

Chester Ismay and Albert Y. Kim

Statistical Inference via Data Science

Contents

List of Tables	xiii
List of Figures	xv
1 Introduction	1
1.1 Introduction for students	2
1.1.1 What you will learn from this book	4
1.1.2 Data/science pipeline	5
1.1.3 Reproducible research	7
1.1.4 Final note for students	8
1.2 Introduction for instructors	8
1.2.1 Who is this book for?	8
1.3 DataCamp	10
1.4 Connect and contribute	13
1.5 About this book	14
1.6 About the authors	15
2 Getting Started with Data in R	17
2.1 What are R and RStudio?	17
2.1.1 Installing R and RStudio	18
2.1.2 Using R via RStudio	19
2.2 How do I code in R?	20
2.2.1 Basic programming concepts and terminology	20
2.2.2 Errors, warnings, and messages	22
2.2.3 Tips on learning to code	23
2.3 What are R packages?	24
2.3.1 Package installation	25
2.3.2 Package loading	27
2.3.3 Package use	27
2.4 Explore your first datasets	28
2.4.1 <code>nycflights13</code> package	28
2.4.2 <code>flights</code> data frame	29
2.4.3 Exploring data frames	30
2.4.4 Identification & measurement variables	33
2.4.5 Help files	34

2.5	Conclusion	35
2.5.1	Additional resources	35
2.5.2	What's to come?	35
I	Data Science via the tidyverse	37
3	Data Visualization	39
3.1	The Grammar of Graphics	40
3.1.1	Components of the Grammar	40
3.1.2	Gapminder data	41
3.1.3	Other components	43
3.1.4	ggplot2 package	43
3.2	Five Named Graphs - The 5NG	44
3.3	5NG#1: Scatterplots	44
3.3.1	Scatterplots via geom_point	45
3.3.2	Over-plotting	48
3.3.3	Summary	52
3.4	5NG#2: Linegraphs	52
3.4.1	Linegraphs via geom_line	54
3.4.2	Summary	55
3.5	5NG#3: Histograms	55
3.5.1	Histograms via geom_histogram	57
3.5.2	Adjusting the bins	60
3.5.3	Summary	62
3.6	Facets	62
3.7	5NG#4: Boxplots	64
3.7.1	Boxplots via geom_boxplot	67
3.7.2	Summary	70
3.8	5NG#5: Barplots	70
3.8.1	Barplots via geom_bar or geom_col	71
3.8.2	Must avoid pie charts!	74
3.8.3	Two categorical variables	76
3.8.4	Summary	80
3.9	Conclusion	82
3.9.1	Summary table	82
3.9.2	Argument specification	83
3.9.3	Additional resources	83
3.9.4	What's to come	84
4	Data Wrangling	87
4.1	The pipe operator: %>%	89
4.2	filter rows	90

4.3	<code>summarize</code> variables	93
4.4	<code>group_by</code> rows	96
4.4.1	Grouping by more than one variable	100
4.5	<code>mutate</code> existing variables	102
4.6	<code>arrange</code> and sort rows	106
4.7	<code>join</code> data frames	108
4.7.1	Matching “key” variable names	109
4.7.2	Different “key” variable names	110
4.7.3	Multiple “key” variables	112
4.7.4	Normal forms	112
4.8	Other verbs	113
4.8.1	<code>select</code> variables	113
4.8.2	<code>rename</code> variables	115
4.8.3	<code>top_n</code> values of a variable	116
4.9	Conclusion	117
4.9.1	Summary table	117
4.9.2	Additional resources	118
4.9.3	What’s to come?	119
5	Data Importing & “Tidy” Data	121
5.1	Importing data	122
5.1.1	Using the console	123
5.1.2	Using RStudio’s interface	124
5.2	Tidy data	125
5.2.1	Definition of “tidy” data	128
5.2.2	Converting to “tidy” data	131
5.2.3	<code>nycflights13</code> package	134
5.3	Case study: Democracy in Guatemala	135
5.4	Conclusion	139
5.4.1	<code>tidyverse</code> package	139
5.4.2	Additional resources	140
5.4.3	What’s to come?	141
II	Data Modeling via <code>moderndive</code>	143
6	Basic Regression	145
6.1	One numerical explanatory variable	147
6.1.1	Exploratory data analysis	148
6.1.2	Simple linear regression	157
6.1.3	Observed/fitted values and residuals	161
6.2	One categorical explanatory variable	164
6.2.1	Exploratory data analysis	164

6.2.2	Linear regression	171
6.2.3	Observed/fitted values and residuals	175
6.3	Related topics	177
6.3.1	Correlation is not necessarily causation	177
6.3.2	Best fitting line	179
6.3.3	<code>get_regression_x()</code> functions	183
6.4	Conclusion	185
6.4.1	Additional resources	185
6.4.2	What's to come?	186
7	Multiple Regression	187
7.1	One numerical & one categorical explanatory variable	188
7.1.1	Exploratory data analysis	189
7.1.2	Interaction model	192
7.1.3	Parallel slopes model	196
7.1.4	Observed/fitted values and residuals	199
7.2	Two numerical explanatory variables	202
7.2.1	Exploratory data analysis	203
7.2.2	Regression plane	208
7.2.3	Observed/fitted values and residuals	210
7.3	Related topics	211
7.3.1	Model selection	211
7.3.2	Correlation coefficient	215
7.3.3	Simpson's Paradox	215
7.4	Conclusion	218
7.4.1	Additional resources	218
7.4.2	What's to come?	218
III	Statistical inference via infer	219
8	Sampling	221
8.1	Sampling activity	221
8.1.1	What proportion of this bowl's balls are red?	221
8.1.2	Using the shovel once	222
8.1.3	Using the shovel 33 times	223
8.1.4	What are we doing here?	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	233
8.2.3	Using the virtual shovel 1000 times	235
8.2.4	Using different shovels	236
8.3	Sampling framework	240

8.3.1	Terminology & notation	241
8.3.2	Statistical definitions	244
8.3.3	The moral of the story	246
8.4	Case study: Polls	249
8.5	Conclusion	252
8.5.1	Central Limit Theorem	252
8.5.2	Summary table	252
8.5.3	Additional resources	254
8.5.4	What's to come?	254
9	Confidence Intervals	255
9.1	Resampling activity	256
9.1.1	What is the average year of circulated US pennies in 2019?	256
9.1.2	Using resampling once	260
9.1.3	Using resampling 33 times	263
9.1.4	What's the plan?	265
9.2	Computer simulation of resampling	267
9.2.1	Using the virtual resample once	267
9.2.2	Using the virtual resample 33 times	268
9.2.3	Using the virtual resample 1000 times	270
9.3	Confidence interval build-up	271
9.3.1	The percentile method	272
9.3.2	The standard error method	273
9.4	The bootstrapping framework	274
9.4.1	The original workflow needed for this	275
9.4.2	The <code>infer</code> package for statistical inference	276
9.4.3	Building confidence intervals with the <code>infer</code> package	284
9.4.4	The percentile method with <code>infer</code>	284
9.4.5	The standard error method with <code>infer</code>	287
9.5	Case study: Revisiting the red ball example	289
9.5.1	Observed statistic	289
9.5.2	Bootstrap distribution for one proportion	290
9.6	Interpreting the confidence interval	292
9.7	Case study: Comparing two proportions	301
9.7.1	Compute the point estimate	302
9.7.2	Bootstrap distribution	304
9.8	Conclusion	307
9.8.1	Comparing bootstrap and sampling distributions	307
9.8.2	Theory-based confidence intervals	313
9.8.3	Summary table	321
9.8.4	Additional resources	322

9.8.5 What's to come?	323
10 Hypothesis Testing	325
10.1 Hypothesis testing activity	326
10.1.1 Question of interest	326
10.1.2 What did we actually observe?	327
10.1.3 Using permuting once	329
10.1.4 Using permuting 33 times	331
10.2 Hypothesis testing with <code>infer</code>	332
10.2.1 Revisiting the <code>infer</code> verb framework	332
10.2.2 The <code>infer</code> pipeline for the activity	333
10.2.3 The “There Is Only One Test” framework	337
10.3 The p-value	339
10.3.1 Corresponding confidence interval	340
10.3.2 Summary	341
10.4 Interpretation of hypothesis testing results	342
10.4.1 Criminal trial analogy	342
10.4.2 Types of errors in hypothesis testing	343
10.4.3 Statistical significance	346
10.5 Case study: comparing two means	346
10.5.1 Randomization/permutation	346
10.5.2 Comparing action and romance movies	347
10.5.3 Sampling → randomization	350
10.5.4 Data	350
10.5.5 Model of H_0	353
10.5.6 Test statistic δ	353
10.5.7 Observed effect δ^*	354
10.5.8 Simulated data	354
10.5.9 Distribution of δ under H_0	355
10.5.10 The p-value	357
10.5.11 Corresponding confidence interval	359
10.6 Conclusion	360
10.6.1 When inference is not needed	360
10.6.2 Problems with p-values	363
10.6.3 Comparing confidence intervals and hypothesis tests .	364
10.6.4 Summary table	364
10.6.5 Building theory-based methods using computation .	365
10.6.6 Additional resources	372
10.6.7 What's to come	372
11 Inference for Regression	375
11.1 Simulation-based Inference for Regression	375

<i>Contents</i>	ix
11.1.1 Data	376
11.1.2 Test statistic δ	376
11.1.3 Observed effect δ^*	377
11.1.4 Model of H_0	377
11.1.5 Simulated data	377
11.1.6 Distribution of δ under H_0	378
11.1.7 The p-value	379
11.2 Bootstrapping for the regression slope	379
11.3 Inference for multiple regression	381
11.3.1 Refresher: Professor evaluations data	381
11.3.2 Refresher: Visualizations	382
11.3.3 Refresher: Regression tables	383
11.3.4 Script of R code	384
11.4 Residual analysis	384
11.4.1 Residual analysis	384
11.4.2 Residual analysis	389
11.4.3 Residual analysis	392
11.4.4 Residual analysis	395
IV Conclusion	399
12 Thinking with Data	401
12.1 Case study: Seattle house prices	404
12.1.1 Exploratory data analysis (EDA)	405
12.1.2 \log_{10} transformations	408
12.1.3 EDA Part II	412
12.1.4 Regression modeling	415
12.1.5 Making predictions	416
12.2 Case study: Effective data storytelling	418
12.2.1 Bechdel test for Hollywood gender representation . . .	418
12.2.2 US Births in 1999	419
12.2.3 Other examples	421
12.2.4 Script of R code	421
Appendix	423
A Statistical Background	423
A.1 Basic statistical terms	423
A.1.1 Mean	423
A.1.2 Median	423
A.1.3 Standard deviation	423
A.1.4 Five-number summary	424
A.1.5 Distribution	424

A.1.6	Outliers	424
A.2	Normal distribution discussion	424
B	Inference Examples	425
B.1	Inference mind map	426
B.2	One mean	426
B.2.1	Problem statement	426
B.2.2	Competing hypotheses	426
B.2.3	Exploring the sample data	428
B.2.4	Non-traditional methods	430
B.2.5	Traditional methods	434
B.2.6	Comparing results	437
B.3	One proportion	437
B.3.1	Problem statement	437
B.3.2	Competing hypotheses	437
B.3.3	Exploring the sample data	438
B.3.4	Non-traditional methods	439
B.3.5	Traditional methods	443
B.3.6	Comparing results	446
B.4	Two proportions	447
B.4.1	Problem statement	447
B.4.2	Competing hypotheses	447
B.4.3	Exploring the sample data	448
B.4.4	Non-traditional methods	449
B.4.5	Traditional methods	453
B.4.6	Check conditions	453
B.4.7	Test statistic	453
B.4.8	State conclusion	454
B.4.9	Comparing results	455
B.5	Two means (independent samples)	455
B.5.1	Problem statement	455
B.5.2	Competing hypotheses	455
B.5.3	Exploring the sample data	456
B.5.4	Non-traditional methods	458
B.5.5	Traditional methods	462
B.5.6	Test statistic	463
B.5.7	Compute p -value	464
B.5.8	State conclusion	464
B.5.9	Comparing results	464
B.6	Two means (paired samples)	465
B.6.1	Competing hypotheses	465
B.6.2	Exploring the sample data	466

B.6.3	Non-traditional methods	467
B.6.4	Traditional methods	471
B.6.5	Comparing results	473
C	Reach for the Stars	475
C.1	Sorted barplots	475
C.2	Interactive graphics	477
	C.2.1 Interactive linegraphs	477
D	Learning Check Solutions	479
D.1	Chapter 2 Solutions	479
D.2	Chapter 3 Solutions	481
D.3	Chapter 4 Solutions	490
D.4	Chapter 5 Solutions	495
D.5	Chapter 6 Solutions	498

List of Tables

3.1	Gapminder 2007 Data: First 6 of 142 countries	41
3.2	Summary of Grammar of Graphics for this plot	42
3.3	Number of flights pre-counted for each carrier.	73
3.4	Summary of 5NG	82
4.1	Summary of data wrangling verbs	117
5.1	Stock Prices (Non-Tidy Format)	129
5.2	Stock Prices (Tidy Format)	130
5.3	Date, Boeing Price, Weather Data	130
6.1	A random sample of 5 out of the 463 courses at UT Austin . .	150
6.2	Linear regression table	158
6.3	Data for the 21st course out of 463	161
6.4	Regression points (for only the 21st through 24th courses) . .	163
6.5	Random sample of 5 out of 142 countries	166
6.6	Life expectancy by continent	170
6.7	Mean life expectancy by continent and relative differences from mean for Africa.	170
6.8	Linear regression table	172
6.9	Regression points (First 10 out of 142 countries)	176
7.1	A random sample of 5 out of the 463 courses at UT Austin . .	190
7.2	Regression table for life expectancy as a function of continent.	193
7.3	Regression table for interaction model.	194
7.4	Comparison of female and male intercepts and age slopes . .	194
7.5	Regression table for parallel slopes model.	198
7.6	Regression points (First 10 out of 463 courses)	201
7.7	Random sample of 5 credit card holders.	204
7.8	Correlation coefficients between credit card debt, credit limit, and income.	206
7.9	Multiple regression table	209
7.10	Regression points (First 10 card holders of 400)	211
7.11	Interaction model regression table	214
7.12	Parallel slopes regression table	214
7.13	Correlation between income (in dollars) and credit card debt .	215

8.1	First 10 out of 33 groups' proportion of 50 balls that are red.	225
8.3	First 10 sampled balls of 50 in virtual sample	230
8.4	First 10 out of 33 virtual proportion of 50 balls that are red.	233
8.6	Comparing standard deviations of proportions red for 3 different shovels.	240
8.7	Three standard errors of the sample proportion based on n = 25, 50, 100.	246
8.8	Scenarios of sampling for inference	253
9.1	First 10 out of 33 friends' mean age of 50 resampled pennies.	264
9.2	First 10 resampled rows of 50 in virtual sample	267
9.3	First 10 out of 33 means from virtual resamples	269
9.5	10 randomly sampled confidence intervals for p for varying confidence levels	297
9.6	33 confidence intervals from 33 tactile samples of size n=50	315
9.7	Scenarios of sampling for inference	322
10.1	First 10 rows of original (left) and permuted (right) data	330
10.4	Scenarios of sampling for inference	365
11.1	Model 1: Regression table with no interaction effect included	383
11.2	Model 2: Regression table with interaction effect included	384
11.3	Countries in Asia with shortest life expectancy	391
12.1	log10-transformed prices, orders of magnitude, and examples	409

List of Figures

1.1	ModernDive Flowchart	3
1.2	Data/Science Pipeline	6
1.3	DataCamp logo	10
2.1	19
2.2	ModernDive flowchart	36
3.1	Life Expectancy over GDP per Capita in 2007	42
3.2	Arrival Delays vs Departure Delays for Alaska Airlines flights from NYC in 2013	46
3.3	Plot with No Layers	47
3.4	Delay scatterplot with alpha=0.2	49
3.5	Regular scatterplot of jitter example data	50
3.6	Jittered scatterplot of jitter example data	51
3.7	Jittered delay scatterplot	51
3.8	Hourly Temperature in Newark for January 1-15, 2013	54
3.9	Plot of Hourly Temperature Recordings from NYC in 2013	56
3.10	Example histogram.	57
3.11	Histogram of hourly temperatures at three NYC airports.	58
3.12	Histogram of hourly temperatures at three NYC airports with white borders.	59
3.13	Histogram of hourly temperatures at three NYC airports with white borders.	59
3.14	Histogram with 40 bins.	60
3.15	Histogram with binwidth 10.	61
3.16	Faceted histogram.	63
3.17	Faceted histogram with 4 instead of 3 rows.	63
3.18	November temperatures.	65
3.19	November temperatures.	65
3.20	November temperatures.	66
3.21	November temperatures.	67
3.22	Invalid boxplot specification	68
3.23	Month by temp boxplot	68
3.24	Barplot when counts are not pre-counted	72
3.25	Barplot when counts are pre-counted	72

3.26 Number of flights departing NYC in 2013 by airline using geom_bar	73
3.27 The dreaded pie chart	75
3.28 The only good pie chart	76
3.29 Stacked barplot comparing the number of flights by carrier and origin	77
3.30 Stacked barplot with color aesthetic used instead of fill.	78
3.31 Side-by-side AKA dodged barplot comparing the number of flights by carrier and origin.	79
3.32 Faceted barplot comparing the number of flights by carrier and origin.	81
3.33 Data Visualization with ggplot2 cheatsheat	84
4.1 Diagram of	90
4.2 Summarize diagram from Data Wrangling with dplyr and tidyr cheatsheet	94
4.3 Another summarize diagram from Data Wrangling with dplyr and tidyr cheatsheet	94
4.4 Group by and summarize diagram from Data Wrangling with dplyr and tidyr cheatsheet	97
4.5 Mutate diagram from Data Wrangling with dplyr and tidyr cheatsheet	102
4.6 Histogram of gain variable	105
4.7 Data relationships in nycflights13 from R for Data Science	109
4.8 Diagram of inner join from R for Data Science	110
4.9 Select diagram from Data Wrangling with dplyr and tidyr cheatsheet	114
4.10 Data Transformation with dplyr cheatsheat	119
5.1	124
5.2 Alcohol consumption in 4 countries.	126
5.3 Tidy data graphic from [R for Data Science](http://r4ds.had.co.nz/tidy-data.html).	129
5.4 Data Import cheatsheat	140
5.5 ModernDive flowchart - On to Part II!	141
6.1 Different correlation coefficients	152
6.2 Instructor evaluation scores at UT Austin	154
6.3 Instructor evaluation scores at UT Austin	155
6.4 Regression line	156
6.5 The concept of a 'wrapper' function.	160
6.6 Example of observed value, fitted value, and residual	162
6.7 Histogram of Life Expectancy in 2007	167

6.8	Life expectancy in 2007	168
6.9	Life expectancy in 2007	169
6.10	Does sleeping with shoes on cause headaches?	177
6.11	Causal graph.	178
6.12	Example of observed value, fitted value, and residual	180
7.1	Colored scatterplot of relationship of teaching and beauty scores	192
7.2	Parallel slopes model of relationship of score with age and gender.	197
7.3	Comparison of interaction and parallel slopes models.	200
7.4	Fitted values for two new professors	201
7.5	Relationship between credit card debt and credit limit/income	207
7.6	Comparison of interaction and parallel slopes models.	213
7.7	Relationship between credit card debt and credit limit/income	216
7.8	Histogram of credit limits and quartiles	217
7.9	Relationship between credit card debt and income for different credit limit groups	217
8.1	A bowl with red and white balls.	222
8.2	Inserting a shovel into the bowl.	223
8.3	Fifty balls from the bowl.	223
8.4	Repeating sampling activity 33 times.	224
8.5	Constructing a histogram of proportions.	224
8.6	Hand-drawn histogram of 33 proportions.	225
8.7	Distribution of 33 proportions based on 33 samples of size 50	227
8.8	Distribution of 33 proportions based on 33 samples of size 50	234
8.9	Comparing 33 virtual and 33 tactile proportions red.	235
8.10	Distribution of 1000 proportions based on 33 samples of size 50	237
8.11	Comparing the distributions of proportion red for different sample sizes	239
8.12	Three sampling distributions of the sample proportion \hat{p}	245
8.13	Three sampling distributions with population proportion p marked in red.	247
8.14	Comparing accuracy and precision	248
9.1	50 US pennies	257
9.2	50 US pennies labelled	257
9.3	50 resampled US pennies labelled	261
9.4	Comparing years of original sample <code>pennies_sample_2</code> and resample <code>pennies_resample</code>	262
9.5	Constructing a histogram of means from resamples.	264
9.6	Distribution of 33 means based on 33 resamples of size 50	266

9.7 Comparing distributions of means from resamples‘	270
9.8 Reliability of 95 percent confidence intervals	295
9.9 Reliability of 90 percent confidence intervals	295
9.10 Sampling distribution for proportion red for n=200 samples of balls	308
9.11 Comparing sampling and bootstrap distributions	312
9.12 33 confidence intervals based on 33 tactile samples of size n=50	317
9.13 100 confidence intervals based on 100 virtual samples of size n=50	319
10.1 Hypothesis Testing Framework	338
10.2 Shaded histogram to show p-value	339
10.3 Type I and Type II errors	344
10.4 Rating vs genre in the population	349
10.5 Faceted histogram of genre vs rating	349
10.6 Genre vs rating for our sample	351
10.7 Genre vs rating for our sample as faceted histogram	352
10.8 Simulated differences in means histogram	356
10.9 Shaded histogram to show p-value	357
10.10 Histogram with vertical lines corresponding to observed statistic	358
10.11 Simulated differences in means histogram	369
11.1 Model 1: no interaction effect included	382
11.2 Model 2: interaction effect included	383
11.3 Plot of residuals over beauty score	386
11.4 Examples of less than ideal residual patterns	387
11.5 Histogram of residuals	387
11.6 Examples of ideal and less than ideal residual patterns	388
11.7 Plot of residuals over continent	390
11.8 Histogram of residuals	391
11.9 Residuals vs credit limit and income	393
11.10 Relationship between credit card balance and credit limit/income	394
11.11 Interaction model histogram of residuals	396
11.12 Interaction model residuals vs predictor	397
12.1 ModernDive Flowchart	402
12.2 Data/Science Pipeline	403
12.3 Exploratory visualizations of Seattle house prices data	407
12.4 House price before and after log10-transformation	411
12.5 House size before and after log10-transformation	412
12.6 Parallel slopes model	413

12.7 Interaction model	414
12.8 Interaction model with facets	414
12.9 Interaction model with prediction	417
B.1 Mind map for Inference	427
C.1 Number of flights departing NYC in 2013 by airline - Descending numbers	476
D.1	485

1

Introduction

Special Announcement

We're excited to announce that we've signed a book deal with CRC Press! We will be publishing our first fully complete online version of ModernDive in Summer 2019, with a corresponding print edition to follow in Fall 2019. Don't worry though, our content will always remain freely available on [ModernDive.com¹](#).



Please note that you are currently looking at the “development version” of ModernDive, which is a work in progress currently being edited and thus subject to frequent change. For the latest “released version” of ModernDive, which changes much less frequently, please visit [ModernDive.com²](#).

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 click [here](#) for more information on how to teach with this book.
- Are you looking to connect with and contribute to ModernDive? Then click [here](#) for information on how.
- Are you curious about the publishing of this book? Then click [here](#) for more information on the open-source technology, in particular R Markdown and the `bookdown` package.

This is version 0.5.0.9000 of ModernDive published on March 25, 2019. For previous versions of ModernDive, see Section 1.5. While a PDF version of this book can be found [here](#)³, this is very much a work in progress with many things that still need to be fixed. We appreciate your patience.

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.5: Understanding the concept of “tidy” data as a standardized data input format for all packages in the `tidyverse`
 - Ch.4: Wrangling data via the `dplyr` package.
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[ismaykim.pdf](#)

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.
 - Ch.10: Conducting hypothesis tests.
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.

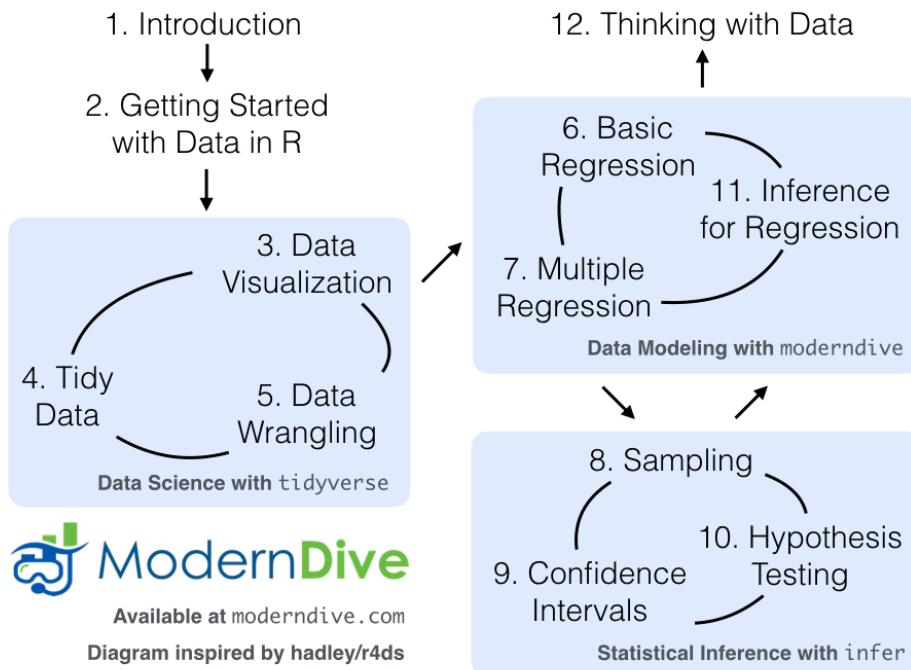


FIGURE 1.1: ModernDive Flowchart

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.
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?⁴ and How many f***ks does Quentin Tarantino give (as measured by the amount of swearing in his films)?⁵. Further discussions on data stories can be found in this Think With Google article⁶.

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

- Middlebury College MATH 116 Introduction to Statistical and Data Sciences⁷ using student collected data.
- Pacific University SOC 301 Social Statistics⁸ using data from the `fivethirtyeight` R package⁹.

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.

⁴http://rpubs.com/ry_lisa_elana/chicago

⁵https://ismayc.github.io/soc301_s2017/group_projects/group4.html

⁶<https://www.thinkwithgoogle.com/marketing-resources/data-measurement/tell-meaningful-stories-with-data/>

⁷https://rudeboybert.github.io/MATH116/PS/final_project/final_project_outline.html#past_examples

⁸https://ismayc.github.io/soc301_s2017/group_projects/index.html

⁹<https://cran.r-project.org/web/packages/fivethirtyeight/vignettes/fivethirtyeight.html>

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 program. 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.com¹⁰.

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

¹⁰<https://stackoverflow.com/>

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”¹¹ in Figure 1.2.

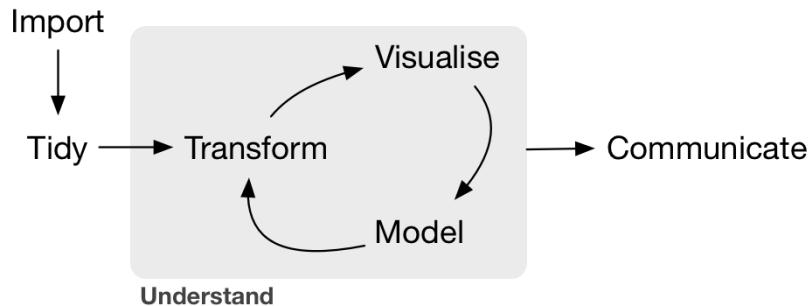


FIGURE 1.2: Data/Science Pipeline

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 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.

¹¹<http://r4ds.had.co.nz/explore-intro.html>

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 a 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” (Gromelund 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`¹² 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

¹²<http://tidyverse.org/>

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 pipeline¹³ should be taught.
- We believe in "minimizing prerequisites to research"¹⁴: students should be answering questions with data as soon as possible.

3. It's all about the data

- 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 statistics¹⁵ 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 Robinson¹⁶, are much more optimistic.
- `dplyr` has made data wrangling much more accessible¹⁷ 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.

¹³<http://r4ds.had.co.nz/introduction.html>

¹⁴<https://arxiv.org/abs/1507.05346>

¹⁵<http://escholarship.org/uc/item/84v3774z>

¹⁶http://varianceexplained.org/r/teach_ggplot2_to_beginners/

¹⁷<http://chance.amstat.org/2015/04/setting-the-stage/>

- This allows for a de-emphasis of traditional probability topics, freeing up room in the syllabus for other topics.
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](#).

1.3 DataCamp



FIGURE 1.3: DataCamp logo

DataCamp is a browser-based interactive platform for learning data science, offering courses on a wide array of courses on data science, analytics, statistics, machine learning, and artificial intelligence, where each course is a combination of lectures and exercises that offer immediate feedback.

The following chapters of ModernDive roughly map to the following closely-

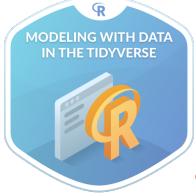
integrated DataCamp courses that use the same R tools and often even the same datasets. By no means is this an exhaustive list of possible DataCamp courses that are relevant to the topics in this book, we recommend these ones in particular to supplement your ModernDive experience.

Click on the image for each course to access its webpage on [datacamp.com¹⁸](https://www.datacamp.com/home). Instructors at accredited universities can sign their class up for a free academic licence at DataCamp For The Classroom¹⁹, giving their students access to all premium courses for 6 months for free.

Chapter	Topic	DataCamp Courses
2	Basic R programming concepts	 a
		 b
3 & 5	Introductory data visualization and wrangling	 a <hr/> ^ahttps://www.datacamp.com/courses/free-introduction-to-r ^bhttps://www.datacamp.com/courses/introduction-to-the-tidyverse

¹⁸<https://www.datacamp.com/home>

¹⁹<https://www.datacamp.com/groups/education>

Chapter	Topic	DataCamp Courses
4 & 5	Data “tidying” and intermediate data wrangling	 ^a https://www.datacamp.com/courses/working-with-data-in-the-tidyverse
6 & 7	Data modeling, basic regression, and multiple regression	 ^a https://www.datacamp.com/courses/modeling-with-data-in-the-tidyverse
9 & 10	Statistical inference: confidence intervals and hypothesis testing	 ^a https://www.datacamp.com/courses/inference-for-numerical-data  ^b https://www.datacamp.com/courses/inference-for-categorical-data

Chapter	Topic	DataCamp Courses
11	Inference for regression	 ^a https://www.datacamp.com/courses/inference-for-linear-regression

1.4 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 list²⁰.
- Contact Albert at [`albert.ys.kim@gmail.com`](mailto:albert.ys.kim@gmail.com)²¹ and Chester at [`chester.ismay@gmail.com`](mailto:chester.ismay@gmail.com)²².
- We're on Twitter at ModernDive²³.

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 issues²⁴ page.
- If you are familiar with GitHub and would like to contribute more, please see Section 1.5 below.

For example, we thank

²⁰<http://eepurl.com/cBkItf>

²¹<mailto:albert.ys.kim@gmail.com>

²²<mailto:chester.ismay@gmail.com>

²³<https://twitter.com/ModernDive>

²⁴https://github.com/moderndive/moderndive_book/issues

- Dr Andrew Heiss²⁵ for contributing Subsection 2.2.3 on “Errors, warnings, and messages”.

The authors would like to thank Nina Sonneborn²⁶, Kristin Bott²⁷, Dr. Jenny Smetzer²⁸, and the participants of our USCOTS 2017 workshop²⁹ for their feedback and suggestions. A special thanks goes to Dr. Yana Weinstein, cognitive psychological scientist and co-founder of The Learning Scientists³⁰, for her extensive contributions.

1.5 About this book

This book was written using RStudio’s bookdown³¹ package by Yihui Xie ([Xie, 2018](#)). This package simplifies the publishing of books by having all content written in R Markdown³². 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 code³³).
 - Available at [ModernDive.com](#)³⁴
- **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 page³⁵
- **Previous versions** Older versions that may be out of date:
 - Version 0.4.0³⁶ released on July 21, 2018 (source code³⁷)

²⁵<https://twitter.com/andrewheiss>

²⁶<https://github.com/nsonneborn>

²⁷<https://twitter.com/rhobott?lang=en>

²⁸<https://www.smith.edu/academics/faculty/jennifer-smetzer>

²⁹<https://www.causeweb.org/cause/uscotus/uscotus17/workshop/>

³⁰<http://www.learningscientists.org/yana-weinstein/>

³¹<https://bookdown.org/>

³²http://rmarkdown.rstudio.com/html_document_format.html

³³https://github.com/moderndive/moderndive_book/releases/tag/v0.5.0

³⁴<https://moderndive.com/>

³⁵https://github.com/moderndive/moderndive_book

³⁶[previous_versions/v0.4.0/index.html](https://github.com/moderndive/moderndive_book/tree/previous_versions/v0.4.0/index.html)

³⁷https://github.com/moderndive/moderndive_book/releases/tag/v0.4.0

- Version 0.3.0³⁸ released on February 3, 2018 (source code³⁹)
- Version 0.2.0⁴⁰ released on August 02, 2017 (source code⁴¹)
- Version 0.1.3⁴² released on February 09, 2017 (source code⁴³)
- Version 0.1.2⁴⁴ released on January 22, 2017 (source code⁴⁵)

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!”

1.6 About the authors

Who we are!

³⁸[previous_versions/v0.3.0/index.html](#)

³⁹https://github.com/moderndive/moderndive_book/releases/tag/v0.3.0

⁴⁰[previous_versions/v0.2.0/index.html](#)

⁴¹https://github.com/moderndive/moderndive_book/releases/tag/v0.2.0

⁴²[previous_versions/v0.1.3/index.html](#)

⁴³https://github.com/moderndive/moderndive_book/releases/tag/v0.1.3

⁴⁴[previous_versions/v0.1.2/index.html](#)

⁴⁵https://github.com/moderndive/moderndive_book/releases/tag/v0.1.2

Chester Ismay



Albert Y. Kim



- Chester Ismay: Senior Curriculum Lead - DataCamp, Portland, OR, USA.
 - Email: chester.ismay@gmail.com⁴⁶
 - Webpage: <http://chester.rbind.io/>
 - Twitter: [old_man_chester](https://twitter.com/old_man_chester)⁴⁷
 - GitHub: <https://github.com/ismayc>
- Albert Y. Kim: Assistant Professor of Statistical & Data Sciences - Smith College, Northampton, MA, USA.
 - Email: albert.ys.kim@gmail.com⁴⁸
 - Webpage: <http://rudeboybert.rbind.io/>
 - Twitter: [rudeboybert](https://twitter.com/rudeboybert)⁴⁹
 - GitHub: <https://github.com/rudeboybert>

⁴⁶ <mailto:chester.ismay@gmail.com>

⁴⁷ https://twitter.com/old_man_chester

⁴⁸ <mailto:albert.ys.kim@gmail.com>

⁴⁹ <https://twitter.com/rudeboybert>

2

Getting Started with Data in R

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

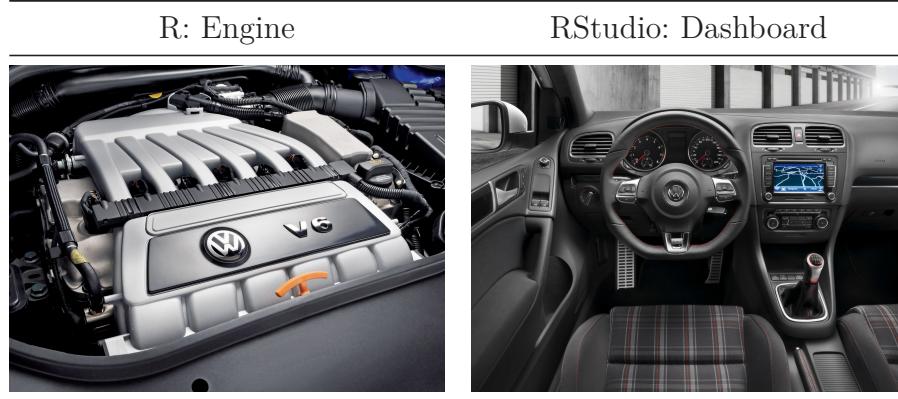
1. What are R and RStudio?
2. How do I code in R?
3. What are R packages?

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.
- RStudio is like a car's dashboard.



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.

If you are still not sure about the difference between R and RStudio IDE, we suggest you watch this DataCamp video¹.

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 R².
 - Click on the download link corresponding to your computer's operating system.
2. **You must do this second:** Download and install RStudio³.
 - Scroll down to "Installers for Supported Platforms"
 - Click on the download link corresponding to your computer's operating system.

If you had trouble with these two steps, we suggest you watch this DataCamp video⁴.

¹<https://campus.datacamp.com/courses/working-with-the-rstudio-ide-part-1/orientation?ex=1>

²<https://cran.r-project.org/>

³<https://www.rstudio.com/products/rstudio/download3/>

⁴<https://campus.datacamp.com/courses/working-with-the-rstudio-ide-part-1/orientation?ex=3>

2.1.2 Using R via RStudio

Recall our car analogy from above. 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. In other words:



After you open RStudio, you should see the following:

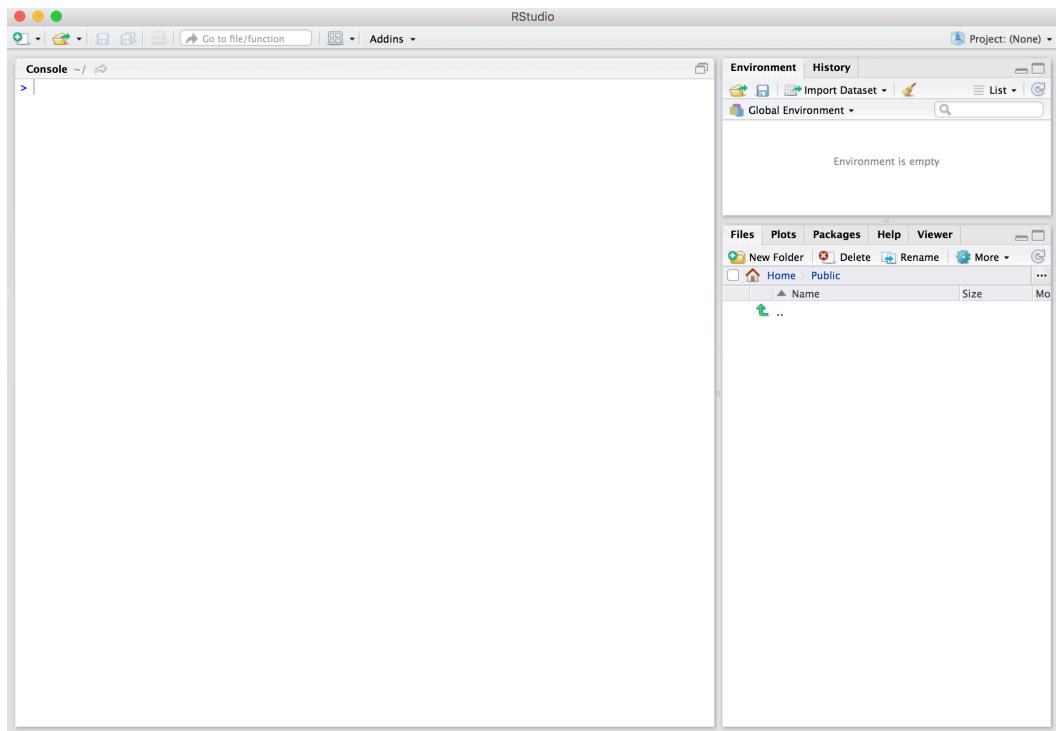


FIGURE 2.1

Note the three panes, which are three panels dividing the screen: The *Con-*

sole 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.

