t statistic assuming that H_0 is true. Much like in Section 10.6.1, it can be mathematically proven that this distribution is a t -distribution with degrees of freedom equal to $df = n-2 = 463-2 = 461$.

Don't worry if you're feeling a little overwhelmed at this point. There is a lot background theory to understand before you can fully make sense of the equations for theory-based methods. That being said, theory-based methods and simulation-based methods for constructing confidence intervals and conducting hypothesis tests often yield consistent results.

In our opinion, a large benefit of simulation-based methods over theory-based is that they are easier for people new to statistical inference to understand. We'll replicate the above analysis using a simulation-based approach with the `infer` package in Section 11.4. In particular, we'll convince you that the sampling/bootstrap distribution of the fitted slope b_1 is indeed bell-shaped.

11.3 Conditions for inference for regression

Similarly to Section 9.3.2 where we could only use the standard-error based method for constructing confidence intervals if the bootstrap distribution was bell shaped, there are certain conditions that need to be met in order for our hypothesis tests and confidence intervals we described in Section 11.2 above to have valid meaning. These conditions must be met for the underlying mathematical and probability theory to work.

For inference for regression, there are four conditions that need to be met. Note the first four letters of these conditions as highlighted in bold below: **L****I****N****E**. This can serve as a nice reminder of what to check whenever linear regression is performed.

1. Linearity of relationship between variables
2. Independence of residuals
3. Normality of residuals
4. Equality of variance

Conditions **L**, **N**, and **E** can be verified through what is known as a **residual analysis**. Condition **I** can only be verified through an understanding of how the data was collected. Before we start verifying these conditions, let's go over a refresher of the concept of residuals.

11.3.1 Residuals refresher

Recall our definition of a residual from Section 6.1.3: it is the observed value minus the fitted value $y - \hat{y}$. Recall that residuals can be thought of as the error or the “lack-of-fit” between the observed value y and the fitted value \hat{y} on the blue regression line in Figure 11.1. We illustrate one particular residual out of 463 in Figure 11.2.

Furthermore, we can automate the calculation of all $n = 463$ residuals by

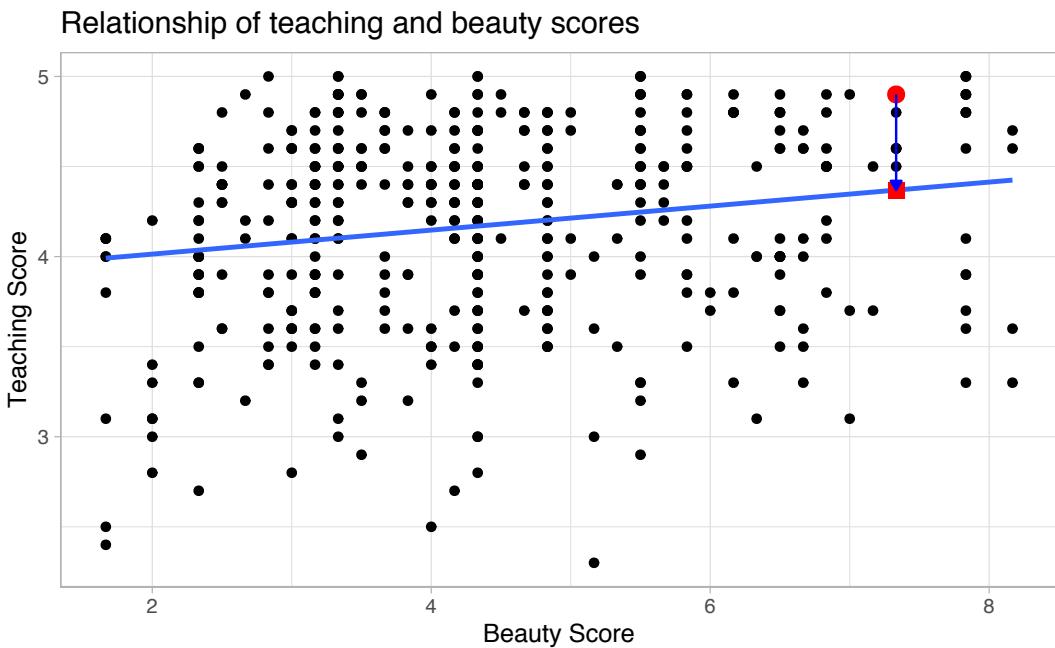


FIGURE 11.2: Example of observed value, fitted value, and residual.

applying the `get_regression_points()` function to `score_model`, which has our regression model saved in it. Observe how the resulting values of `residual` are roughly equal to `score - score_hat` (there is a slight difference due to rounding error).

```
# Fit regression model:
score_model <- lm(score ~ bty_avg, data = evals_ch6)
# Get regression points:
regression_points <- get_regression_points(score_model)
regression_points
```

```
# A tibble: 463 x 5
  ID score bty_avg score_hat residual
  <int> <dbl>    <dbl>     <dbl>     <dbl>
1     1 4.7      5       4.214    0.486
2     2 4.100    5       4.214   -0.114
3     3 3.9      5       4.214   -0.314
4     4 4.8      5       4.214    0.586
5     5 4.600    3       4.08     0.52
6     6 4.3      3       4.08     0.22
7     7 2.8      3       4.08   -1.28
8     8 4.100    3.333   4.102   -0.002
```

```

9      9 3.4   3.333      4.102 -0.702
10     10 4.5   3.16700    4.091  0.40900
# ... with 453 more rows

```

Ideally when we fit a regression model, we'd like there to be *no systematic pattern* to these residuals. We'll be more specific as to what we mean by *no systematic pattern* when we see Figure 11.2, but let's keep this notion imprecise for now. This residual analysis to verify conditions 1, 3, and 4 can be performed via appropriate data visualizations. While there are more sophisticated statistical approaches that can also be done, we'll will focus on the simpler approaches of evaluating plots.

11.3.2 Linearity of relationship

The first condition is that the relationship between the outcome variable y and the explanatory variable x must be **Linear**. Recall the scatterplot in Figure 11.1. We had the explanatory variable beauty score x on the x-axis and the outcome variable teaching score y on the y-axis. Would you say that the relationship between x and y is linear? It's hard to say, because of the scatter of the points about the line. Recall that the correlation coefficient between teaching and beauty score was 0.187, indicating a weakly positive linear relationship. In the authors' opinions, we feel this relationship is "linear enough".

Let's present an example where the relationship between x and y is very clearly not linear in Figure 11.3. In this case, the points clearly do not form a line, but rather a U-shaped polynomial line. In this case, any inference for regression would not be valid.

11.3.3 Independence of residuals

The second condition is that the residuals must be **Independent**. In other words, the different observations in our data must be independent of one another.

For our UT Austin data, while there is data on 463 courses, there are only 94 unique instructors. In other words, the same professor can be included more than once in our data. The original `evals` data frame that we used to construct the `evals_ch6` data frame has a variable `prof_ID` of an anonymized identification variable for the professor:

```

evals %>%
  select(ID, prof_ID, score, bty_avg)

```

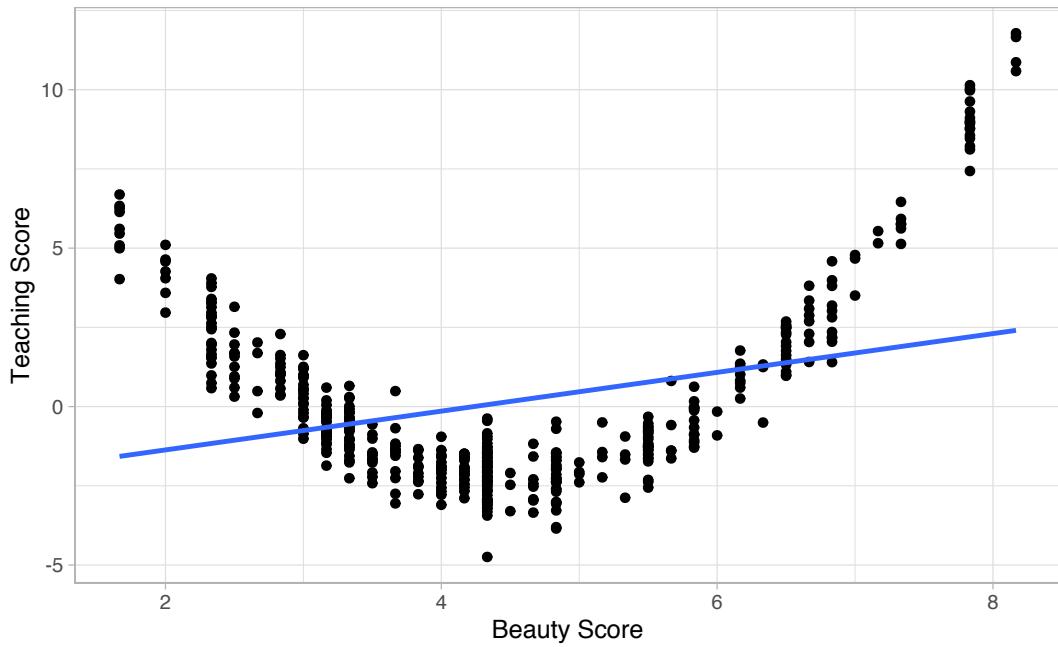


FIGURE 11.3: Example of clearly non-linear relationship.

```
# A tibble: 463 x 4
  ID prof_ID score bty_avg
  <int> <int> <dbl>   <dbl>
1     1      1  4.7    5
2     2      1 4.100   5
3     3      1  3.9    5
4     4      1  4.8    5
5     5      2 4.600   3
6     6      2  4.3    3
7     7      2  2.8    3
8     8      3 4.100  3.333
9     9      3  3.4    3.333
10    10     4  4.5   3.16700
# ... with 453 more rows
```

For example, the professor with ID equal to 1 taught the first 4 courses in the data, the professor with ID equal to 2 taught the next 3, and so on. Given that the same professor taught the first four courses, it is reasonable to expect that the teaching `score` for these courses are related. If a professor gets a high `score` in one class, chances are fairly good they'll get a high `score` in another. This dataset thus provides different information than if we had 463 unique instructors teaching the 463 courses.

In this case we say there exists *dependence* between observations. The first four courses taught by professor 1 are dependent, the next 3 courses taught by professor 2 are related, and so on. Any proper analysis of this data needs to take into account that we have *repeated measures* for the same profs.

So in this case, the independence conditions is not met. What does this mean for our analysis? We'll address this in Subsection 11.3.6 below, after we check the remaining two conditions.

11.3.4 Normality of residuals

Let's now get a little more precise in our definition of *no systematic pattern* in the residuals. The third condition is that the residuals should follow a **Normal distribution**. Furthermore, the center of this distribution should be 0. In other words, sometimes the regression model will make positive errors: $y - \hat{y} > 0$. Other times, the regression model will make equally negative errors: $y - \hat{y} < 0$. However, *on average* the errors should be 0.

The simplest way to check for this condition of the normality of the residuals is to look at a histogram, which we visualize in Figure 11.4.

```
ggplot(regression_points, aes(x = residual)) +
  geom_histogram(binwidth = 0.25, color = "white") +
  labs(x = "Residual")
```

This histogram seems to indicate that we have more positive residuals than negative. Since the residual $y - \hat{y}$ is positive when $y > \hat{y}$, it seems our regression model's fitted teaching scores \hat{y} tend to *underestimate* the true teaching scores y . This histogram has a slight *left-skew* in that there is a tail on the left. Another way to say this is this data exhibits a *negative skew*. Is this a problem? Again, there is a certain amount of subjectivity in the response. In the authors' opinion, while there is a slight skew or pattern to the residuals, it isn't drastic. On the other hand, others might disagree with our assessment.

Let's present an example where the residuals clearly follow a normal distribution and an example where they don't in Figure 11.5. In this case of the model yielding the clearly non-normal residuals on the right, any inference for regression would not be valid.

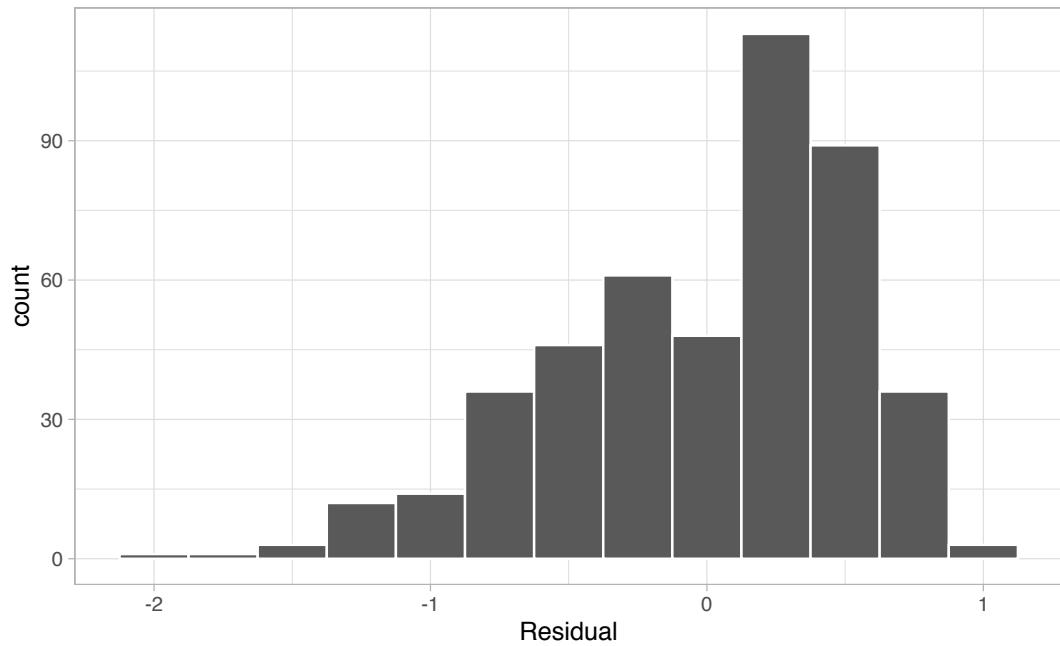


FIGURE 11.4: Histogram of residuals.

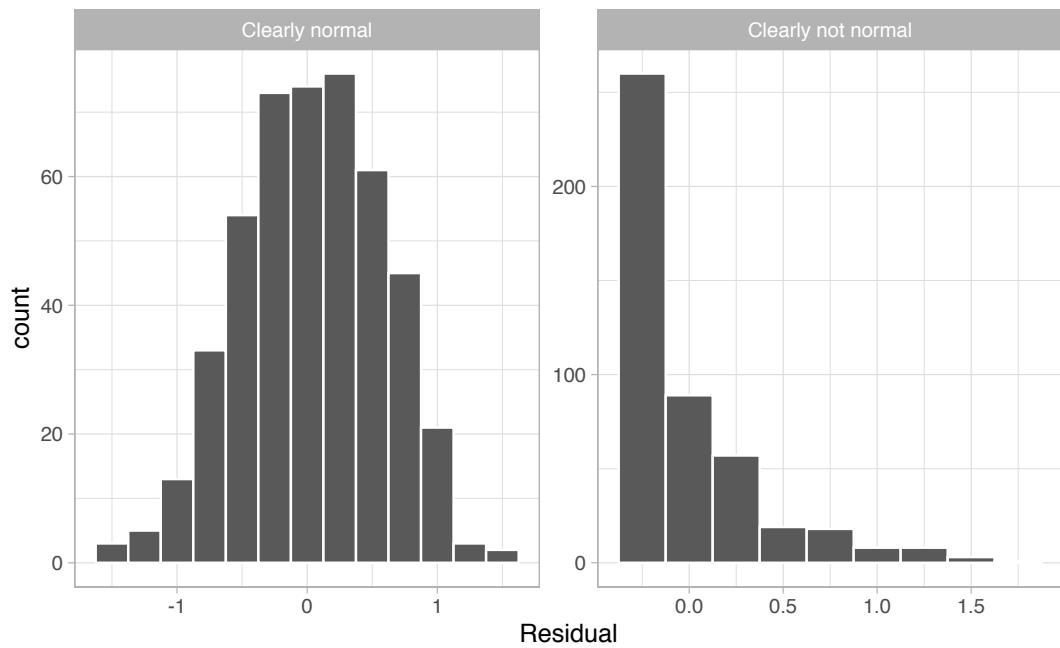


FIGURE 11.5: Example of clearly normal and clearly non-normal residuals.

11.3.5 Equality of variance

Continuing with our ideal that there be *no systematic pattern* to the residuals, the fourth and final condition is that the residuals should exhibit Equal variance for across all values of x . In other words, the value and spread of the residuals should not depend on the value of x .

First, recall the scatterplot in Figure 11.1. We had the explanatory variable beauty score x on the x-axis and the outcome variable teaching score y on the y-axis. Instead, let's create a scatterplot that has the same values on the x-axis, but now with the residual $y - \hat{y}$ on the y-axis instead in Figure 11.6 below.

```
ggplot(regression_points, aes(x = bty_avg, y = residual)) +
  geom_point() +
  labs(x = "Beauty Score", y = "Residual") +
  geom_hline(yintercept = 0, col = "blue", size = 1)
```

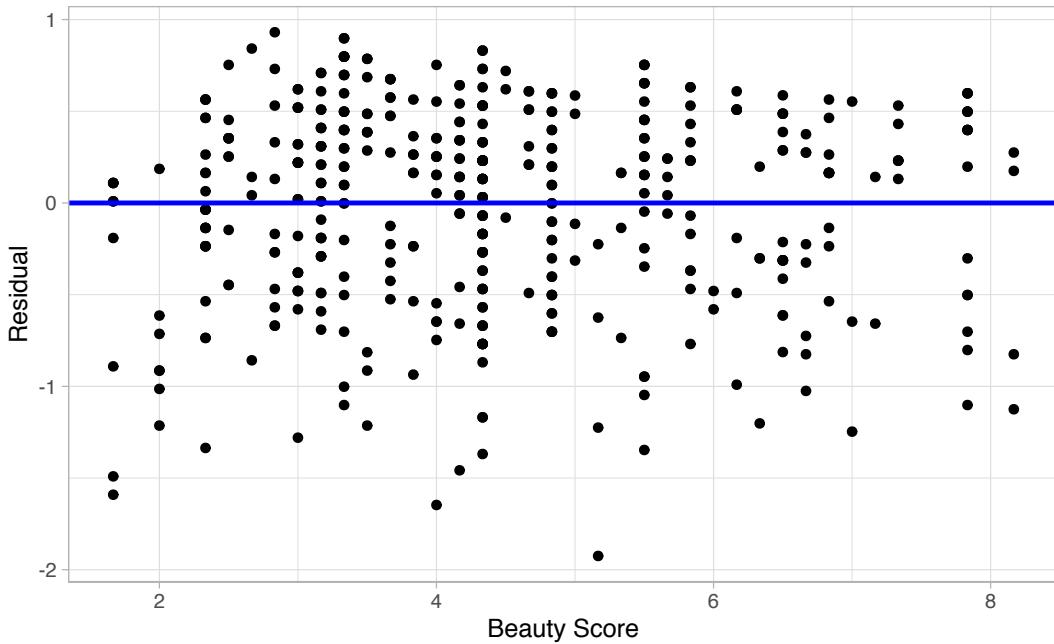


FIGURE 11.6: Plot of residuals over beauty score.

You can think of this plot as a version of the plot with the regression line in blue Figure 11.1, but with the regression line flattened out to $y = 0$. Looking at this plot, would you say that the spread of the residuals around the blue line is constant? This question is rather qualitative and subjective in nature, thus different people may respond with different answers to the above question. For

example, some people might say that there is slightly more variation in the residuals for smaller values of x than with for higher ones. However, it can be argued that there isn't a *drastic* inequality.

Let's present an example where the residuals clearly do not have equal variance for all values of x in Figure 11.7. The spread of the residuals increases as the value of x increases; this situation is known as *heteroskedasticity*. In this case of the model yielding these residuals on the right, any inference for regression would not be valid.

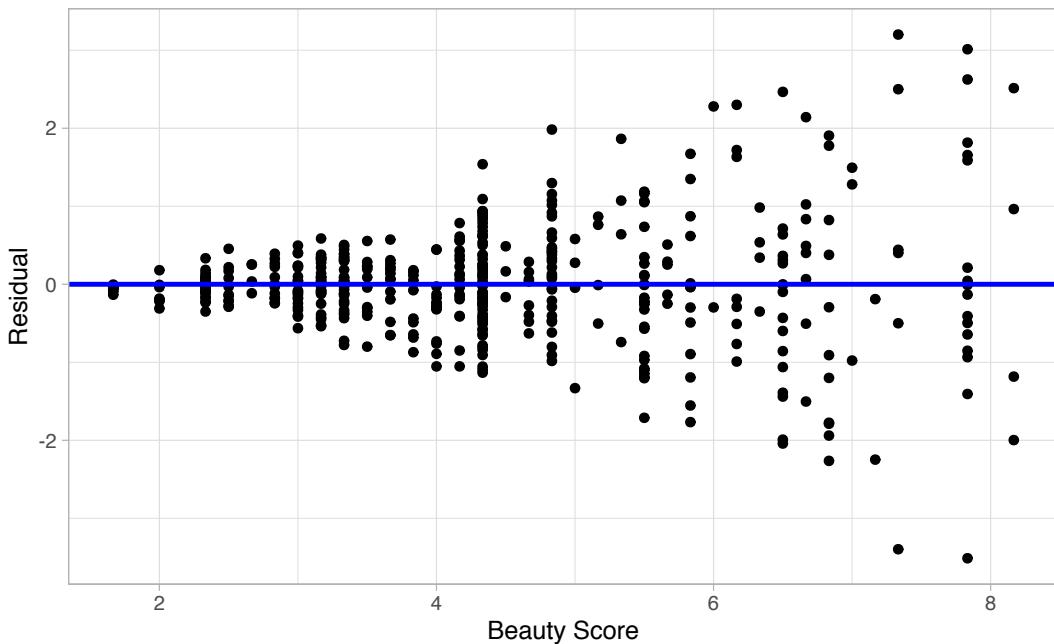


FIGURE 11.7: Example of clearly non-equal variance.

Learning check

(LC11.1) Continuing with our regression using `age` as the explanatory variable and teaching `score` as the outcome variable, use the `get_regression_points()` function to get the observed values, fitted values, and residuals for all 463 instructors. Perform a residual analysis and look for any systematic patterns in the residuals. Ideally, there should be little to no pattern.

11.3.6 What's the conclusion?

Let's list our four conditions again and indicate whether or not they were satisfied in our analysis:

1. Linearity of relationship between variables: Yes
2. Independence of residuals: No
3. Normality of residuals: Somewhat
4. Equality of variance: Yes

So what does this mean for the results of our confidence intervals and hypothesis tests in Section 11.2? Our model of `score ~ bty_avg` should be viewed as a preliminary analysis that needs to be improved on.

First, the **I**ndependence condition. The fact that there exist dependencies between different rows in `evals_ch6` must be addressed. In more advanced statistics courses, you'll learn how incorporate the fact that there are 94 instructors teaching multiple courses. One such technique is called *hierarchical AKA multilevel modeling*.

Second, when conditions **L**, **N**, **E** are not met, it often means there is a shortcoming in our model. For example, is using only a single explanatory variable beauty score sufficient? Should we be including more information via a multiple regression model as we did in Chapter 7? Or perhaps a more sophisticated form of regression modeling, like *logistic* or *Poisson* regression is necessary?

In our case, the best we can do is view the result suggested by our confidence intervals and hypothesis testing that there is in fact a relationship between teaching and beauty scores as preliminary. However further investigation is warranted, in particular by improving the model so that the 4 conditions are met. When the 4 conditions are at least roughly met, then we can put more faith into our confidence intervals and p-values.

The conditions for inference in regression problems are a key part of regression analysis that are of vital importance to the processes of constructing confidence intervals and conducting hypothesis tests. However, it is often the case with regression analysis in the real-world that not all the conditions are completely. Furthermore, as you saw in verifying there is a level of subjectivity in the residual analyses to verify the **L**, **N**, and **E** conditions. So what can you do? We as authors advocate for transparency in communicating all results and

letting the stakeholders of analyses know about any shortcomings of a model as needed or if the model is “good enough.”

11.4 Simulation-based inference for regression

Recall in Section 11.2.5 when we interpreted all the third through seventh columns of the regression table in Table 11.1, we stated in Subsection 11.2.5 that R doesn’t do simulations to compute these values. Rather R uses theory-based methods that involve mathematical formulas.

In this section, we’ll use the simulation-based methods you previously learned in Chapters 9 and 10 to replicate the above results. In particular, we’ll use the `infer` package workflow to

- Construct a 95% confidence interval for the population slope β_1 using bootstrap resampling with replacement. We did this previously in Sections 9.4 with the `pennies` data and 9.6 with the `mythbusters_yawn` data.
- Conduct a hypothesis test of $H_0 : \beta_1 = 0$ vs $H_A : \beta_1 \neq 1$ using permuting/shuffling. We did this previously in Sections 10.3 with the `promotions` data and 10.5 with the `movies_sample` IMDb data.

11.4.1 Confidence interval for slope

We construct the bootstrap distribution for the fitted slope b_1 by following the steps below. We then use this bootstrap distribution to construct a 95% confidence interval for β_1 . Remember that these denote a range of plausible values for an unknown true population slope β_1 quantifying the relationship between teaching score on beauty score.