If however you would like an in depth explanation right now however, we suggest you watch following DataCamp video⁵.

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 statistical software programs like Excel, STATA, or SAS that provide point and click⁶ interfaces, R is an interpreted language⁷, meaning 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

To introduce you to many of these basic programming concepts and terminology, we direct you to the following DataCamp online interactive tutorials. For each of the tutorials, we give a list of the basic programming concepts covered. 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

⁵<https://campus.datacamp.com/courses/working-with-the-rstudio-ide-part-1/orientation?ex=5>

⁶https://en.wikipedia.org/wiki/Point_and_click

⁷https://en.wikipedia.org/wiki/Interpreted_language

of the entire book and we encourage this process as much as possible as you learn any new skill.

- From the Introduction to R⁸ course complete the following chapters. As you work through the chapters, carefully note the important terms and what they are used for. We recommend you do so in a notebook that you can easily refer back to.
 - Chapter 1 Intro to basics⁹:
 - * Console pane: where you enter in commands
 - * Objects: where values are saved, how to assign values to objects.
 - * Data types: integers, doubles/numerics, logicals, characters.
 - Chapter 2 Vectors¹⁰:
 - * 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)`.
 - Chapter 4 Factors¹¹:
 - * *Categorical data* (as opposed to *numerical data*) are represented in R as `factors`.
 - Chapter 5 Data frames¹²:
 - * 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.
- From the Intermediate R¹³ course complete the following chapters:
 - Chapter 1 Conditionals and Control Flow¹⁴:
 - * Testing for equality in R using `==` (and not `=` which is typically used for 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:

⁸<https://www.datacamp.com/courses/free-introduction-to-r>

⁹<https://campus.datacamp.com/courses/free-introduction-to-r/chapter-1-intro-to-basics-1?ex=1>

¹⁰<https://campus.datacamp.com/courses/free-introduction-to-r/chapter-2-vectors-2?ex=1>

¹¹<https://campus.datacamp.com/courses/free-introduction-to-r/chapter-4-factors-4?ex=1>

¹²<https://campus.datacamp.com/courses/free-introduction-to-r/chapter-5-data-frames?ex=1>

¹³<https://www.datacamp.com/courses/intermediate-r>

¹⁴<https://campus.datacamp.com/courses/intermediate-r/chapter-1-conditionals-and-control-flow?ex=1>

`(2 + 1 == 3) & (2 + 1 == 4)` returns FALSE while `(2 + 1 == 3) | (2 + 1 == 4)` returns TRUE.

- Chapter 3 Functions¹⁵:

- * Concept of functions: they 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 below 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 “Warning”, 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 below, or when you

¹⁵<https://campus.datacamp.com/courses/intermediate-r/chapter-3-functions?ex=1>

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:

R: A new phone

R Packages: Apps you can download



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.

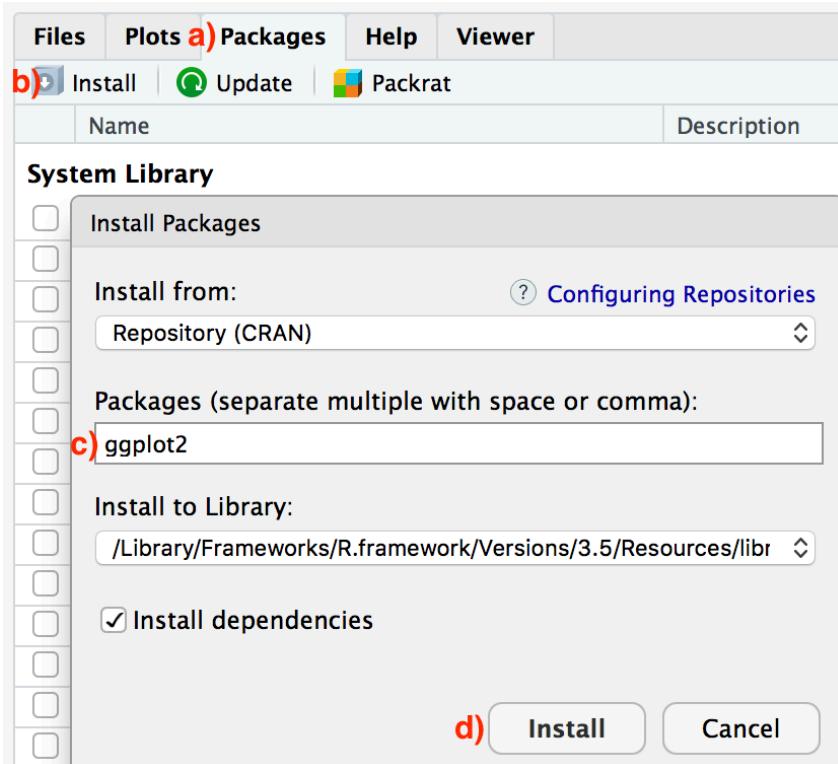
Let's now show you how to perform these two steps for the `ggplot2` package for data visualization.

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”
 - c) Type the name of the package under “Packages (separate multiple with space or comma):” In this case, type `ggplot2`
 - d) Click “Install”



2. **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 hitting enter. 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 above steps.

Learning check

(LC2.1) Repeat the above 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 above 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 above 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.” Almost all new users forget do this when starting out, and it is a little annoying to get used. 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 (Wickham, 2018). 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 `$` 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 below.

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 Markdown¹⁶, the latter code produces output that is much more legible and reader-friendly.

4. \$ operator

Lastly, the \$ operator allows us to 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)` below:

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

¹⁶<http://rmarkdown.rstudio.com/lesson-1.html>

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
```

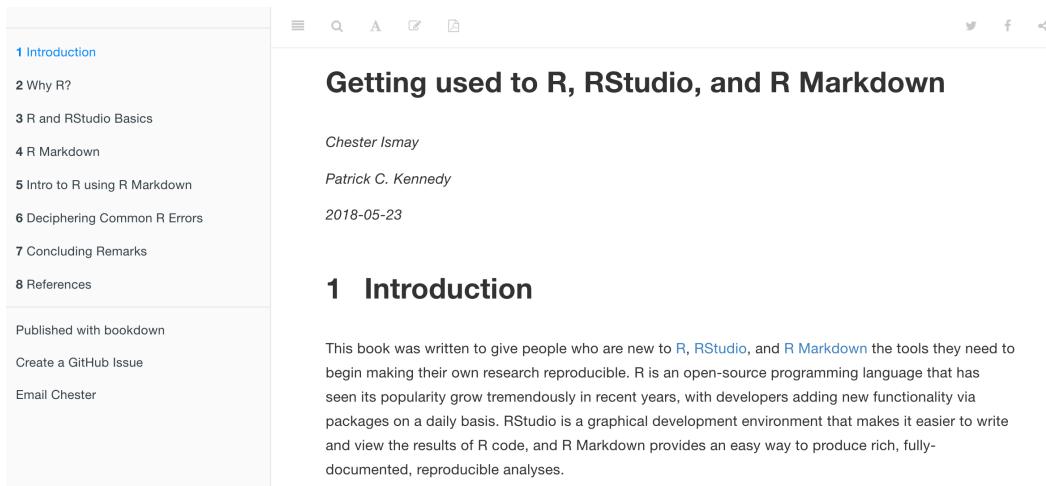
A help file should pop-up in the Help pane of RStudio. If you have questions about a function or data frame included in an R package, you should get in the habit of consulting the help file right away.

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 Markdown¹⁷ 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.



The screenshot shows a book page titled "Getting used to R, RStudio, and R Markdown" by Chester Ismay and Patrick C. Kennedy, published on 2018-05-23. The left sidebar contains a table of contents with chapters 1 through 8. The main content area includes a brief introduction and a note about the book's purpose.

1 Introduction
2 Why R?
3 R and RStudio Basics
4 R Markdown
5 Intro to R using R Markdown
6 Deciphering Common R Errors
7 Concluding Remarks
8 References

Published with bookdown
Create a GitHub Issue
Email Chester

Getting used to R, RStudio, and R Markdown
Chester Ismay
Patrick C. Kennedy
2018-05-23

1 Introduction

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.

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

¹⁷<https://rbasics.netlify.com/>

to add to our toolbox for data exploring that provides additional insight to what the `view()` and `glimpse()` functions can provide.

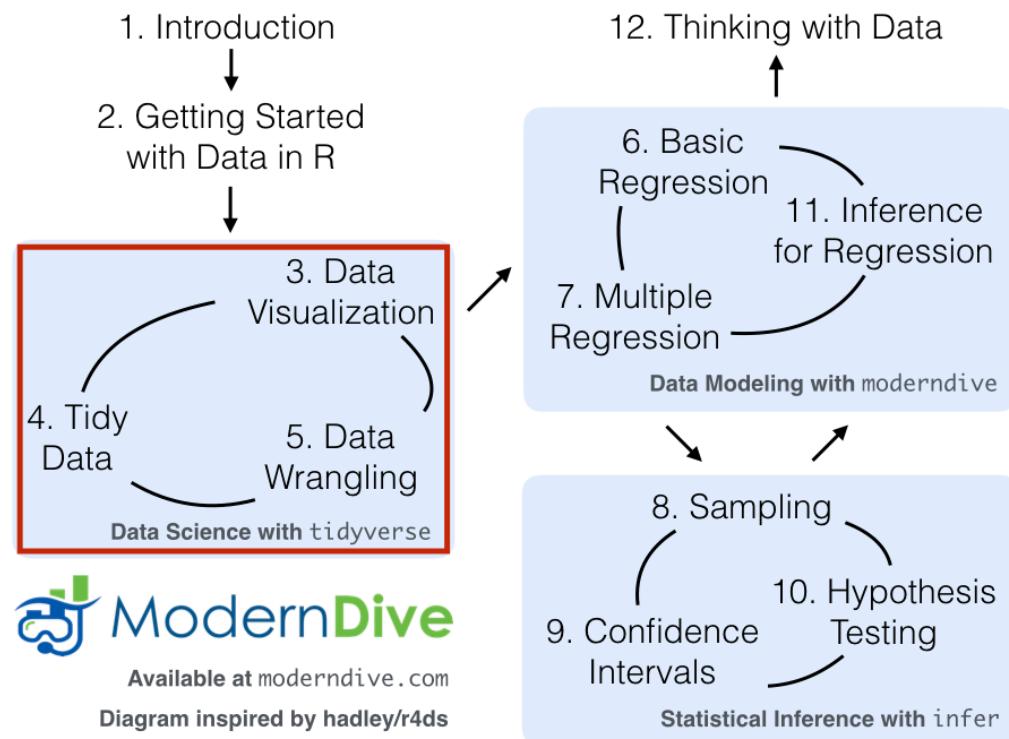


FIGURE 2.2: ModernDive flowchart

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).

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

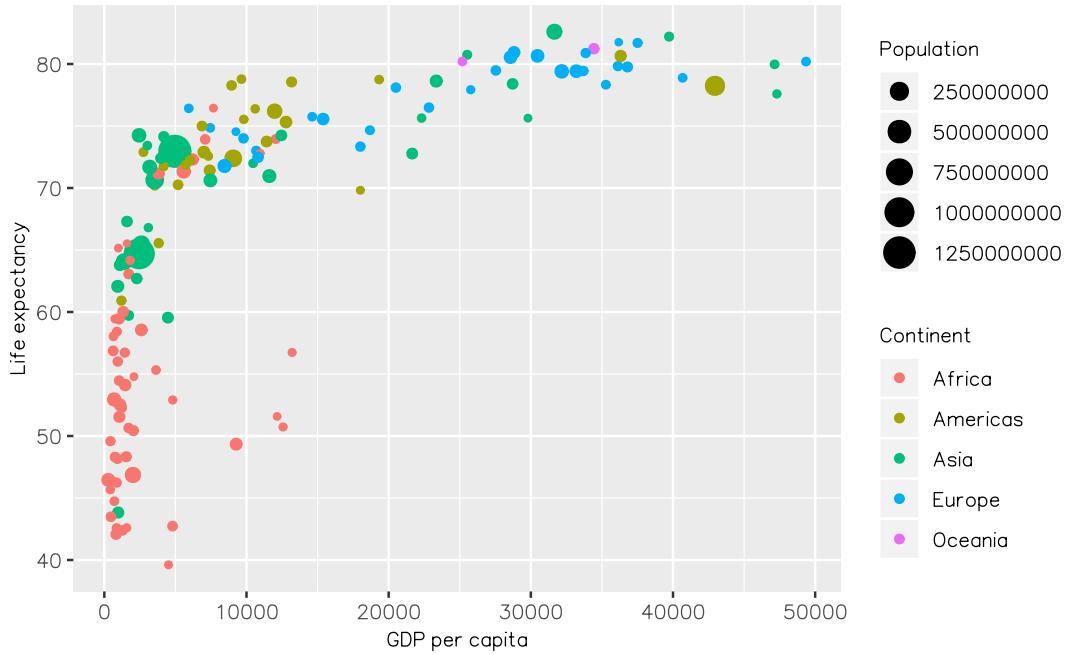


FIGURE 3.1: Life Expectancy over GDP per Capita in 2007

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., 2018). 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 a given 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 `alaska_flights` consisting of only the 714 Alaska Airlines flights.

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?

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 are points, set 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).
```

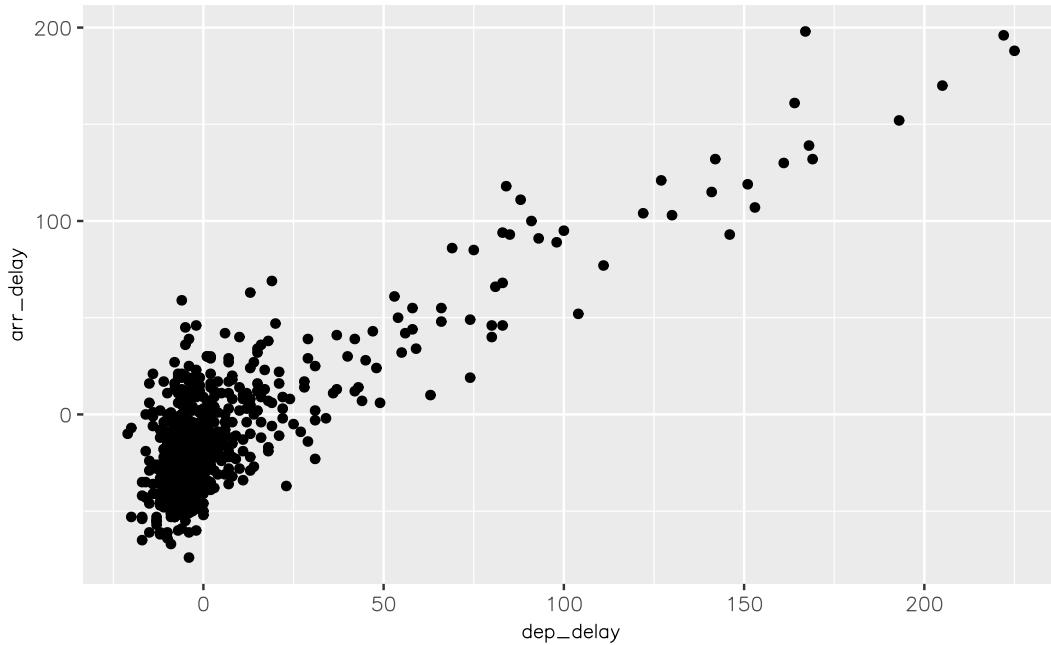


FIGURE 3.2: Arrival Delays vs Departure Delays for Alaska Airlines flights from NYC in 2013

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))
```

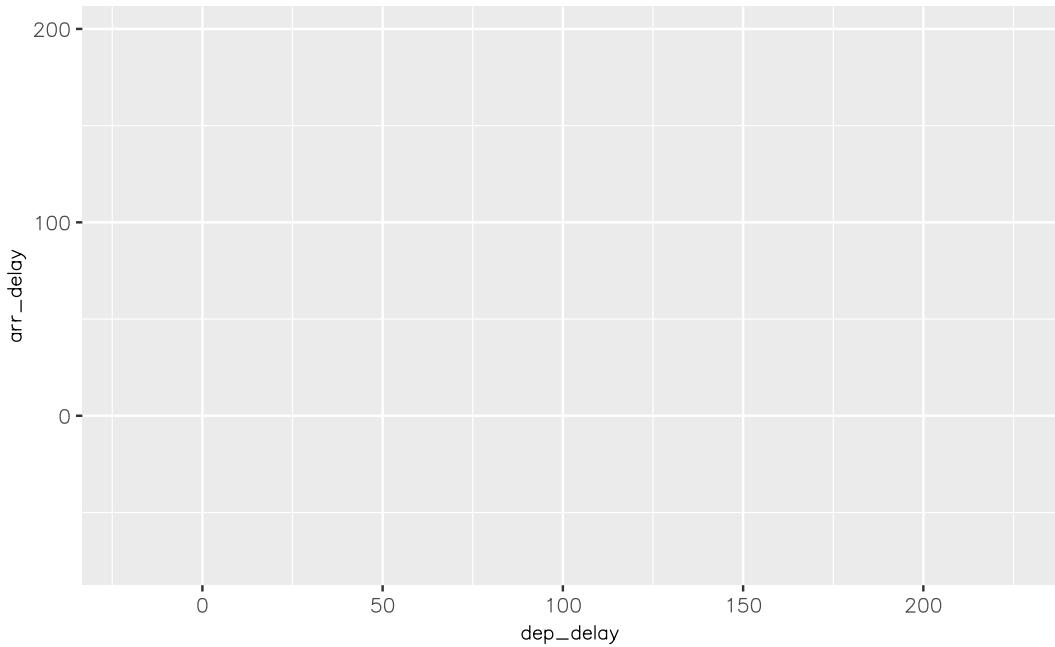


FIGURE 3.3: Plot with No Layers

Learning check

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

(LC3.3) What variables (not necessarily in the `flights` data frame) would

you expect to have a negative correlation (i.e. a negative relationship) with `dep_delay`? Why? Remember that we are focusing on numerical variables here.

(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

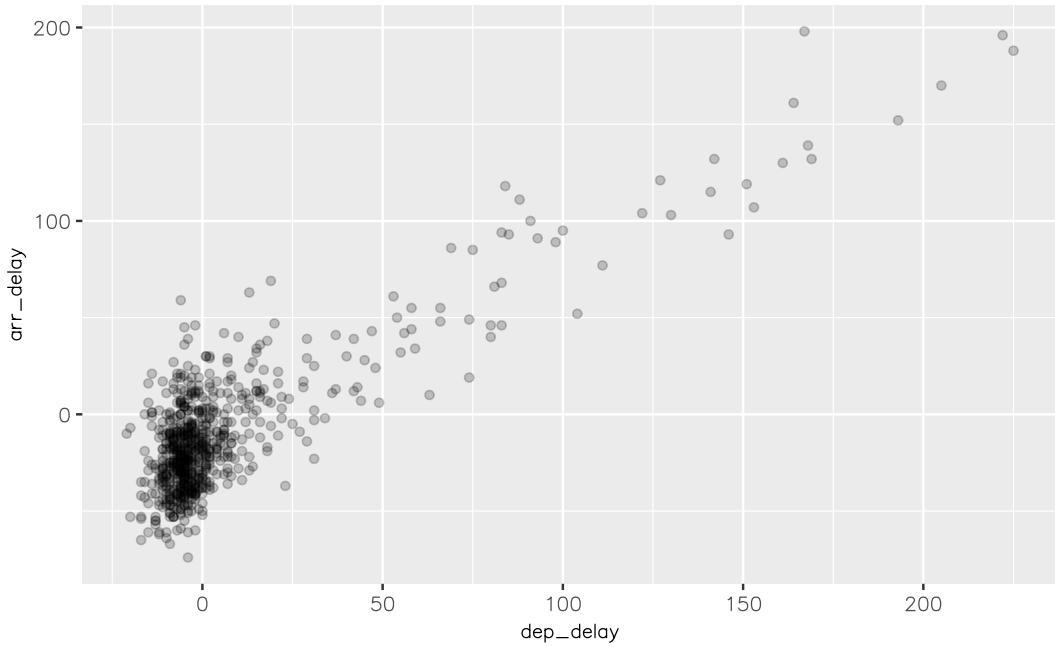


FIGURE 3.4: Delay scatterplot with $\text{alpha}=0.2$

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

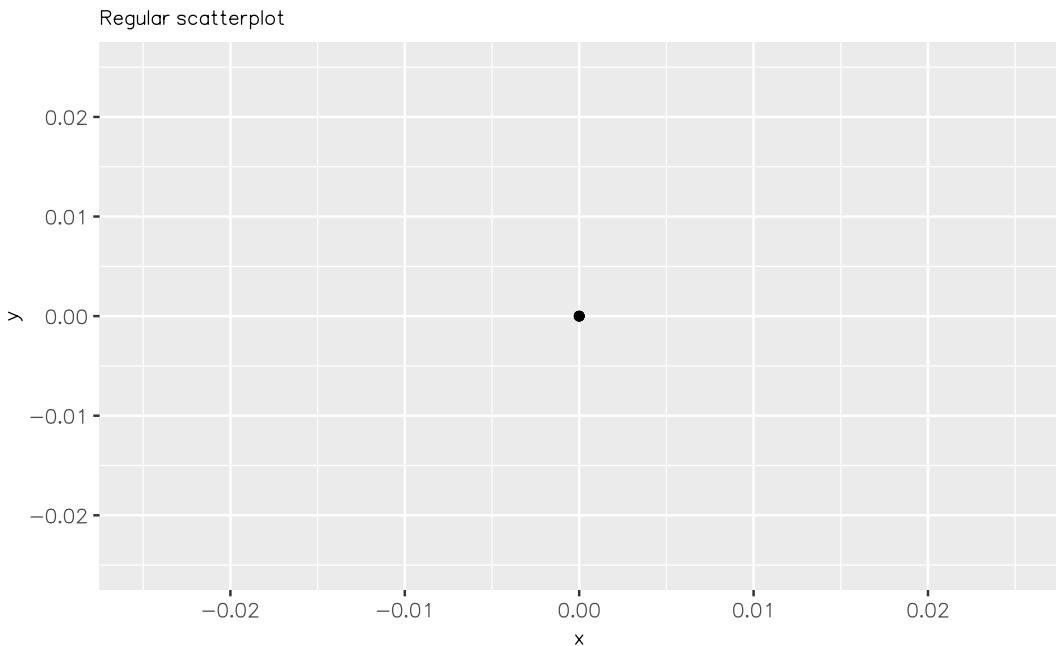


FIGURE 3.5: Regular scatterplot of jitter example data

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

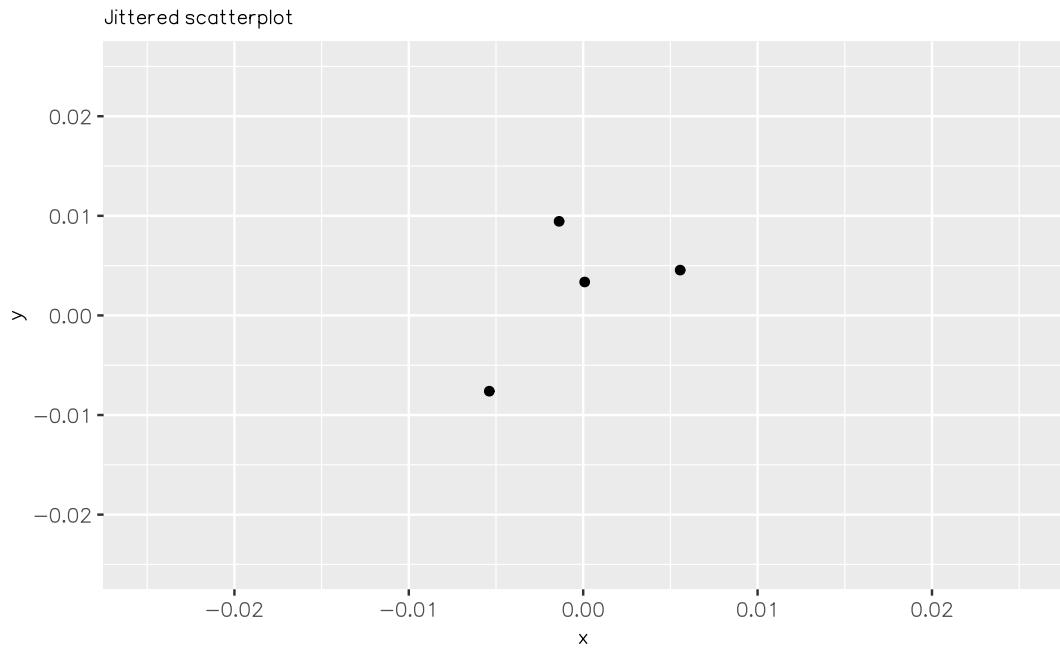


FIGURE 3.6: Jittered scatterplot of jitter example data

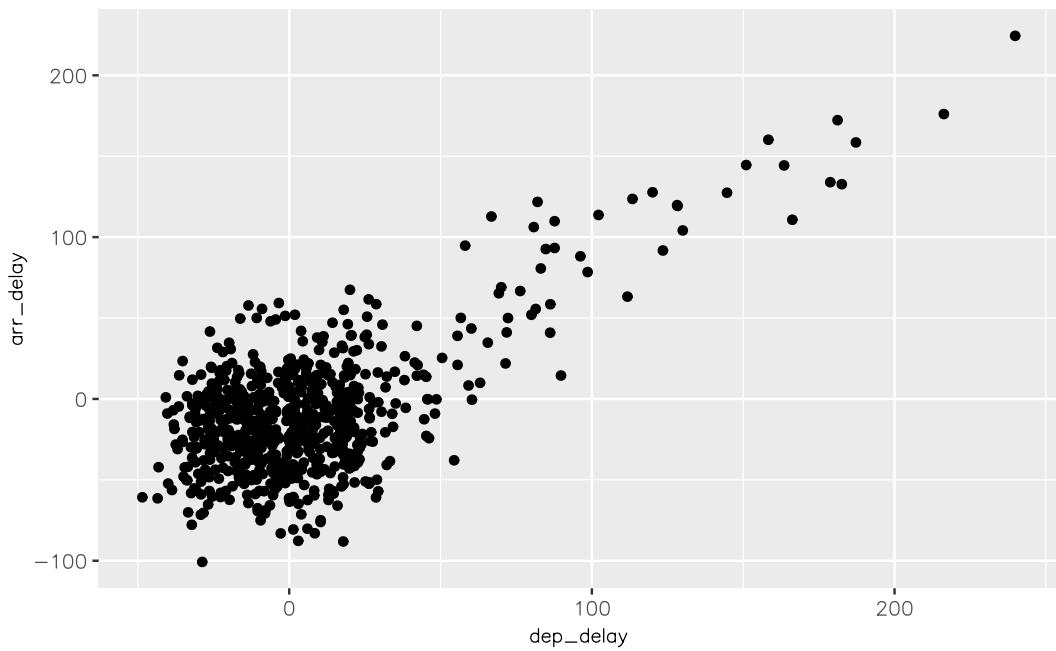


FIGURE 3.7: Jittered delay scatterplot

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

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()
```

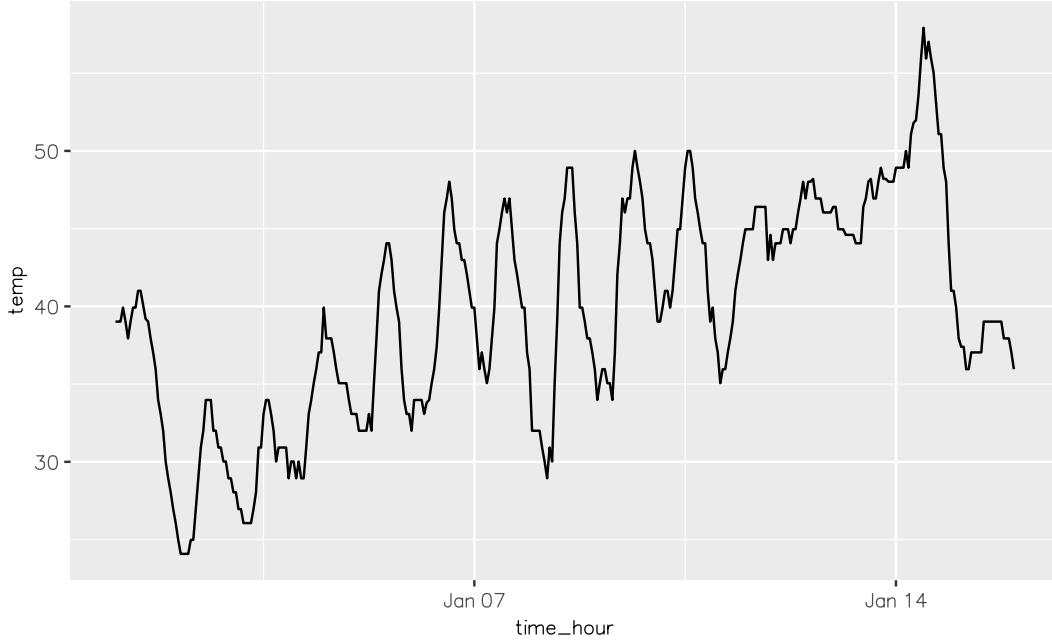


FIGURE 3.8: Hourly Temperature in Newark for January 1-15, 2013

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.

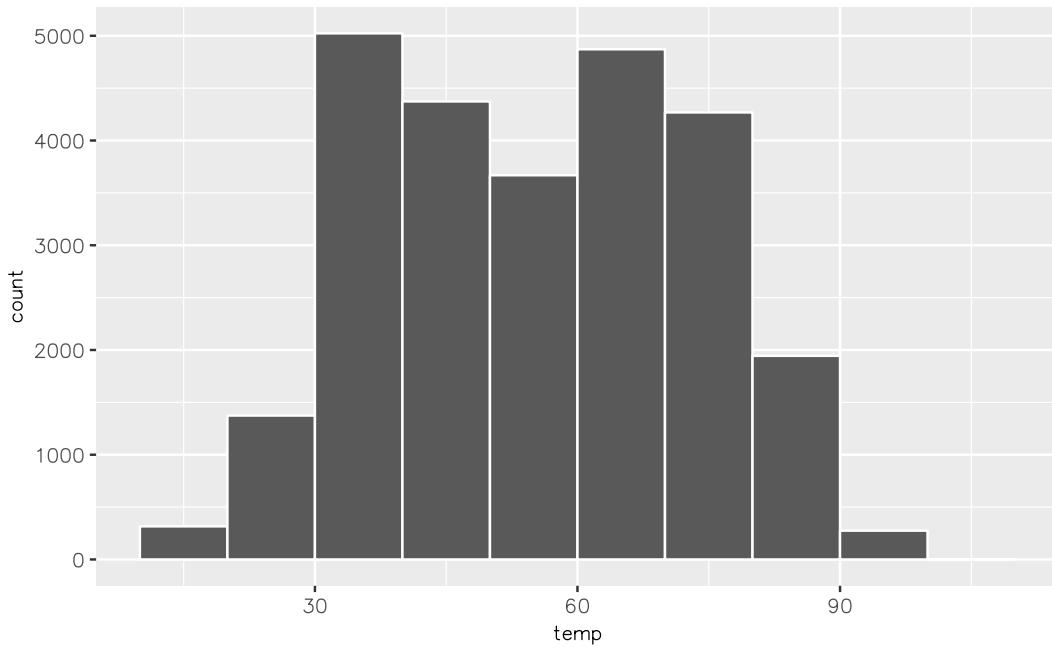


FIGURE 3.10: Example histogram.

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

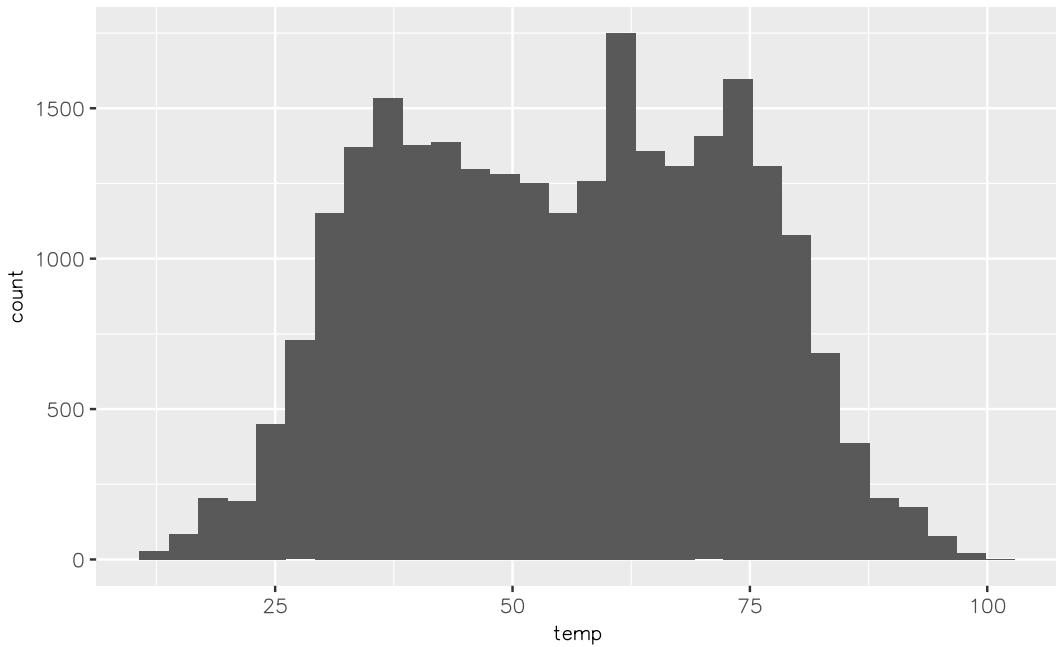


FIGURE 3.11: Histogram of hourly temperatures at three NYC airports.

warning 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")
```

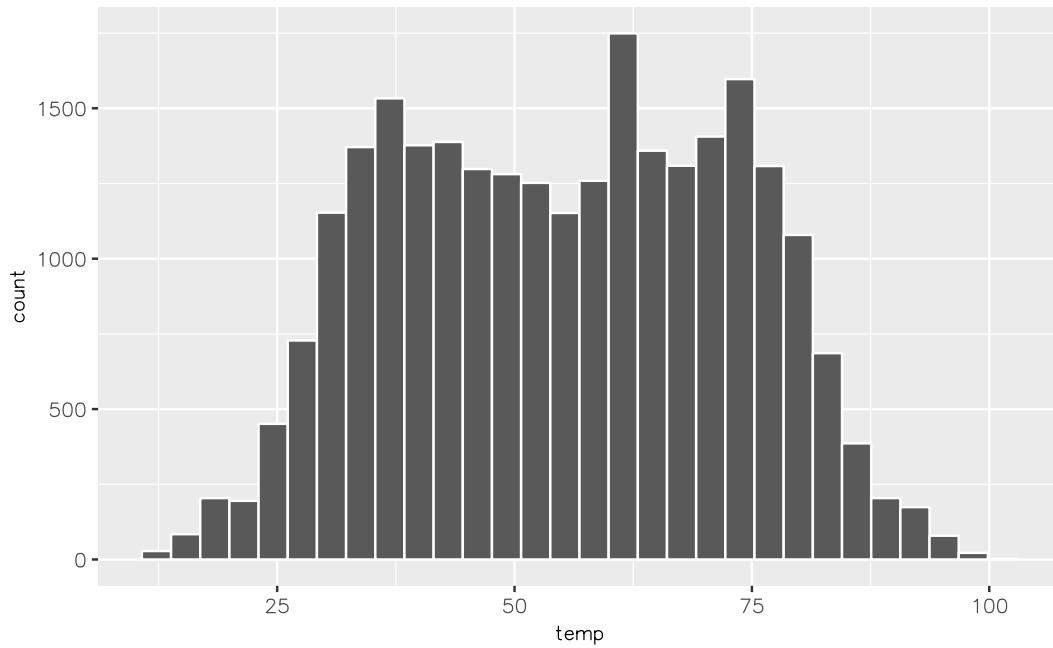


FIGURE 3.12: Histogram of hourly temperatures at three NYC airports with white borders.

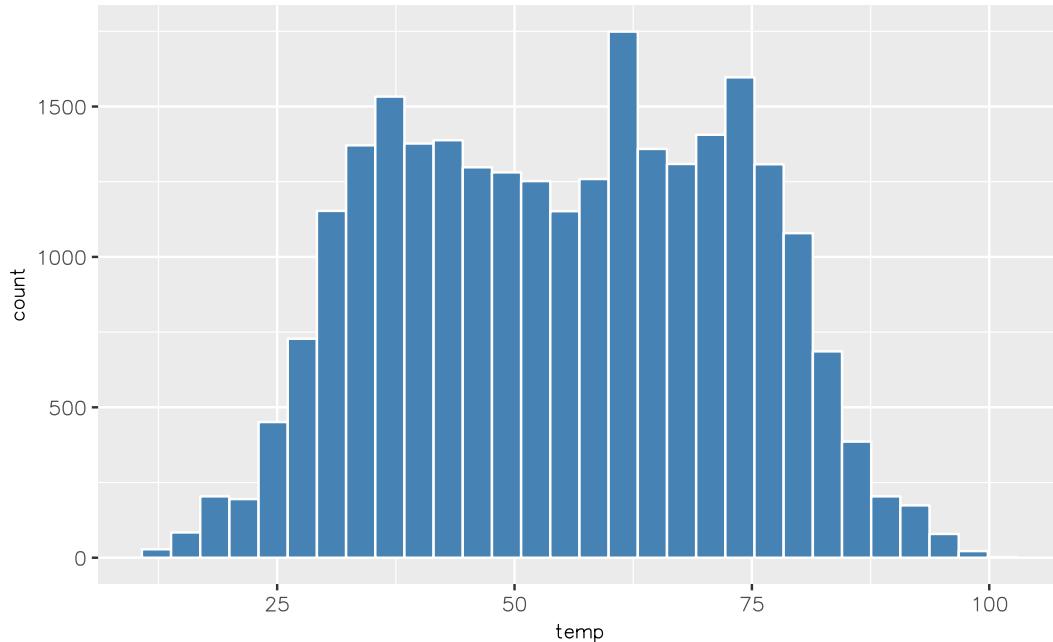


FIGURE 3.13: Histogram of hourly temperatures at three NYC airports with white borders.

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")
```

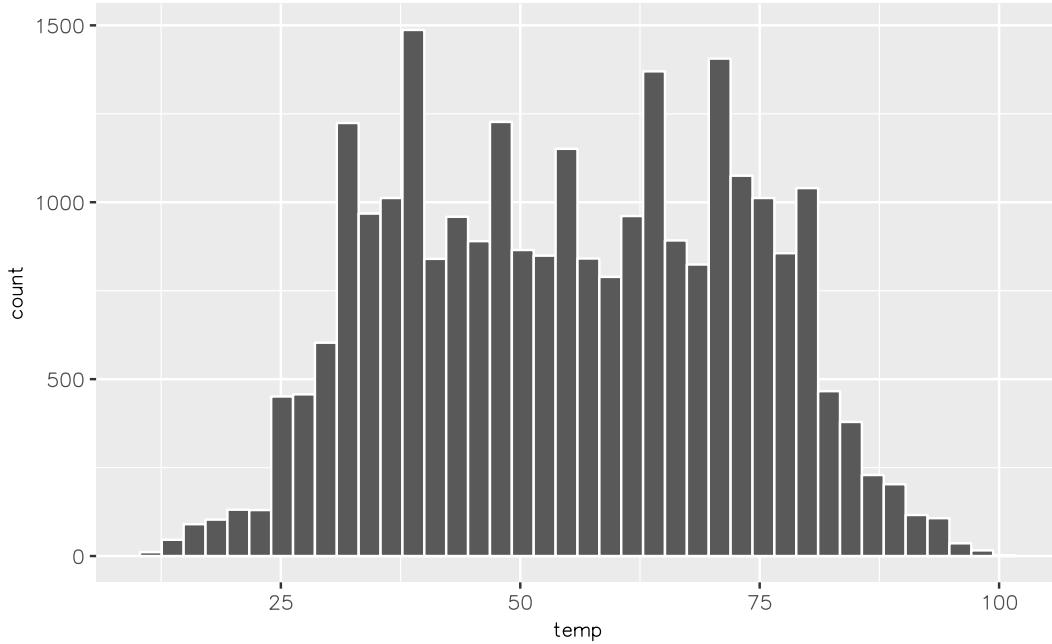


FIGURE 3.14: Histogram with 40 bins.