1. `specify()` the variables of interest in `evals_ch6` with the formula: `score ~ bty_avg`.
2. `generate()` replicates by using `bootstrap` resampling with replacement from the original sample of 463 courses. We generate `reps = 1000` such replicates.
3. `calculate()` the summary statistic of interest: the fitted `slope b1`

Here, the bootstrapping with replacement is done row-by-row. Thus, the original pairs of points `score` and `bty_avg` are linked together and may be repeated with each resample.

```
bootstrap_distn_slope <- evals_ch6 %>%
  specify(formula = score ~ bty_avg) %>%
  generate(reps = 1000, type = "bootstrap") %>%
  calculate(stat = "slope")
```

```
bootstrap_distn_slope
```

```
# A tibble: 1,000 x 2
  replicate      stat
  <int>      <dbl>
1       1  0.0651055
2       2  0.0382313
3       3  0.108056 
4       4  0.0666601
5       5  0.0715932
6       6  0.0854565
7       7  0.0624868
8       8  0.0412859
9       9  0.0796269
10      10 0.0761299
# ... with 990 more rows
```

Observe how we have 1000 values of the bootstrapped slope b_1 . Let's visualize the resulting bootstrap distribution in Figure 11.8. Recall from Section 9.7.1 that shape of the bootstrap distribution of b_1 closely approximates the shape of the sampling distribution of b_1 , in particular the spread.

```
visualize(bootstrap_distn_slope)
```

Percentile-method

First, let's compute the 95% confidence interval for β_1 using the percentile method, in other words by identifying the 2.5th and 97.5th percentiles which include the middle 95% of values. Recall that this method does not require the bootstrap distribution to be normally shaped.

```
percentile_ci <- bootstrap_distn_slope %>%
  get_confidence_interval(type = "percentile", level = 0.95)
percentile_ci
```

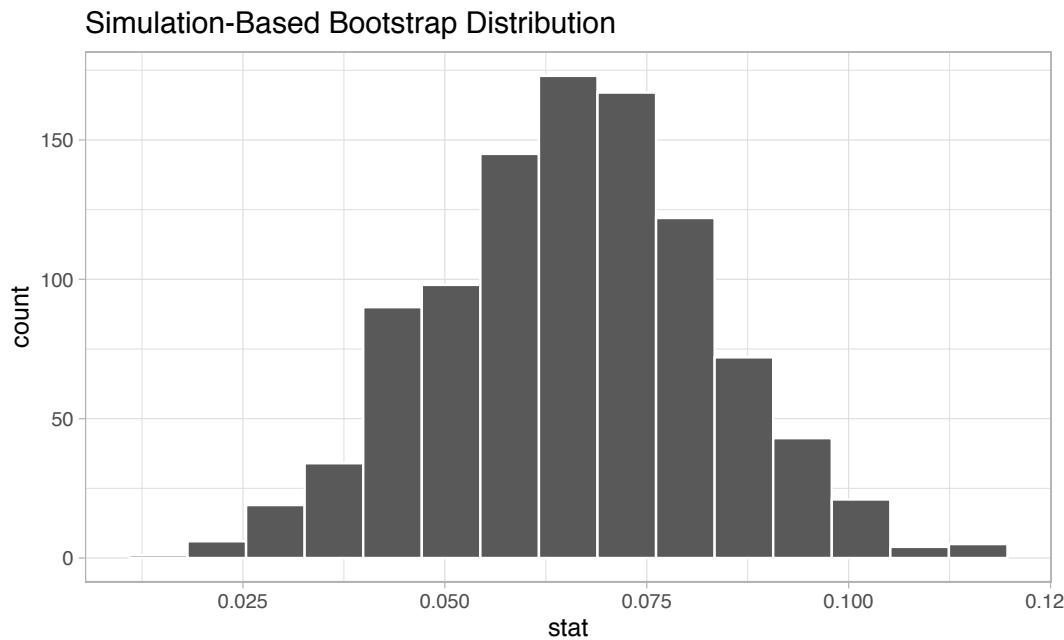


FIGURE 11.8: Bootstrap distribution.

```
# A tibble: 1 × 2
  `2.5%`   `97.5%
  <dbl>     <dbl>
1 0.0323411 0.0990027
```

The resulting percentile-based 95% confidence interval for β_1 of (0.032, 0.099) is similar to the confidence interval in the regression Table 11.1 of (0.035, 0.099).

Standard error method

Since the bootstrap distribution in Figure 11.8 appears to be roughly bell-shaped, we can also construct a 95% confidence interval for β_1 using the standard error method. This necessitates the fitted slope b_1 . While we saw in the regression table in Table 11.1 that this was $b_1 = 0.067$, we can also use the infer pipeline with the `generate()` step removed:

```
observed_slope <- evals %>%
  specify(score ~ bty_avg) %>%
  calculate(stat = "slope")
observed_slope
```

```
# A tibble: 1 × 1
```

```

stat
<dbl>
1 0.0666370

se_ci <- bootstrap_distn_slope %>%
  get_ci(level = 0.95, type = "se", point_estimate = observed_slope)
se_ci

# A tibble: 1 × 2
  lower     upper
  <dbl>     <dbl>
1 0.0333767 0.0998974

```

The resulting standard error-based 95% confidence interval for β_1 of (0.033, 0.1) is however slightly different than the confidence interval in the regression Table 11.1 of (0.035, 0.099).

Comparing all three

Let's compare all three confidence intervals in Figure 11.9, where the percentile-based confidence interval is in red, the standard error based confidence interval is in blue, and the theory-based confidence interval from the regression table is in green.

```

visualize(bootstrap_distn_slope) +
  shade_confidence_interval(endpoints = percentile_ci, fill = NULL,
                             color = "red") +
  shade_confidence_interval(endpoints = se_ci, fill = NULL,
                             color = "blue") +
  shade_confidence_interval(endpoints = c(0.035, 0.099), fill = NULL,
                             color = "darkgreen")

```

Observe that all three are quite similar!

11.4.2 Hypothesis test for slope

We will set up this hypothesis testing process consistent with the “There is Only One Test” diagram in Figure 10.14 using the `infer` package. Let’s conduct a hypothesis test of $H_0 : \beta_1 = 0$ vs $H_A : \beta_1 \neq 0$. We need to think about what it means for β_1 to be zero as stated in the null hypothesis H_0 . If $\beta_1 = 0$, we said earlier that there is no relationship between the teaching and beauty scores.

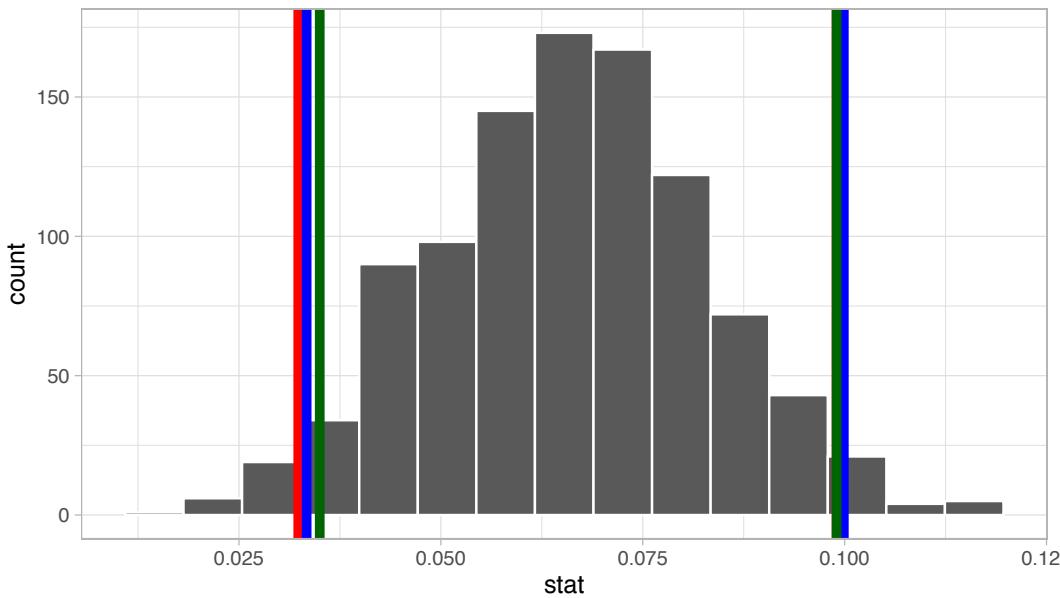


FIGURE 11.9: Two 95 percent confidence intervals: percentile method in red, standard error method in blue.

Assuming this particular null hypothesis means that in our “hypothesized universe” there is no relationship between `score` and `bty_avg` and thus we can shuffle/permute the `bty_avg` variable to no consequence. We, therefore, have another example of shuffling/permuting of data under the null hypothesis.

We construct the null distribution of the fitted slope b_1 by following the steps. Recall from Section @ref() that the null distribution is merely the sampling distribution of our test statistic b_1 assuming the null hypothesis H_0 is true.

1. `specify()` the variables of interest in `evals_ch6` with the formula: `score ~ bty_avg`.
2. `hypothesize()` the null hypothesis of `independence`. Recall from Section 10.3 that this is a step that needs to be added for hypothesis test.
3. `generate()` replicates by permuting/shuffling the explanatory variable `bty_avg` from the original sample of 463 courses. We generate `reps = 1000` such replicates.
4. `calculate()` the test statistic of interest: the fitted slope b_1

To further reinforce the process being done in the pipeline, we’ve added the `type = "permute"` argument to `generate()`. In this case, we permuted the values of one variable across the values of the other 1000 times and calculated a “slope” coefficient for each of these 1000 generated samples.

```
null_distn_slope <- evals %>%
  specify(score ~ bty_avg) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 1000, type = "permute") %>%
  calculate(stat = "slope")
```

Let's visualize the resulting null distribution for the fitted slope b_1 in Figure 11.10. Notice how it is centered at $b_1 = 0$. This is because in our hypothesized universe, there is no relationship between `score` and `bty_avg`, or in other words $\beta_1 = 0$. Thus the most typical fitted slope b_1 we observe across our simulations is 0. Observe also how there is variation around this central value of 0.

```
visualize(null_distn_slope)
```

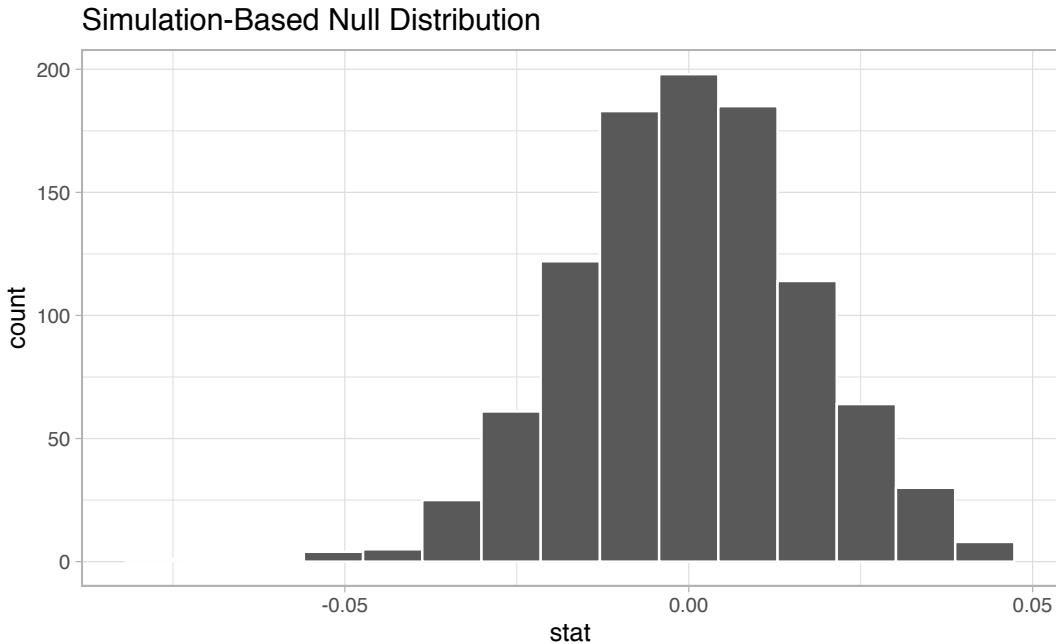


FIGURE 11.10: Null distribution.

Let's visualize this p-value in the above null distribution by comparing it to the observed test statistic of $b_1 = \text{c}(bty_avg = 0.0666370370198145)$. We'll do this by adding a `shade_p_value()` layer in Figure 11.11

```
visualize(null_distn_slope) +
  shade_p_value(obs_stat = observed_slope, direction = "both")
```

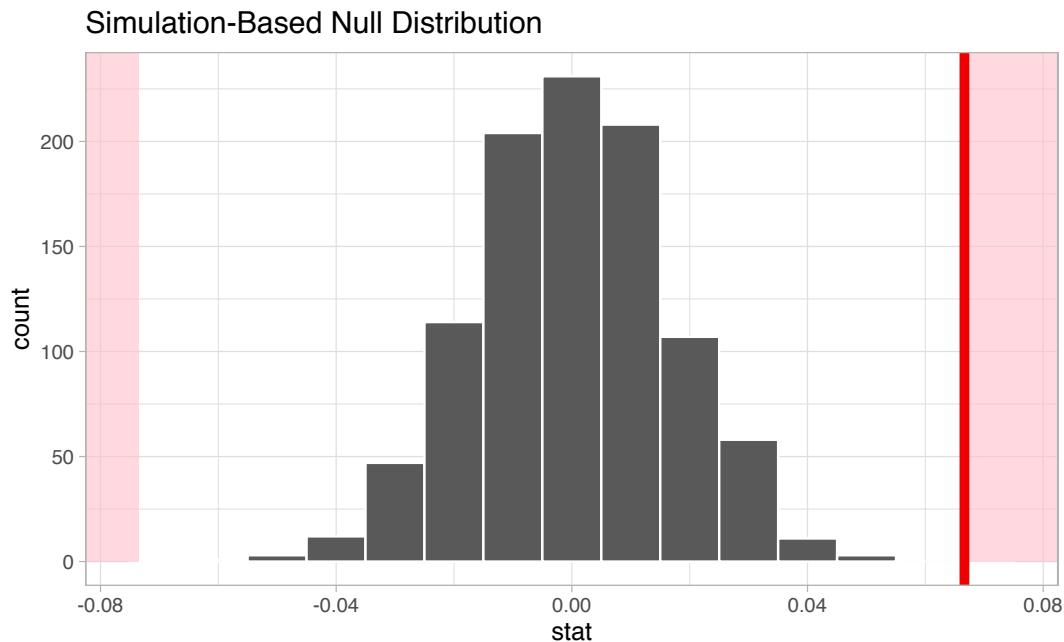


FIGURE 11.11: Null distribution and p-value.

Since 0.067 falls far to the right of this null distribution and doesn't overlap it, we can say that we have a *p*-value of 0. We, thus, have evidence that suggests there is a relationship between the beauty score and teaching score of University of Texas faculty members. What is the numerical value of the *p*-value? Let's use the `get_p_value()` function with the same inputs as the `shade_p_value()` function from above

```
null_distn_slope %>%
  get_p_value(obs_stat = observed_slope, direction = "both")
```

```
# A tibble: 1 × 1
  p_value
  <dbl>
1      0
```

This matches the *p*-value of 0 in the regression table in Table 11.1. When the conditions for inference for regression are met and the null distribution has the characteristic bell shape, we are likely to see similar results between the simulation-based and theoretical results.

Learning check

(LC11.2) Repeat the inference above but this time for the correlation coefficient instead of the slope. Note the implementation of `stat = "correlation"` in the `calculate()` function of the `infer` package.

11.5 Conclusion

11.5.1 Summary

We've now completed our last two sampling scenarios out of six first introduced in Table 8.8 on the six scenarios of sampling for inference. Armed with the regression modeling techniques you learned in Chapters 6 and 7, your understanding of sampling for inference in Chapter 8, and tools for statistical inference like confidence intervals and hypothesis tests in Chapters 9 and 10, you're now able to study a wide array of data!

TABLE 11.4: Scenarios of sampling for inference

Scenario	Population parameter	Notation	Point estimate	Notation.
1	Population proportion	p	Sample proportion	\hat{p}
2	Population mean	μ	Sample mean	\bar{x} or $\hat{\mu}$
3	Difference in population proportions	$p_1 - p_2$	Difference in sample proportions	$\hat{p}_1 - \hat{p}_2$
4	Difference in population means	$\mu_1 - \mu_2$	Difference in sample means	$\bar{x}_1 - \bar{x}_2$
5	Population regression slope	β_1	Fitted regression slope	b_1 or $\hat{\beta}_1$
6	Population regression intercept	β_0	Fitted regression intercept	b_0 or $\hat{\beta}_0$

11.5.2 Additional resources

An R script file of all R code used in this chapter is available here¹.

11.5.3 What's to come

You've now concluded the last major part of the book on "Statistical Inference via `infer`." The closing chapter concludes with a case study on house prices in Seattle, Washington in the US. You'll see there how the principles in this book can apply to help you be a great storyteller with data!

¹[scripts/11-inference-for-regression.R](#)

Part IV

Conclusion

12

Tell the Story with Data

Recall in Section 1.1 “Introduction for students” and at the end of chapters throughout this book, we displayed the “ModernDive flowchart” mapping your journey through this book.

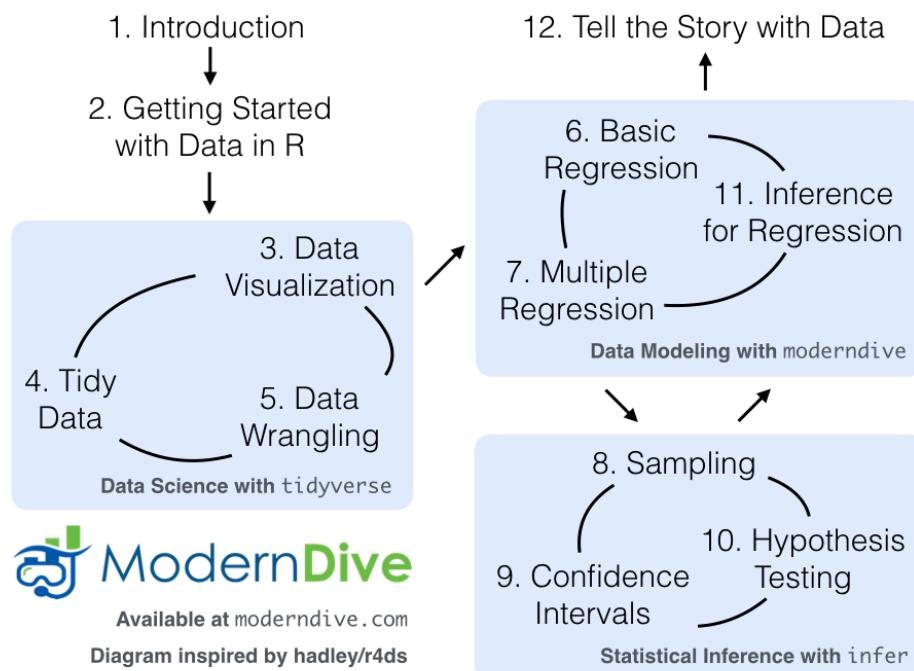


FIGURE 12.1: ModernDive Flowchart.

Let’s get a refresher of what you’ve covered so far. You first got started with data in Chapter 2, where you learned about the difference between R and RStudio, started coding in R, started understanding what R packages are, and explored your first dataset: all domestic departure flights from a New York City airport in 2013. Then:

1. **Data science:** You assembled your data science toolbox using `tidyverse` packages. In particular:
 - Ch.3: Visualizing data via the `ggplot2` package.
 - Ch.4: Wrangling data via the `dplyr` package.
 - Ch.5: Understanding the concept of “tidy” data as a standardized data input format for all packages in the `tidyverse` and how to import data from files.
2. **Data modeling:** Using these data science tools and helper functions from the `moderndive` package, you started performing data modeling. In particular:
 - Ch.6: Constructing basic regression models.
 - Ch.7: Constructing multiple regression models.
3. **Statistical inference:** Once again using your newly acquired data science tools, you unpacked statistical inference using the `infer` package. In particular:
 - Ch.8: Understanding the role that sampling variability plays in statistical inference using both tactile and virtual simulations of sampling from a “bowl” with an unknown proportion of red balls.
 - Ch.9: Building confidence intervals.
 - Ch.10: Conducting hypothesis tests.
4. **Data modeling revisited:** Armed with your new understanding of statistical inference, you revisited and reviewed the models you constructed in Ch.6 & Ch.7. In particular:
 - Ch.11: Interpreting both the statistical and practical significance of the results of the models.

All this was our approach of guiding you through your first experiences of “thinking with data”¹, an expression originally coined by Diane Lambert of Google. How the philosophy underlying this expression guided our mapping of the flowchart above was well put in the introduction to the “Practical Data Science for Stats”² collection of pre-prints focusing on the practical side of data science workflows and statistical analysis, curated by Jennifer Bryan³ and Hadley Wickham⁴:

There are many aspects of day-to-day analytical work that are almost absent from the conventional statistics literature and curriculum. And yet these ac-

¹<https://arxiv.org/pdf/1410.3127.pdf>

²<https://peerj.com/collections/50-practicaldatascistats/>

³<https://twitter.com/jennybryan?lang=en>

⁴<https://twitter.com/hadleywickham?lang=en>

tivities account for a considerable share of the time and effort of data analysts and applied statisticians. The goal of this collection is to increase the visibility and adoption of modern data analytical workflows. We aim to facilitate the transfer of tools and frameworks between industry and academia, between software engineering and statistics and computer science, and across different domains.

In other words, in order to be equipped to “think with data” in the 21st century, future analysts need preparation going through the entirety of the “Data/Science Pipeline”⁵ we also saw earlier and not just parts of it.

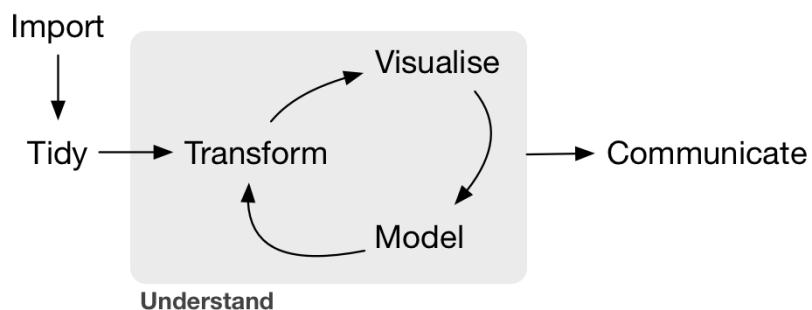


FIGURE 12.2: Data/Science Pipeline.

In Section 12.1, we’ll take you through full-pass of the “Data/Science Pipeline” where we’ll analyze the sale price of houses in Seattle, WA, USA. In Section 12.2, we’ll present you with examples of effective data storytelling, in particular the articles from the data journalism website FiveThirtyEight.com⁶, many of whose source datasets are accessible from the `fivethirtyeight` R package. We believe that you should do more than just think with data but be able to tell the story with data. Let’s explore how this might be done!

Needed packages

Let’s load all the packages needed for this chapter (this assumes you’ve already installed them). Read Section 2.3 for information on how to install and load R packages.

⁵<http://r4ds.had.co.nz/explore-intro.html>

⁶<https://fivethirtyeight.com/>

```
library(ggplot2)
library(dplyr)
library(moderndive)
library(fivethirtyeight)
```

12.1 Case study: Seattle house prices

Kaggle.com⁷ is a machine learning and predictive modeling competition website that hosts datasets uploaded by companies, governmental organizations, and other individuals. One of their datasets is the House Sales in King County, USA⁸ consisting of homes sold in between May 2014 and May 2015 in King County, Washington, USA, which includes the greater Seattle metropolitan area. This CC0: Public Domain⁹ licensed dataset is included in the `moderndive` package in the `house_prices` data frame. We'll refer to this as the "Seattle house prices" dataset.

The dataset consists of 21,613 houses and 21 variables describing these houses; for a full list of these variables see the help file by running `?house_prices` in the R console. In this case study, we'll create a model using multiple regression where:

- The outcome variable y is the sale price of houses.
- The two explanatory/predictor variables we'll use are
 1. x_1 : house size `sqft_living`, as measured by square feet of living space.
Here one square foot is about 0.09 square meters.
 2. x_2 : house `condition`, a categorical variable with 5 levels. Here, 1 indicates "poor" and 5 indicates "excellent."

Let's load all the packages needed for this case study (this assumes you've already installed them). If needed, read Section 2.3 for information on how to install and load R packages.

```
library(ggplot2)
library(dplyr)
library(moderndive)
```

⁷<https://www.kaggle.com/>

⁸<https://www.kaggle.com/harlfoxem/housesalesprediction>

⁹<https://creativecommons.org/publicdomain/zero/1.0/>

12.1.1 Exploratory data analysis (EDA)

A crucial first step before any formal modeling is an exploratory data analysis, commonly abbreviated as EDA. Exploratory data analysis can give you a sense of your data, help identify issues with your data, bring to light any outliers, and help inform model construction. There are three basic approaches to EDA:

1. Most fundamentally, just looking at the raw data. For example using RStudio’s `View()` spreadsheet viewer or the `glimpse()` function from the `dplyr` package
2. Creating visualizations like the ones using `ggplot2` from Chapter 3
3. Computing summary statistics using the `dplyr` data wrangling tools from Chapter 4

First, let’s look the raw data using `View()` and the `glimpse()` function. Explore the dataset. Which variables are numerical and which are categorical? For the categorical variables, what are their levels? Which do you think would be useful variables to use in a model for house price? In this case study, we’ll only consider the variables `price`, `sqft_living`, and `condition`. An important thing to observe is that while the `condition` variable has values 1 through 5, these are saved in R as `fct` factors. This is R’s way of saving categorical variables. So you should think of these as the “labels” 1 through 5 and not the numerical values 1 through 5.

```
View(house_prices)
glimpse(house_prices)
```

```
Observations: 21,613
Variables: 21
$ id              <chr> "7129300520", "6414100192", "5631...
$ date            <date> 2014-10-13, 2014-12-09, 2015-02-...
$ price           <dbl> 221900, 538000, 180000, 604000, 5...
$ bedrooms        <int> 3, 3, 2, 4, 3, 4, 3, 3, 3, 3, ...
$ bathrooms        <dbl> 1.00, 2.25, 1.00, 3.00, 2.00, 4.5...
$ sqft_living     <int> 1180, 2570, 770, 1960, 1680, 5420...
$ sqft_lot         <int> 5650, 7242, 10000, 5000, 8080, 10...
$ floors          <dbl> 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 2.0...
$ waterfront       <lgl> FALSE, FALSE, FALSE, FALSE...
```

```
$ view      <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
$ condition <fct> 3, 3, 3, 5, 3, 3, 3, 3, 3, 3, ...
$ grade      <fct> 7, 7, 6, 7, 8, 11, 7, 7, 7, 8, ...
$ sqft_above <int> 1180, 2170, 770, 1050, 1680, 3890...
$ sqft_basement <int> 0, 400, 0, 910, 0, 1530, 0, 0, 73...
$ yr_built    <int> 1955, 1951, 1933, 1965, 1987, 200...
$ yr_renovated <int> 0, 1991, 0, 0, 0, 0, 0, 0, 0, ...
$ zipcode     <fct> 98178, 98125, 98028, 98136, 98074...
$ lat         <dbl> 47.5, 47.7, 47.7, 47.5, 47.6, 47....
$ long        <dbl> -122, -122, -122, -122, -122, -12...
$ sqft_living15 <int> 1340, 1690, 2720, 1360, 1800, 476...
$ sqft_lot15   <int> 5650, 7639, 8062, 5000, 7503, 101...
```