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")
```

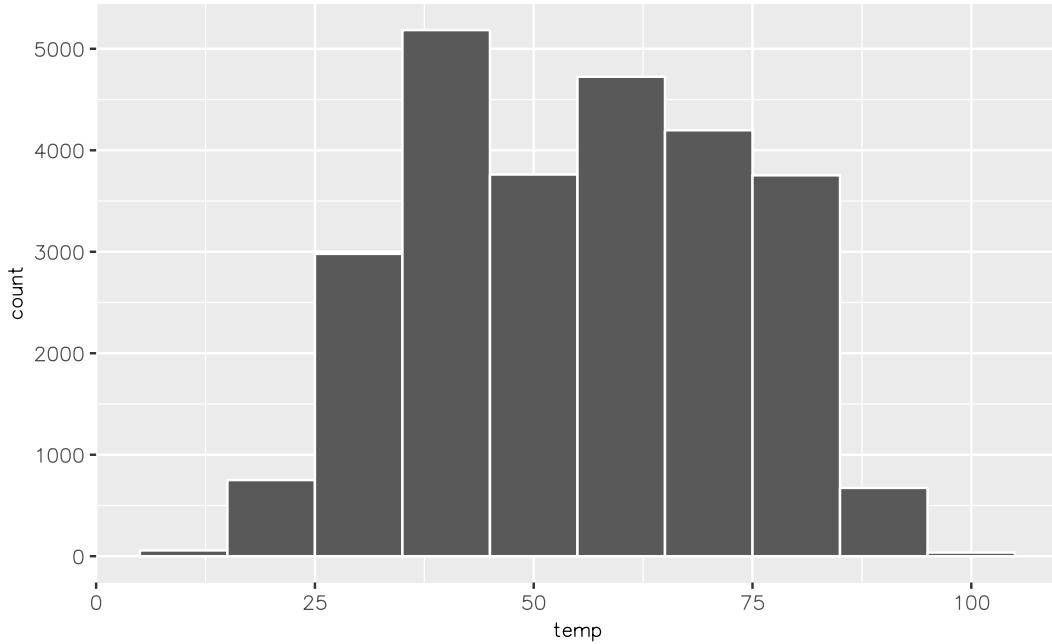


FIGURE 3.15: Histogram with binwidth 10.

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)
```

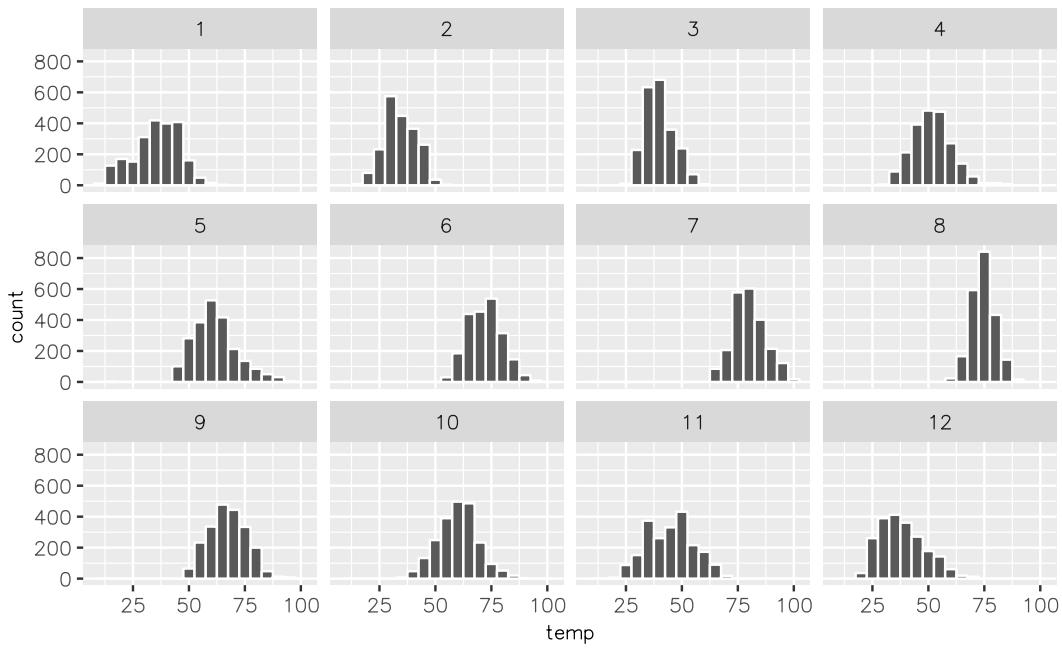


FIGURE 3.16: Faceted histogram.

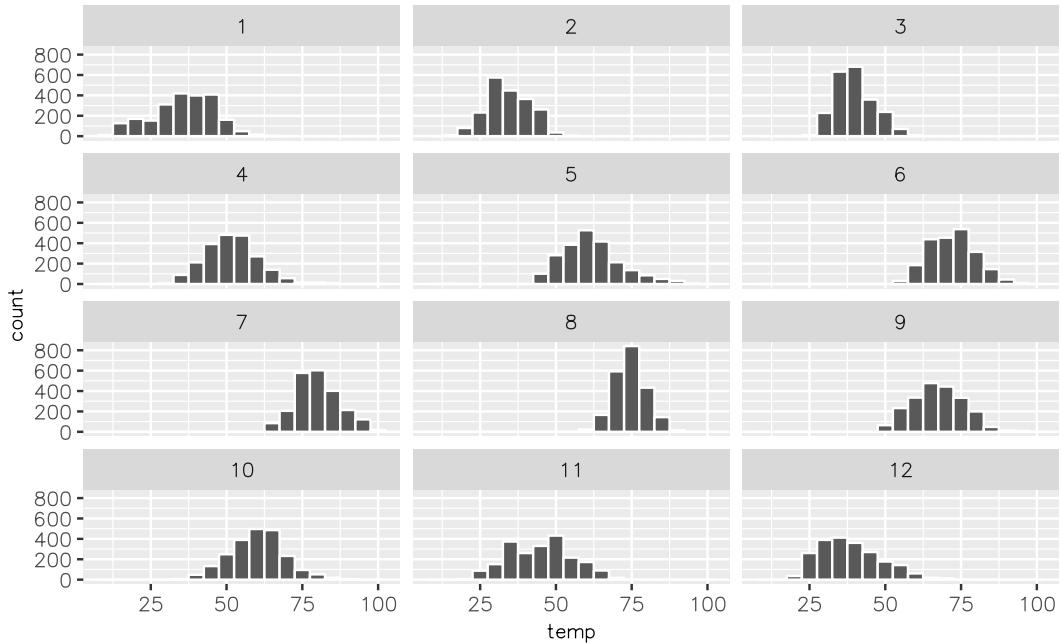


FIGURE 3.17: Faceted histogram with 4 instead of 3 rows.

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

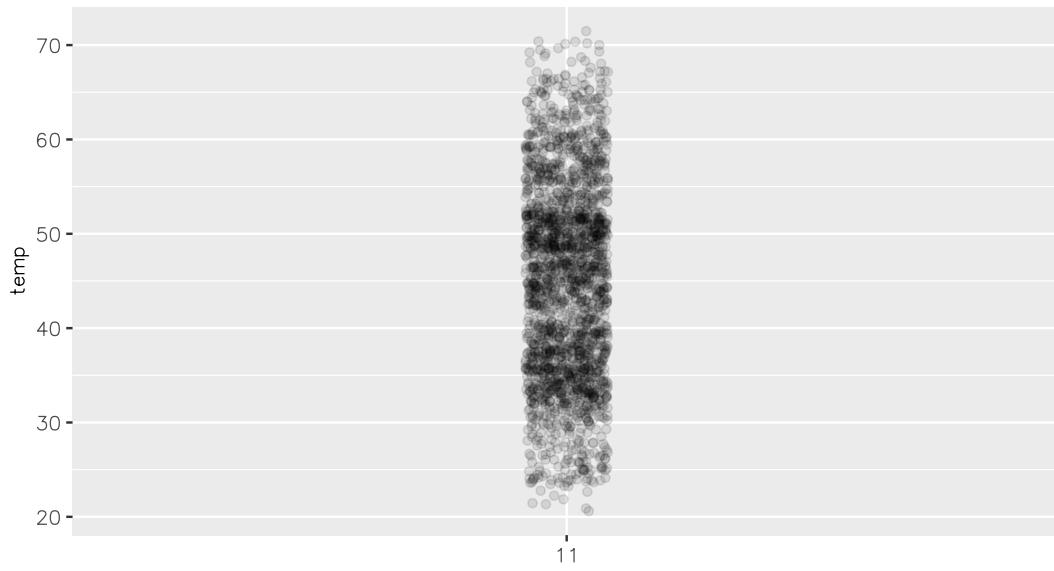


FIGURE 3.18: November temperatures.

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

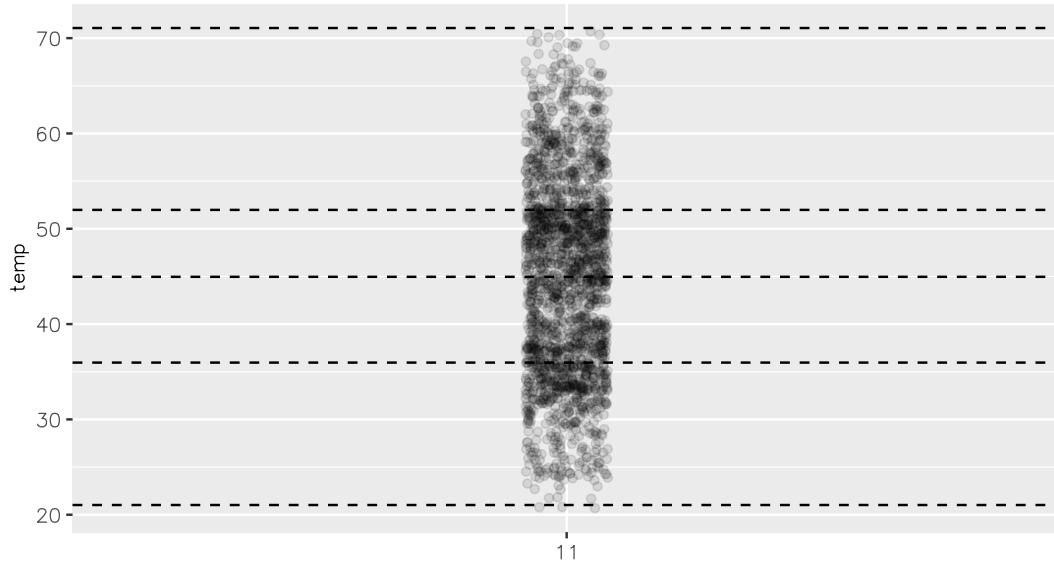


FIGURE 3.19: November temperatures.

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

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

1. 25% of points (about 534 observations) fall below the bottom edge of

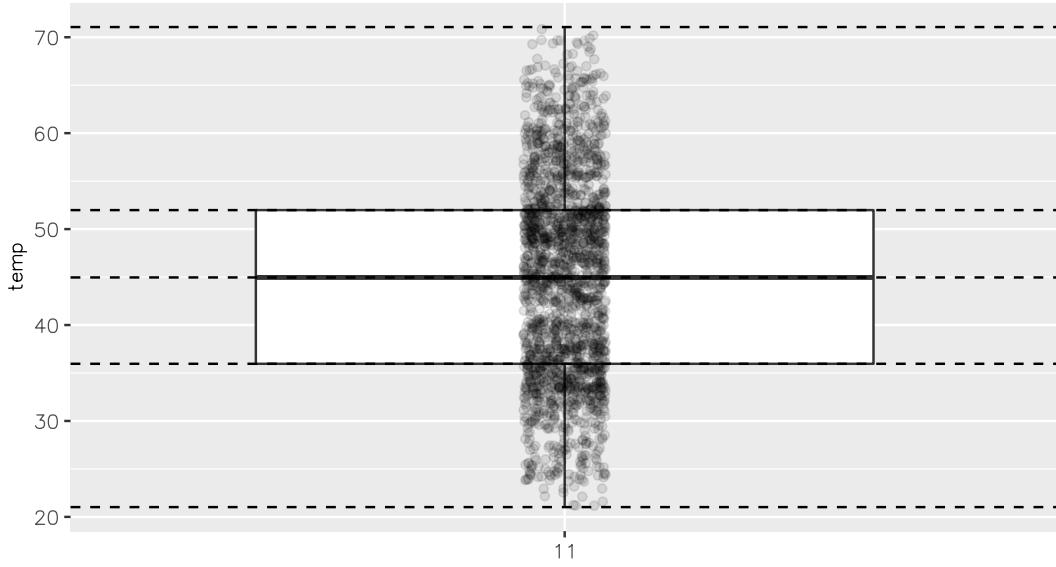


FIGURE 3.20: November temperatures.

- 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 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

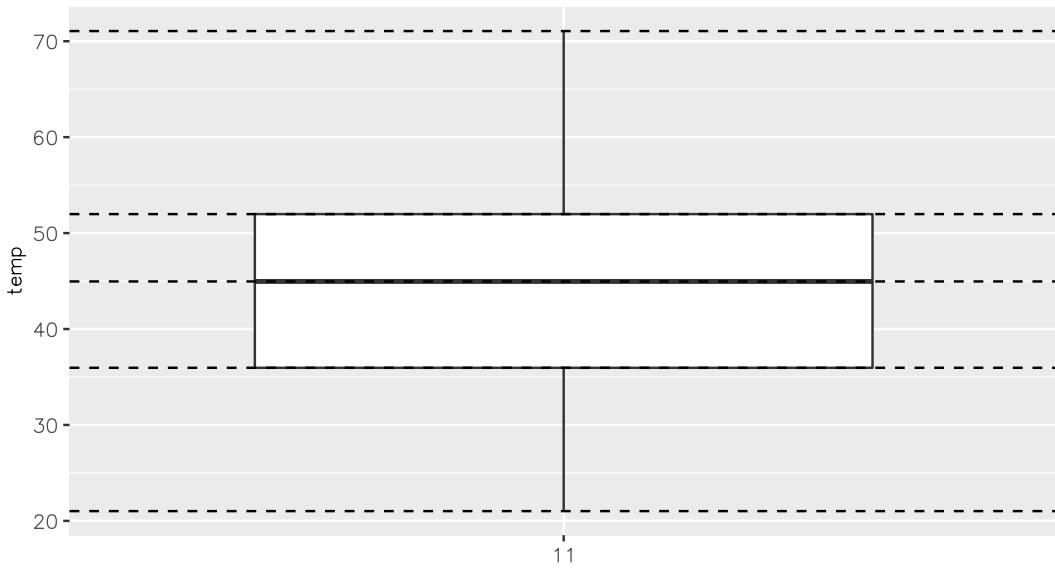


FIGURE 3.21: November temperatures.

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. 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.

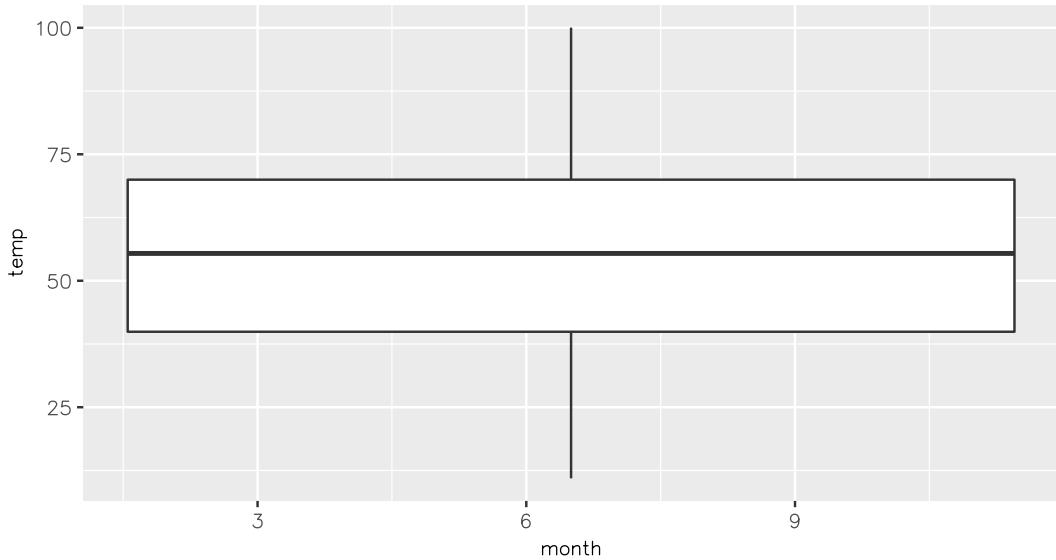


FIGURE 3.22: Invalid boxplot specification

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()
```

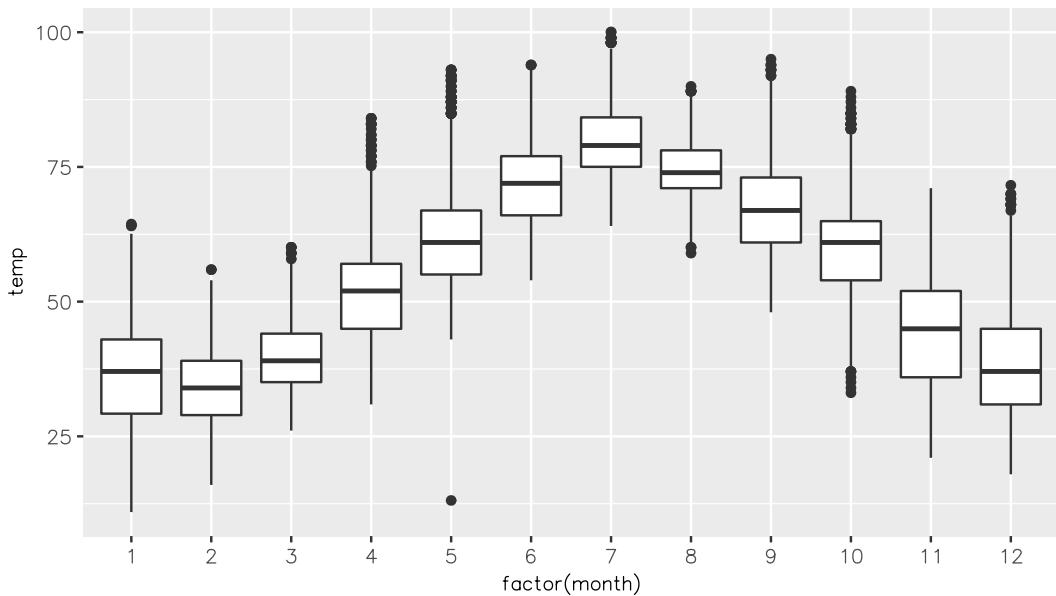


FIGURE 3.23: Month by temp boxplot

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 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 out-

liers 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()
```

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()
```

Compare the barplots in Figures 3.24 and 3.25. They are identical because

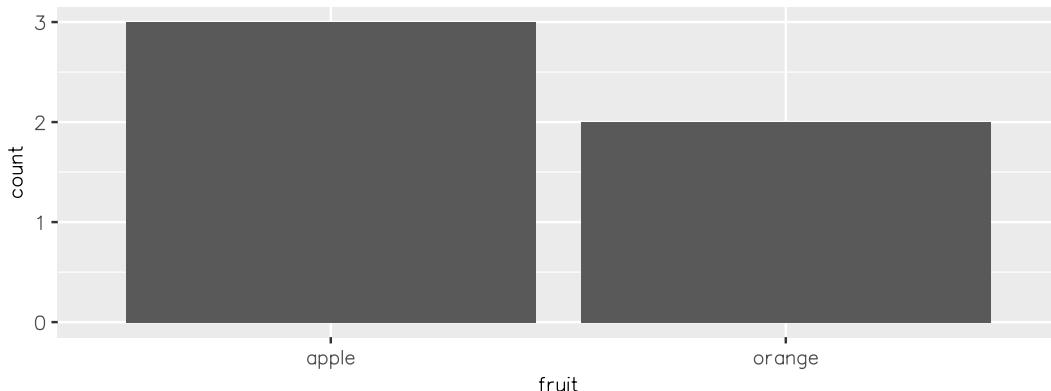


FIGURE 3.24: Barplot when counts are not pre-counted

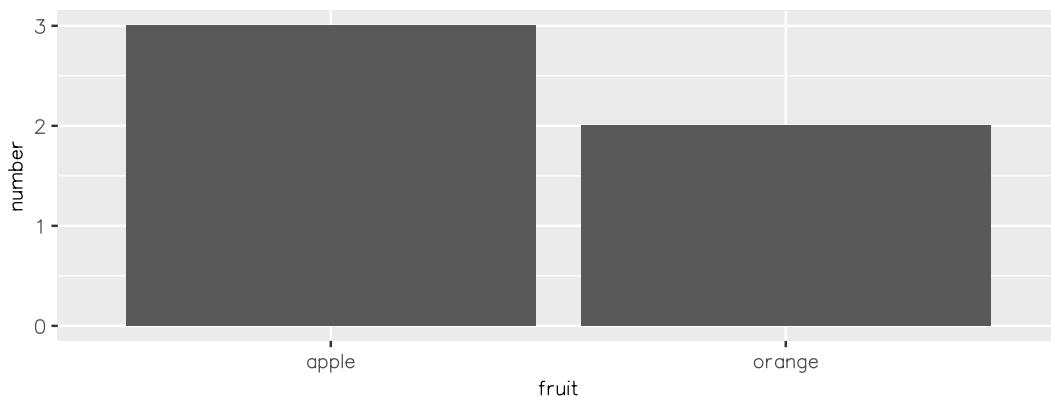


FIGURE 3.25: Barplot when counts are pre-counted

they reflect count of the same 5 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()
```

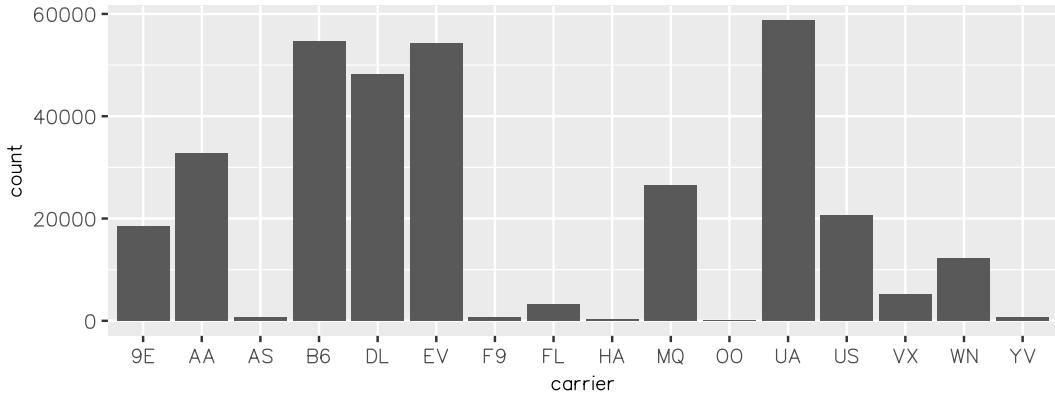


FIGURE 3.26: Number of flights departing NYC in 2013 by airline using `geom_bar()`

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.

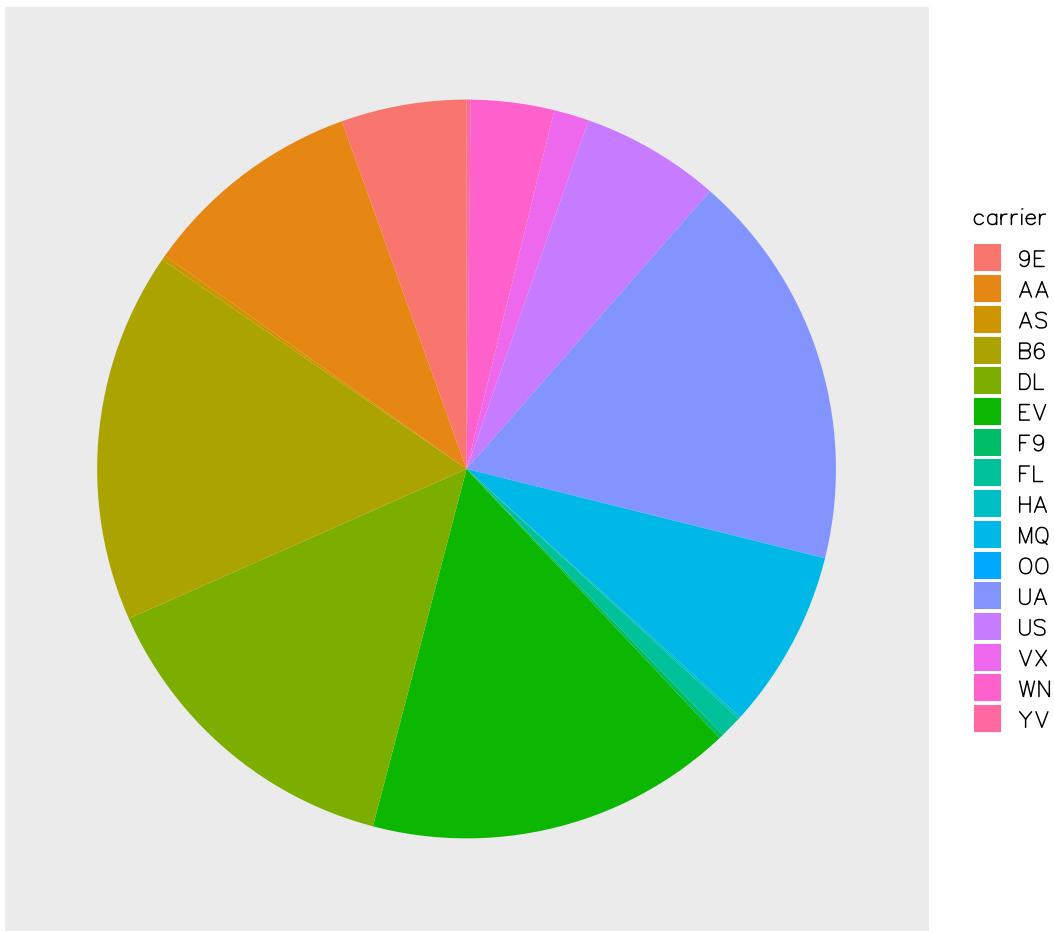


FIGURE 3.27: The dreaded pie chart

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.

There may be one exception of a pie chart not to avoid courtesy Nathan Yau at FlowingData.com⁴, but we will leave this for the reader to decide:



FIGURE 3.28: The only good pie chart

Learning check

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

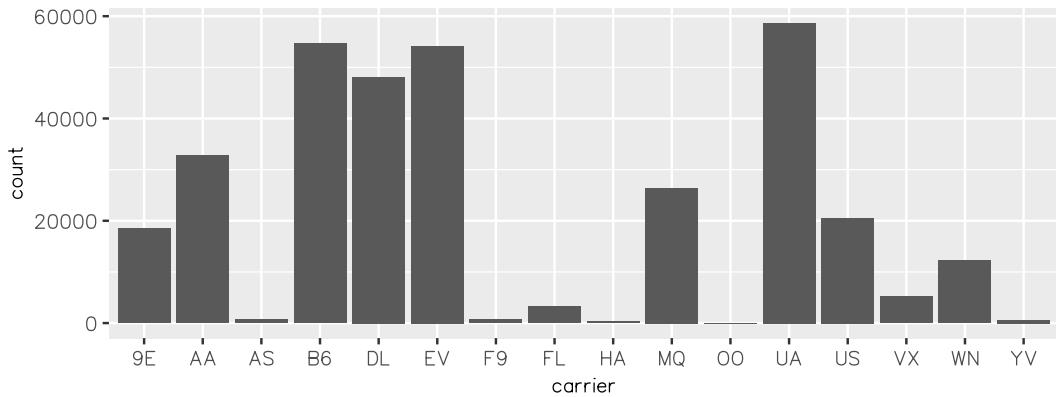
(LC3.31) Why do you think people continue to use pie charts?

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:

⁴<https://flowingdata.com/2008/09/19/pie-i-have-eaten-and-pie-i-have-not-eaten/>

```
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()
```

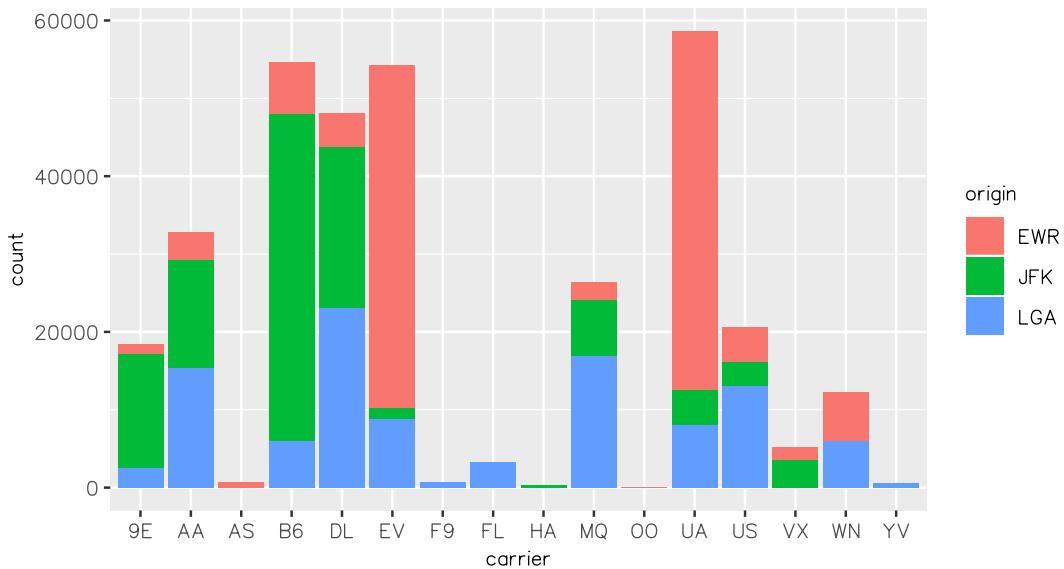


FIGURE 3.29: Stacked barplot comparing the number of flights by carrier and origin.

Figure 3.29 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 3.30 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()
```

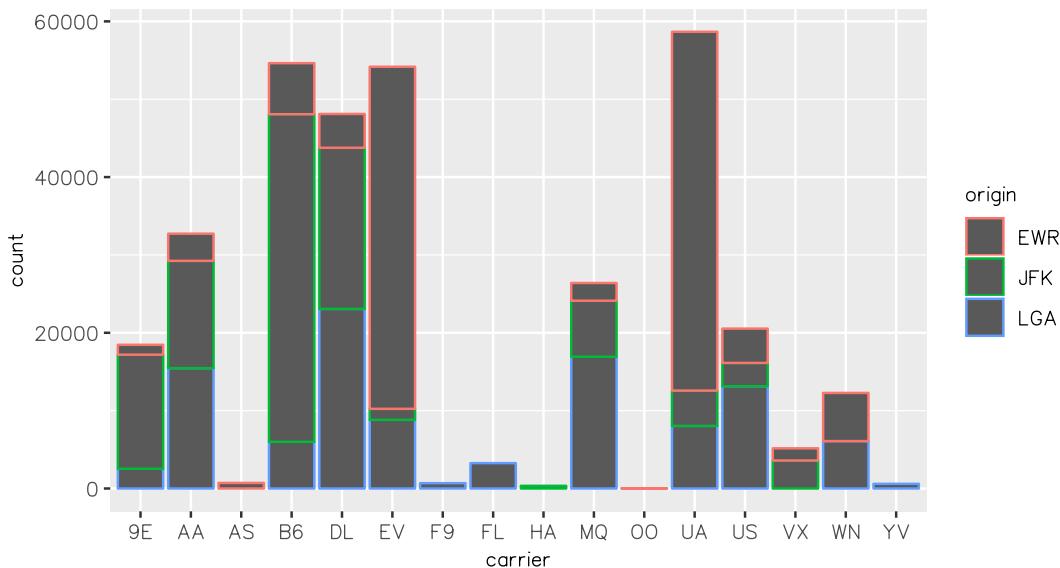


FIGURE 3.30: Stacked barplot with color aesthetic used instead of fill.

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")
```

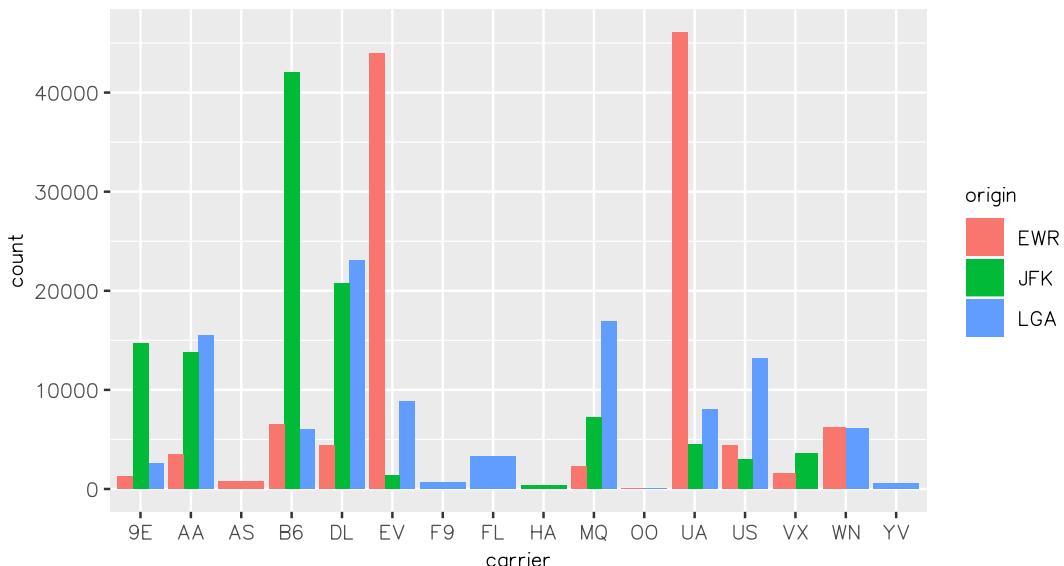


FIGURE 3.31: Side-by-side AKA dodged barplot comparing the number of flights by carrier and origin.

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.

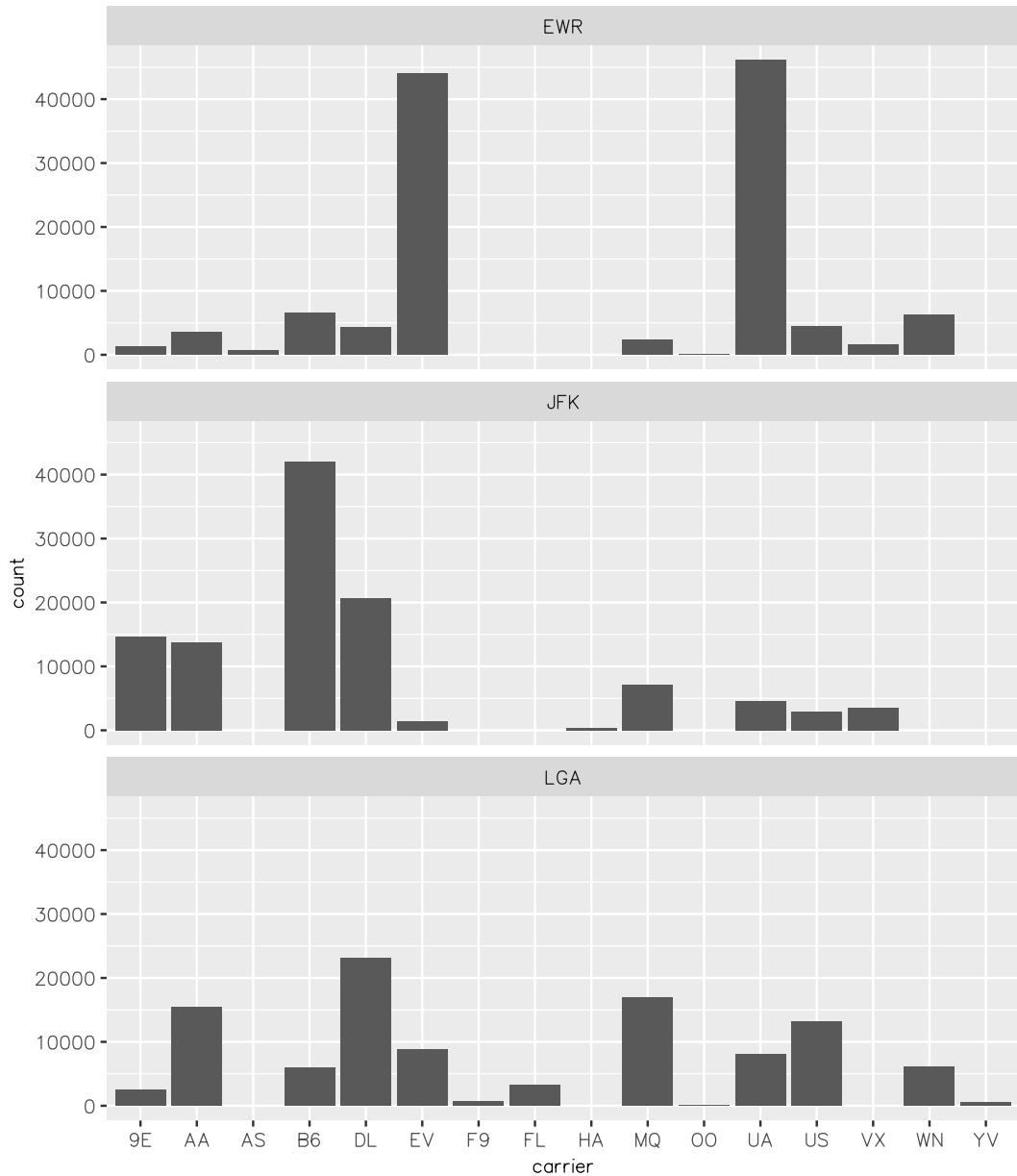


FIGURE 3.32: Faceted barplot comparing the number of flights by carrier and origin.

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

	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()'	Faceted 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()'	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 here⁵.

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":

⁵[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<MAPPINGS>,  
  stat = <STAT>, position = <POSITION>)+  
  <COORDINATE FUNCTION>  
  <FACET FUNCTION>  
  <SCALE FUNCTION>  
  <THEME FUNCTION>
```

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

ggplot(data = mpg, aes = cty, y = hwy) # Begins a plot that you finish by adding layers to. Add one geom function per layer.

ggplot() +
 geom<function>(aes<MAPPINGS> data geom
 ggplot(x = cty, y = hwy, data = mpg, geom = "point")
Creates a complete plot with given data, geom, and mappings. Supersedes the usual defaults.
last_plot() Returns the last plot.
ggplot("plot.png", width = 5, height = 5) # Saves last plot
as a 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(x=long, y=lat))`
- `a + geom_blank()` (useful for setting limits)
- `b + geom_curve(aes(yend = lat + 1, xend = long + 1), curvature=0.2, xend, yend, alpha, angle, color, curvature, linetype, size)`
- `a + geom_rect(aes(xmin=long - 1, xmax=long + 1, ymin=lat - 1, ymax=lat + 1), xmax, ymin, ymax, alpha, color, fill, group, linetype, size)`
- `b + geom_ribbon(aes(min=unemploy - 900, max=unemploy + 900), x, ymax, ymin, alpha, color, fill, group, linetype, size)`

LINE SEGMENTS

- `continuous x , continuous y`
- `e <- ggplot(mpg, 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_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 = cyl), nudge_x = 1, nudge_y = 1, check_overlap = TRUE), x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust`

TWO VARIABLES

- `continuous x , continuous y`
- `f <- ggplot(mpg, aes(class, label = "bin"))`
- `f + geom_col(), x, y, alpha, color, fill, group, linetype, size`
- `f + geom_bar(), x, y, alpha, color, fill, group, linetype, size`
- `f + geom_boxplot(), x, y, lower, middle, upper, ymin, ymax, alpha, color, fill, group, linetype, size, weight`
- `f + geom_dodgeplot(binaxis = "y", stackdir = "center"), x, y, alpha, color, fill, group, linetype, size, weight`
- `f + geom_violin(scale = "area"), x, y, alpha, color, fill, group, linetype, size, weight`

continuous x , continuous y

- `f <- ggplot(mpg, aes(class, label = "bin"))`
- `f + geom_col(), x, y, alpha, color, fill, group, linetype, size`
- `f + geom_bar(), x, y, alpha, color, fill, group, linetype, size`
- `f + geom_boxplot(), x, y, lower, middle, upper, ymin, ymax, alpha, color, fill, group, linetype, size, weight`
- `f + geom_dodgeplot(binaxis = "y", stackdir = "center"), x, y, alpha, color, fill, group, linetype, size, weight`
- `f + geom_violin(scale = "area"), x, y, alpha, color, fill, group, linetype, size, weight`

ONE VARIABLE continuous

- `c <- ggplot(mpg, aes(hwy)); c + ggplot(mpg)`
- `c + geom_area(stat = "bin")`
- `c + geom_density(kernel = "gaussian")`
- `c + geom_dotplot()`
- `c + geom_freqpoly()`
- `c + geom_histogram(binwidth = 5)`
- `c + geom_linerplot()`
- `c + geom_qqplot(sample = hwy)`

discrete x , discrete y

- `g <- ggplot(diamonds, aes(cut, color))`
- `g + geom_count()`
- `x, y, alpha, color, fill, shape, size, stroke`

discrete x , continuous y

- `seals <- c(wt, sepal.length, sepal.width, petal.length, petal.width)`
- `l <- ggplot(seals, aes(x = z))`
- `l + geom_contour()`
- `x, y, z, alpha, color, group, linetype, size, weight`

THREE VARIABLES

- `seals <- c(wt, sepal.length, sepal.width, petal.length, petal.width)`
- `l <- ggplot(seals, aes(x = z))`
- `l + geom_raster(aes(fill = z), interpolate = FALSE)`
- `x, y, alpha, fill`
- `l + geom_tile(aes(fill = z))`
- `x, y, alpha, color, fill, linetype, size, width`

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, y, alpha, color, fill, linetype, size`
- `i + geom_line()`
- `i, y, alpha, color, group, linetype, size`
- `i + geom_step(direction = "hv")`
- `i, y, alpha, color, group, linetype, size`

visualizing error

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

maps

- `data <- data.frame(murder = USArrests$Murder, state = USArrests$State, region = USArrests$Region)`
- `map <- map_data("state")`
- `k <- ggplot(data, aes(fill = murder))`
- `k + geom_map(aes(map_id = state), map = map)`
- `k + expand_limits(map$long, y = map$lat)`
- `map_alpha, alpha, color, fill, linetype, size`

FIGURE 3.33: 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 6.4.2 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 6 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 6 listed in the introduction to this chapter; you'll see some examples of these near in Section 4.8. However, just with these 6 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)

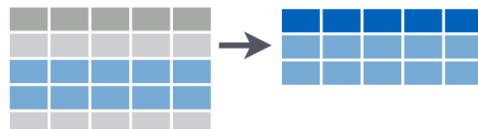


FIGURE 4.1: Diagram of

The `filter()` function here works much like the “Filter” option in Microsoft

Excel; it allows you to specify criteria about the values of 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 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.

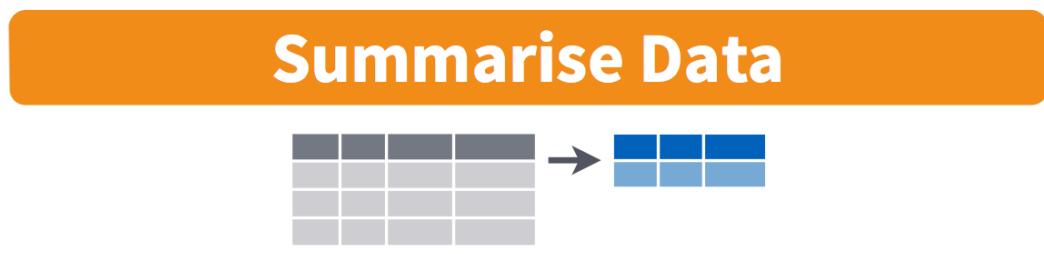


FIGURE 4.2: Summarize diagram from Data Wrangling with dplyr and tidyverse cheatsheet



FIGURE 4.3: Another summarize diagram from Data Wrangling with dplyr and tidyverse cheatsheet

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”



FIGURE 4.4: Group by and summarize diagram from Data Wrangling with dplyr and tidyverse cheatsheet

temperature observations by the values of another variable, in this case by the 12 values of the variable `month`. Run the following code:

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

month	mean	std_dev
1	35.6	10.22
2	34.3	6.98
3	39.9	6.25
4	51.7	8.79
5	61.8	9.68
6	72.2	7.55
7	80.1	7.12
8	74.5	5.19
9	67.4	8.47
10	60.1	8.85
11	45.0	10.44
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

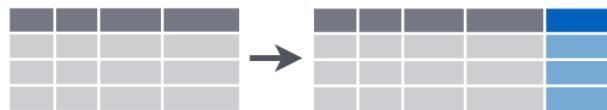


FIGURE 4.5: Mutate diagram from Data Wrangling with dplyr and tidyverse cheatsheet

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 Farenheit °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 `dep_delay`, `arr_delay`, and the resulting `gain` variables for the first 5 rows in our new `flights` data frame:

```
# A tibble: 5 x 3
  dep_delay arr_delay   gain
     <dbl>     <dbl> <dbl>
1       2       11    -9
2       4       20   -16
3       2       33   -31
4      -1      -18    17
5      -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`.

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

```

	min	q1	median	q3	max	mean	sd	missing
	-196	-3	7	17	109	5.66	18	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)

```

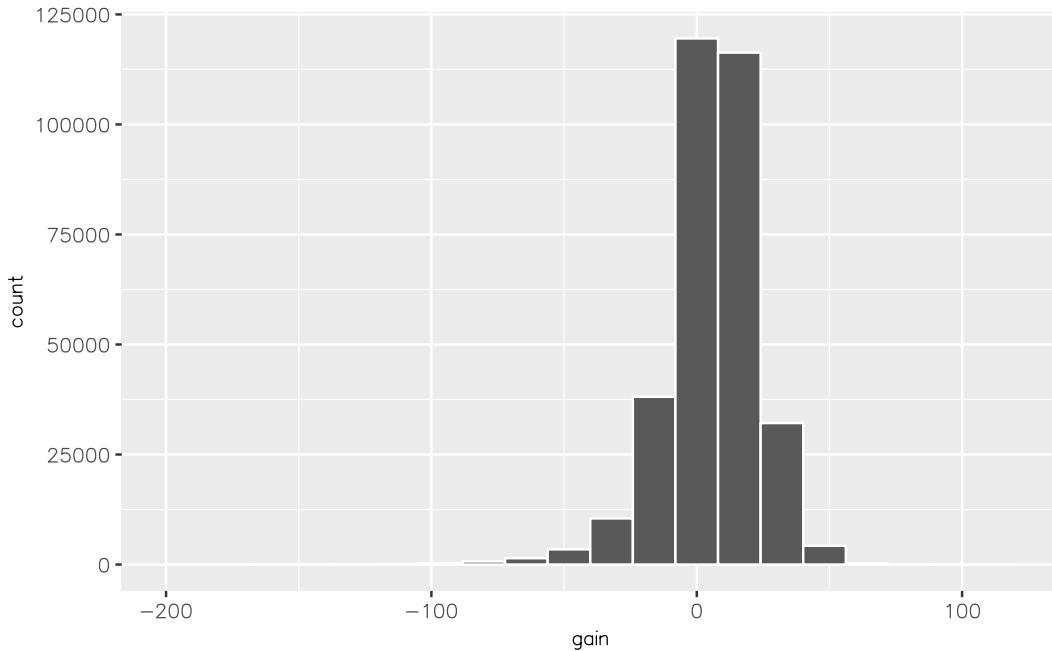


FIGURE 4.6: Histogram of gain variable

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” ([Grolemund 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?

(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 American 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:

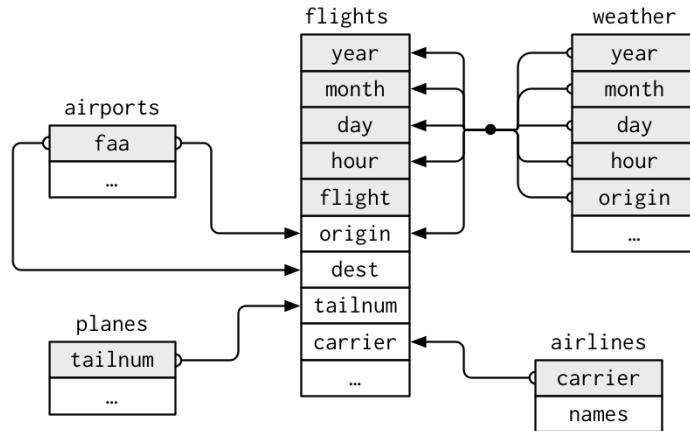


FIGURE 4.7: Data relationships in `nycflights13` from R for Data Science

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 (Grollemund 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.

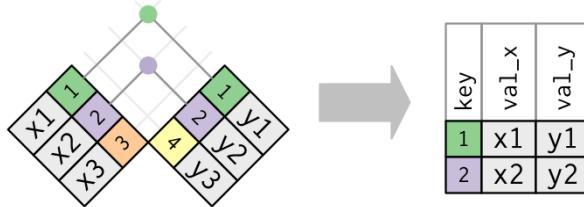


FIGURE 4.8: Diagram of inner join from R for Data Science

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)

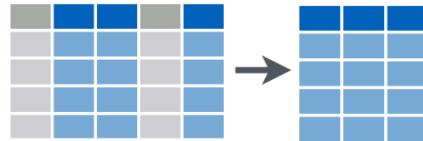


FIGURE 4.9: Select diagram from Data Wrangling with dplyr and tidyr cheatsheet

```
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)
```

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. I 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 of 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 4.1. 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 4.1: 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

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

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.

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 coding right away, but 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 4.1 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 wran-

³[scripts/04-wrangling.R](#)

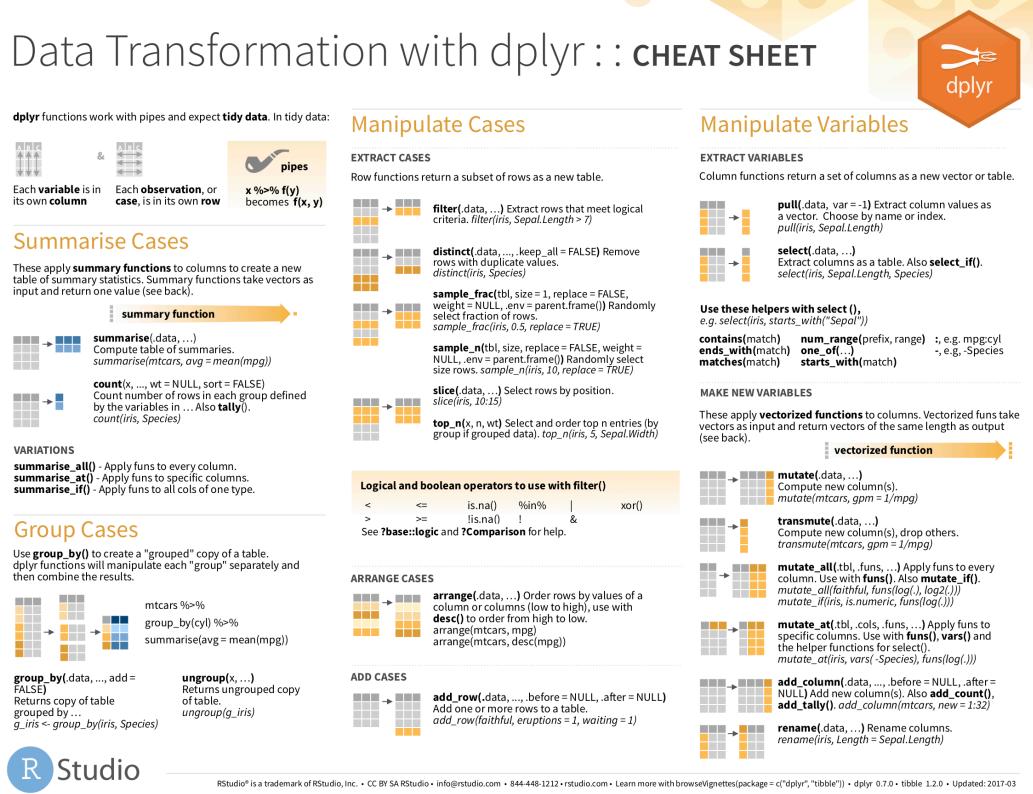


FIGURE 4.10: Data Transformation with dplyr cheatsheat

gling check out Chapter 5⁴ of Garrett Grolemund and Hadley Wickham's and Garrett's book ([Grolemund and Wickham, 2016](#)).

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.

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

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 clicking [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:

The screenshot shows the 'Import Excel Data' dialog box. At the top, there's a 'File/Url:' field containing the path `~/Desktop/dem_score.xlsx`. Below it is a 'Data Preview:' table showing data for 50 countries from 1952 to 1992. The table has columns for each year from 1952 to 1992. The 'Import Options:' section includes fields for 'Name:' (set to `dem_score`), 'Max Rows:' (set to 0), 'First Row as Names' (checked), 'Sheet:' (set to 'Default'), 'Skip:' (set to 0), 'Open Data Viewer' (checked), and 'Range:' (set to `A1:D10`). The 'Code Preview:' section contains the R code for reading the file:

```
library(readxl)
dem_score <- read_excel("Desktop/dem_score.xlsx")
View(dem_score)
```

At the bottom right, there are 'Import' and 'Cancel' buttons.

FIGURE 5.1

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.

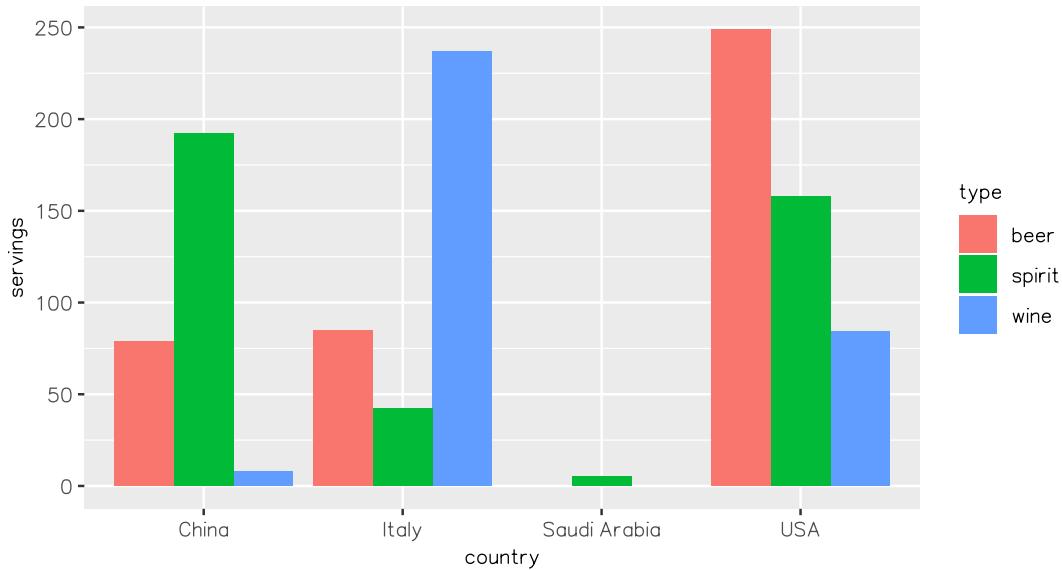


FIGURE 5.2: Alcohol consumption in 4 countries.

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.
-

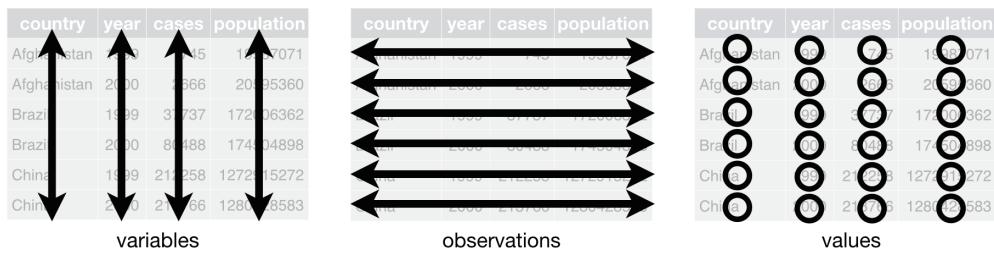


FIGURE 5.3: Tidy data graphic from [R for Data Science](<http://r4ds.had.co.nz/tidy-data.html>).

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:

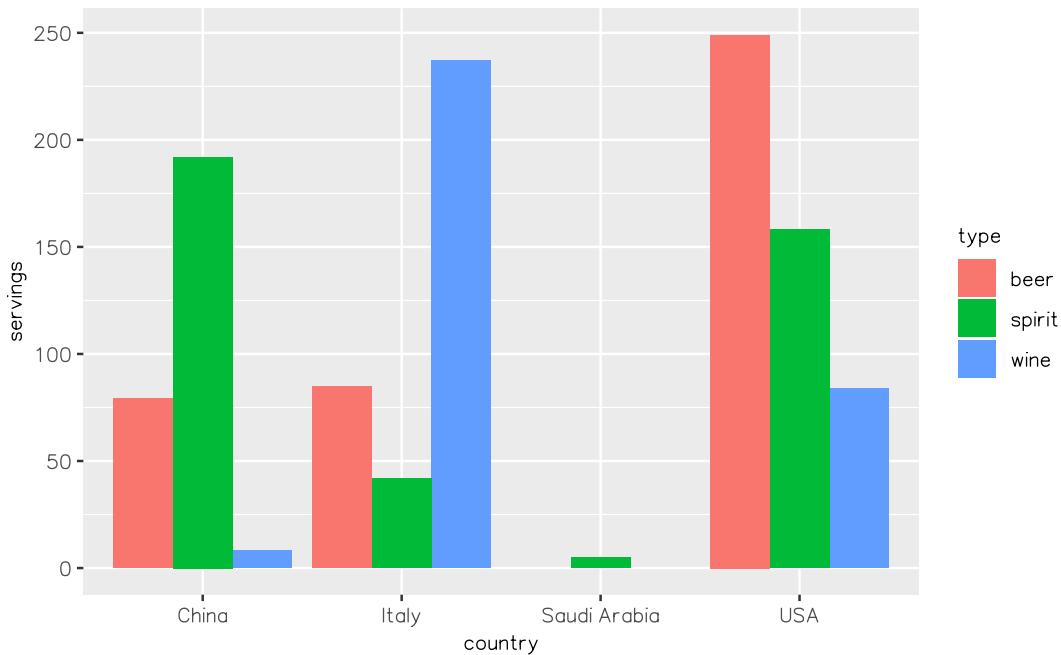
1. `key` is the name of the column/variable in the new “tidy” frame that contains the column names of the original data frame that you want to tidy. Observe how we set `key = type` and in the resulting `drinks_smaller_tidy` the column `type` contains the three types of alcohol `beer`, `spirit`, and `wine`.
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 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 (Golemund and Wickham, 2016).

Learning check

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

```
airline_safety
```

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

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:

```
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?

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

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
# ... with 2 more variables: `1987` <dbl>, `1992` <dbl>
  country `1952` `1957` `1962` `1967` `1972` `1977` `1982` 
  <chr>     <dbl>   <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 `tidyverse` 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 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.

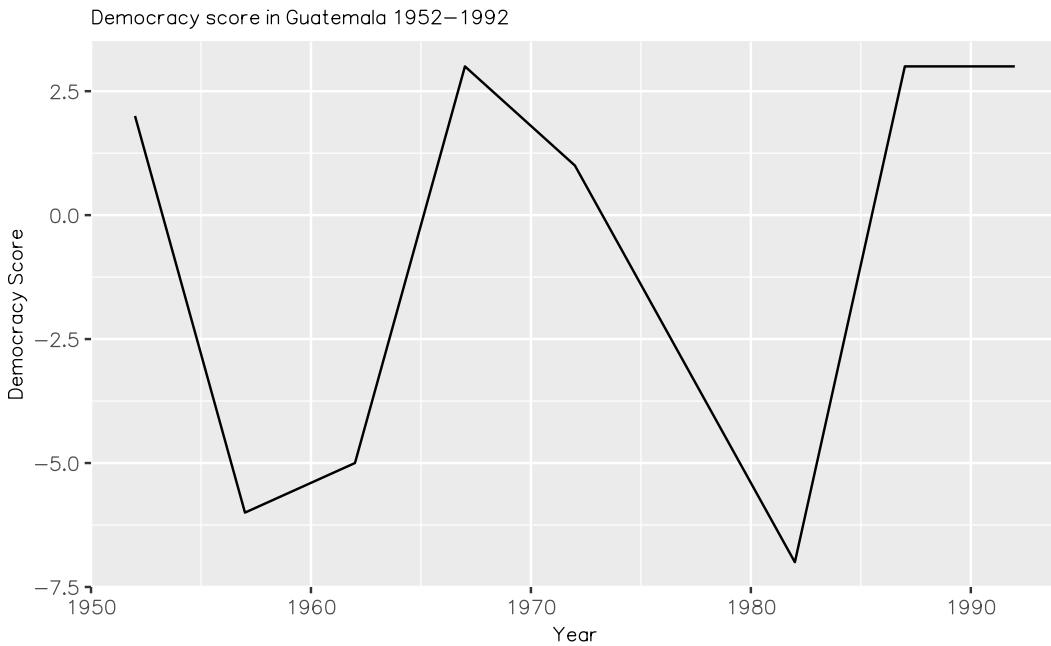
However, observe in the output for `guat_dem_tidy` that the `year` variable is of type `chr` or character. Before we can plot this variable on the x-axis, we need to convert it into a numerical variable using the `as.numeric()` function within the

`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", title = "Democracy score 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, `tidyverse` 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.

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”.

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!

⁹<http://r4ds.had.co.nz/>

¹⁰[scripts/05-tidy.R](#)

¹¹<https://www.rstudio.com/resources/cheatsheets/>

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:

	Comma Delimited Files	Semi-colon Delimited Files	Files with Any Delimiter	Fixed Width Files	Tab Delimited Files
<code>read_csv("file.csv")</code>	<code>read_csv("file.csv")</code> To make file run: <code>write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")</code>	<code>read_csv2("file.csv")</code> <code>write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file2.csv")</code>	<code>read_delim("file.txt", delim = "")</code> <code>write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.txt")</code>	<code>read_fwf("file.fwf", col_positions = c(1, 3, 5))</code> <code>write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")</code>	<code>read_tsv("file.tsv")</code> Also <code>read_table()</code> . <code>write_file(x = "a\tb\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")</code>
<code>read_csv("file.csv")</code>	<code>1 2 3</code>	<code>1 2 3</code>	<code>1 2 3</code>	<code>1 2 3</code>	<code>1 2 3</code>
<code>read_csv("file.csv")</code>	<code>4 5 NA</code>	<code>4 5 NA</code>	<code>4 5 NA</code>	<code>4 5 NA</code>	<code>4 5 NA</code>
<code>read_csv("file.csv")</code>	<code>1 2 3</code>	<code>1 2 3</code>	<code>1 2 3</code>	<code>1 2 3</code>	<code>1 2 3</code>
<code>read_csv("file.csv")</code>	<code>4 5 NA</code>	<code>4 5 NA</code>	<code>4 5 NA</code>	<code>4 5 NA</code>	<code>4 5 NA</code>

USEFUL ARGUMENTS

<code>example file</code>	<code>1 2 3</code>	<code>Skip lines</code>
<code>write_file("a,b,c\n1,2,3\n4,5,NA", "file.csv")</code>	<code>4 5 NA</code>	<code>read_csv(f, skip = 1)</code>
<code>No header</code>	<code>A B C</code>	<code>Read in a subset</code>
<code>read_csv(f, col_names = FALSE)</code>	<code>1 2 3</code>	<code>read_csv(f, n_max = 1)</code>
<code>Provide header</code>	<code>A B C</code>	<code>Missing Values</code>
<code>read_csv(f, col_names = c("x", "y", "z"))</code>	<code>NA 2 3</code>	<code>read_csv(f, na = c("1", ""))</code>
<code>read_csv(f, na = c("1", ""))</code>	<code>4 5 NA</code>	

Read Non-Tabular Data

<code>read_file(file, locale = default_locale())</code>	<code>read_file_raw(file)</code>
<code>Read each line into its own string</code>	<code>Read each line into a raw vector</code>
<code>read_lines(file, skip = 0, n_max = -1, na = character(), locale = default_locale(), progress = interactive())</code>	<code>read_lines_raw(file, skip = 0, n_max = -1, progress = interactive())</code>
<code>Read Apache style log files</code>	<code>read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive())</code>

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • studio.com • Learn more at tidyverse.org • readr 1.1.0 • tibble 1.2.12 • tidy 0.6.0 • Updated: 2017-01



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 results.

```
## Parsed with column specification:
##   #> age = col_integer(),
##   #> sex = col_character(),
##   #> earn = col_double()
##   #>
```

`age is an integer`

`sex is a 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_factor(levels, ordered = FALSE)`

• `col_integer()`

• `col_logical()`

• `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)`

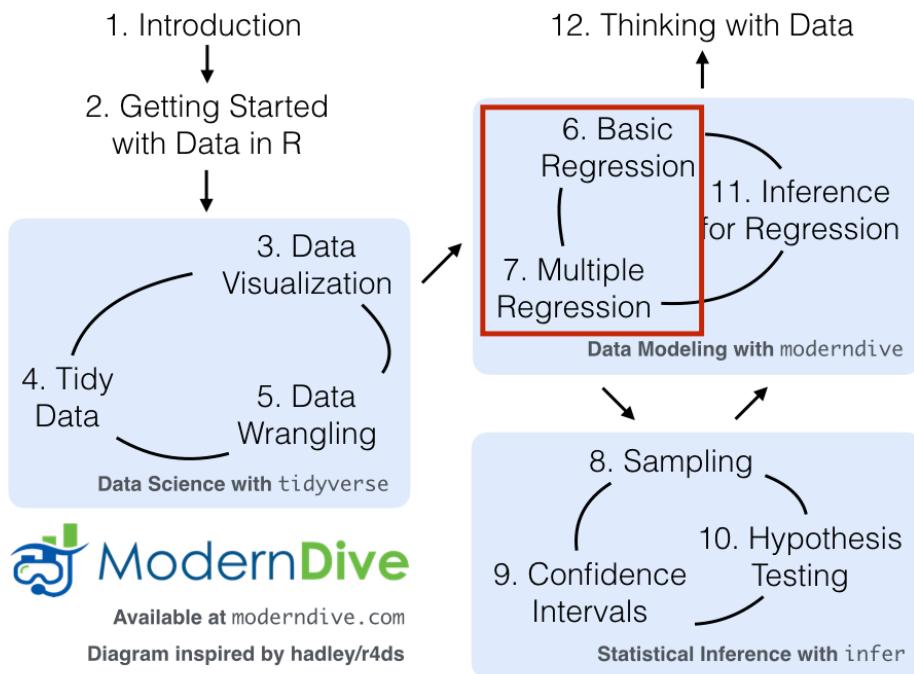


FIGURE 5.5: ModernDive flowchart - On to Part II!

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 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 . 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.

Let's perform the first common step in an exploratory data analysis: looking

¹<https://www.openintro.org/stat/data/?data=evals>

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, 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. In it is important to note that the *observational unit* are individual courses and not individual instructors. 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 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 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.

²<https://www.openintro.org/stat/data/?data=evals>

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
290	3.6	6.67	34
341	4.9	3.50	43
199	3.3	2.33	47
47	4.4	4.67	33
215	4.7	3.67	60

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 above `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
bty_avg	0	463	463	4.42	1.53	1.67	3.17	4.33	5.5
score	0	463	463	4.17	0.54	2.3	3.8	4.3	4.6

p100 hist



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
- `hist`: A quick snapshot of the histogram

Looking at the above 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* sum-

mary 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.

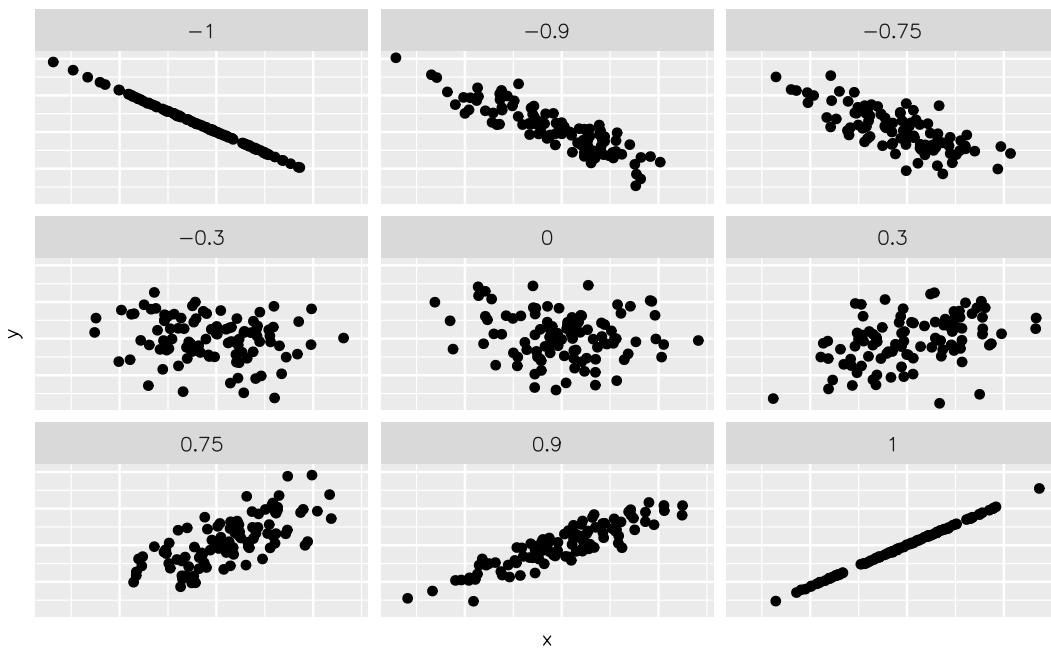


FIGURE 6.1: Different correlation coefficients

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.” 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()`:

```
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 below.

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")
```

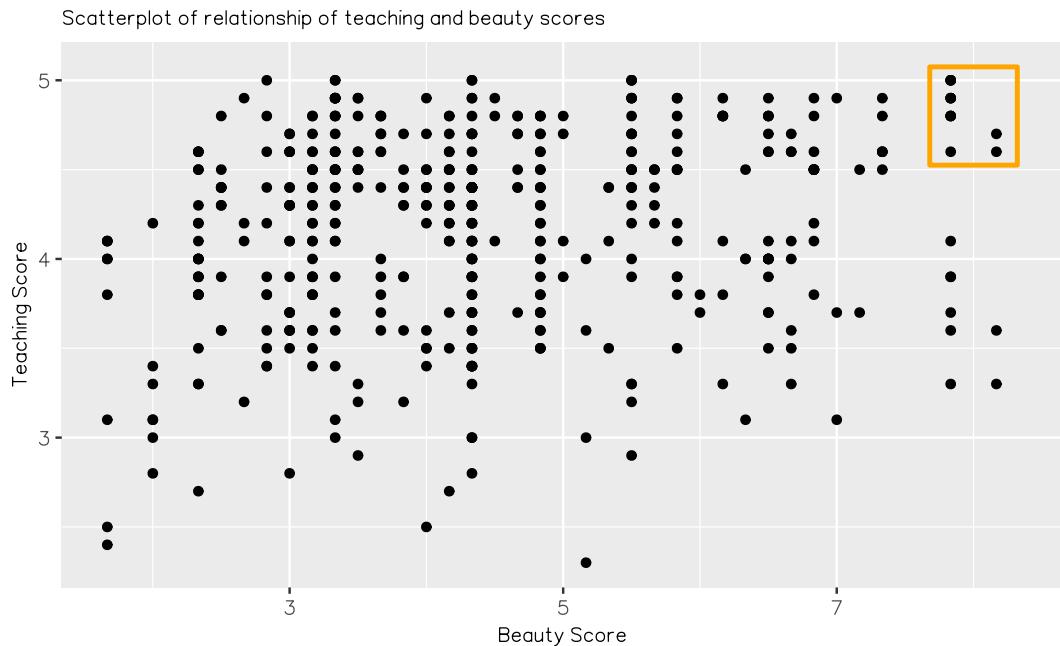


FIGURE 6.2: Instructor evaluation scores at UT Austin

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")
```

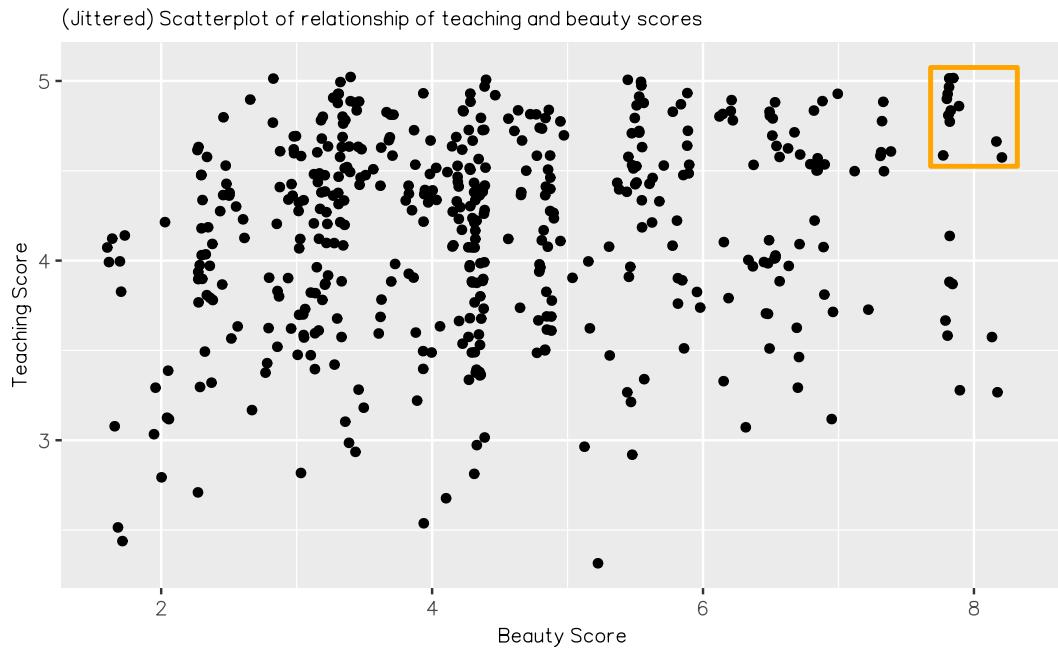


FIGURE 6.3: Instructor evaluation scores at UT Austin

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)
```

The blue line in the resulting Figure 6.4 is called a “regression line”. The re-

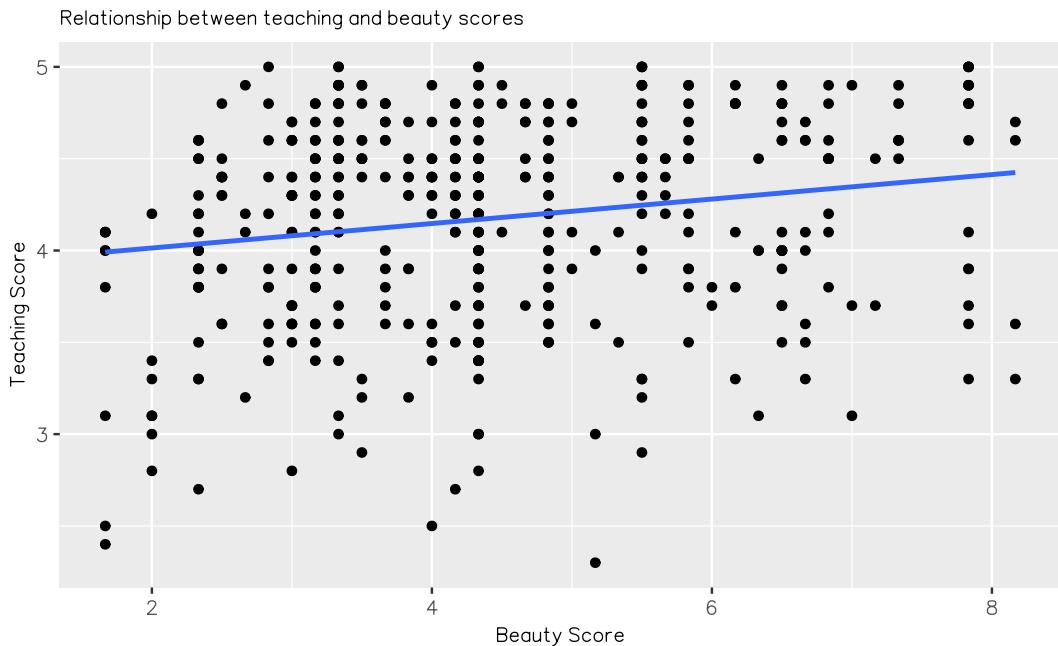


FIGURE 6.4: Regression line

gression 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 below, 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:

- 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 $x = 0$ and the slope coefficient 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 is below (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 $b_1 = b_{\text{bty_avg}}$ for `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 as beauty scores go up, so also do teaching scores go up. 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 ~ 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.

Wrapper Function



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 exploratory data analysis above?

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 regression table above:

$$\hat{y} = b_0 + b_1 \cdot x = 3.88 + 0.067 \cdot 7.333 = 4.369$$

- 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 $y - \hat{y} = 4.9 - 4.369 = 0.531$.

Now say we want to compute both the

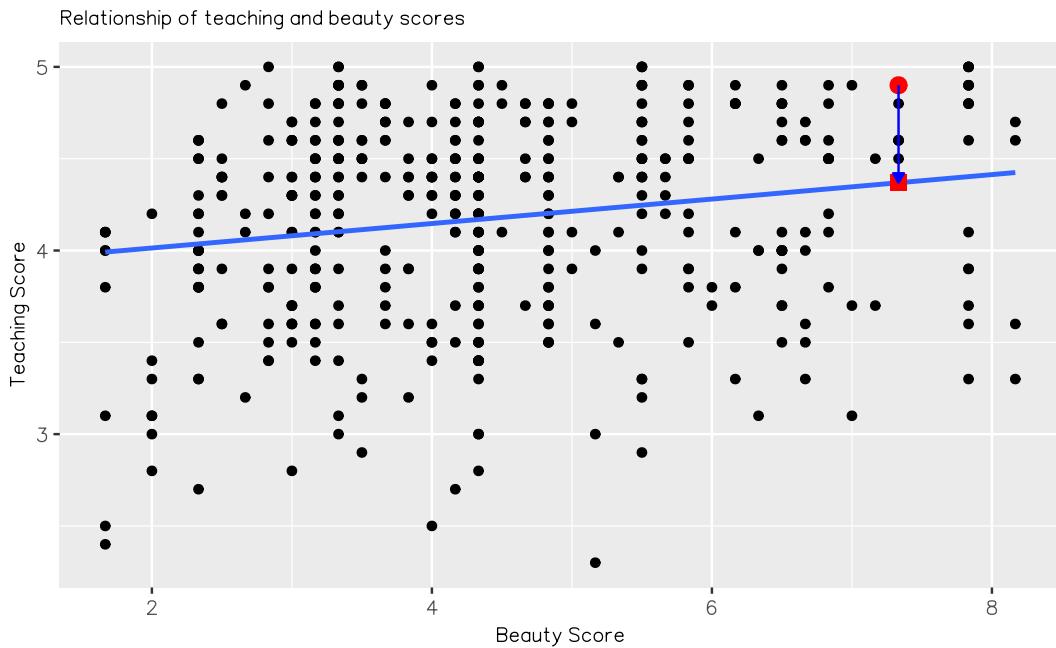


FIGURE 6.6: Example of observed value, fitted value, and residual

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 above 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.