Let's now perform the second possible approach to EDA: creating visualizations. Since `price` and `sqft_living` are numerical variables, an appropriate way to visualize of these variables' distributions would be using a histogram via `geom_histogram()` as seen in Section 3.5. However, since `condition` is categorical, a barplot using `geom_bar()` yields an appropriate visualization of its distribution. Recall from Section 3.8 that since `condition` is not "pre-counted", we use a `geom_bar()` and not a `geom_col()`. In Figure 12.3, we display all three of these visualizations at once.

```
# Histogram of house price:
ggplot(house_prices, aes(x = price)) +
  geom_histogram(color = "white") +
  labs(x = "price (USD)", title = "House price")

# Histogram of sqft_living:
ggplot(house_prices, aes(x = sqft_living)) +
  geom_histogram(color = "white") +
  labs(x = "living space (square feet)", title = "House size")

# Barplot of condition:
ggplot(house_prices, aes(x = condition)) +
  geom_bar() +
  labs(x = "condition", title = "House condition")
```

We observe the following:

1. In the histogram for `price`:
 - We see that a majority of houses are less than 2 million (2,000,000) dollars.

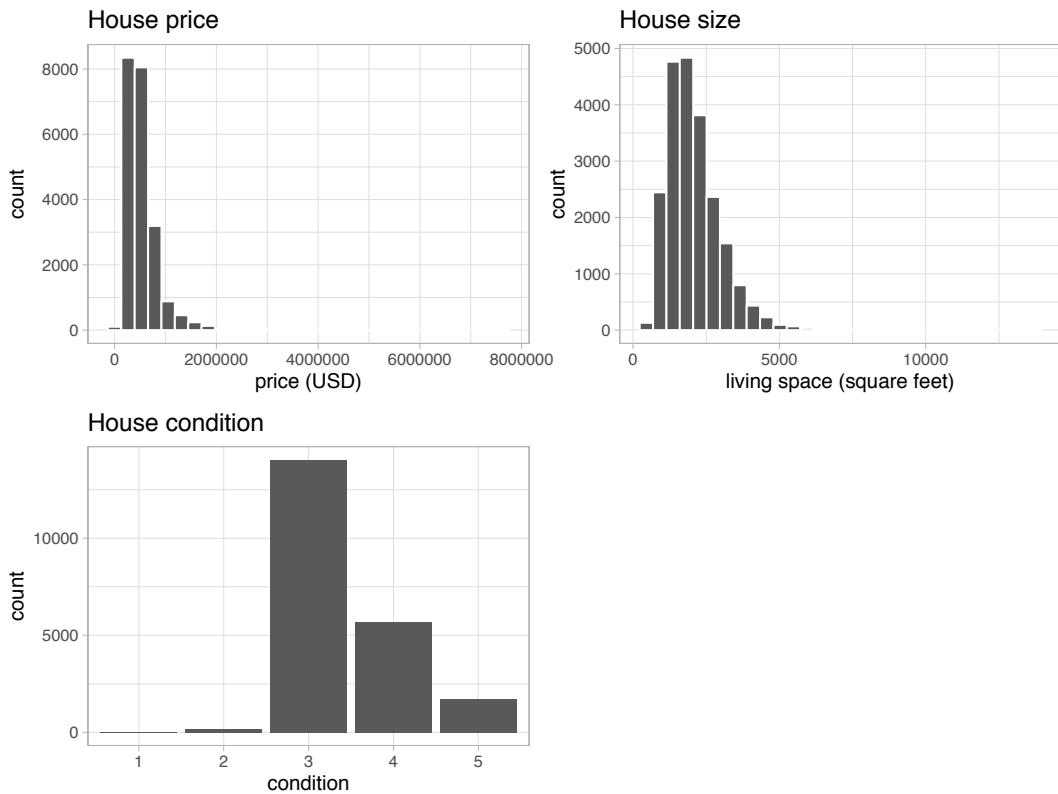


FIGURE 12.3: Exploratory visualizations of Seattle house prices data.

- The x-axis stretches out far to the right to 8 million dollars, even though there appear to be no houses in the plot.
2. In the histogram for size `sqft_living`
 - Most houses appear to have less than 5000 square feet of living space. For comparison a standard American football field is about 57,600 square feet, whereas a standard soccer (association football) field is about 64,000 square feet.
 - The x-axis exhibits the same stretched out behavior to the right as for `price`.
 3. Most houses are of condition 3, 4, or 5.

In the case of `price`, why does the x-axis stretch so far to the right? It is because there are a very small number of houses with price closer to 8 million; these prices are outliers in this case. We say this variable is “right skewed” as exhibited by the long right tail. This skew makes it difficult to compare prices of the less expensive houses as the more expensive houses dominate the scale of the x-axis. This is similarly the case for `sqft_living`.

Let’s now perform the third possible approach to EDA: computing summary

statistics. In particular, let's compute four summary statistics using the `summarize()` data wrangling verb from Section 4.3.

- Two measures of center: the mean and median
- Two measures of variability/spread: the standard deviation and the interquartile-range ($IQR = 3rd\ quartile - 1st\ quartile$)

```
house_prices %>%
  summarize(
    mean_price = mean(price),
    median_price = median(price),
    sd_price = sd(price),
    IQR_price = IQR(price)
  )

# A tibble: 1 x 4
  mean_price median_price sd_price IQR_price
  <dbl>        <dbl>      <dbl>      <dbl>
1     540088.      450000.   367127.    323050
```

Observe the following:

1. The mean `price` of \$540,088 is larger than the median of \$450,000. This is because the small number of very expensive outlier houses prices are inflating the average. Whereas since the median is the “middle” value, it is not as sensitive to such large values at the high end. This is why the news typically report median house prices and not average house prices when describing the real estate market. We say here that the median is more “robust to outliers” than the mean.
2. Similarly, while both the standard deviation and IQR are both measures of spread and variability, the IQR is more “robust to outliers.”

If you repeat the above `summarize()` for `sqft_living`, you'll find a similar relationship between mean vs median and standard deviation vs IQR given its similar right-skewed nature. Is there anything we can do about this right-skew? Again, this could potentially be an issue because we'll have a harder time discriminating between houses at the lower end of `price` and `sqft_living`, which might lead to a problem when modeling. We can in fact address this issue by using a log base 10 transformation, which we cover next.

12.1.2 \log_{10} transformations

At its simplest, $\log_{10}()$ transformations return base 10 *logarithms*. For example, since $1000 = 10^3$, $\log_{10}(1000)$ returns 3. To undo a \log_{10} -transformation, we raise 10 to this value. For example, to undo the previous \log_{10} -transformation and return the original value of 1000, we raise 10 to this value 10^3 by running $10^{(3)} = 1000$. \log -transformations allow us to focus on multiplicative changes instead of additive ones, thereby emphasizing changes in “orders of magnitude.” We can also frame these as being percentage increases and that will be the focus here. Let’s illustrate this idea in Table 12.1 with examples of prices of consumer goods in US dollars.

TABLE 12.1: \log_{10} -transformed prices, orders of magnitude, and examples

Price	$\log_{10}(\text{Price})$	Order of magnitude	Examples
\$1	0	Singles	Cups of coffee
\$10	1	Tens	Books
\$100	2	Hundreds	Mobile phones
\$1,000	3	Thousands	High definition TV's
\$10,000	4	Tens of thousands	Cars
\$100,000	5	Hundreds of thousands	Luxury cars & houses
\$1,000,000	6	Millions	Luxury houses

Let’s break this down:

1. When purchasing a cup of coffee, we tend to think of prices ranging in single dollars e.g. \$2 or \$3. However when purchasing, say, mobile phones, we don’t tend to think in prices in single dollars such as \$676 or \$757, but tend to round to the nearest unit of hundreds of dollars such as \$200 or \$500.
2. Let’s say we want to know the \log_{10} -transformed value of \$76. Even if this would be hard to compute without a calculator, we know that its \log_{10} value is between 1 and 2, since \$76 is between \$10 and \$100. In fact, $\log_{10}(76)$ is 1.880814.
3. \log_{10} -transformations are *monotonic*, meaning they preserve orderings. So if Price A is lower than Price B, then $\log_{10}(\text{Price A})$ will also be lower than $\log_{10}(\text{Price B})$.
4. Most importantly, increments of one in \log_{10} correspond to multiplicative changes and not additive ones. For example, increasing from $\log_{10}(\text{Price})$ of 3 to 4 corresponds to a multiplicative increase by a factor of 10: \$100 to \$1000.

Let's create new log10-transformed versions of the right-skewed variable `price` and `sqft_living` using the `mutate()` function from Section 4.5, but we'll give the latter the name `log10_size`, which is a little more succinct and descriptive a variable name.

```
house_prices <- house_prices %>%
  mutate(
    log10_price = log10(price),
    log10_size = log10(sqft_living)
  )
```

Let's first display the before and after effects of this transformation on these variables for only the first 10 rows of `house_prices`:

```
house_prices %>%
  select(price, log10_price, sqft_living, log10_size)
```

	price	log10_price	sqft_living	log10_size
	<dbl>	<dbl>	<int>	<dbl>
1	221900	5.34616	1180	3.07188
2	538000	5.73078	2570	3.40993
3	180000	5.25527	770	2.88649
4	604000	5.78104	1960	3.29226
5	510000	5.70757	1680	3.22531
6	1225000	6.08814	5420	3.73400
7	257500	5.41078	1715	3.23426
8	291850	5.46516	1060	3.02531
9	229500	5.36078	1780	3.25042
10	323000	5.50920	1890	3.27646

Observe in particular:

- The house in the sixth row with `price` \$1,225,000, which is just above one million dollars. Since 10^6 is one million, its `log10_price` is 6.09. Contrast this with all other houses with `log10_price` less than six.
- Similarly, there is only one house with size `sqft_living` less than 1000. Since $1000 = 10^3$, its the lone house with `log10_size` less than 3.

Let's now visualize the before and after effects of this transformation for `price` in Figure 12.4.

```
# Before:
ggplot(house_prices, aes(x = price)) +
  geom_histogram(color = "white") +
  labs(x = "price (USD)", title = "House price: Before")

# After:
ggplot(house_prices, aes(x = log10_price)) +
  geom_histogram(color = "white") +
  labs(x = "log10 price (USD)", title = "House price: After")
```

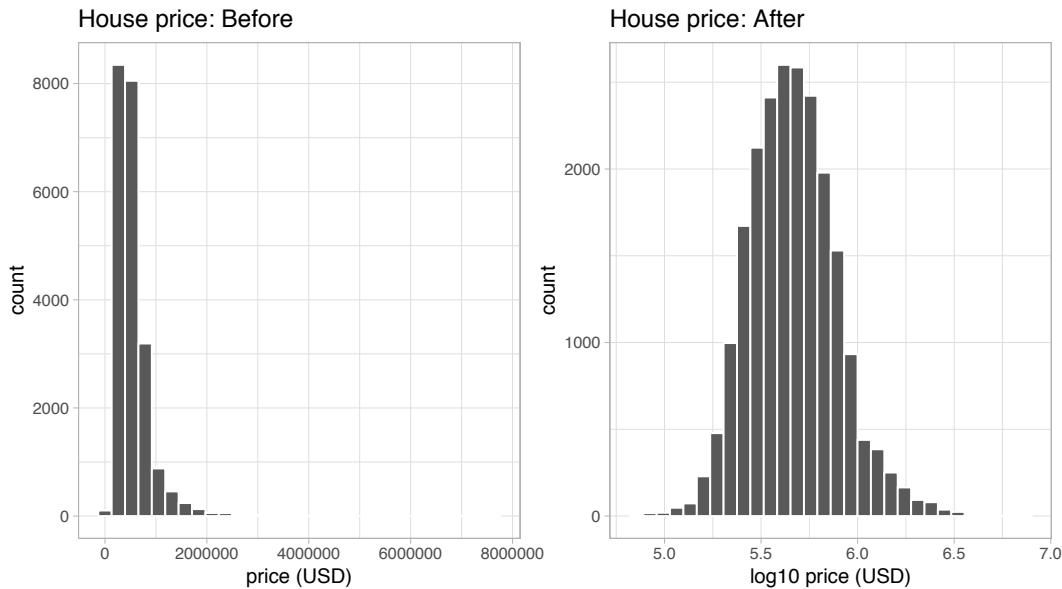


FIGURE 12.4: House price before and after log10-transformation.

Observe that after the transformation, the distribution is much less skewed, and in this case, more symmetric and bell-shaped. Note this isn't always necessarily the case. Now you can now better discriminate between house prices at the lower end of the scale. Let's do the same for size where the before variable is `sqft_living` and the after variable is `log10_size`. Observe in Figure 12.5 that the log10-transformation has a similar effect of un-skewing the variable. Again, we emphasize that while in these two cases the resulting distributions are more symmetric and bell-shaped, this is not always necessarily the case.

```
# Before:
ggplot(house_prices, aes(x = sqft_living)) +
  geom_histogram(color = "white") +
```

```

labs(x = "living space (square feet)", title = "House size: Before")

# After:
ggplot(house_prices, aes(x = log10_size)) +
  geom_histogram(color = "white") +
  labs(x = "log10 living space (square feet)", title = "House size: After")
```

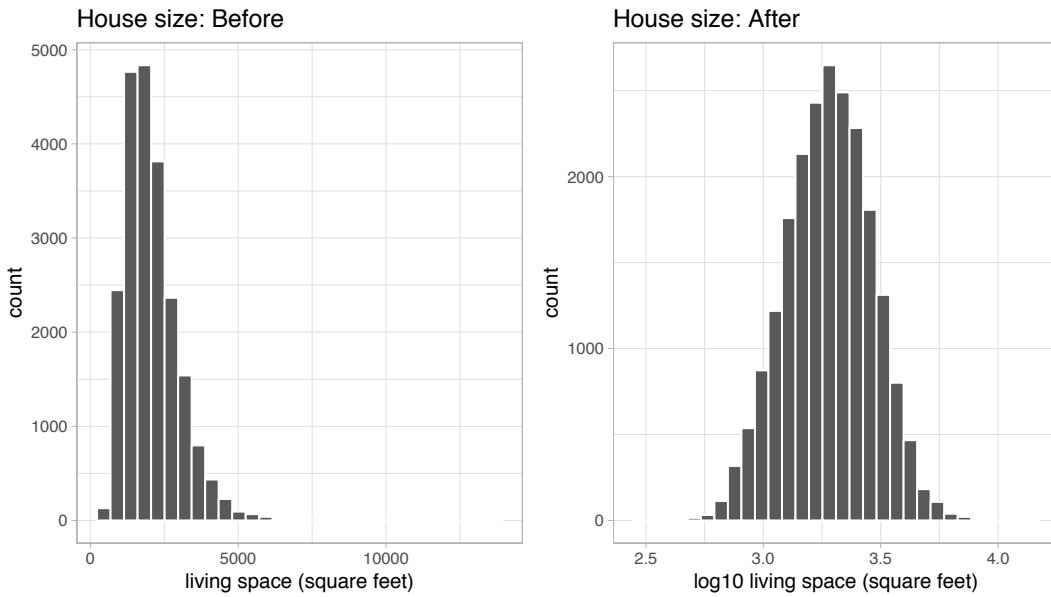


FIGURE 12.5: House size before and after log10-transformation.

Given the now un-skewed nature of `log10_price` and `log10_size`, we are going to revise our modeling structure:

- We'll use a new outcome variable y `log10_price` of houses
- The two explanatory/predictor variables we'll use are
 1. x_1 : A modified version of house size: `log10_size`
 2. x_2 : House condition will remain unchanged

12.1.3 EDA Part II

Let's continue our exploratory data analysis from Subsection 12.1.1 above. The earlier EDA you performed was *univariate* in nature in that we only considered one variable at a time. The goal of modeling, however, is to explore relationships between variables. So we must *jointly* consider the relationship between the outcome variable `log10_price` and the explanatory/predictor variables `log10_size` (numerical) and `condition` (categorical). We viewed such a

modeling scenario in Section 7.1 using the `evals` dataset, where the outcome variable was teaching `score`, the numerical explanatory/predictor variable was instructor `age`, and the categorical explanatory/predictor variable was (binary) `gender`.

We have two possible visual models. Either a (1) parallel slopes model in Figure 12.6 where we have a different regression line for each of the 5 possible `condition` levels, each with a different intercept but the same slope:

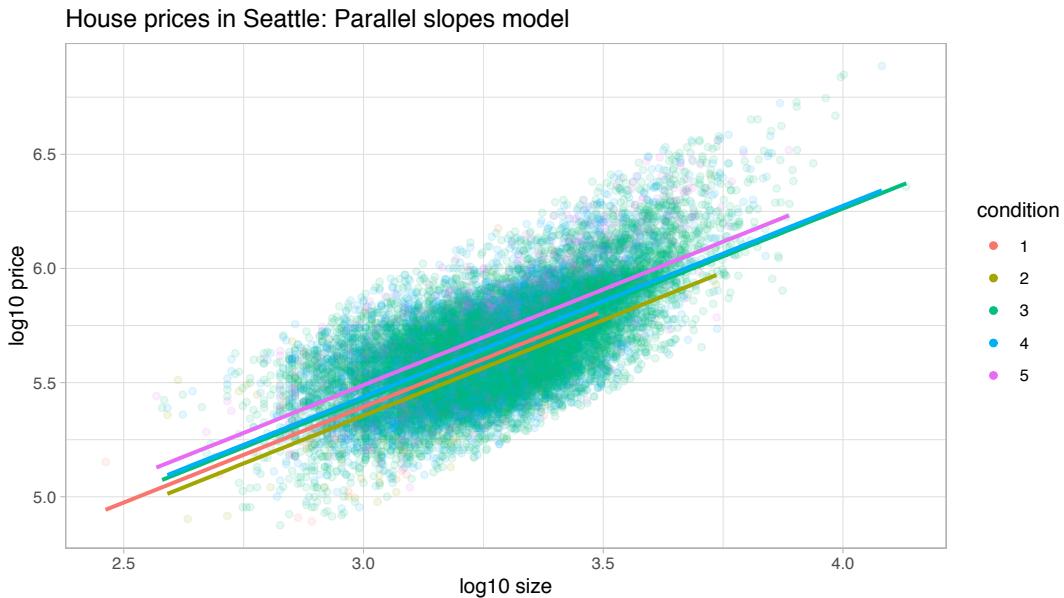


FIGURE 12.6: Parallel slopes model.

Or (2) an interaction model in Figure 12.7, where we allow each regression line to not only have different intercepts, but different slopes as well:

```
ggplot(house_prices, aes(x = log10_size, y = log10_price, col = condition)) +
  geom_point(alpha = 0.1) +
  labs(y = "log10 price", x = "log10 size", title = "House prices in Seattle") +
  geom_smooth(method = "lm", se = FALSE)
```

In both cases, we see there is a positive relationship between house price and size, meaning as houses are larger, they tend to be more expensive. Furthermore, in both plots it seems that houses of condition 5 tend to be the most expensive for most house sizes as evidenced by the fact that the purple line is highest, followed by condition 4 and 3. As for condition 1 and 2, this pattern isn't as clear, as if you recall from the univariate barplot of `condition` in Figure 12.3 there are very few houses of condition 1 or 2. This reality is more appar-

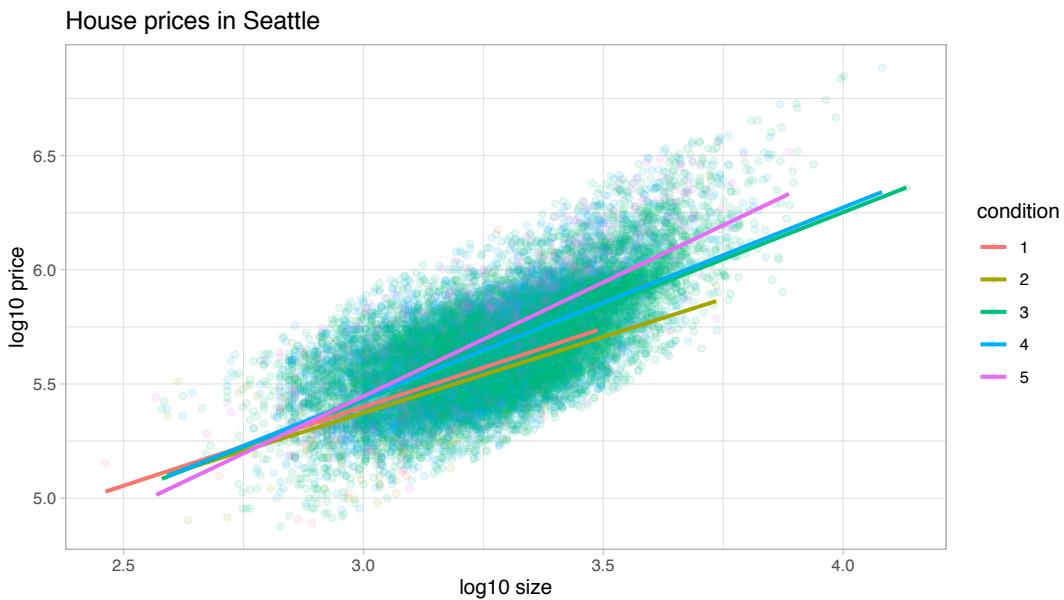


FIGURE 12.7: Interaction model.

ent in an alternative visualization to Figure 12.7. This is displayed in Figure 12.8 using facets instead:

```
ggplot(house_prices, aes(x = log10_size, y = log10_price, col = condition)) +
  geom_point(alpha = 0.3) +
  labs(y = "log10 price", x = "log10 size", title = "House prices in Seattle") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~condition)
```

Which exploratory visualization of the interaction model is better, the one in Figure 12.7 or Figure 12.8? There is no universal right answer; you need to make a choice depending on what you want to convey, and own it.

12.1.4 Regression modeling

For now let's focus on the latter, interaction model we've visualized in Figure 12.8 above. What are the 5 different slopes and intercepts for the condition = 1, condition = 2, ..., and condition = 5 lines in Figure 12.8? To determine these, we first need the values from the regression table:

```
# Fit regression model:
price_interaction <- lm(log10_price ~ log10_size * condition, data = house_prices)
```

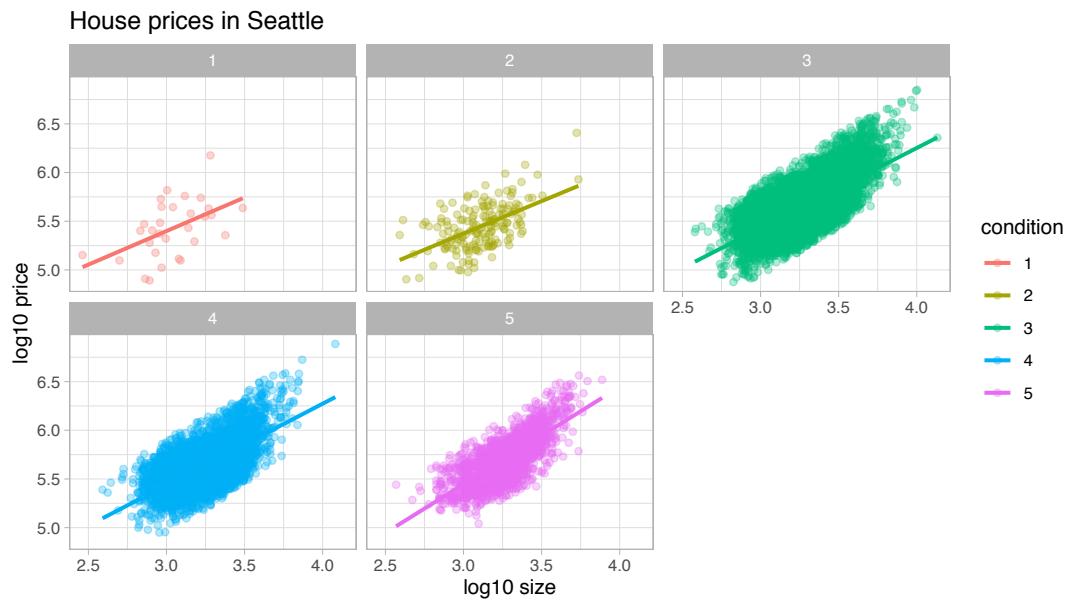


FIGURE 12.8: Interaction model with facets.

```
# Get regression table:
get_regression_table(price_interaction)
```

```
# A tibble: 10 x 7
  term    estimate std_error statistic p_value lower_ci upper_ci
  <chr>     <dbl>     <dbl>     <dbl>     <dbl>    <dbl>    <dbl>
1 inte~     3.33      0.451     7.38      0       2.446
2 log1~    0.69000   0.148     4.652     0       0.399
3 cond~     0.047     0.498     0.094     0.925   -0.93
4 cond~    -0.367     0.452    -0.812     0.417   -1.25300
5 cond~    -0.398     0.453    -0.879     0.38    -1.286
6 cond~    -0.883     0.457    -1.931     0.053   -1.779
7 log1~   -0.024     0.163    -0.148     0.882   -0.34400
8 log1~    0.133     0.148     0.893     0.372   -0.158
9 log1~    0.146000   0.149     0.979     0.328   -0.146000
10 log1~   0.31       0.15      2.067     0.039   0.016
# ... with 1 more variable: upper_ci <dbl>
```

Recall from Section 7.1.2 on how to interpret the outputs where there exists an *interaction term*, where in this case the “baseline for comparison” group for the categorical variable `condition` are the condition 1 houses. We’ll write our answers as

$$\widehat{\log 10(\text{price})} = \hat{\beta}_0 + \hat{\beta}_{\text{size}} * \log 10(\text{size})$$

for all five condition levels separately:

1. Condition 1: $\widehat{\log 10(\text{price})} = 3.33 + 0.69 * \log 10(\text{size})$
2. Condition 2: $\widehat{\log 10(\text{price})} = (3.33 + 0.047) + (0.69 - 0.024) * \log 10(\text{size}) = 3.38 + 0.666 * \log 10(\text{size})$
3. Condition 3: $\widehat{\log 10(\text{price})} = (3.33 - 0.367) + (0.69 + 0.133) * \log 10(\text{size}) = 2.96 + 0.823 * \log 10(\text{size})$
4. Condition 4: $\widehat{\log 10(\text{price})} = (3.33 - 0.398) + (0.69 + 0.146) * \log 10(\text{size}) = 2.93 + 0.836 * \log 10(\text{size})$
5. Condition 5: $\widehat{\log 10(\text{price})} = (3.33 - 0.883) + (0.69 + 0.31) * \log 10(\text{size}) = 2.45 + 1 * \log 10(\text{size})$

These correspond to the regression lines in the exploratory visualization of the interaction model in Figure 12.7 above. For homes of all 5 condition types, as the size of the house increases, the price increases. This is what most would expect. However, the rate of increase of price with size is fastest for the homes with condition 3, 4, and 5 of 0.823, 0.836, and 1 respectively; these are the three most largest slopes out of the five.