```
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:

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

- 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 above 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` = $4.25 = 3.88 + 0.067 \cdot 5.50$ 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.

As 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
Namibia	52.9	Africa	4811
Portugal	78.1	Europe	20510
Iran	71.0	Asia	11606
Brazil	72.4	Americas	9066
Italy	80.5	Europe	28570

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:

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.

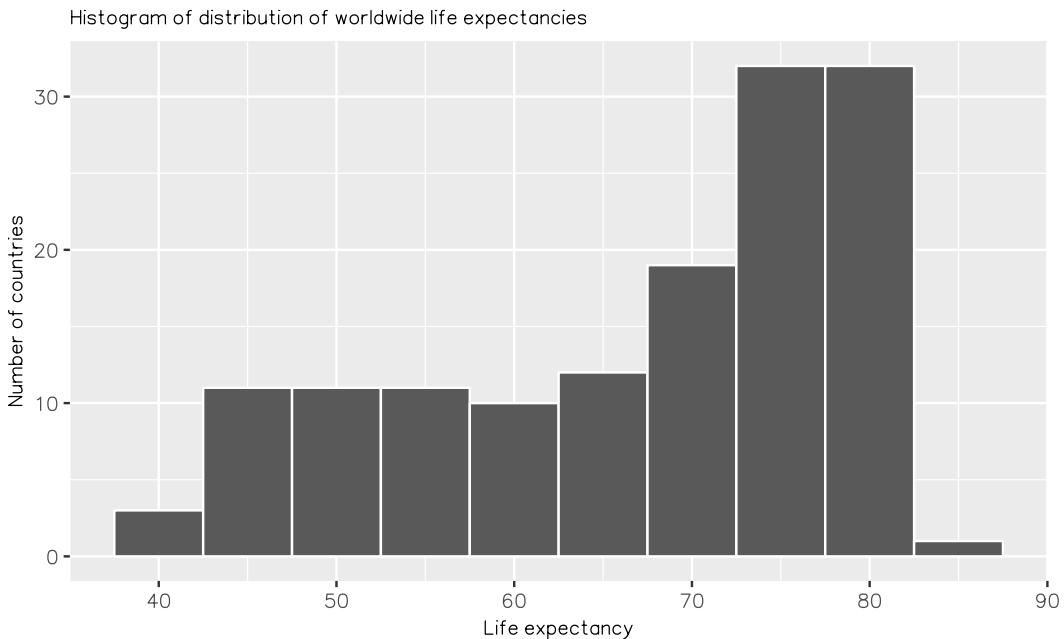


FIGURE 6.7: Histogram of Life Expectancy in 2007

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)
```

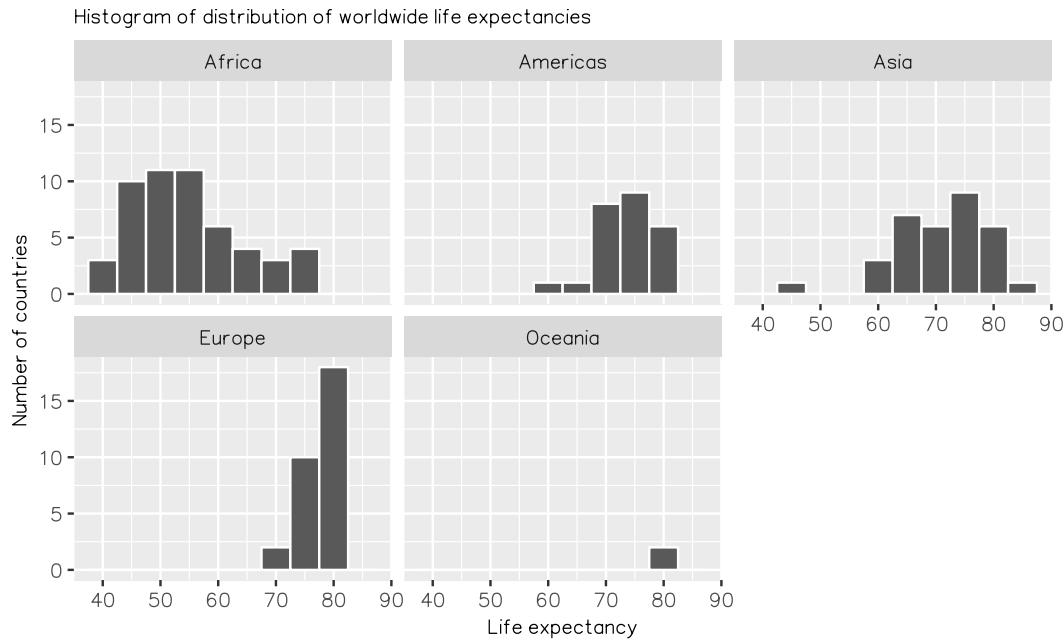


FIGURE 6.8: Life expectancy in 2007

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 greatly influenced by the fact that there are only two countries in Oceania: Australia and New Zealand.

Recall that 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")
```

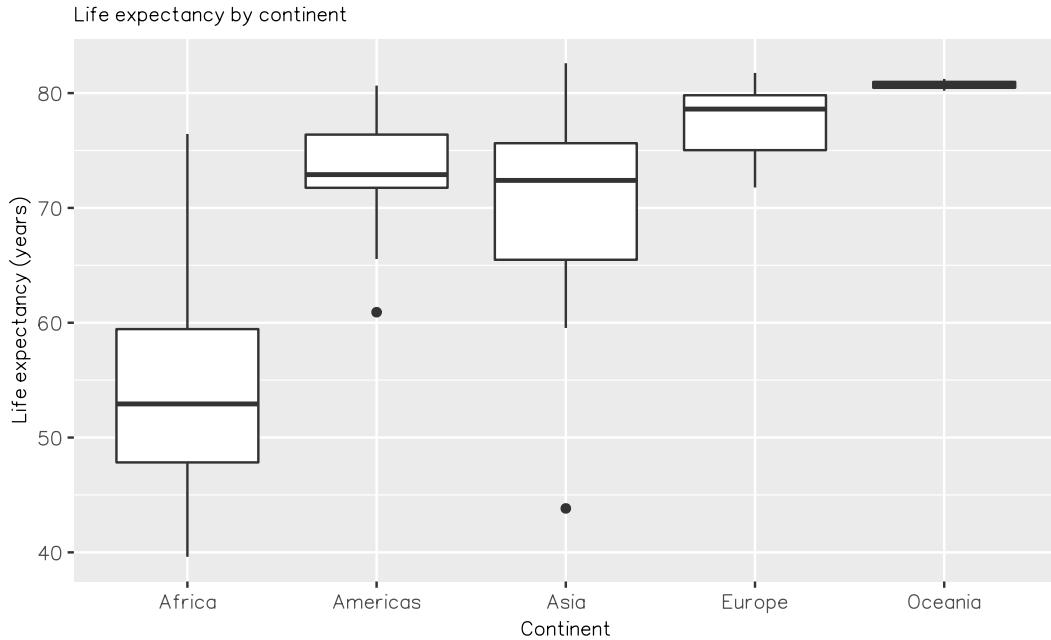


FIGURE 6.9: Life expectancy in 2007

Some people prefer comparing the distributions of a numerical variable between different levels of a categorical variable using a boxplot instead of 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 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, indicating 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.

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

```
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 Africa chosen as the “baseline for comparison” group. For no other reason than it comes first alphabetically of the five continents. Note that you can change this baseline group to be another continent if you manipulate the variable `continent`'s factor “levels”

(see Chapter 15³ of Garrett Grolemund and Hadley Wickham’s and Garrett’s book (Grolemund and Wickham, 2016)).

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 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 $\hat{y} = \widehat{\text{life exp}}$ 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:

³<https://r4ds.had.co.nz/factors.html>

$$\begin{aligned}
\widehat{\text{life exp}} &= b_0 + b_{\text{Amer}} \cdot \mathbb{1}_{\text{Amer}}(x) + b_{\text{Asia}} \cdot \mathbb{1}_{\text{Asia}}(x) + \\
&\quad b_{\text{Euro}} \cdot \mathbb{1}_{\text{Euro}}(x) + b_{\text{Ocean}} \cdot \mathbb{1}_{\text{Ocean}}(x) \\
&= 54.8 + 18.8 \cdot \mathbb{1}_{\text{Amer}}(x) + 15.9 \cdot \mathbb{1}_{\text{Asia}}(x) + \\
&\quad 22.8 \cdot \mathbb{1}_{\text{Euro}}(x) + 25.9 \cdot \mathbb{1}_{\text{Ocean}}(x) \\
&= 54.8 + 18.8 \cdot 0 + 15.9 \cdot 0 + 22.8 \cdot 0 + 25.9 \cdot 0 \\
&= 54.8
\end{aligned}$$

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 70.7 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 exploratory data analysis above?

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 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 negative 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!”

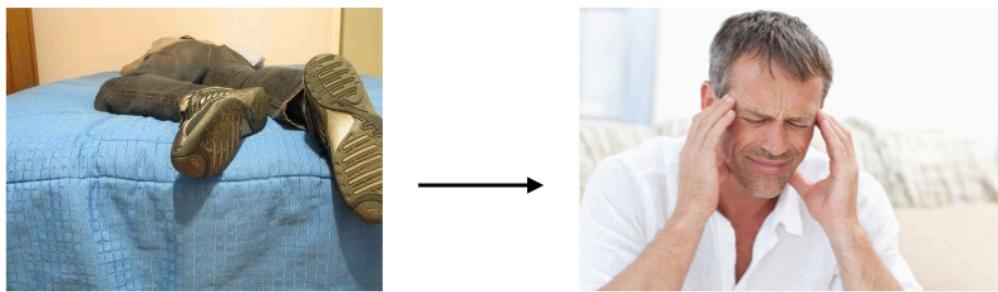


FIGURE 6.10: Does sleeping with shoes on cause headaches?

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."

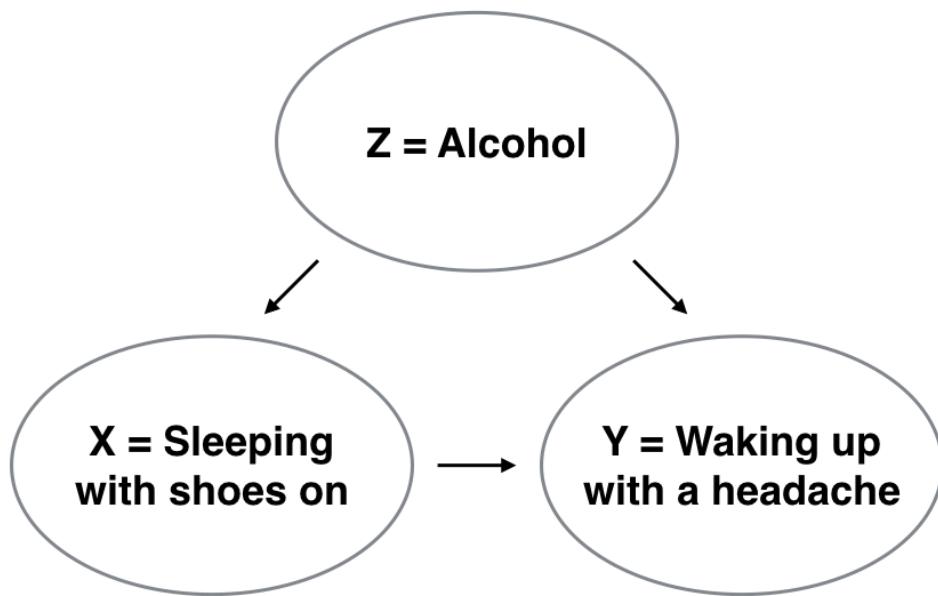


FIGURE 6.11: Causal graph.

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.

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 $x = \text{bty_avg} = 7.333$. This value is computed using the intercept and slope in the regression table above:

$$\hat{y} = b_0 + b_1 \cdot x = 54.806 + 18.802 \cdot 7.333 = 4.369$$

- 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 the above 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

⁴<http://www.tylervigen.com/spurious-correlations>

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.

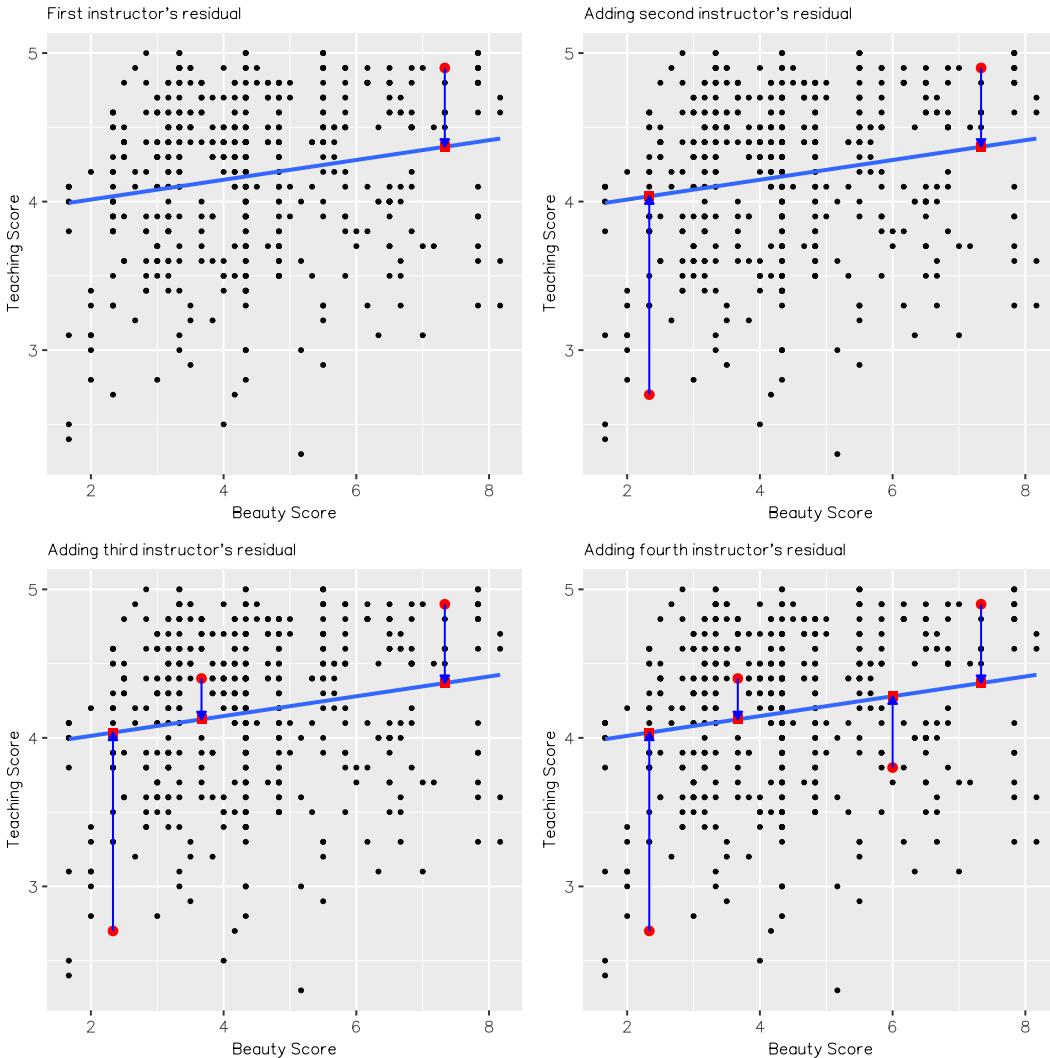


FIGURE 6.12: Example of observed value, fitted value, and residual

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:

```
# 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))

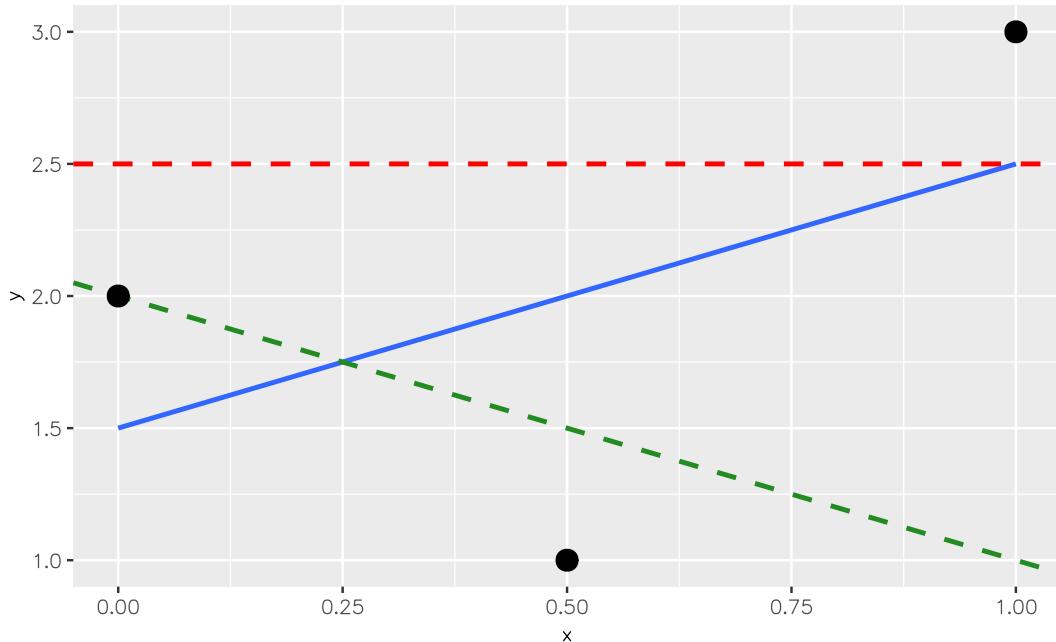
# A tibble: 1 x 1
  sum_of_squared_residuals
  <dbl>
1 132.
```

Any other line drawn in the above 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 plot below 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.

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)
```

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:

⁵<https://broom.tidyverse.org/>

⁶<https://github.com/sfirke/janitor>

```
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)
```

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 editorial decision to “wrap” all the above code into `get_regression_points()`, 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_points()`:

```
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"))
```

⁷<https://broom.tidyverse.org/>

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.

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”! Click on the image below to do so:



10

⁸https://github.com/moderndive/moderndive/blob/master/R/regression_functions.R

⁹<scripts/06-regression.R>

¹⁰<http://guessthecorrelation.com/>

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.

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
290	3.6	34	male
341	4.9	43	male
199	3.3	47	male
47	4.4	33	female
215	4.7	60	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
  gender      0      463 463        2
                top_counts ordered
  mal: 268, fem: 195, NA: 0    FALSE

-- Variable type:integer -----
variable missing complete   n  mean   sd p0 p25 p50 p75
  age        0      463 463 48.37 9.8 29  42  48  57
p100      hist
  73   

-- Variable type:numeric -----
variable missing complete   n  mean   sd p0 p25 p50 p75
  score       0      463 463 4.17 0.54 2.3  3.8  4.3  4.6
p100      hist
  5   
```

Observe for example that we have no missing data, courses taught by 268 male vs 195 female instructors, and an 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 variables: `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:

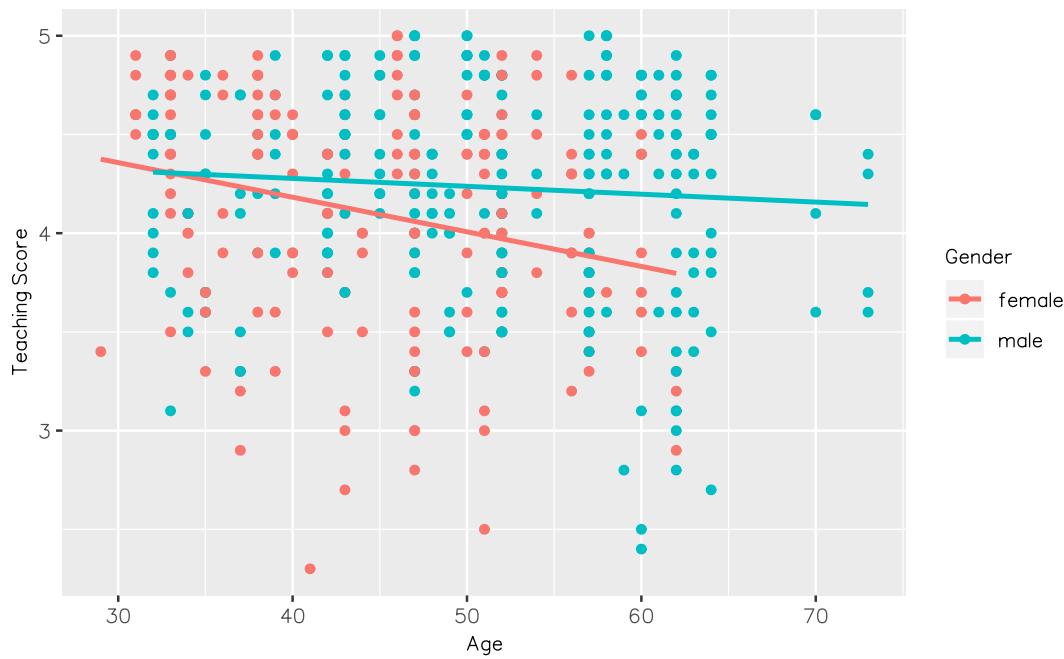


FIGURE 7.1: Colored scatterplot of relationship of teaching and beauty scores

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. as instructor's age, so also do they tend to receive lower teaching scores), the slope for age for the female instructors is *more* negative. In other words, the female instructors are paying a harsher penalty in their teaching scores than the male instructors do.

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 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

for every additional year in age for female instructors, there is an associated *decrease* of on average 0.018 units in teaching score. 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 female instructors are paying a heavier price for aging in the evaluations they receive from students.

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

$$\begin{aligned}\hat{y} &= \widehat{\text{score}} = 4.883 - 0.018 \cdot \text{age} - 0.446 \cdot \mathbb{1}_{\text{is male}}(x) + 0.014 \cdot \text{age} \cdot \mathbb{1}_{\text{is male}}(x) \\ &= 4.883 - 0.018 \cdot \text{age} - 0.446 \cdot 1 + 0.014 \cdot \text{age} \cdot 1 \\ &= 4.883 - 0.018 \cdot \text{age} - 0.446 + 0.014 \cdot \text{age} \\ &= (4.883 - 0.446) + (-0.018 + 0.014) * \text{age} \\ &= 4.437 - 0.004 \cdot \text{age}\end{aligned}$$

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 $\text{age:gendermale} = 0.014$ 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 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)
```

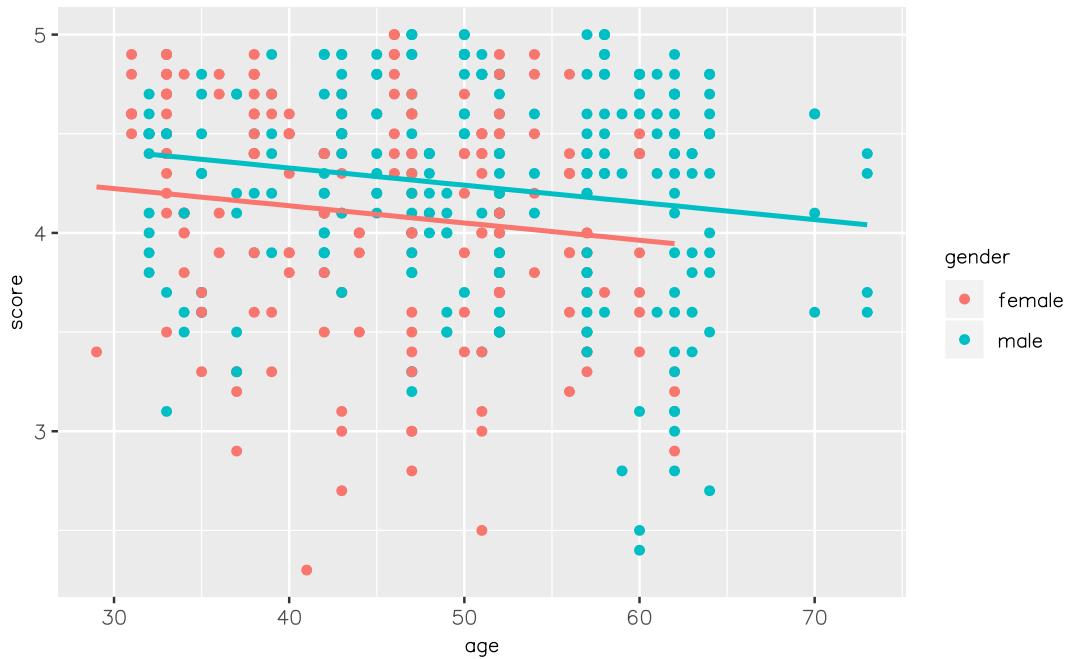


FIGURE 7.2: Parallel slopes model of relationship of score with age and gender.

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 $y \sim x_1 * x_2$, 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_interaction <- lm(score ~ age * gender, data = evals_ch7)
# Get regression table:
get_regression_table(score_model_interaction)
```

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

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



FIGURE 7.3: Comparison of interaction and parallel slopes models.

`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.

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.

**FIGURE 7.4:** Fitted values for two new professors

```
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:

²<http://www-bcf.usc.edu/~gareth/ISL/>

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:
 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 5 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 5 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
306	0	1924	24.5	165	50
121	0	1402	27.2	128	67
288	0	3098	60.4	272	69
128	0	3411	57.2	259	72
249	0	905	15.7	93	38

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
credit_limit      0        400 400 4735.6 2308.2 855
debt            0        400 400 520.01 459.76  0
p25          p50      p75  p100      hist
3088     4622.5 5872.75 13913 
```

```

68.75 459.5 863      1999  

-- Variable type:numeric -----
variable missing complete    n   mean     sd    p0    p25
income        0       400 400 45.22 35.24 10.35 21.01
p50    p75    p100      hist
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)
```

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.

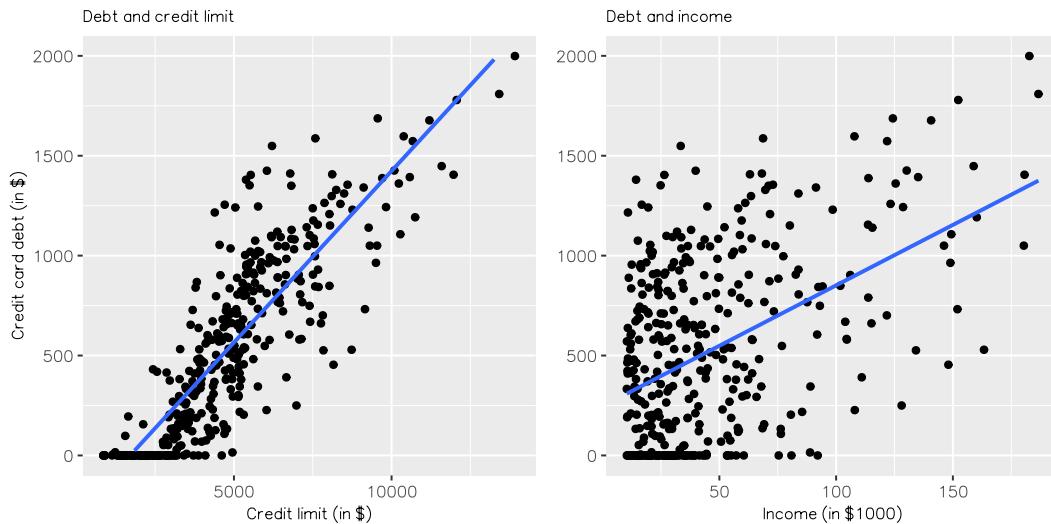


FIGURE 7.5: Relationship between credit card debt and credit limit/income

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 $y \text{ debt}$ 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 $x_1 \text{ income}$
 2. The second numerical explanatory variable $x_2 \text{ credit_limit}$

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.

Learning check

(LC7.2) Conduct a new exploratory data analysis with the same outcome

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 $\hat{y} = \hat{\text{debt}}$.

$$\begin{aligned}\hat{y} &= b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 \\ \hat{\text{debt}} &= b_0 + b_{\text{limit}} \cdot \text{limit} + b_{\text{income}} \cdot \text{income} \\ &= -387.179 + 0.263 \cdot \text{limit} - 7.663 \cdot \text{income}\end{aligned}$$

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:

$$\text{Interaction : } \hat{y} = \widehat{\text{score}} = b_0 + b_{\text{age}} \cdot \text{age} + b_{\text{male}} \cdot \mathbb{1}_{\text{is male}}(x) + b_{\text{age,male}} \cdot \text{age} \cdot \mathbb{1}_{\text{is male}}$$

$$\text{Parallel slopes : } \hat{y} = \widehat{\text{score}} = b_0 + b_{\text{age}} \cdot \text{age} + b_{\text{male}} \cdot \mathbb{1}_{\text{is male}}(x)$$

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 7.6.

```
# 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")
```

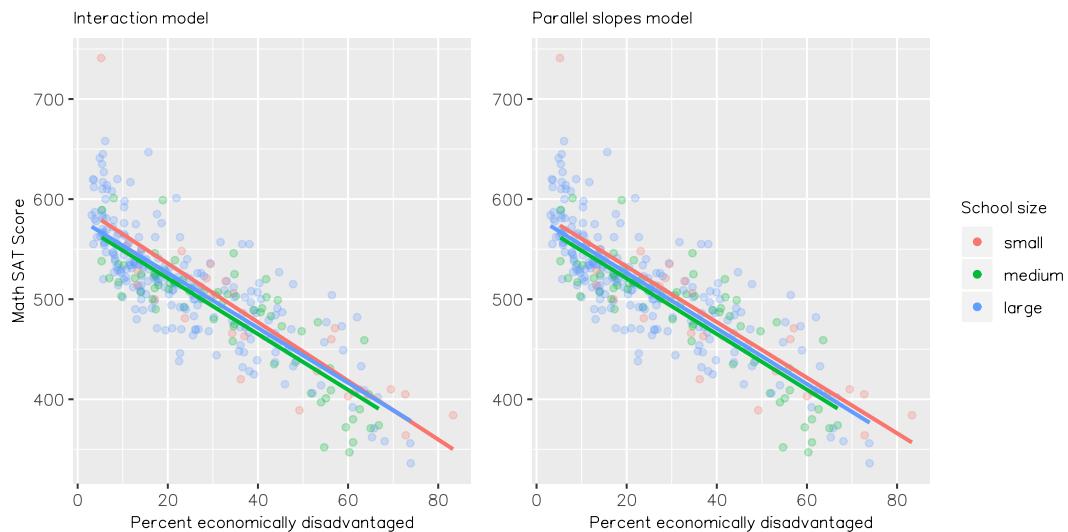


FIGURE 7.6: Comparison of interaction and parallel slopes models.

Looking closely at the left-hand plot of Figure 7.6, 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 percentage of students that are disadvantaged `perc_disadvan:size`_{medium} = 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 7.8 shows a histogram of all 400 values of `credit_limit`, along with vertical red lines that cut up the data into quartiles, meaning:

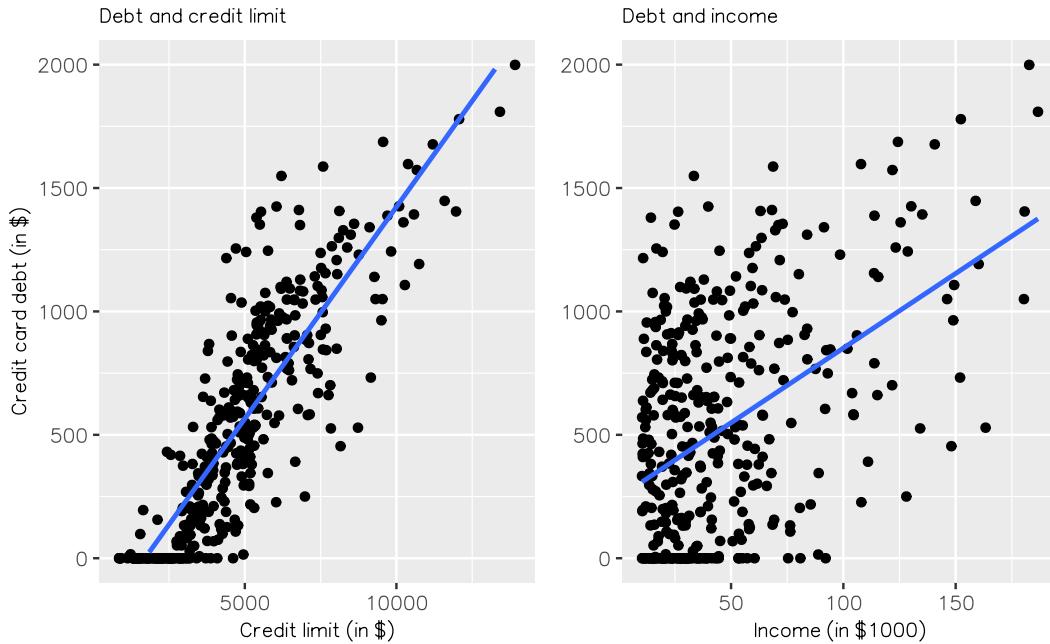


FIGURE 7.7: Relationship between credit card debt and credit limit/income

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 7.9 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 disappears! Only for the high bracket does the relationship stay somewhat

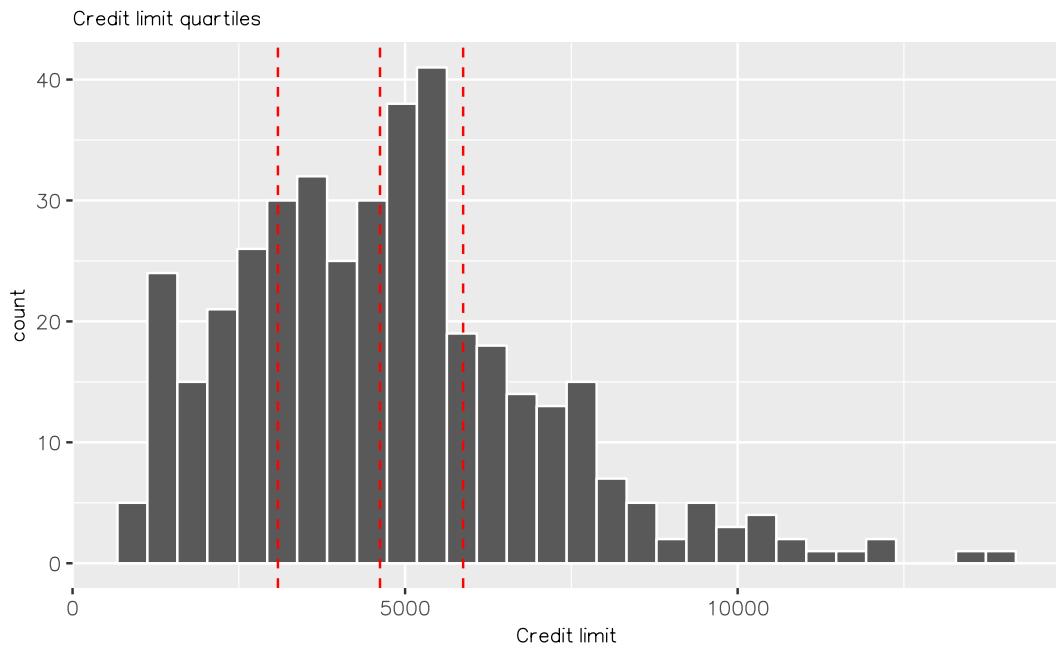


FIGURE 7.8: Histogram of credit limits and quartiles



FIGURE 7.9: Relationship between credit card debt and income for different credit limit groups

positive. In this example, credit limit is a *confounding variable* for credit card debt and income. This is a phenomenon known as Simpson's Paradox³ 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 here⁴.

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 "assumptions for inference for regression". On to the next one!

³https://en.wikipedia.org/wiki/Simpson%27s_paradox

⁴[scripts/07-multiple-regression.R](#)

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). If needed, read Section 2.3 for information on how to install and load R packages.