12.1.5 Making predictions

Say you're a realtor and someone calls you asking you how much their home will sell for. They tell you that it's in condition = 5 and is sized 1900 square feet. What do you tell them? We first make this prediction visually in Figure 12.9. The predicted `log10_price` of this house is marked with a black dot. This is where the two following lines intersect:

- The purple regression line for the condition = 5 homes and
- The vertical dashed black line at `log10_size` equals 3.28, since our predictor variable is the log10-transformed square feet of living space and $\log 10(1900) = 3.28$.

Eyeballing it, it seems the predicted `log10_price` seems to be around 5.72. Let's now obtain the exact numerical value for the prediction using the values of the intercept and slope for the condition = 5 that we computed using the regression table output. We use the equation for the condition = 5 line, being sure to `log10()` the square footage first.

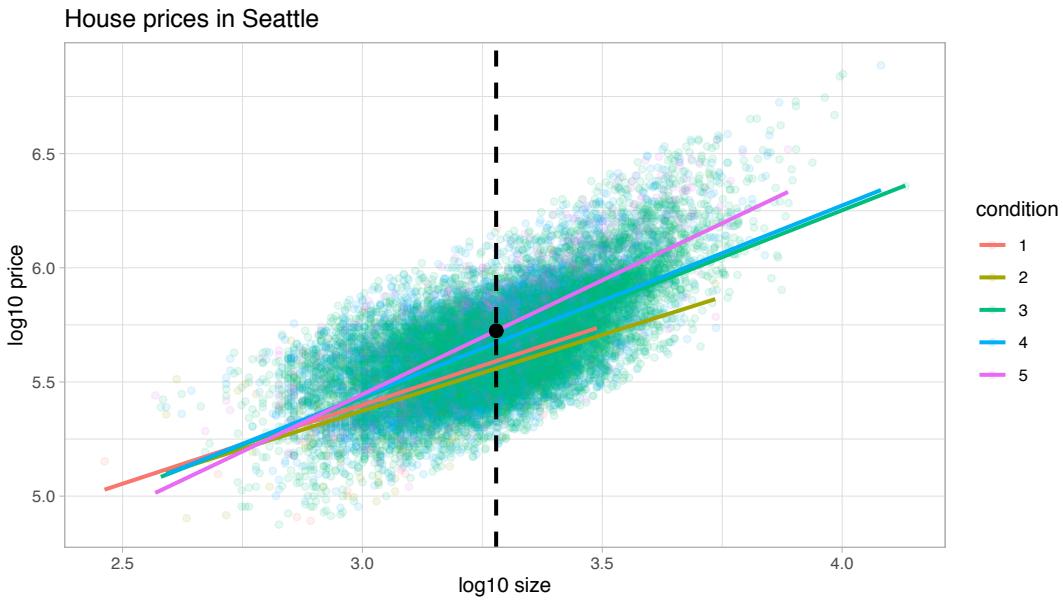


FIGURE 12.9: Interaction model with prediction.

```
2.45 + 1 * log10(1900)
```

```
[1] 5.73
```

This value is very close to our earlier visually made prediction of 5.72. But wait! We were using the outcome variable `log10_price` as our outcome variable! So if we want a prediction in terms of `price` in dollar units, we need to un-log this by taking a power of 10 as described in Section 12.1.2.

```
10^(2.45 + 1 * log10(1900))
```

```
[1] 535493
```

So we our predicted price for this home of condition 5 and size 1900 square feet is \$535,493.

Learning check

(LC12.1) Check that the LINE conditions are met for inference to be made in this Seattle house prices example.

(LC12.2) Repeat the regression modeling in Subsection 12.1.4 and the prediction making you just did on the house of condition 5 and size 1900 square

feet in Subsection 12.1.5, but using the parallel slopes model you visualized in Figure 12.6. Hint: it's \$524,807!

12.2 Case study: Effective data storytelling

As we've progressed throughout this book, you've seen how to work with data in a variety of ways. You've learned effective strategies for plotting data by understanding which types of plots work best for which combinations of variable types. You've summarized data in table form and calculated summary statistics for a variety of different variables. Further, you've seen the value of inference as a process to come to conclusions about a population by using a random sample. Lastly, you've explored how to use linear regression and the importance of checking the conditions required to make it a valid procedure. All throughout, you've learned many computational techniques and focused on reproducible research in writing R code. We now present another case study, but this time of the “effective data storytelling” done by data journalists around the world. Great data stories don't mislead the reader, but rather engulf them in understanding the importance that data plays in our lives through the captivation of storytelling.

12.2.1 Bechdel test for Hollywood gender representation

We recommend you read and analyze this article by Walt Hickey entitled The Dollar-And-Cents Case Against Hollywood's Exclusion of Women¹⁰ on the Bechdel test, an informal test of gender representation in a movie created by Alison Bechdel. More information is at <https://bechdeltest.com/>. As you read over it, think carefully about how Walt is using data, graphics, and analyses to paint the picture for the reader of what the story is he wants to tell. In the spirit of reproducibility, the members of FiveThirtyEight have also shared the data and R code¹¹ that they used to create for this story and many more of their articles on GitHub¹².

¹⁰<http://fivethirtyeight.com/features/the-dollar-and-cents-case-against-hollywoods-exclusion-of-women/>

¹¹<https://github.com/fivethirtyeight/data/tree/master/bechdel>

¹²<https://github.com/fivethirtyeight/data>

ModernDive co-authors Chester Ismay¹³ and Albert Y. Kim¹⁴ along with Jennifer Chun¹⁵ went one step further by creating the `fivethirtyeight` R package¹⁶. The `fivethirtyeight` package takes FiveThirtyEight's article data from GitHub, “tames”¹⁷ it so that it's novice-friendly, and makes all data, documentation, and the original article easily accessible via an R package. The package homepage also includes a list of all `fivethirtyeight` data sets¹⁸ included.

Furthermore, example “vignettes” of fully reproducible start-to-finish analyses of some of these data using `dplyr`, `ggplot2`, and other packages in the `tidyverse` is available here¹⁹. For example, a vignette showing how to reproduce one of the plots at the end of the above article on the Bechdel test is available here²⁰.

12.2.2 US Births in 1999

Here is another example involving the `us_births_1994_2003` data frame of all births in the United States between 1994 and 2003. For more information on this data frame including a link to the original article on FiveThirtyEight.com, check out the help file by running `?us_births_1994_2003` in the console. First, let's load all necessary packages:

```
library(ggplot2)
library(dplyr)
library(fivethirtyeight)
```

It's always a good idea to preview your data, either by using RStudio's spreadsheet `View()` function or using `glimpse()` from the `dplyr` package below:

```
# Preview data
glimpse(us_births_1994_2003)
```

```
Observations: 3,652
Variables: 6
$ year           <int> 1994, 1994, 1994, 1994, 1994, 199...
$ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
```

¹³https://twitter.com/old_man_chester?lang=en

¹⁴<https://twitter.com/rudeboybert>

¹⁵<https://twitter.com/jchunn206>

¹⁶<https://fivethirtyeight-r.netlify.com/>

¹⁷http://rpubs.com/rudeboybert/fivethirtyeight_tamedata

¹⁸<https://fivethirtyeight-r.netlify.com/articles/fivethirtyeight.html#data-sets>

¹⁹<https://fivethirtyeight-r.netlify.com/articles/>

²⁰<https://fivethirtyeight-r.netlify.com/articles/bechdel.html>

```
$ date_of_month <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11...
$ date          <date> 1994-01-01, 1994-01-02, 1994-01-...
$ day_of_week   <ord> Sat, Sun, Mon, Tues, Wed, Thurs, ...
$ births        <int> 8096, 7772, 10142, 11248, 11053, ...
```

We'll focus on the number of `births` for each `date`, but only for births that occurred in 1999. Recall we achieve this using the `filter()` command from `dplyr` package:

```
US_births_1999 <- US_births_1994_2003 %>%
  filter(year == 1999)
```

Since `date` is a notion of time, which has a sequential ordering to it, a line-graph (also known as a “time series” plot) would be more appropriate than a scatterplot. In other words, use a `geom_line()` instead of `geom_point()`:

```
ggplot(US_births_1999, aes(x = date, y = births)) +
  geom_line() +
  labs(x = "Data", y = "Number of births", title = "US Births in 1999")
```

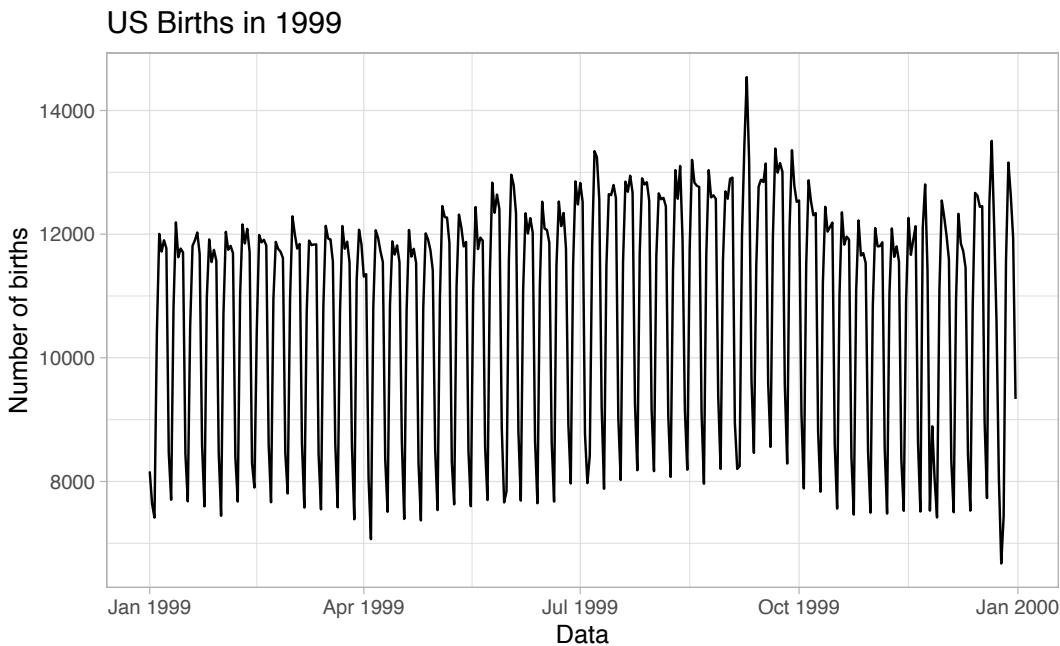


FIGURE 12.10: Number of births in US in 1999.

We see a big valley occurring just before January 1st, 2000, mostly likely due to the holiday season. However, what about the major peak of over 14,000

births occurring just before October 1st, 1999? What could be the reason for this anomalously high spike?

Time to think with data and further tell the story with data! How could statistical modeling help you here? What types of statistical inference would be helpful? What else can you find and where can you take this analysis? We leave these questions to you as the reader to explore and examine. Remember to get in touch with us via our contact info in Section 1.5. We'd love to see what you come up with!

12.2.3 Script of R code

An R script file of all R code used in this chapter is available here²¹.

Concluding remarks

If you've come to this point in the book, we'd suspect that you know a thing or two about how to work with data in R. You've also gained a lot of knowledge about how to use simulation techniques to determine statistical significance and how these techniques build an intuition about traditional inferential methods like the *t*-test. The hope is that you've come to appreciate data wrangling, tidy datasets, and the power of data visualization. Actually, the data visualization part may be the most important thing here. If you can create truly beautiful graphics that display information in ways that the reader can clearly decipher, you've picked up a great skill. Let's hope that that skill keeps you creating great stories with data into the near and far distant future. Thanks for coming along for the ride as we dove into modern data analysis using R!

²¹ [scripts/12-tell-the-story-with-data.R](#)

A

Statistical Background

A.1 Basic statistical terms

A.1.1 Mean

The mean AKA average is the most commonly reported measure of center. It is commonly called the “average” though this term can be a little ambiguous. The mean is the sum of all of the data elements divided by how many elements there are. If we have n data points, the mean is given by:

$$\text{Mean} = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

A.1.2 Median

The median is calculated by first sorting a variable’s data from smallest to largest. After sorting the data, the middle element in the list is the **median**. If the middle falls between two values, then the median is the mean of those two values.

A.1.3 Standard deviation

We will next discuss the **standard deviation** of a sample dataset pertaining to one variable. The formula can be a little intimidating at first but it is important to remember that it is essentially a measure of how far to expect a given data value is from its mean:

$$\text{Standard deviation} = \sqrt{\frac{(x_1 - \text{Mean})^2 + (x_2 - \text{Mean})^2 + \cdots + (x_n - \text{Mean})^2}{n - 1}}$$

A.1.4 Five-number summary

The **five-number summary** consists of five values: minimum, first quantile AKA 25th percentile, second quantile AKA median AKA 50th percentile, third quantile AKA 75th, and maximum. The quantiles are calculated as

- first quantile (Q_1): the median of the first half of the sorted data
- third quantile (Q_3): the median of the second half of the sorted data

The *interquartile range* is defined as $Q_3 - Q_1$ and is a measure of how spread out the middle 50% of values is. The five-number summary is not influenced by the presence of outliers in the ways that the mean and standard deviation are. It is, thus, recommended for skewed datasets.