```
library(dplyr)
library(ggplot2)
library(moderndive)
```

8.1 Sampling 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?

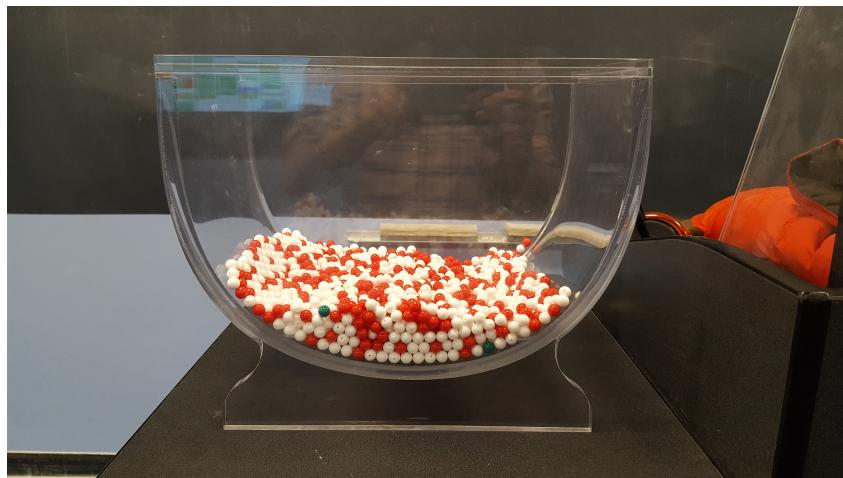


FIGURE 8.1: A bowl with red and white balls.

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 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



FIGURE 8.2: Inserting a shovel into the bowl.

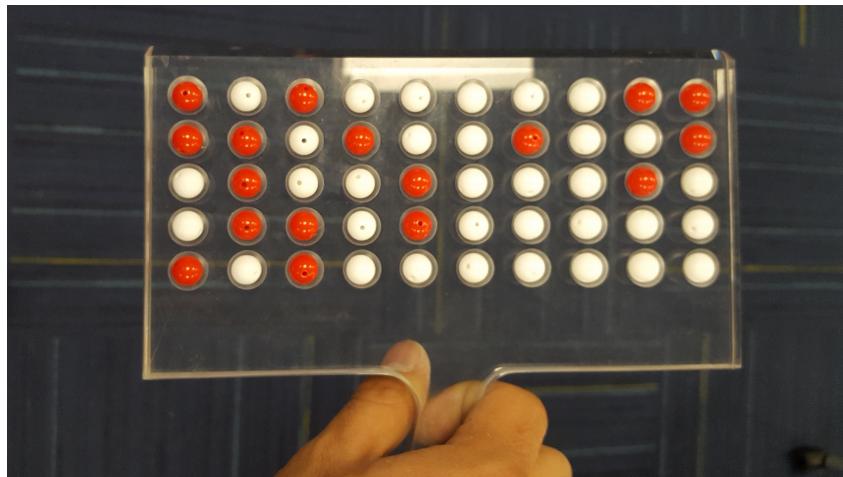


FIGURE 8.3: Fifty balls from the bowl.

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.

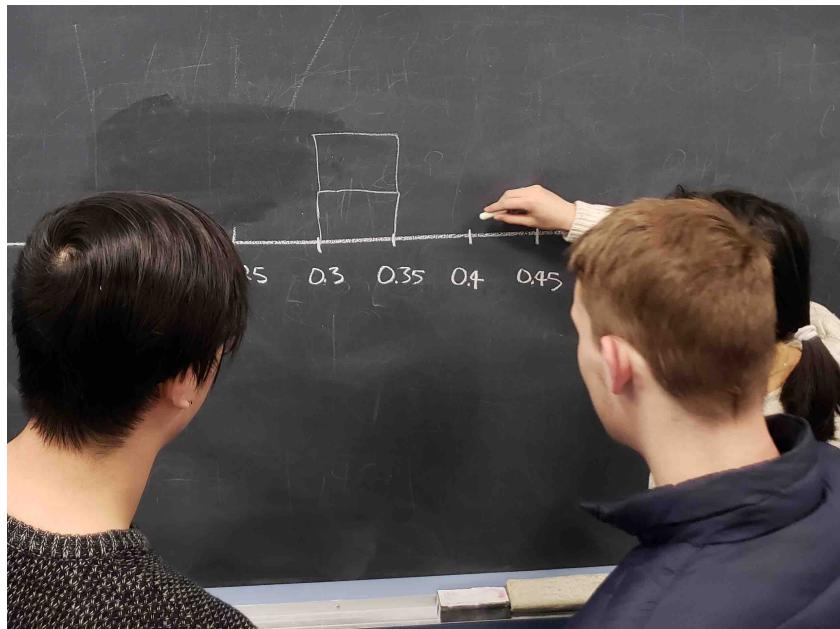


FIGURE 8.5: Constructing a histogram of proportions.

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.

Observe the following about the histogram in Figure 8.6:

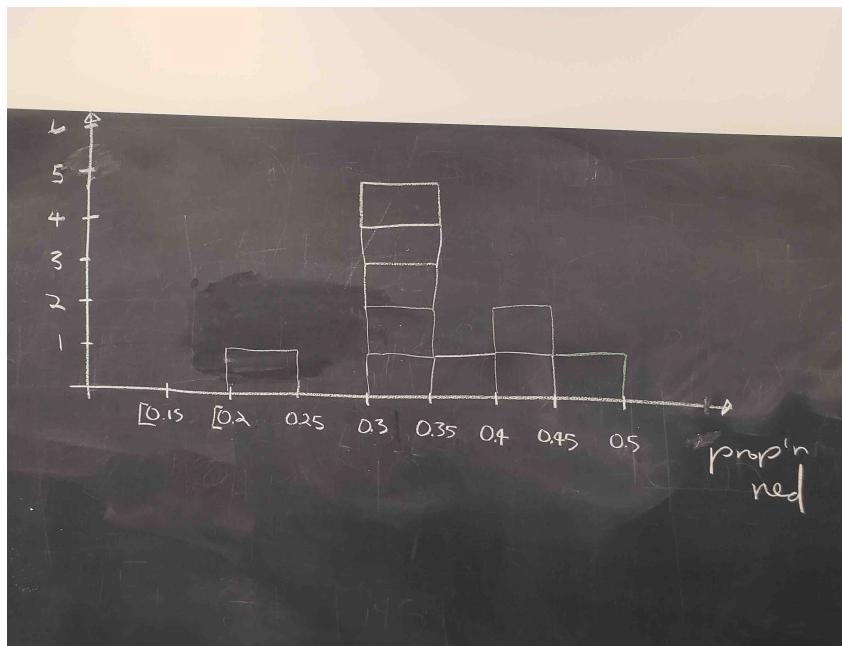


FIGURE 8.6: Hand-drawn histogram of 33 proportions.

- 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

TABLE 8.1: First 10 out of 33 groups' proportion of 50 balls that are red.
(continued)

group	replicate	red_balls	prop_red
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 are we doing here?

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

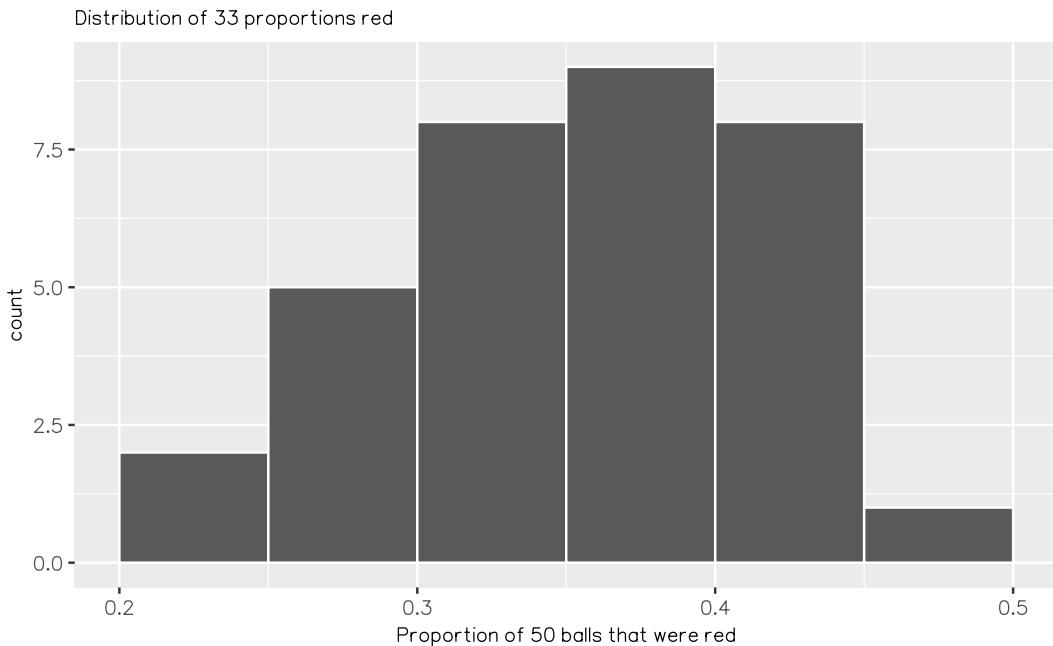


FIGURE 8.7: Distribution of 33 proportions based on 33 samples of size 50

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.

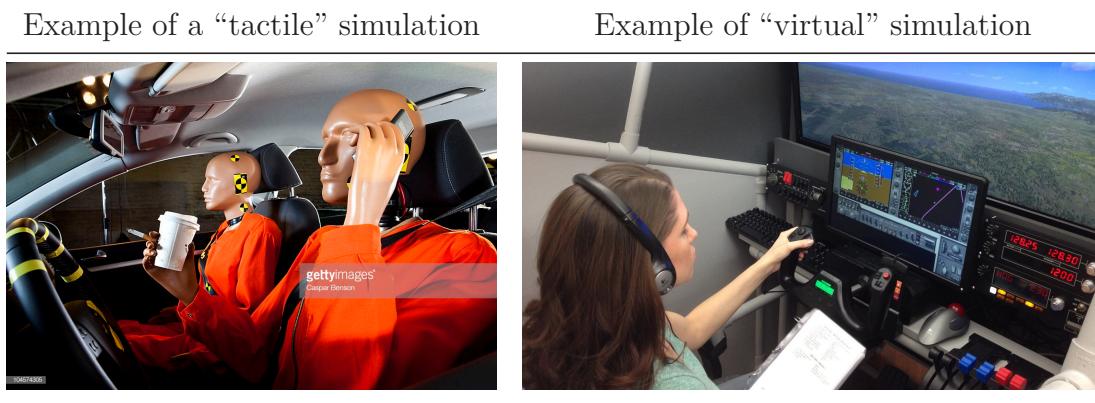
We'll close this chapter by generalizing the above sampling from the bowl activity to other scenarios, distinguishing between *random sampling* and *random assignment*, presenting the theoretical result underpinning all our results, and presenting a few mathematical formulas that relate to the concepts and ideas explored in this chapter.

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 a simulation states: “A simulation is an approximate imitation of the operation of a process or system.”¹ One example of simulations in practice are 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 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.

¹Wikipedia entry for simulation²



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 x 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 con-

tains 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 8.3.

TABLE 8.3: First 10 sampled balls of 50 in virtual sample

replicate	ball_ID	color
1	1500	white
1	1767	red
1	1035	white
1	245	white
1	1121	white
1	1828	white
1	721	white
1	1729	red
1	770	white
1	1499	red

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     1500 white FALSE
2       1     1767 red   TRUE
3       1     1035 white FALSE
4       1      245 white FALSE
5       1     1121 white FALSE
6       1     1828 white FALSE
7       1      721 white FALSE
8       1     1729 red   TRUE
9       1      770 white FALSE
10      1     1499 red   TRUE
# ... 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        17
```

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       17      0.34
```

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       17      0.34
```

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 8.3 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 8.4. As one would expect, there is variation in the resulting `prop_red` proportions red for the first 10 out 33 repeated/replicated samples.

TABLE 8.4: First 10 out of 33 virtual proportion of 50 balls that are red.

replicate	red	prop_red
1	18	0.36

TABLE 8.4: First 10 out of 33 virtual proportion of 50 balls that are red.
(continued)

replicate	red	prop_red
2	20	0.40
3	19	0.38
4	18	0.36
5	15	0.30
6	18	0.36
7	19	0.38
8	13	0.26
9	23	0.46
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")
```

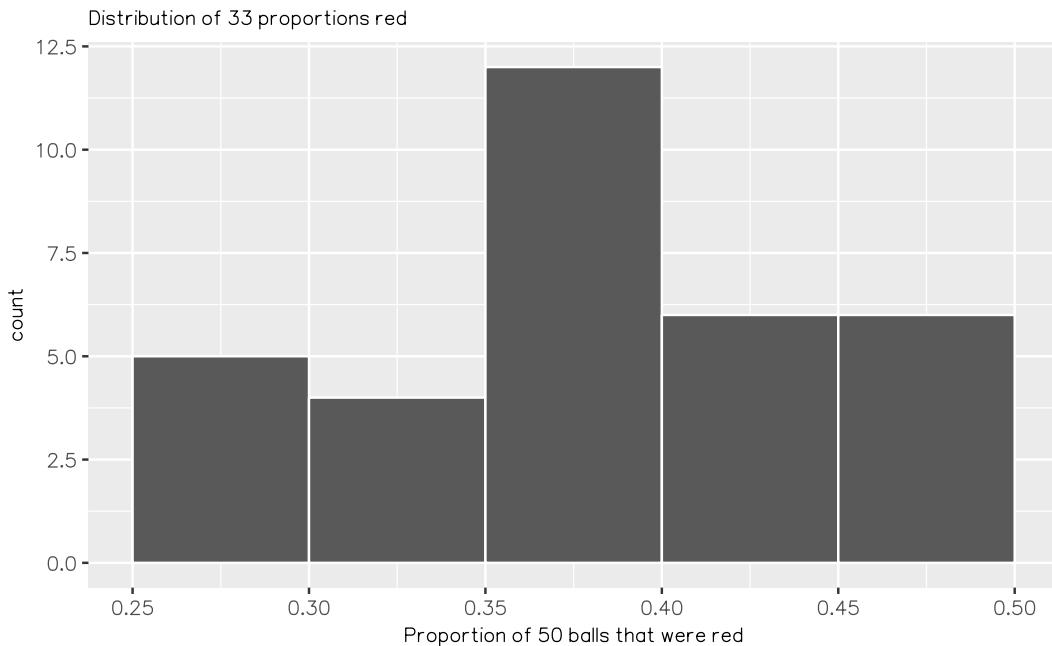


FIGURE 8.8: Distribution of 33 proportions based on 33 samples of size 50

Observe that occasionally we obtained proportions red that are less than 0.3 =

30%, while on the other hand we 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 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.

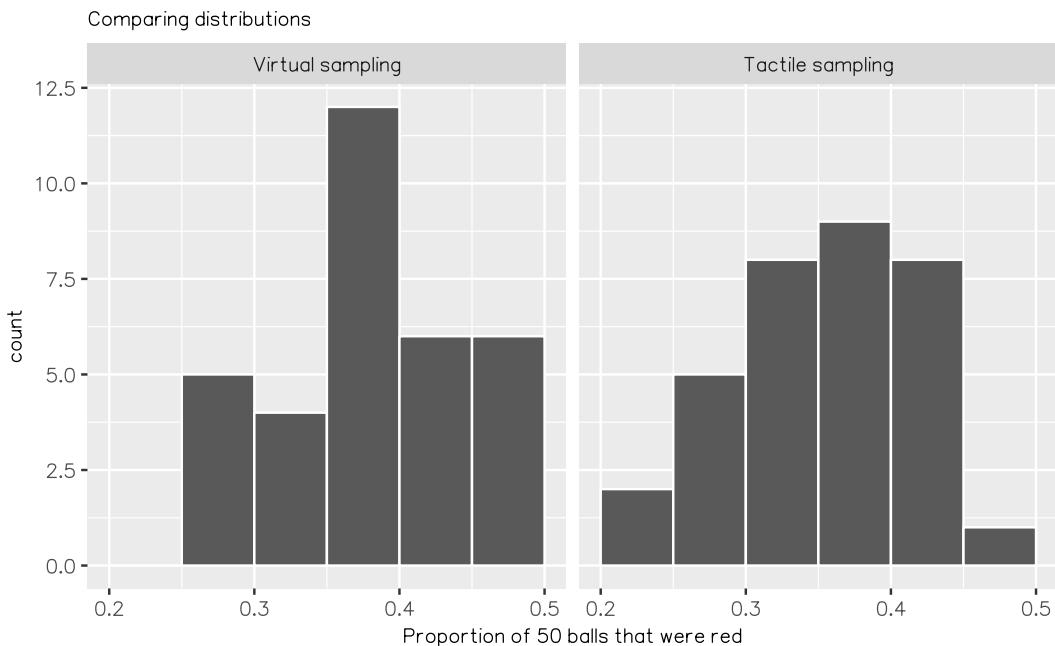


FIGURE 8.9: Comparing 33 virtual and 33 tactile proportions red.

8.2.3 Using the virtual shovel 1000 times

Now say we want 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")
```

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 Appendix A for a brief discussion on properties of the Normal distribution.

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.

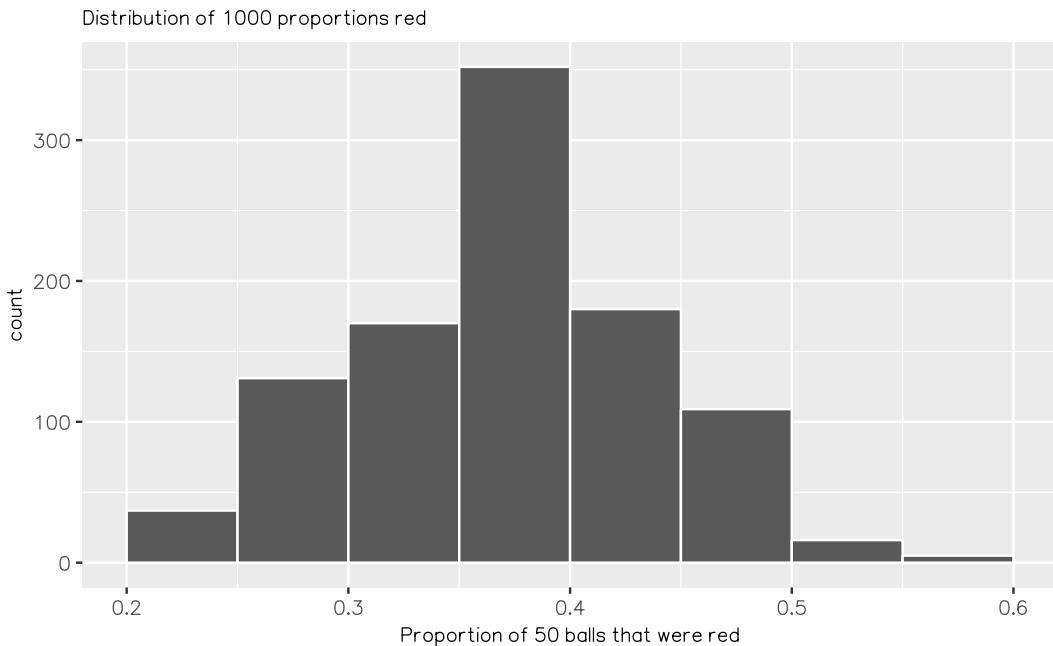
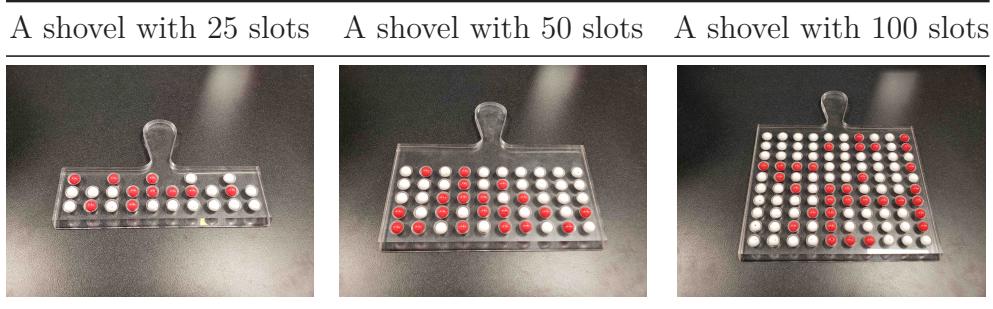


FIGURE 8.10: Distribution of 1000 proportions based on 33 samples of size 50



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/relicated samples at 1000:

1. Virtually use the appropriate shovel to generate 1000 samples with `size` balls.
2. Compute the resulting 1000 replicated 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 8.11. What do you observe?

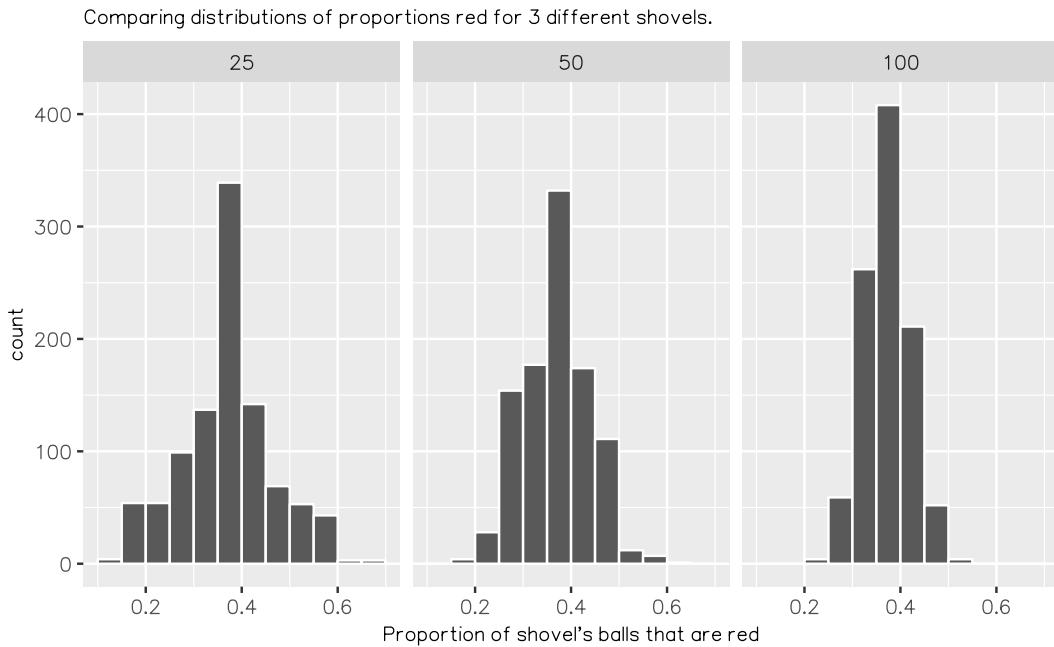


FIGURE 8.11: Comparing the distributions of proportion red for different sample sizes

Observe that as the sample size increases, the spread of the 1000 replicates of the proportion red decreases. In other words, as the sample size increases, there are less differences due to sampling variation and the distribution centers more tightly around the same value. Eyeballing Figure 8.11, 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 3 measures of spread of the distributions in Table 8.6.

TABLE 8.6: Comparing standard deviations of proportions red for 3 different shovels.

Number of slots in shovel	Standard deviation of proportions red
25	0.096
50	0.068
100	0.048

As we observed visually in Figure 8.11, 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.

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 8.11 and Table 8.6, 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 terminology 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. 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 are 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 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 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.
-

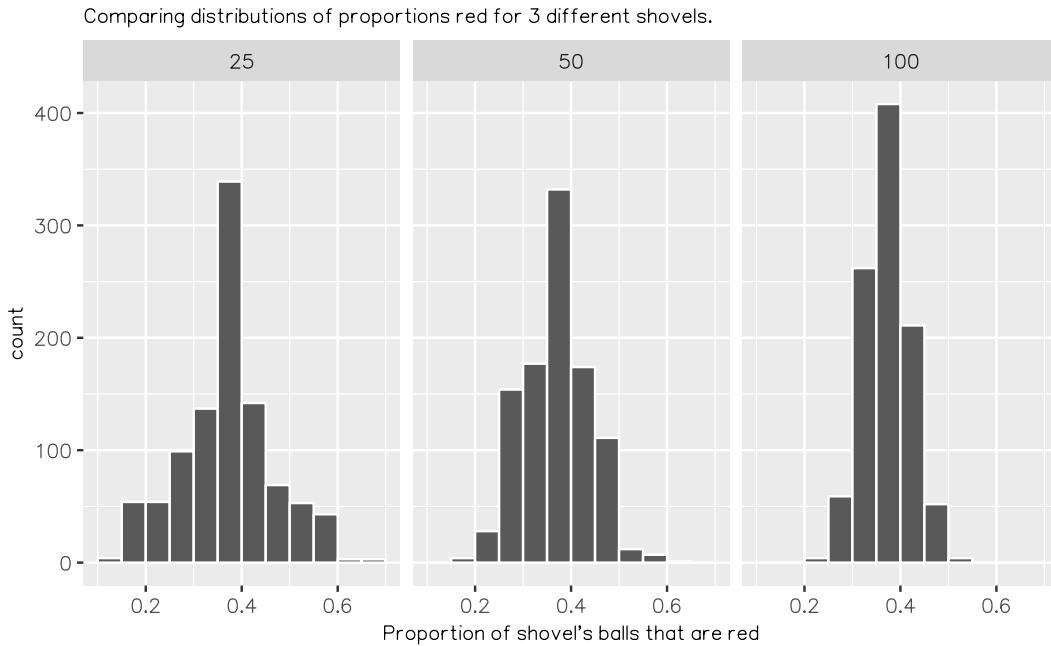
Restricting consideration to a shovel with 50 slots from our simulations,

- 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`”.

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 8.11 again below.



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 8.6, which we display again below:

Number of slots in shovel	Standard deviation of proportions red
25	0.096
50	0.068
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 random sample to 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 8.11 but using our new terminology, notation, and definitions relating to sampling in Figure 8.12.

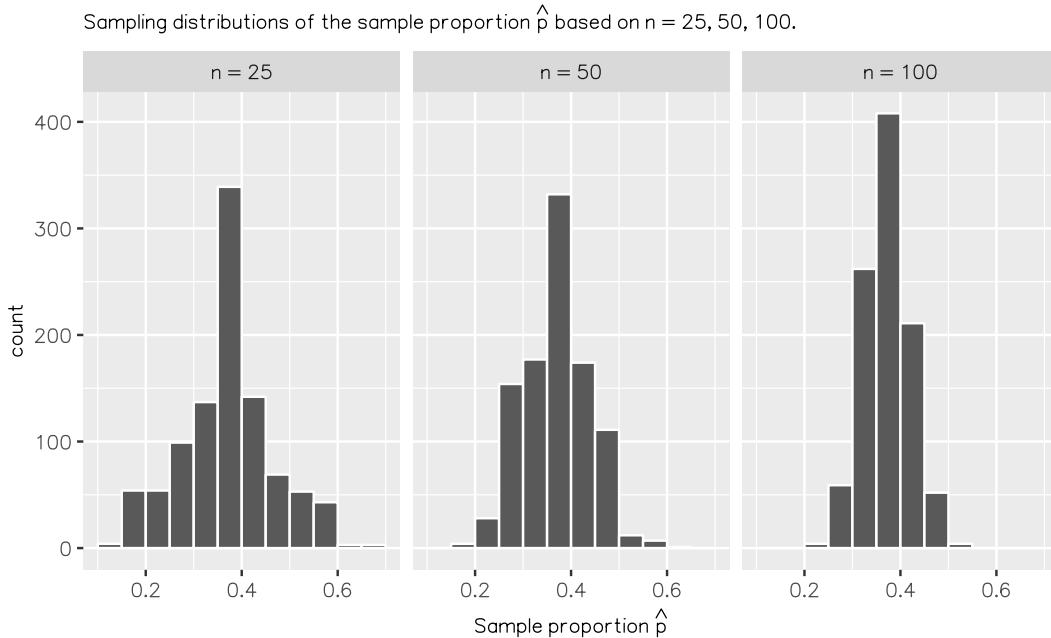


FIGURE 8.12: Three sampling distributions of the sample proportion \hat{p} .

Furthermore, let's re-display Table 8.6 but using our new terminology, notation, and definitions relating to sampling in Table 8.7.

TABLE 8.7: Three standard errors of the sample proportion based on $n = 25, 50, 100$.

Sample size	Standard error of \widehat{p}
$n = 25$	0.096
$n = 50$	0.068
$n = 100$	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.

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 on the contexts 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 8.11 and 8.12, but now with a vertical red line marking the true population proportion p of balls that are red = 37.5% in Figure 8.13. 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 p .

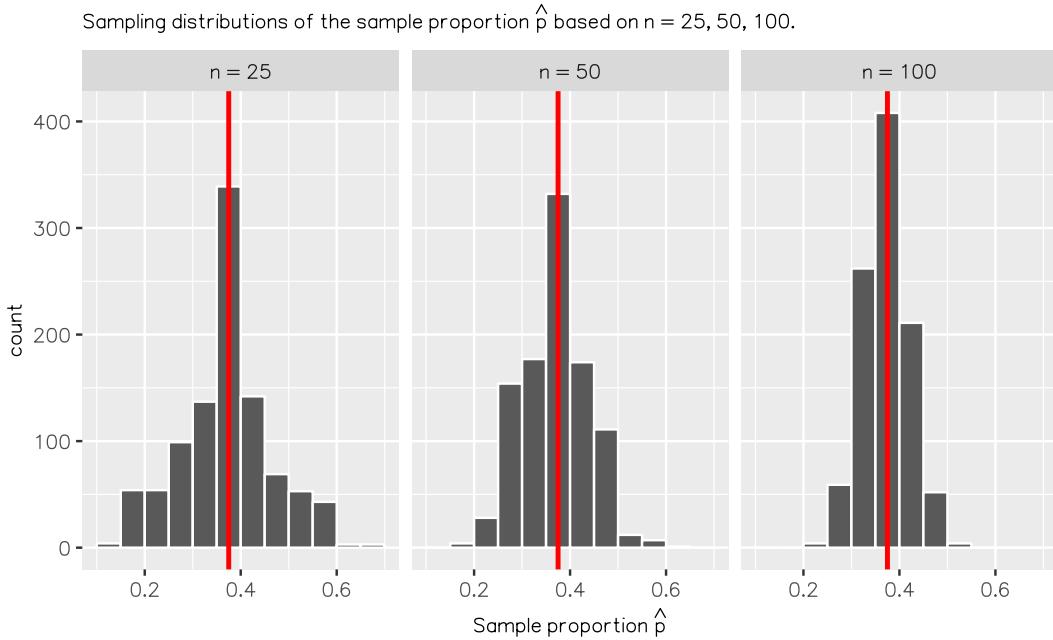


FIGURE 8.13: Three sampling distributions with population proportion p marked in red.

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 8.13. 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 8.14 illustrates the difference.

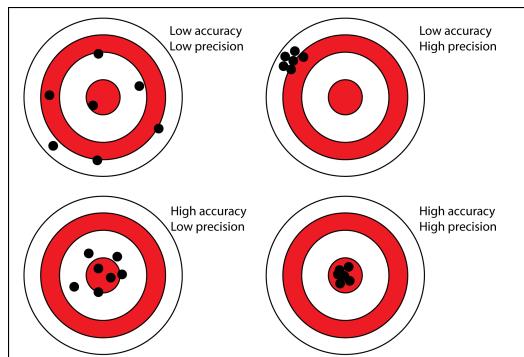


FIGURE 8.14: Comparing accuracy and precision

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 what did we do any of this for?” In other words, “If you already knew the value of the true unknown population parameter, then why did we do any sampling?” You might also be asking: “Why did we take 1000 repeated/replicated samples 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. This was also done to show the power of the technique of sampling when trying to estimate a population parameter. Since we knew the value was 37.5%, we could show just how well the different sample sizes approximated this value in their sampling distributions. We present one case study of a real-life sampling scenario in the next section: polling.

8.4 Case study: Polls

In December 4, 2013 National Public Radio in the US reported on a recent, at the time, poll of President Obama's approval rating among young Americans aged 18-29 in an article Poll: Support For Obama Among Young Americans Eroding³. 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.

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.

³<https://www.npr.org/sections/itsallpolitics/2013/12/04/248793753/poll-support-for-obama-among-young-americans-eroding>

4. **Sampling:** How do you collect the sample of size n individuals or observations?
 - Simulation: Using a shovel with n slots.
 - 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?
 - Obama poll: Is the sample of $n = 2089$ young Americans representative of all young Americans aged 18-29?
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?
 - 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 41% of *all* young Americans approve of Obama?
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? For example, if this was conducted using only mobile phone numbers, would people without mobile phones be included? What if those who disapproved of Obama were less likely to agree to take part in the poll? What about if this were an internet poll on a certain news website? Would non-readers of this website be included? We need to ask the Har-

vard University Institute of Politics pollsters about their *sampling methodology*.

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 need to ask the Harvard University Institute of Politics pollsters about their *sampling methodology*.

Once again, let's revisit the sampling paradigm:

- 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.
-

8.5 Conclusion

8.5.1 Central Limit Theorem

What you did in Sections 8.1 and 8.2 (in particular in Figure 8.11 and Table 8.6) was demonstrate a very famous theorem, or mathematically proven truth, called the *Central Limit Theorem*. It loosely states that when sample means and sample proportions are based on larger and larger sample sizes, the sampling distribution of these two point estimates become more and more normally shaped and more and more narrow. In other words, their sampling distributions become more normally distributed and the spread/variation of these sampling distributions as quantified by their standard errors gets smaller. Shuyi Chiou, Casey Dunn, and Pathikrit Bhattacharyya created the following 3m38s video at <https://www.youtube.com/embed/jvoxEYmQHNM> explaining this crucial statistical theorem using the average weight of wild bunny rabbits and the average wing span of dragons as examples. Enjoy!

8.5.2 Summary table

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 a 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 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.

Note that the sample mean is traditionally noted as \bar{x} but can also be thought

of as an estimate of the population mean μ . Thus, it can also be denoted as $\hat{\mu}$ as shown below in the table.

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	$\hat{\mu}$ or \bar{x}
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	Sample regression slope	$\hat{\beta}_1$ or b_1
6	Population regression intercept	β_0	Sample regression intercept	$\hat{\beta}_0$ or b_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.

8.5.3 Additional resources

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

8.5.4 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*. 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% being 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*.

⁴ [scripts/08-sampling.R](#)

9

Confidence Intervals

In preparation for our first print edition to be published by CRC Press in Fall 2019, we're remodeling this chapter a bit. Don't expect major changes in content, but rather only minor changes in presentation. Our remodeling will be complete and available online at [ModernDive.com¹](https://ModernDive.com) by early Summer 2019!

In Chapter 8, we explored the process of sampling from a representative sample to build a sampling distribution. The motivation there was to use multiple samples from the same population to visualize and attempt to understand the variability in the statistic from one sample to another. Furthermore, recall our concepts and terminology related to sampling from the beginning of Chapter 8:

Generally speaking, we learned that if the sampling of a sample of size n is done at *random*, then the resulting sample is *unbiased* and *representative* of the *population*. Thus, any result based on the sample can *generalize* to the population, and hence the **point estimate/sample statistic** computed from this sample is a “good guess” of the unknown population parameter of interest.

Specific to the bowl, we learned that if we properly mix the balls first we ensure the randomness of samples extracted using the shovel with $n = 50$ slots. Further, then the contents of the shovel will “look like” the contents of the bowl. Thus, any results based on the sample of $n = 50$ balls can generalize to the large bowl of $N = 2400$ balls. Hence the sample proportion red \hat{p} of the $n = 50$ balls in the shovel is a “good guess” of the true population proportion red p of the $N = 2400$ balls in the bowl.

We emphasize that we used a point estimate/sample statistic, in this case the sample proportion \hat{p} , to estimate the unknown value of the population parameter, in this case, the population proportion p . In other words, we are using the sample to **infer** about the population.

In most cases, we don't have the population values as we did with the bowl of balls. We only have a single sample of data from a larger population. We'd like to be able to make some reasonable guesses about population parameters using that single sample to create a range of plausible values for a population parameter. This range of plausible values is known as a **confidence interval** and will be the focus of this chapter. And how do we use a single sample to get some idea of how other samples might vary in terms of their statistic values? One common way this is done is via a process known as **bootstrapping** that will be the focus of the beginning sections of this chapter.

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(janitor)
library(moderndive)
library(infer)
```

9.1 Resampling activity

As we did in Chapter 8, we'll begin with a hands-on activity.

9.1.1 What is the average year of circulated US pennies in 2019?

In order to begin to answer this question, we obtained a sample of 50 US pennies collected from a bank in the US. An image of these pennies is below.

Each of the pennies was numbered as 1-50 starting in the top left and ending in the bottom right progressing row by row. This is shown with the "ID" values on top of and near the middle of each penny for clarity. The year of mint is also shown on top of and to the right side of the penny as well.

**FIGURE 9.1:** 50 US pennies**FIGURE 9.2:** 50 US pennies labelled

The `moderndive` package contains this data on the pennies collected and minted in the United States in 2019. Let's explore this sample data first:

```
pennies_sample_2
```

```
# A tibble: 50 x 2
  ID    year
  <int> <int>
1     1    2002
```

```
2     2  1986
3     3  2017
4     4  1988
5     5  2008
6     6  1983
7     7  2008
8     8  1996
9     9  2004
10    10 2000
# ... with 40 more rows
```

The `pennies_sample_2` data frame has rows corresponding to a single penny with two variables:

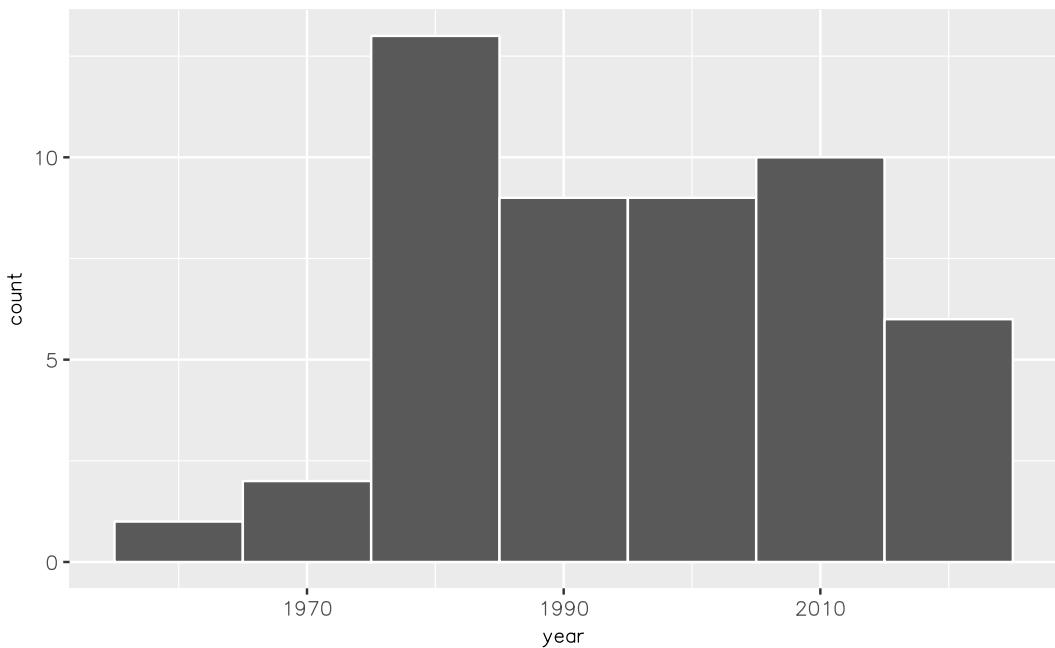
- `ID` of the penny stored as an integer to identify the penny from other pennies in the sample and
- `year` of minting as shown on the penny and stored as an integer.

Suppose we are interested in understanding some properties of the mean year of **all** US pennies using this data on 50 pennies collected in 2019. How might we go about that? Let's begin by understanding some of the properties of `pennies_sample_2` using data visualization from Chapter 3 and data wrangling from Chapter 4.

Exploratory data analysis on original sample

First, let's visualize the values in this sample as a histogram:

```
ggplot(pennies_sample_2, aes(x = year)) +
  geom_histogram(binwidth = 10, color = "white")
```



We see a skewed distribution here that has many values in the 1990s through 2010s with only a few older than 1970. If `pennies_sample_2` is a representative sample from the population, we'd expect the age of all US pennies collected in 2019 to have a similar shape, a similar spread, and similar measures of central tendency like the mean.

So where does the mean value fall for this sample? This point will be known as our **point estimate** and provides us with a single number that could serve as the guess to what the true population mean year might be. Recall how to find this using the `dplyr` package:

```
x_bar <- pennies_sample_2 %>%
  summarize(stat = mean(year))
```

We've denoted this *sample mean* as \bar{x} , which is the standard symbol for denoting the mean of a sample. Our point estimate is, thus, $\bar{x} = 1995.44$. Note that this is just one sample though providing just one guess at the population mean. What if we'd like to have another guess?

This should all sound similar to what we did in Chapter 8. There instead of collecting just a single scoop of balls, we had many different students use the shovel to scoop different samples of red and white balls. We then calculated a sample statistic (the sample proportion) from each sample. But, we don't

have a population to pull from here with the pennies. We only have this one sample.

The process of **resampling** allows us to use a single sample to generate many different samples that will act as our way of approximating a sampling distribution. Let's see how this works using our sample of fifty pennies.

9.1.2 Using resampling once

The fifty pennies represent one possible sample from all of the pennies in our population. As we saw in Chapter 8, there are other potential samples that could have been selected. All we know about the population is that it contains pennies with years equal to those in our sample. In our sample, we drew three pennies with a year of 1999. But will we always draw exactly three pennies with a year of 1999 if we drew other samples of size 50? More than likely not. We might draw 0, 1, 2, or even all 50 pennies as being from 1999. The same can be said for the other 26 years that are represented in our sample.

Taking our sample of fifty pennies, we can perform a resampling process to obtain another potential sample from the population. This will help us get a sense for the variability in another sample like what we did in Chapter 8. Here's the process:

1. First, pretend that each of the 50 values of `year` in `pennies_sample_2` was written on a small piece of paper. Recall that these values were 2002, 1986, 2017, 1988, 2008, etc. We also note the `ID` values here on each sheet of paper of 1, 2, 3, 4, 5, etc. to keep track of how resampling works.
2. Now, put the 50 small pieces of paper into a receptacle such as a baseball cap.
3. Shake up the pieces of paper.
4. Draw “at random” from the cap to select one piece of paper.
5. Write down the value of `age` and `ID` on this piece of paper. Say that it is 1976 for `year` and 32 for `ID`.
6. Now, place this piece of paper corresponding to the 32nd coin with a value of 1976 back into the cap.
7. Draw “at random” again from the cap to select a piece of paper. Note that this is the *sampling with replacement* part since you may draw this 39th coin again.
8. Repeat this process until you have drawn 50 pieces of paper and written down the `ID` and `year` values for these 50 pieces of paper. Completing this repetition produces ONE resample.

Let's enter these values into R in a tibble called `pennies_resample` and observe the first ten rows of this resample.

```
# A tibble: 50 x 2
  ID    year
  <int> <int>
1 32    1976
2 37    1962
3 22    1976
4 6     1983
5 24    2017
6 39    2015
7 16    2015
8 37    1962
9 17    2016
10 32   1976
# ... with 40 more rows
```

Now we can view what the actual pennies sampled would look like corresponding to this resample:



FIGURE 9.3: 50 resampled US pennies labelled

Note the different `id` values and `year` values for each penny here. There are also some pennies drawn multiple times and some pennies not drawn at all from our original sample.

Exploratory data analysis on the resample

Let's look at the distribution of years in the original sample of `pennies_sample_2` compared to the resample of `pennies_resample`:

```
ggplot(pennies_sample_2, aes(x = year)) +
  geom_histogram(binwidth = 10, color = "white") +
  labs(title = "50 US pennies labelled")
ggplot(pennies_resample, aes(x = year)) +
  geom_histogram(binwidth = 10, color = "white") +
  labs(title = "50 resampled US pennies labelled")
```

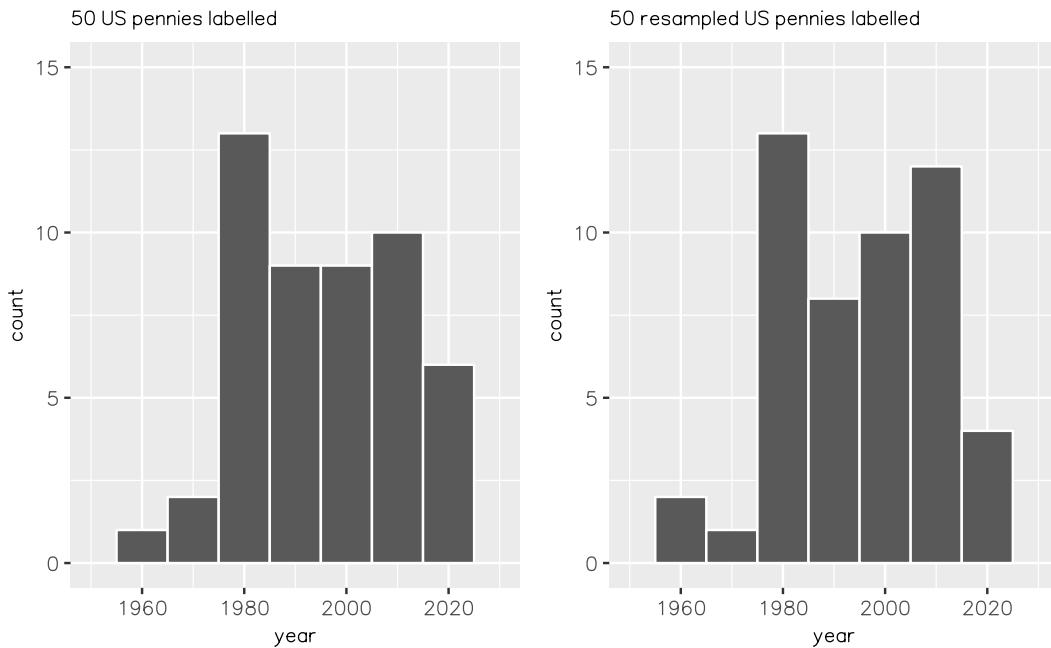


FIGURE 9.4: Comparing years of original sample `pennies_sample_2` and resample `pennies_resample`

Note the similarities and differences between these two distributions. The general shape of the two distributions is similar. This won't always be the case for all resamples since there is some chance that values that don't appear frequently in the original sample won't be drawn in the resample. Note also that the range of possible values taken on for the years is the same as what was seen in `pennies_sample_2`. In other words, we don't see any years such as 1948 or 1952 in the `pennies_resample` data. This will always be the case with resamples since they are, by their name, sampling again from the original sample.

So where does the mean of the `year` variable for this resample of `pennies_resample` fall? Any guesses? Let's have `dplyr` help us out here:

```
resample_mean <- pennies_resample %>%
  summarize(stat = mean(year))
```

The value of the resampled mean is 1994.82. We might have guessed that our mean here is similar to what we saw for the mean of `pennies_sample_2` of 1995.44 based on the distributions in the histogram in Figure 9.4.

What if we repeated several times this exercise of resampling 50 times from our sample of pennies? Would we obtain the same 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? This should remind you of what we did in Chapter 8. Let's do some more resamplings and observe the results with the help of 33 friends again.

9.1.3 Using resampling 33 times

Each of our 33 friends will do the following:

1. Each takes 50 sheets of paper that show the different `ID` and `year` pairings from the `pennies_sample_2` original sample.
2. Then, each puts the 50 small pieces of paper into something like a baseball cap.
3. Shake up the pieces of paper.
4. Draw “at random” from the cap to select one piece of paper.
5. Write down the value of `age` and `ID` on this piece of paper.
6. Next, place this piece of paper just drawn back into the cap to have 50 small pieces of paper again.
7. Draw “at random” again from the cap to select a piece of paper.
8. Repeat this process until each of your friends have drawn 50 pieces of paper and each has written down the `ID` and `year` values for these 50 pieces of paper.

Now each of your 33 friends has a resample of penny `ID` and `year` pairings. You now ask your friends to each compute the mean of the `year` values for each of their resamples. Using a similar strategy to what was done in Chapter 8, we build out our histogram manually:

Observe the following about the histogram in Figure 9.5:

- Only one of your friends has a mean year between 1991 and 1992.

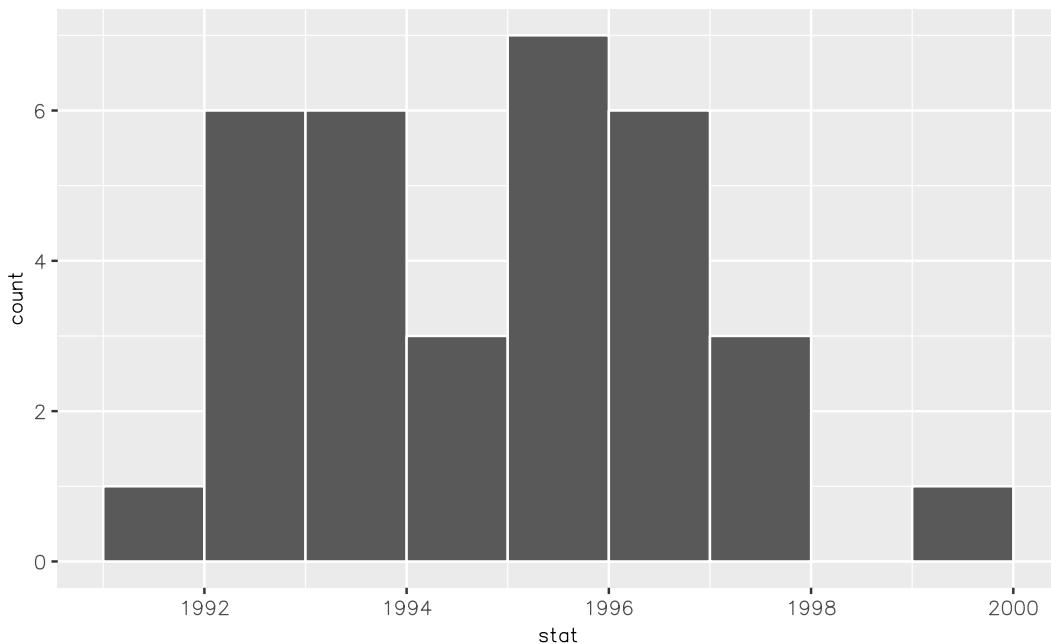


FIGURE 9.5: Constructing a histogram of means from resamples.

- On the other side of the distribution, only one of your friends has a mean year between 1999 and 2000.
- The most frequently occurring range is between 1995 and 1996 with 8 entries, but 1992 to 1993, 1993 to 1994, and 1996 to 1997 each have 6 entries.
- The distribution is starting to show a bit of normality, but that might be a stretch to guess at with this small of a sample size.

The different mean values reported by your 33 friends are stored in the `tactile_sample_means` tibble shown below.

```
tactile_resample_means
View(tactile_resample_means)
```

Let's display only the first 10 out of 33 rows of `tactile_resample_means`'s contents in Table 9.1.

TABLE 9.1: First 10 out of 33 friends' mean age of 50 resampled pennies.

friend	replicate	stat
1	1	1995
2	2	1997
3	3	1993
4	4	1994

TABLE 9.1: First 10 out of 33 friends' mean age of 50 resampled pennies.
(continued)

friend	replicate	stat
5	5	1993
6	6	1993
7	7	1997
8	8	1992
9	9	1997
10	10	1996

Recall the `replicate` column here similar to the one shown in Chapter 8 enumerating each of the 33 groups. This again corresponds to each row being viewed as one instance of a replicated activity. The activity here is the sampling with replacement from the original sample 50 times to create a resample.

The distribution of the 33 means of these resamples is given using `geom_histogram()` with `binwidth = 1` in Figure 9.6. As with the similar plot from Chapter 8 for the proportion of red balls, this computer-generated histogram matches our hand-drawn histogram from the earlier Figure 9.5.

```
ggplot(tactile_resample_means, mapping = aes(x = stat)) +
  geom_histogram(binwidth = 1, color = "white", boundary = 1990) +
  scale_x_continuous(breaks = seq(1990, 2000, 2))
```

9.1.4 What's the plan?

We've just gone over how to use the technique of *resampling* to make a guess as to what the variability would look like if we were to take samples from a population. Thus, in essence, the *resampling distribution* would be an approximation of the *sampling distribution* discussed in Chapter 8. We saw that we obtained many different means from these resamples and began to see the underpinnings of the normal distribution take shape in the first 33 resamples made by our friends.

In Section 9.2 we'll use *computer simulation* to imitate the hands-on resampling activity conducted here. We can use a computer to do the resampling many more times than just having our friends do this for us. This will allow us to better understand the distribution of means from many different resamples. Following these simulations, in Section 9.3 we'll explicitly articulate our goals for this chapter: understanding the concept of resampling variation, defining

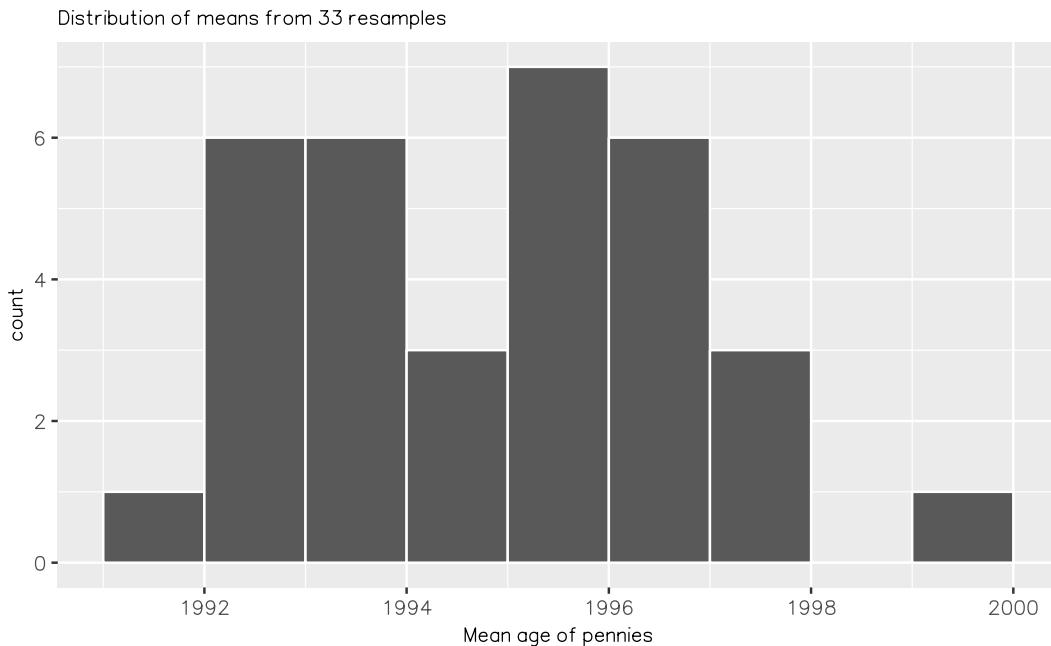


FIGURE 9.6: Distribution of 33 means based on 33 resamples of size 50

the statistical idea of a *confidence interval* by building on our pennies example, and discussing how confidence intervals can be interpreted.

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 these ideas together with a case study in Section 9.7. Here we will be analyzing an experiment done about yawning on the US television show Mythbusters. The chapter concludes with a comparison of a sampling distribution and a bootstrap distribution using the balls data from Chapter 8. We'll also discuss briefly the normal distribution and how these traditionally taught theoretical results tie in with the methods developed throughout the chapter.

9.2 Computer simulation of resampling

We've completed a tactile example of resampling. We'll next explore how this can be done virtually using the computer.

9.2.1 Using the virtual resample once

Recall that `pennies_sample_2` is stored as a dataset in the `moderndive` package. We've used the `rep_sample_n()` function in Chapter 8 to take samples from a population using `replace = FALSE` as the default. If we change this to `replace = TRUE`, we can also use this function to resample from a given original sample. Let's try this out:

```
virtual_resample <- pennies_sample_2 %>%
  rep_sample_n(size = 50, replace = TRUE, reps = 1)
```

Note here that the `size` argument should always be the same as the original sample size. We are doing this resampling one time and, thus, `reps = 1`. So what does `virtual_resample` look like?

```
View(virtual_resample)
```

We'll display only the first 10 out of 50 rows of `virtual_shovel`'s contents in Table 8.3.

TABLE 9.2: First 10 resampled rows of 50 in virtual sample

replicate	ID	year
1	48	1988
1	41	1992
1	49	2006
1	7	2008
1	32	1976
1	29	1988
1	18	1996
1	37	1962
1	44	2015
1	10	2000

The `ID` variable identifies which of the pennies from `pennies_sample_2` are in-

cluded in our resample of 50 coins and `year` denotes the year minted on the penny. The `replicate` column here is always the value of 1 corresponding to us only having `reps = 1`. It will take values between 1 and 33 next. But before we get to resampling multiple times, let's compute the mean `year` in our virtual resample of size 50 using `dplyr`.

```
virtual_resample %>%
  summarize(resample_mean = mean(year))

# A tibble: 1 x 2
  replicate resample_mean
  <int>        <dbl>
1       1        1995.48
```

Note that tibbles will try to print as pretty as possible which may result in numbers being rounded. In this chapter, we have set the default number of values to be printed to six in tibbles with `options(pillar.sigfig = 6)`.

9.2.2 Using the virtual resample 33 times

Let's now extend this to have 33 virtual friends help us understand the variability in the means from 33 resamples of size 50.

```
virtual_resamples <- pennies_sample_2 %>%
  rep_sample_n(size = 50, replace = TRUE, reps = 33)
View(virtual_resamples)
```

Just as we did with `virtual_resample`, we'll take the tibble `virtual_resamples` with $33 \times 50 = 1650$ rows corresponding to 33 resamples of size 50 pennies and then compute the resulting 33 means. We'll use the same `dplyr` verb as we did in the previous section, but compute the mean for each of our virtual friends:

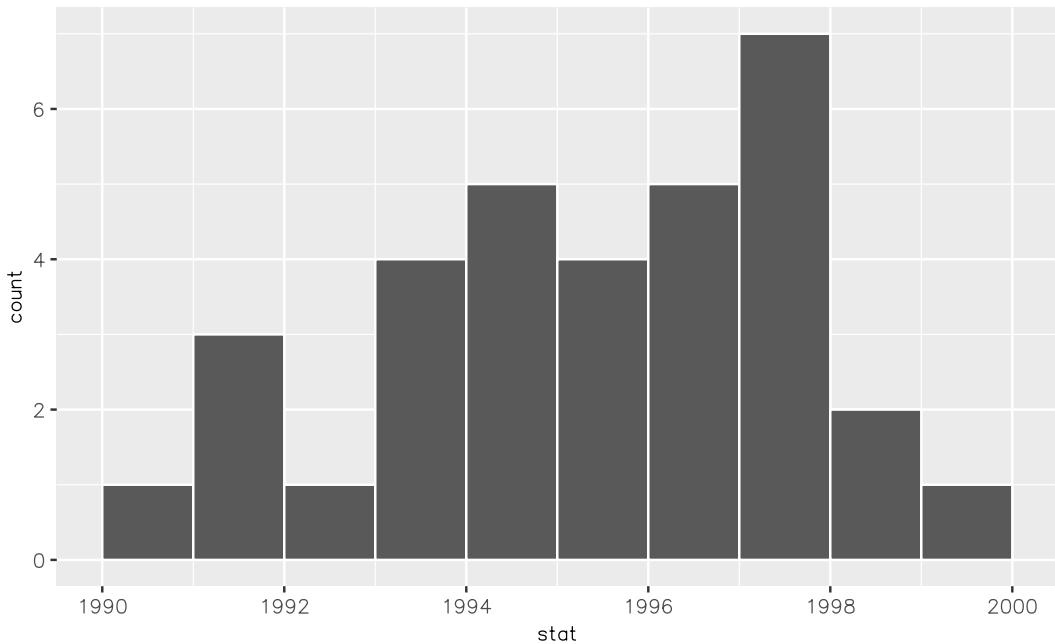
```
virtual_resample_means <- virtual_resamples %>%
  group_by(replicate) %>%
  summarize(stat = mean(year))
View(virtual_resample_means)
```

TABLE 9.3: First 10 out of 33 means from virtual resamples

replicate	stat
1	1995
2	1992
3	1994
4	1992
5	1997
6	1996
7	1999
8	1998
9	1996
10	1992

By looking at these first 10 rows, we can see values in the same range as those obtained from our friends' resamples in Table 9.1. Let's next visualize these 33 means:

```
ggplot(virtual_resample_means, aes(x = stat)) +
  geom_histogram(binwidth = 1, color = "white", boundary = 1990) +
  scale_x_continuous(breaks = seq(1990, 2000, 2))
```



Let's next compare the two distributions of the tactile resampling done by friends and the virtual resampling done by the computer:



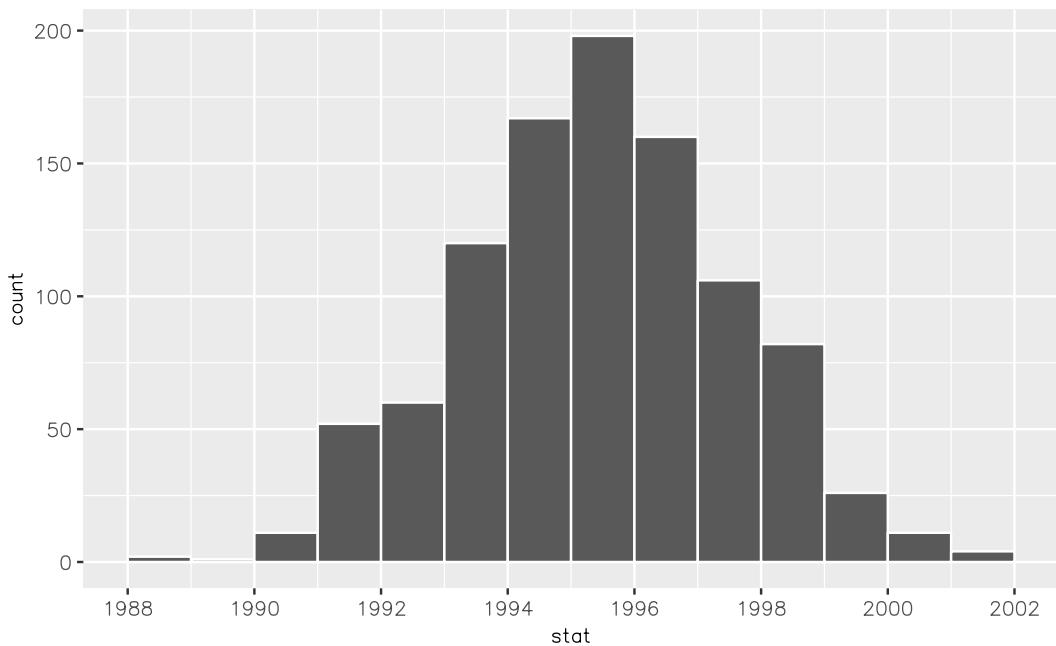
FIGURE 9.7: Comparing distributions of means from resamples⁴

9.2.3 Using the virtual resample 1000 times

Remember that one of the goal of resampling is to get an estimate for what the sampling distribution of the statistic of interest, the mean year here, looks like. To do so we'll need lots of different replicates to better understand the shape and the variability from one resample to the next. Let's extend our 33 replicates above to look at 1000 instead. With our goal being to get to 1000 resample means, we can use the `%>%` to get us there in one chain:

```
virtual_resample_means <- pennies_sample_2 %>%
  rep_sample_n(size = 50, replace = TRUE, reps = 1000) %>%
  group_by(replicate) %>%
  summarize(stat = mean(year))
```

Let's next look at what this distribution of 1000 means from 1000 resamples looks like:



Note here the bell shape starting to become more apparent. We now have a general sense for the range of values that the mean may take on in these resamples from this histogram. 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 look at the mean:

```
virtual_resample_means %>%
  summarize(mean_of_means = mean(stat))
```

The mean of the 1000 means from 1000 resamples is 1995.441. Note that this is quite close to the mean of our original sample: 1995.44. This will always be the case when we have 1000 or so replicates since each of the resamples is based on the original sample.

9.3 Confidence interval build-up

Definition: Confidence Interval

A *confidence interval* (CI) gives a range of plausible values for a parameter. It depends on a specified *confidence level* with higher confidence levels corresponding to wider confidence intervals and lower confidence levels corre-

sponding to narrower confidence intervals. Common confidence levels include 90%, 95%, and 99%.

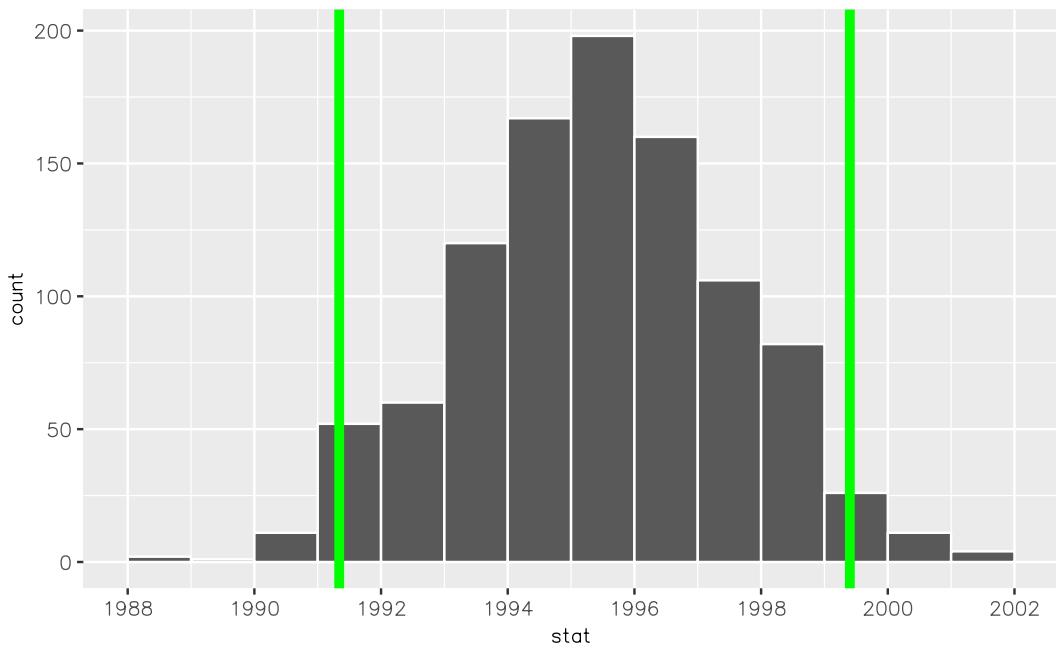
Usually, we don't just begin sections with a definition, but *confidence intervals* are simple to define and play an important role in the sciences and any field that uses data. We've also just shown the build-up to them visually in the previous section.

You can think of a confidence interval as playing the role of a net when fishing. Instead of just trying to catch a fish with a single spear (estimating an unknown parameter by using a single point estimate/statistic), we can use a net to try to provide a range of possible locations for the fish (use a range of possible values based around our statistic to make a plausible guess as to the location of the parameter).

The resampling process will provide statistics that have a distribution with center at (or extremely close to) the mean of the original sample. The distribution of statistics brought forth by resampling, also called the *resampling distribution* provides us a guess as to what the variability in different sample means may look like only using the original sample as our guide. We can quantify this variability in the form of a 95% confidence interval in a couple of different ways.

9.3.1 The percentile method

Recall that the actual mean year for all pennies in circulation in the US is unknown to us. But by finding a confidence interval we have a way to make an educated guess as to what a range of plausible values is for the unknown parameter. One way to calculate this range is to use the middle 95% of the `virtual_resample_means` to determine our endpoints. Our endpoints are thus at the 2.5th and 97.5th percentiles. We can visualize these percentiles on our distribution with green vertical lines:



Using the percentile method, our range of plausible values for the mean age of US pennies in circulation in 2011 is 1991.339 years to 1999.401 years. You'll see in later sections how to compute these values.

You can see that 95% of the data stored in the `stat` variable in `virtual_resample_means` falls between the two endpoints with 2.5% to the left outside of the shading and 2.5% to the right outside of the shading. The cut-off points that provide our range are shown with the green lines.

9.3.2 The standard error method

If the resampling distribution is close to symmetric and bell-shaped, we can also use a shortcut formula for determining the lower and upper endpoints of the confidence interval. This is done by using the formula $\bar{x} \pm (\text{multiplier} * SE)$, where \bar{x} is our original sample mean and SE stands for **standard error** and corresponds to the standard deviation of the resampling distribution. The value of *multiplier* here is the appropriate percentile of the standard normal distribution. We'll go into this further in Section 9.8.

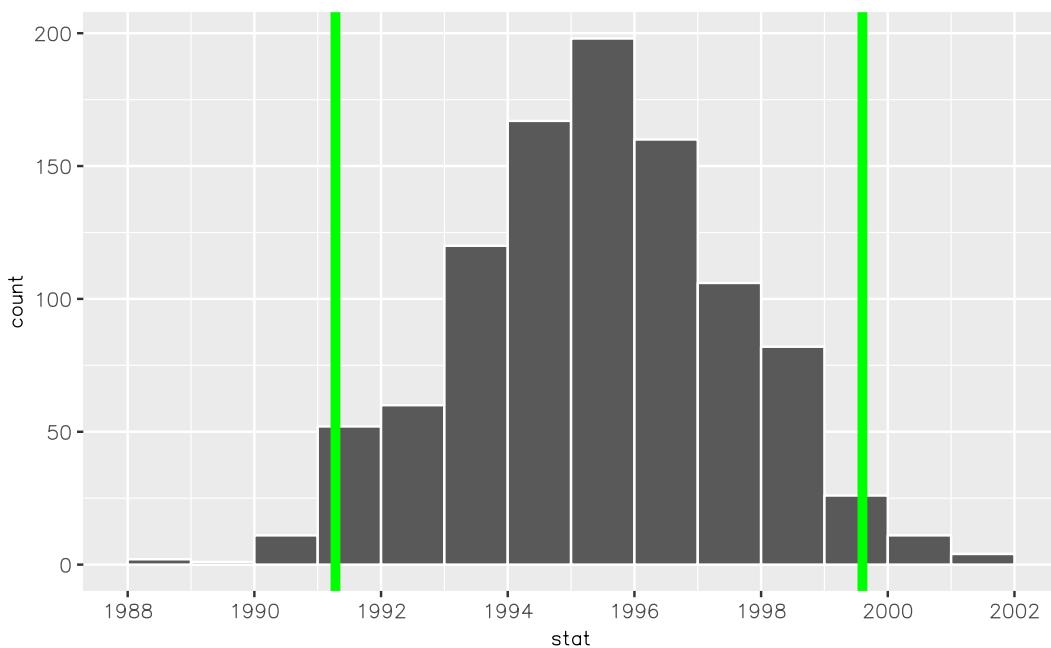
Definition: standard error

The *standard error* is the standard deviation of the sampling distribution.

The variability of the sampling distribution may be approximated by the variability of the resampling distribution. Traditional theory-based methodologies

for inference also have formulas for standard errors, assuming some conditions are met.

Let's next see how the lower and upper values of the confidence interval compare across the two methods of "percentile" and "standard error."



We see that both methods produce nearly identical confidence intervals with the percentile method being [1991.34, 1999.4] and the standard error method being [1991.28, 1999.6]. This is to be expected since the resampling distribution is roughly bell-shaped. Now that you've seen how these methods work, let's dig into the formalities of the process and the code needed to calculate them.

9.4 The bootstrapping framework

The way in which we used resamples to get a range of plausible for an unknown parameter is known as **bootstrapping**. To better understand this term, we harken back to the idea of pulling oneself up by their bootstraps. To "pull oneself up by their bootstraps" means to "succeed only by one's own efforts or abilities."² From a statistical perspective, we have pulled ourselves up from

²https://en.wiktionary.org/wiki/pull_oneself_up_by_one%27s_bootstraps

our bootstraps using a single sample (`pennies_sample_2`) to get an idea of the grander sampling distribution. Thus, we've used only the “effort” of the single original sample to approximate a larger goal of the variability in the sampling distribution.

Bootstrapping uses a process of sampling **with replacement** from our original sample to create new **bootstrap samples** of the *same* size as our original sample. We can again make use of the `rep_sample_n()` function to explore what one such bootstrap sample would look like. Remember that we are randomly sampling from the original sample here with replacement and that we always use the same sample size for the bootstrap samples as the size of the original sample (`pennies_sample_2`).

9.4.1 The original workflow needed for this

We saw earlier in Section 9.2 how to develop these bootstrap samples, bootstrap statistics, and the resulting *bootstrap distribution*. We called this *bootstrap distribution* the *resampling distribution* before to drill home the idea of this being based on resampling (sampling with replacement) instead of sampling without replacement as seen in Chapter 8. Let's revisit the flow using the `%>%` but this time call it `bootstrap_distribution`.

First, we repeatedly (`reps = 1000`) sampled with replacement (`replace = TRUE`) from the original sample with the same size of the resample (`size = 50`) using the `rep_sample_n()` function in the `infer` package:

```
pennies_sample_2 %>%
  rep_sample_n(size = 50, replace = TRUE, reps = 1000)
```

Next, since we are looking to get the mean value for year across each `replicate`, we used the `dplyr` verb `group_by()` to set up that grouping for our next step:

```
pennies_sample_2 %>%
  rep_sample_n(size = 50, replace = TRUE, reps = 1000) %>%
  group_by(replicate)
```

Lastly, we finish off the grouping by using `summarize()` from `dplyr` to get our mean value for year from each `replicate`. We also name this resulting 1000 row data frame `bootstrap_distribution`:

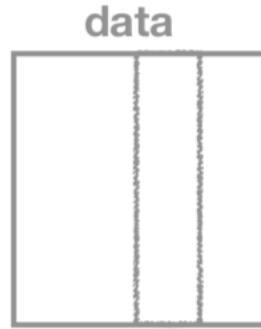
```
bootstrap_distribution <- pennies_sample_2 %>%
  rep_sample_n(size = 50, replace = TRUE, reps = 1000) %>%
  group_by(replicate) %>%
  summarize(stat = mean(year))
```

So for the case where we are bootstrapping a single variable like `year` that we have seen so far, we can get by with using the `rep_sample_n()` function and a couple `dplyr` verbs to get the bootstrap distribution. For more complicated inferential techniques we need a little more firepower though. Let's check out the `infer` package for tidy statistical inference!

9.4.2 The `infer` package for statistical inference

The `infer` package makes great use of the `%>%` to create a pipeline for statistical inference. The goal of the package is to provide a way for its users to explain the computational process of confidence intervals and hypothesis tests using the code as a guide. The verbs build in order here, so you'll want to start with `specify()` and then continue through the others as needed.

Specify variables



specify()

The `specify()` function is used primarily to choose which variables will be the focus of the statistical inference. In addition, a setting of which variable will act as the `explanatory` and which acts as the `response` variable is done here. For proportion problems similar to those in Chapter 8, we can also give which

of the different levels we would like to have as a `success`. We'll see further examples of these options in this chapter, Chapter 10, and in Appendix B.

To begin to create a confidence interval for the population mean year of US pennies in 2019, we start by using `specify()` to choose which variable in our `pennies_sample_2` data we'd like to work with. This can be done in one of two ways:

1. Using the `response` argument:

```
pennies_sample_2 %>%
  specify(response = year)
```

```
Response: year (integer)
# A tibble: 50 x 1
  year
  <int>
 1 2002
 2 1986
 3 2017
 4 1988
 5 2008
 6 1983
 7 2008
 8 1996
 9 2004
10 2000
# ... with 40 more rows
```

2. Using `formula` notation:

```
pennies_sample_2 %>%
  specify(formula = year ~ NULL)
```

```
Response: year (integer)
# A tibble: 50 x 1
  year
  <int>
 1 2002
 2 1986
 3 2017
```

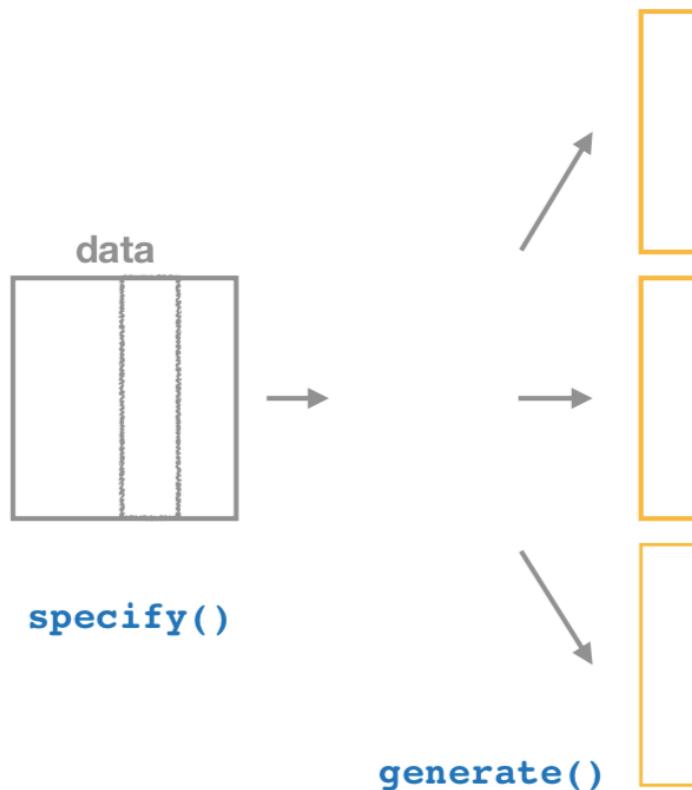
```

4 1988
5 2008
6 1983
7 2008
8 1996
9 2004
10 2000
# ... with 40 more rows

```

Note that the formula notation uses the common R methodology to include the response y variable on the left of the \sim and the explanatory x variable on the right of the “tilde.” Recall that you used this notation frequently with the `lm()` function in Chapters 6 and 7 when fitting regression models. Either notation works just fine, but preference is usually given here for the `formula` notation to further build on the ideas from earlier chapters.

Generate replicates



After `specify()`ing the variables we'd like in our inferential analysis, we next feed that into the `generate()` verb. The `generate()` verb's main argument is

`reps`, which is used to give how many different repetitions one would like to perform. Another argument here is `type`, which is automatically determined by the kinds of variables passed into `specify()`. We can also be explicit and set this `type` to be `type = "bootstrap"`. If you are not explicit, `infer` will send you a message just to make sure this is what you are wanting. This `type` argument will be further used in hypothesis testing in Chapter 10 as well. Make sure to check out `?generate` to see the options here and use the `?operator` to better understand other verbs as well.

Let's `generate()` 1000 bootstrap samples:

```
thousand_bootstrap_samples <- pennies_sample_2 %>%
  specify(response = year) %>%
  generate(reps = 1000, type = "bootstrap")
```

We can use the `dplyr count()` function to help us understand what the `thousand_bootstrap_samples` data frame looks like:

```
thousand_bootstrap_samples %>%
  count(replicate)

# A tibble: 1,000 x 2
# Groups:   replicate [1,000]
  replicate     n
  <int> <int>
1       1     50
2       2     50
3       3     50
4       4     50
5       5     50
6       6     50
7       7     50
8       8     50
9       9     50
10      10    50
# ... with 990 more rows
```

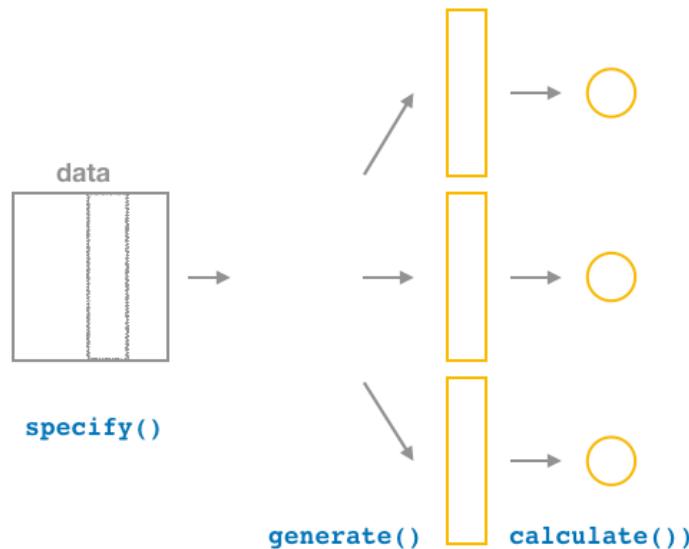
Notice that each `replicate` has 50 entries here. Now that we have 1000 different bootstrap samples, our next step is to `calculate` the bootstrap statistics for each sample.

Comparing back to original workflow

Note that the steps up to this point of the `infer` pipeline produce the same procedure as what we saw before with `rep_sample_n()`. In other words, the following two code chunks produce similar results:

<pre># With infer pipeline pennies_sample_2 %>% specify(response = year) %>% generate(reps = 1000)</pre>	<pre># Without infer pipeline pennies_sample_2 %>% rep_sample_n(size = 50, replace = TRUE, reps = 1000)</pre>
--	--

Calculate summary statistics



After `generate()`ing many different samples, we next want to condense those samples down into a single statistic for each replicated sample. As seen in the diagram, the `calculate()` function is helpful here.

As we did at the beginning of this chapter, we now want to calculate the mean `year` for each bootstrap sample. To do so, we use the `stat` argument and set it to "mean" below. The `stat` argument has a variety of different options here and we will see further examples of this throughout the remaining chapters.