A.1.5 Distribution

The **distribution** of a variable/dataset corresponds to generalizing patterns in the dataset. It often shows how frequently elements in the dataset appear. It shows how the data varies and gives some information about where a typical element in the data might fall. Distributions are most easily seen through data visualization.

A.1.6 Outliers

Outliers correspond to values in the dataset that fall far outside the range of “ordinary” values. In regards to a boxplot (by default), they correspond to values below $Q_1 - (1.5 * IQR)$ or above $Q_3 + (1.5 * IQR)$.

Note that these terms (aside from **Distribution**) only apply to quantitative variables.

A.2 Normal distribution

In Figure A.1 we visualize three normal curves/distributions

- The black normal curve has mean $\mu = 0$ and standard deviation $\sigma = 1$.
- The blue normal curve has mean $\mu = 0$ and standard deviation $\sigma = 3$.
- The orange normal curves has mean $\mu = 7$ and standard deviation $\sigma = 1$.

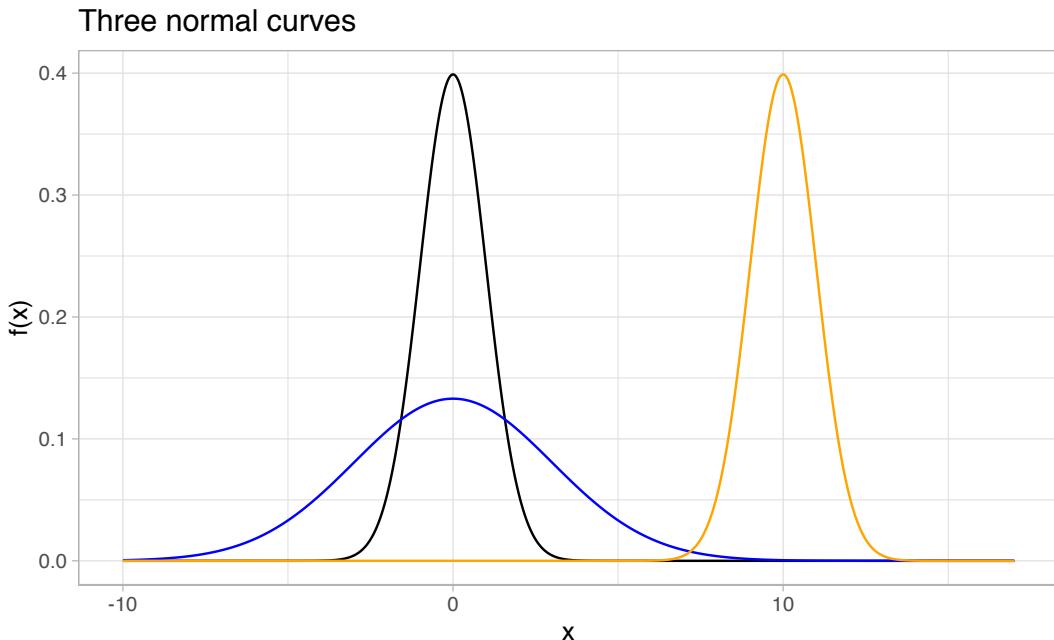


FIGURE A.1: Three examples of normal curves.

Some notes about these curves:

- The black curve is a specific case of normal distribution called the **standard normal AKA z-curve AKA z-distribution**
- The blue normal curve has the same center as the black curve, but more spread/variation.
- The orange normal curve has a different center than the black curve, but same spread/variation.

In Figure A.2, we illustrate some useful “rules of thumb” of how values that form a normal curve *distribute*:

1. 68.27% of values lie within ± 1 standard deviation σ of mean μ i.e. between $(\mu - \sigma, \mu + \sigma)$
2. 95.45% of values lie within ± 2 standard deviations σ of mean μ i.e. between $(\mu - 2\sigma, \mu + 2\sigma)$
3. 99.73% of values lie within ± 3 standard deviations σ of mean μ i.e. between $(\mu - 3\sigma, \mu + 3\sigma)$

Some notes:

1. So about two-thirds of values that follow a bell-curve are within ± 1 standard deviation σ of mean μ .

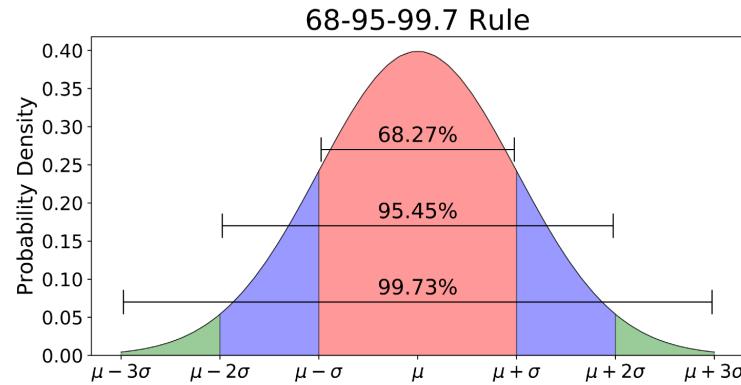


FIGURE A.2: Normal rules.

2. You might also see a similar statement of: 95% of values lie within ± 1.96 standard deviations σ of mean μ
3. Only $100\% - 99.73\% = 0.27\%$ of values lie outside of the interval $(\mu - 3\sigma, \mu + 3\sigma)$. In other words, almost all values lie within the interval that's 6σ lengths wide. This is where the term “six sigma”¹ from manufacturing reliability originates.

¹https://en.wikipedia.org/wiki/Six_Sigma

Bibliography

- Chihara, L. M. and Hesterberg, T. C. (2011). *Mathematical Statistics with Resampling and R*. John Wiley and Sons, Hoboken, NJ.
- Diez, D. M., Barr, C. D., and Çetinkaya Rundel, M. (2014). *Introductory Statistics with Randomization and Simulation*. First edition edition.
- Grolemund, G. and Wickham, H. (2016). *R for Data Science*.
- Ismay, C. (2016). *Getting used to R, RStudio, and R Markdown*.
- Robbins, N. (2013). *Creating More Effective Graphs*. Chart House.
- Wickham, H. (2014). Tidy data. *Journal of Statistical Software*, Volume 59(Issue 10).
- Wickham, H. (2018). *nycflights13: Flights that Departed NYC in 2013*. R package version 1.0.0.
- Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., Woo, K., and Yutani, H. (2019). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.2.0.
- Wickham, H. and Henry, L. (2019). *tidyverse: Easily Tidy Data with 'spread()' and 'gather()' Functions*. R package version 0.8.3.
- Wilkinson, L. (2005). *The Grammar of Graphics (Statistics and Computing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.
- Xie, Y. (2019). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.12.

Index

- Abelson, Hal, 4
adding transparency to plots, 46
- Baggerly, Keith, 6
barplot
 faceted, 78
 side-by-side, 77
 stacked, 75
- Bechdel, Alison, 442
- bias, 253
- bivariate, 150
- bookdown, 11
- Boolean algebra, 19
- bootstrap, 277
 colloquial definition, 289
 distribution, 277
 approximation of sampling distribution, 325
 resamples, 286
 statistical reference, 290
- boxplots, 62
 side-by-side, 65
 whiskers, 64
- Bryan, Jenny, 426
- categorical, 28
- Central Limit Theorem, 257
- Cobb, George, 8
- colors(), 56
- computational reproducibility, 7
- computer simulation, 227
- conditionals, 18
- confidence interval, 286
 analogy to fishing, 285
 impact of confidence level on interval width, 313
- impact of sample size on interval width, 314
interpretation, 309
- confidence level, 286
- console, 18
- correlation (coefficient), 150
- CSV file, 120
- data analysis, 5
 exploratory, 146
- data frame, 119
- data frames, 18, 25
- data science pipeline, 8
- data types, 18
- degrees of freedom, 381
- distribution, 37, 53, 225
 normal, 257
 standard normal, 259
- dplyr
 arrange(), 104
 cheatsheet, 116
 desc(), 106
 filter, 88
 filter(), 51
 glimpse(), 28
 group_by(), 95
 inner_join(), 107
 by, 109
 mutate(), 101
 n(), 98
 rename(), 113
 sample_n(), 148
 select(), 112
 summarize(), 92
 top_n(), 114
 ungroup(), 97

factor(), 66
 factors, 18
 five named graphs, 41, 80
 frequencies, 68
 functions, 19
 argument order, 81
 na.rm argument, 93
 wrapper, 158

 generalizability, 253
 geom_histogram()
 bins, 58
 binwidth, 58
 ggplot2
 +, 44
 aes(), 39
 alpha, 46
 cheatsheet, 81
 data, 39
 diamonds, 96
 facet, 41
 facet_wrap(), 60, 166
 fill, 56
 geom, 40
 geom_bar(), 69
 geom_col(), 70
 geom_histogram(), 55, 103
 geom_jitter(), 47
 geom_line(), 52, 135
 geom_point(), 49
 geom_smooth(), 153
 ggplot(), 41, 43
 mapping, 43
 position, 41, 77
 GitHub issues, 10
 Grammar of Graphics, The, 38
 Grolemund, Garrett, 5, 103

 heteroskedasticity, 411
 Hickey, Walt, 442
 histograms, 54
 bins, 54
 hypothesis, 347

hypothesis test, 348
 one-sided alternative, 348
 two-sided alternative, 348
 hypothesis testing
 alternative hypothesis, 348
 null distribution, 349
 null hypothesis, 348
 observed test statistic, 349
 p-value, 349
 reject the null hypothesis, 350
 significance level, 350
 test statistic, 349
 tradeoff between alpha and beta, 368
 Type I error, 366
 Type II error, 366
 US criminal trial analogy, 364

 infer
 calculate(), 295, 356
 generate(), 293, 355
 get_confidence_interval(), 299
 get_p_value(), 359
 hypothesize(), 353
 observed statistic shortcut, 291
 rep_sample_n(), 280
 shade_confidence_interval(), 299
 shade_p_value(), 357
 specify(), 292, 352
 switching between tests and confidence intervals, 360
 visualize(), 296, 357
 interquartile range (IQR), 64

 joining data
 key variable, 107

 knitr
 kable(), 30

 Lambert, Diane, 426
 levels, 68
 linegraphs, 50

- literate programming, 4
- lm(), 157
- long data format, 126
- margin of error, 332
- mean(), 93
- missing values, 93
- moderndive
 - get_correlation(), 150
 - get_regression_points(), 183
 - get_regression_table(), 156, 395
 - mythbusters_yawn, 315
 - pennies_sample, 266
- objects, 18
- observational unit, 147
- Occam's Razor, 209
- operator
 - assignment (<-), 43
 - in, 91
 - or, 90
- operators, 89
 - ==, 89
 - ?, 31
 - dollar sign, 30
 - logical, 19
 - not, 90
 - pipe, 87
- outliers, 67
- overplotting, 46
- p-hacking, 390
- permutation, 346
- pie chart, 73
- pie charts
 - problems with, 73
- plots, 37
- point estimate, 253
- programming language basics, 18
- quantitative, 28
- R, 15
 - errors, 19
- formula notation, 150
- installation, 16
- messages, 19
- packages, 21
- warnings, 19
- R Markdown, 11
- R packages, 21
 - broom
 - augment(), 183
 - tidy(), 182
 - dplyr, 21
 - fivethirtyeight, 443
 - gapminder
 - gapminder, 162
 - ggplot2, 21, 38
 - infer, 21
 - installation, 23
 - ISLR
 - Credit, 201
 - janitor
 - clean_names(), 182
 - loading, 24
 - loading error, 24
 - moderndive, 21, 145
 - nycflights13, 26, 42, 51, 88, 388
 - readr
 - cheatsheet, 138
 - read_csv, 121
 - skimr
 - skim(), 149
 - tidyr, 129
 - cheatsheet, 138
 - utils
 - View(), 28
- regression
 - basic, 144
 - conditions for inference (LINE), 404
- equation of a line, 155
 - intercept, 156
 - slope, 156
- fitted value, 155

- interpretation of the slope, 157
 - line, 154
 - linear, 144
 - multiple linear, 186
 - interactions model, 194
 - parallel slopes model, 196
 - observed values, 174
 - plane, 206
 - residual, 160
 - simple linear, 146, 394
 - resampling, 270
 - Robinson, David, 9
 - RStudio, 15
 - import data, 122
 - installation, 16
 - sample statistic, 253
 - sampling, 226, 252
 - census, 252
 - population, 252
 - population parameter, 252
 - random, 253
 - representative, 253
 - variation, 227
 - with replacement, 279
 - sampling distribution
 - relationship to sample size, 246
 - sampling distributions, 245
 - scatterplots, 42
 - sd(), 93
 - Simpson's Paradox, 216
 - skew, 166, 267
 - standard deviation, 240
 - standard error, 246, 277, 398
 - statistics, 5
 - tidy data, 126
 - tidyrr
 - gather(), 129
 - two-sample t-statistic, 381
 - univariate, 150
 - using == instead of =, 43
- variables
- confounding, 176
 - explanatory/predictor/independent, 143
 - response/outcome/dependent, 143
- vectors, 18
- Wickham, Hadley, 5, 103, 126
- wide data format, 126
- Wilkinson, Leland, 37
- Xie, Yihui, 7, 11
- z-score, 380