```
bootstrap_distribution <- pennies_sample_2 %>%
  specify(response = year) %>%
  generate(reps = 1000) %>%
  calculate(stat = "mean")
```

Setting `type = "bootstrap"`` in `generate()`.

bootstrap_distribution

```
# A tibble: 1,000 x 2
  replicate    stat
  <int>    <dbl>
1       1 1997.42
2       2 1996.1
3       3 1995.64
4       4 1994.78
5       5 1994.46
6       6 1995.42
7       7 1995.98
8       8 1992.18
9       9 1993.06
10      10 1992.22
# ... with 990 more rows
```

We see that the resulting data has 1000 rows and 2 columns corresponding to the 1000 replicates and the mean for each bootstrap sample.

Comparing back to original workflow

We can see that the `calculate()` step does what the `group_by() %>% summarize()` steps do in the original workflow:

<pre># With infer pipeline pennies_sample_2 %>% specify(response = year) %>% generate(reps = 1000) %>% calculate(stat = "mean")</pre>	<pre># Without infer pipeline pennies_sample_2 %>% rep_sample_n(size = 50, replace = TRUE, reps = 1000) %>% group_by(replicate) %>% summarize(stat = mean(year))</pre>
--	--

Observed statistic / point estimate calculations

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_2 %>%
  summarize(stat = mean(year))
```

```
# A tibble: 1 × 1
  stat
  <dbl>
1 1995.44
```

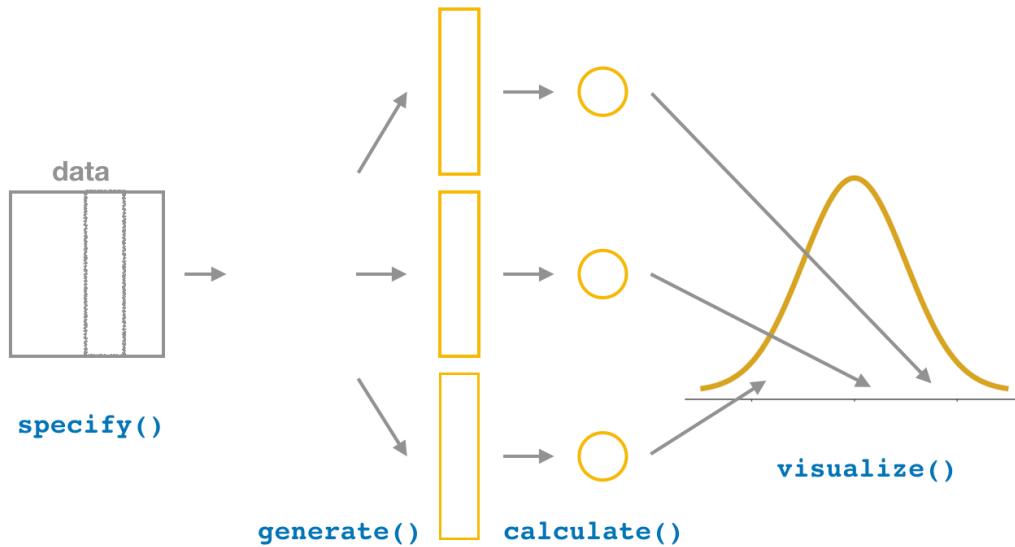
This can also be done by skipping the `generate()` step in the pipeline feeding `specify()` directly into `calculate()`:

```
pennies_sample_2 %>%
  specify(response = year) %>%
  calculate(stat = "mean")
```

```
# A tibble: 1 × 1
  stat
  <dbl>
1 1995.44
```

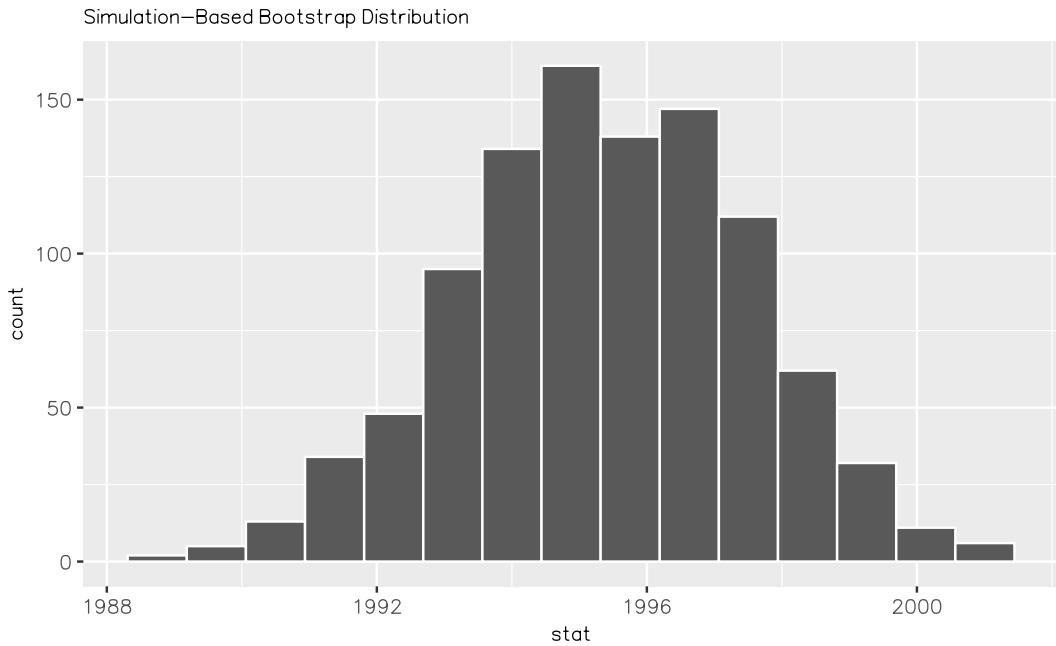
This shortcut will be particularly useful when the calculation of the observed statistic is tricky to do using `dplyr` alone. This is particularly the case when working with more than one variable as will be seen in Chapter 10.

Visualize the results



The `visualize()` verb provides a simple way to view the bootstrap distribution as a histogram of the `stat` variable values. It has many other arguments that one can use as well including the shading of the histogram values corresponding to the confidence interval values.

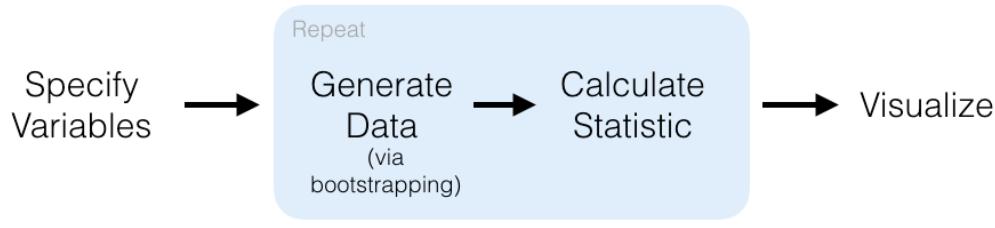
```
bootstrap_distribution %>% visualize()  
# or  
visualize(bootstrap_distribution)
```



The shape of this resulting distribution may look familiar to you. It resembles the well-known normal (bell-shaped) curve.

The following diagram recaps the `infer` pipeline for creating a bootstrap distribution.

Confidence Interval in `infer`



```
specify(response) %>% generate(reps) %>% calculate(stat) %>% visualize()
```

9.4.3 Building confidence intervals with the `infer` package

Recall how we showed two different methods for building a range of plausible values for an unknown parameter in Section 9. Let's now check out how the `infer` package and some new functions were used to get us there. There's also some additional functionality to further assist with visualizing the intervals built-in!

9.4.4 The percentile method with `infer`

Recall the percentile method of looking at the middle 95% of values with the lower endpoint at the 2.5th percentile and the upper endpoint at the 97.5th percentile. This can be done with `infer` using the `get_confidence_interval()` function. You can also use the alias `get_ci()` if you'd like the short version. That's what we use here.

```
bootstrap_distribution %>%
  get_ci(level = 0.95, type = "percentile")
```

```
# A tibble: 1 x 2
`2.5%` `97.5%
<dbl>   <dbl>
1 1991.16 1999.46
```

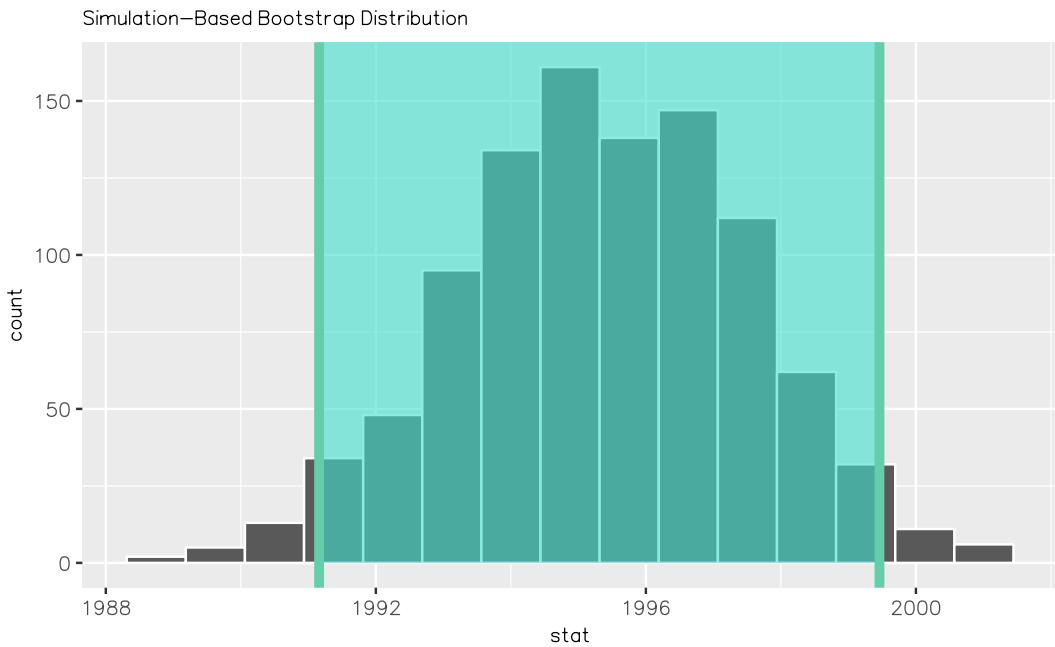
These options are the default values for `level` and `type` so we can also just do:

```
percentile_ci <- bootstrap_distribution %>%
  get_ci()
percentile_ci
```

```
# A tibble: 1 x 2
`2.5%` `97.5%
<dbl>   <dbl>
1 1991.16 1999.46
```

Now we see where the values obtained in Section 9 come from. Using the percentile method, our range of plausible values for the mean year of US pennies in circulation in 2019 is 1991.159 to 1999.46. We can further use the `visualize()` function and the `shade_confidence_interval()` function, or alias `shade_ci()`, to view this. We use the `endpoints` argument to be those stored with name `percentile_ci`.

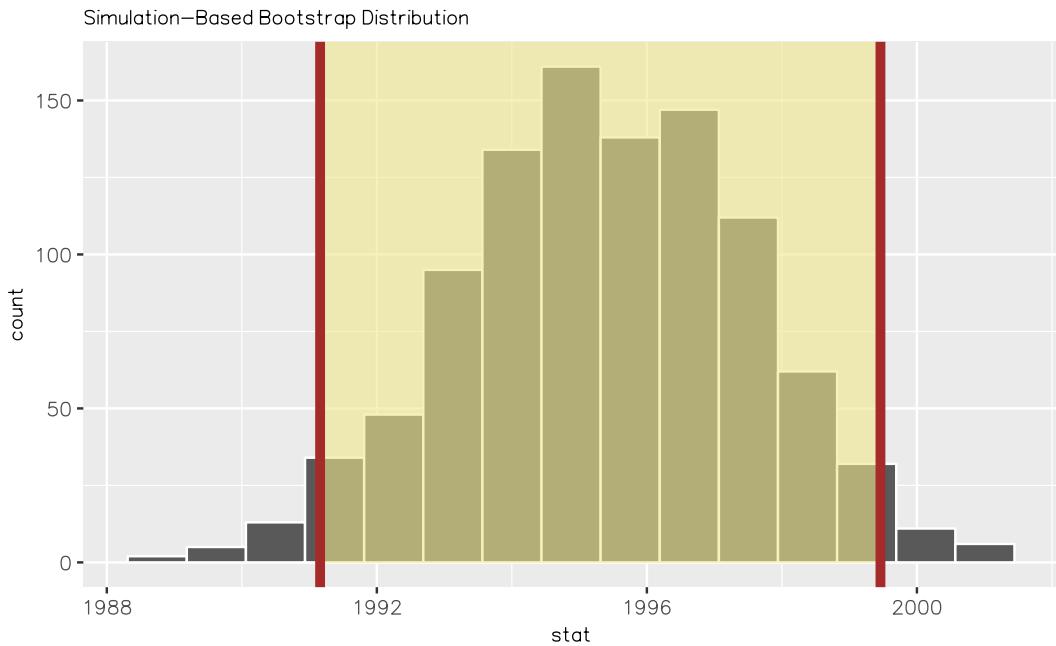
```
visualize(bootstrap_distribution) +
  shade_ci(endpoints = percentile_ci)
```



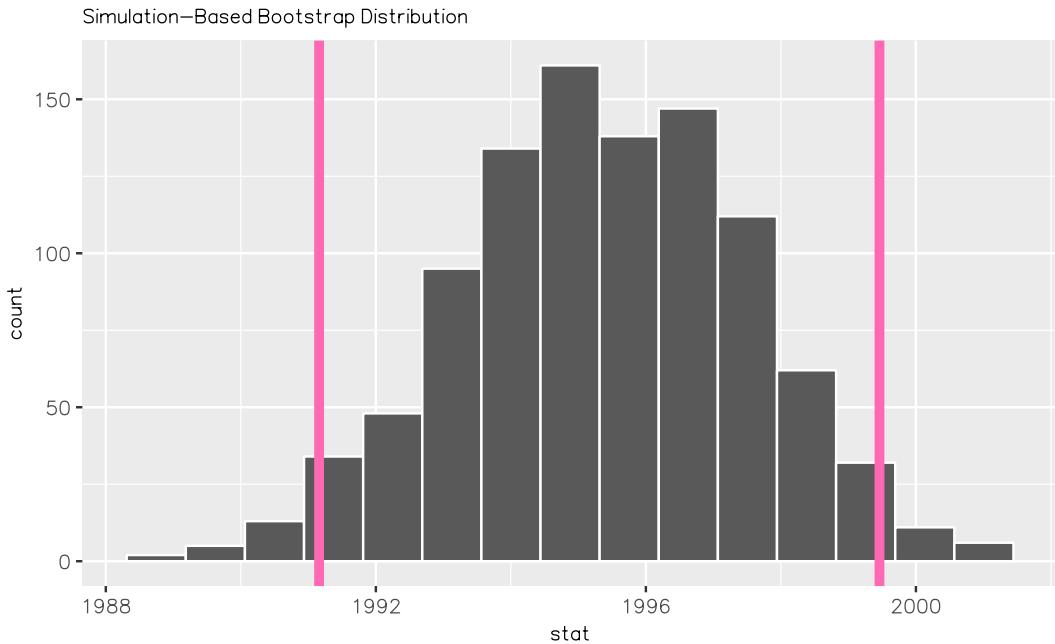
You can see that 95% of the data stored in the `stat` variable in `bootstrap_distribution` falls between the two endpoints with 2.5% to the left outside of the shading and 2.5% to the right outside of the shading. The cut-off points that provide our range are shown with the darker lines.

Note that you can change the colors here as you wish using the `color` and `fill` arguments.

```
visualize(bootstrap_distribution) +  
  shade_ci(endpoints = percentile_ci,  
            color = "brown",  
            fill = "khaki")
```



```
visualize(bootstrap_distribution) +  
  shade_ci(endpoints = percentile_ci,  
            color = "hotpink",  
            fill = NULL)
```



9.4.5 The standard error method with `infer`

Recall the formula $\bar{x} \pm (\text{multiplier} * SE)$, where \bar{x} is our original sample mean and SE stands for **standard error** and corresponds to the standard deviation of the bootstrap distribution. This is the formula for using the standard error method for calculating a confidence interval.

The *multiplier* is automatically calculated when `level` is provided with `level = 0.95` being the default. (95% of the values in a standard normal distribution fall within 1.96 standard deviations of the mean, so *multiplier* = 1.96 for `level = 0.95`, for example.) As mentioned, this formula assumes that the bootstrap distribution is symmetric and bell-shaped. This is often the case with bootstrap distributions, especially those in which the original distribution of the sample is not highly skewed.

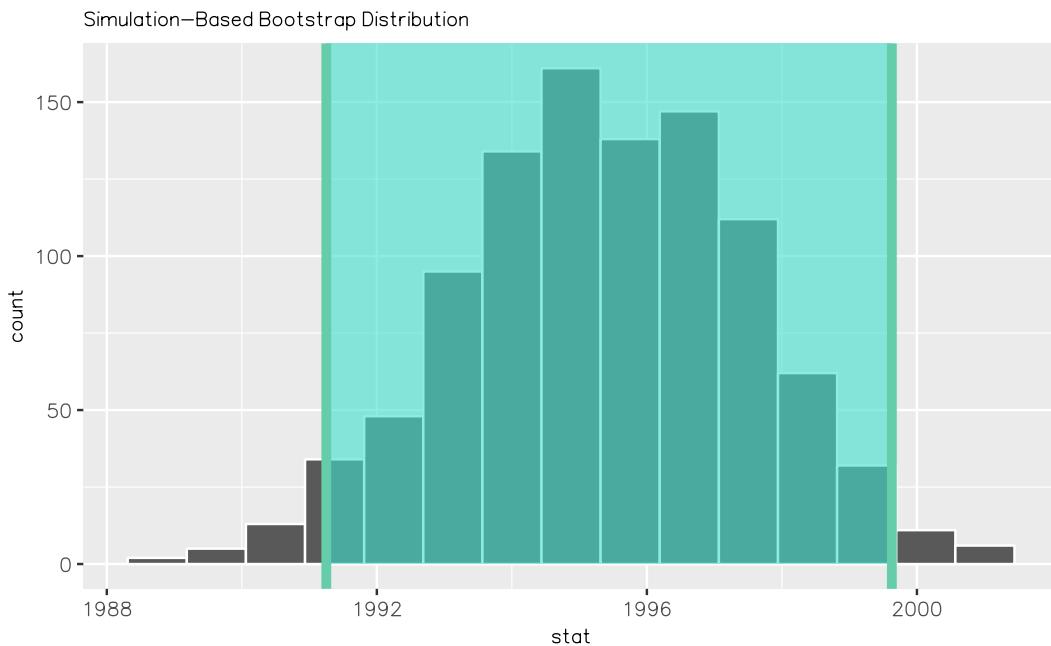
This $\bar{x} \pm (\text{multiplier} * SE)$ formula is implemented in the `get_ci()` function as shown with our pennies problem using the bootstrap distribution's variability as an approximation for the sampling distribution's variability. We'll see more on this approximation shortly.

Note that the center of the confidence interval (the `point_estimate`) must be provided for the standard error confidence interval.

```
standard_error_ci <- bootstrap_distribution %>%
  get_ci(type = "se", point_estimate = x_bar)
standard_error_ci
```

```
# A tibble: 1 x 2
  lower    upper
  <dbl>    <dbl>
1 1991.25 1999.63
```

```
visualize(bootstrap_distribution) +
  shade_ci(endpoints = standard_error_ci)
```



As noted in Section 9 both methods produce similar confidence intervals.

Percentile	Standard error
[1991.16, 1999.46]	[1991.25, 1999.63]

The `[lower, upper]` notation here corresponds to the `lower` value being the smallest included entry in the confidence interval and `upper` being the largest included entry in the confidence interval.

9.5 Case study: Revisiting the red ball example

Let's revisit our exercise of trying to estimate the proportion of red balls in the bowl from Chapter 8. We are now interested in determining a confidence interval for the population parameter p , the proportion of balls that are red out of the total $N = 2400$ red and white balls.

We will use the first sample reported from Ilyas and Yohan in Subsection 8.1.3 for our point estimate. They observed 21 red balls out of the 50 in their shovel. This data is stored in the `tactile_shovel_1` data frame in the `moderndive` package.

```
tactile_shovel_1
```

```
# A tibble: 50 x 1
  color
  <chr>
  1 red
  2 red
  3 red
  4 white
  5 white
  6 white
  7 white
  8 white
  9 red
 10 white
# ... with 40 more rows
```

9.5.1 Observed statistic

To compute the proportion that are red in this data we can use the `specify() %>% calculate()` workflow. Note the use of the `success` argument here to clarify which of the two colors "red" or "white" we are interested in.

```
p_hat <- tactile_shovel_1 %>%
  specify(formula = color ~ NULL, success = "red") %>%
  calculate(stat = "prop")
p_hat
```

```
# A tibble: 1 × 1
  stat
  <dbl>
1 0.42
```

9.5.2 Bootstrap distribution for one proportion

Next, we want to calculate many different bootstrap samples and their corresponding bootstrap statistic (the proportion of red balls). We've done 1000 in the past, but let's go up to 10,000 now to better see the resulting distribution. Recall that this is done by including a `generate()` function call in the middle of our pipeline:

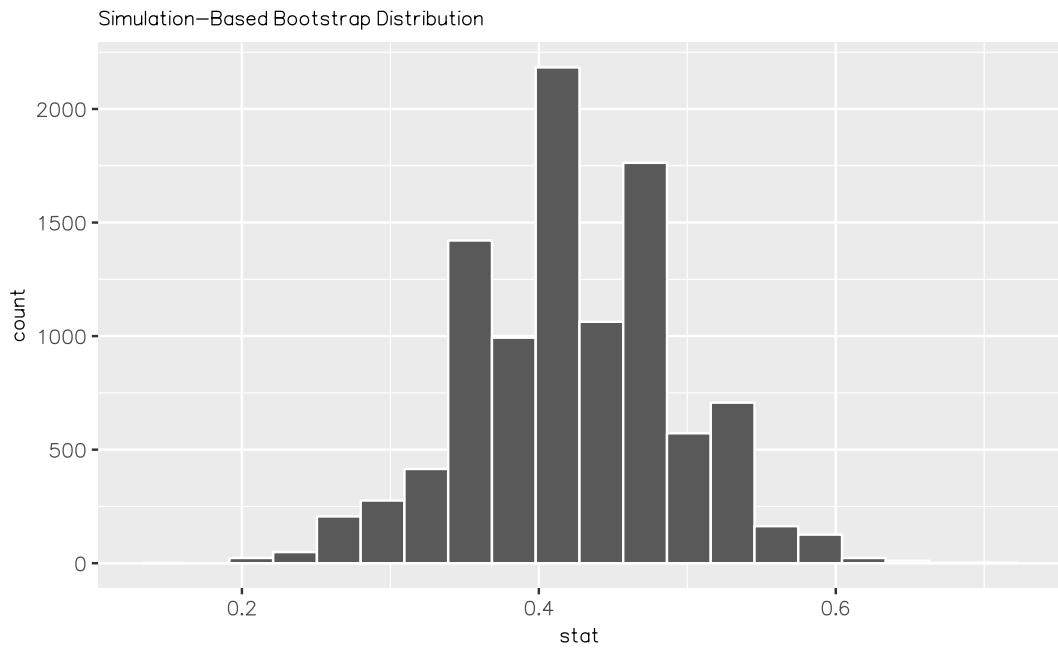
```
tactile_shovel_1 %>%
  specify(formula = color ~ NULL, success = "red") %>%
  generate(reps = 10000, type = "bootstrap")
```

This results in 50 rows for each of the 10,000 replicates. Lastly, we finish the `infer` pipeline by adding back in the `calculate()` step.

```
bootstrap_props <- tactile_shovel_1 %>%
  specify(formula = color ~ NULL, success = "red") %>%
  generate(reps = 10000, type = "bootstrap") %>%
  calculate(stat = "prop")
```

Let's `visualize()` what the resulting bootstrap distribution looks like as a histogram. We've adjusted the number of bins here as well to better see the resulting shape.

```
visualize(bootstrap_props, bins = 20)
```

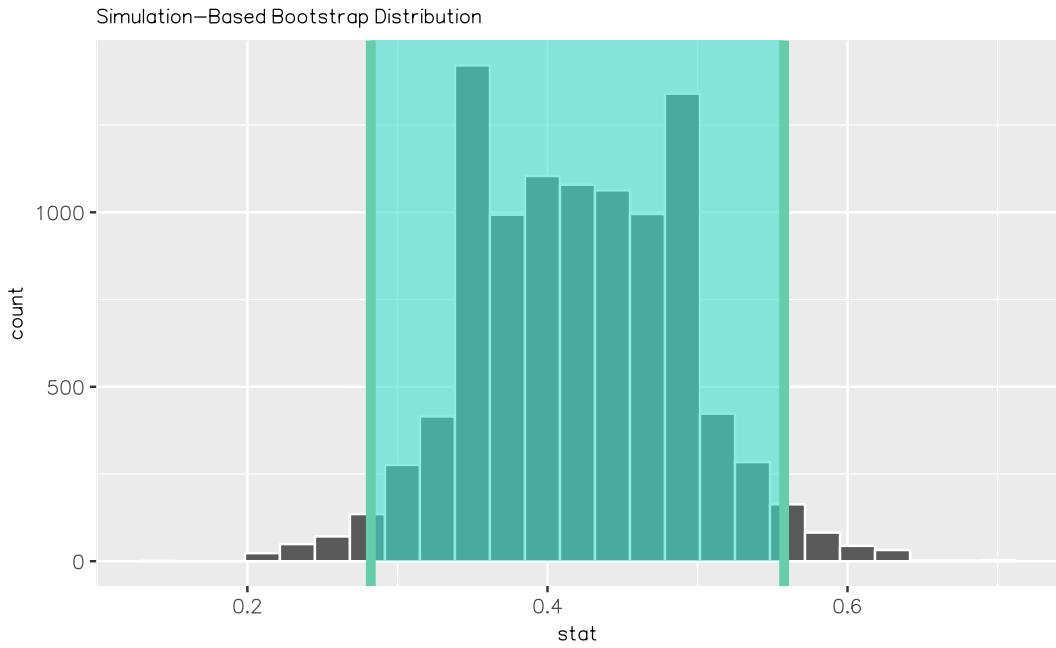


We see that the resulting distribution is symmetric and bell-shaped so it doesn't much matter which confidence interval method we choose. Let's use the standard error method to create a 95% confidence interval.

```
standard_error_ci <- bootstrap_props %>%
  get_ci(type = "se", level = 0.95, point_estimate = p_hat)
standard_error_ci
```

```
# A tibble: 1 x 2
  lower     upper
  <dbl>     <dbl>
1 0.282244 0.557756
```

```
visualize(bootstrap_props, bins = 25) +
  shade_ci(endpoints = standard_error_ci)
```



We are “95% confident” that the true proportion of red balls in the bowl is between 0.282 and 0.558.

9.6 Interpreting the confidence interval

One key to working with confidence intervals is to also understand how best to interpret them. From the previous example, this level of confidence (95%) is based on the standard error-based method including the true proportion 95% of the time if many different samples (not just the one we used) were collected and confidence intervals were created following this standard error-based method. Let’s dig into what this means further by exploring the confidence intervals based on other samples to see how they compare to the one we just calculated. By the end of this section, you should have an understanding as to what “95% confident” means and how best to use that knowledge when you see that language used in other contexts.

As shown above in Subsection 9.5.2, one range of plausible values for the population proportion of red balls (the true proportion of all red balls in the entire bowl), denoted by p , is $[0.28, 0.56]$. Recall that this confidence interval is based on bootstrapping using `tactile_shovel_1`.

To best understand how to interpret a confidence interval, it is important

to see how the process works when we have a known population parameter. 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 . Remember this isn't usually the case of knowing the population parameter, but we'll see why this is useful for our build-up shortly. Let's do this two ways to review both the `infer` and `dplyr` pipelines:

```
bowl %>%
  specify(formula = color ~ NULL, success = "red") %>%
  calculate(stat = "prop")
```

```
# A tibble: 1 × 1
  stat
  <dbl>
1 0.375
```

```
bowl %>%
  summarize(stat = mean(color == "red"))
```

```
# A tibble: 1 × 1
  stat
  <dbl>
1 0.375
```

Both methods return 0.375 as the proportion of red balls in the population of all balls in the bowl. So did our “95% confident” guess above of [0.28, 0.56] contain the “true value” for the population?

Yes, the population proportion (0.375) does fall in this confidence interval. 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 parameter value as well? Let's try it out by pulling another sample from `bowl` of size 50:

```
bowl_sample_2 <- bowl %>%
  sample_n(size = 50)
```

Note the use of the `sample_n()` function in the `dplyr` package here. This does the same thing as `rep_sample_n(reps = 1)` but omits the extra `replicate` column.

We next create an `infer` pipeline to generate a standard error-based 95% confidence interval for p . Recall that we first need a `point_estimate` to act as the

center of our standard error-based confidence interval. We calculate this with name `prop_red_2`.

```
prop_red_2 <- bowl_sample_2 %>%
  specify(formula = color ~ NULL, success = "red") %>%
  calculate(stat = "prop")
standard_error_ci_2 <- bowl_sample_2 %>%
  specify(formula = color ~ NULL, success = "red") %>%
  generate(reps = 1000, type = "bootstrap") %>%
  calculate(stat = "prop") %>%
  get_ci(type = "se", point_estimate = prop_red_2)
standard_error_ci_2
```

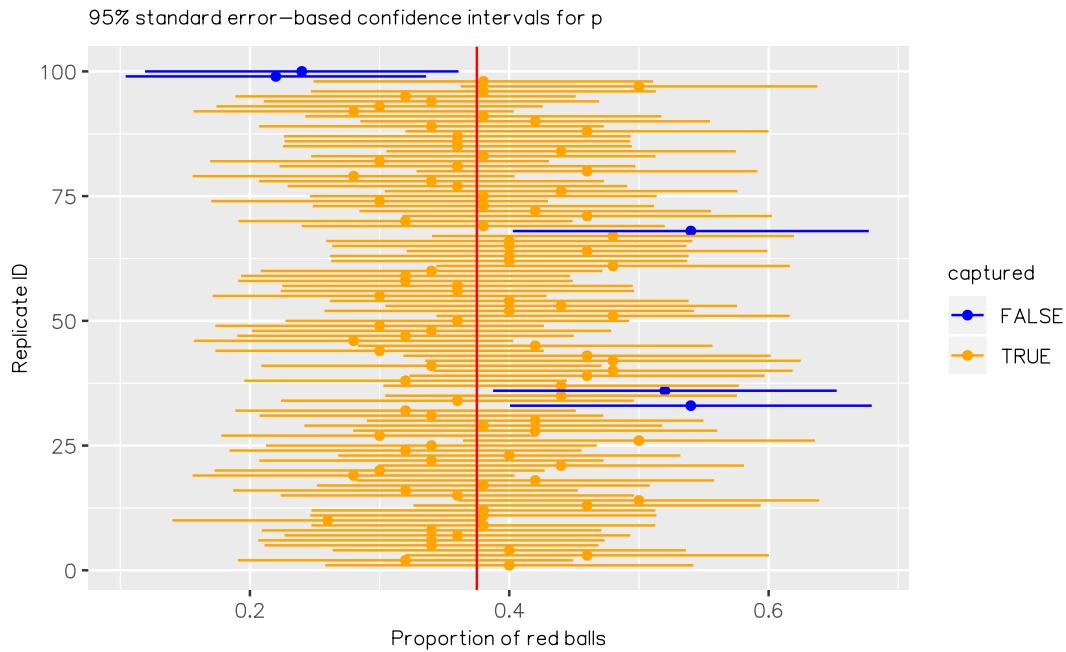
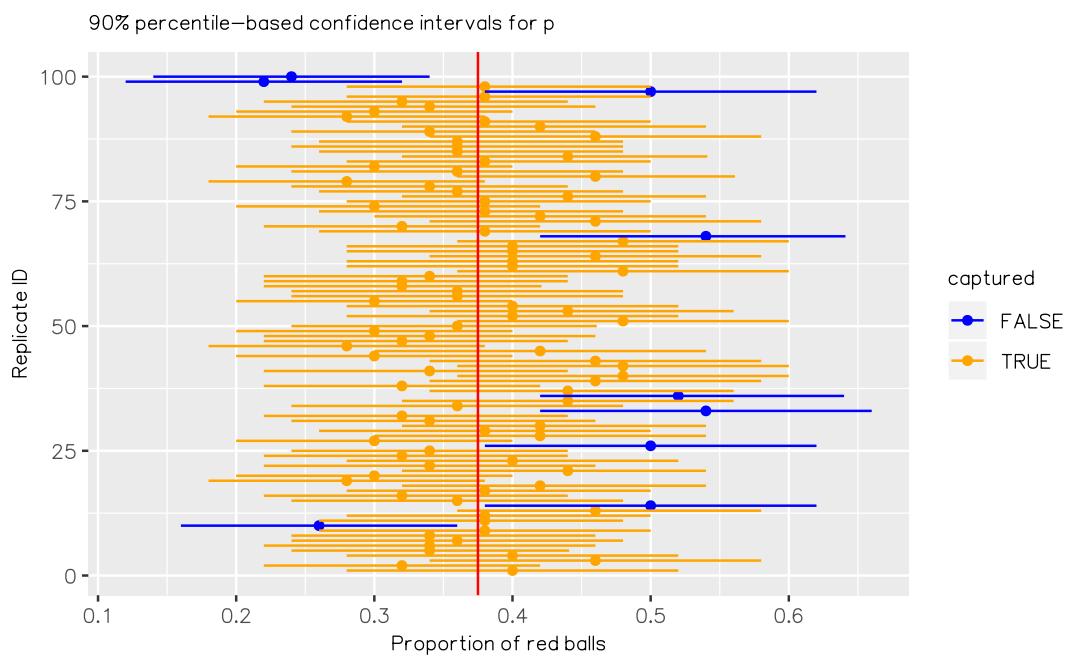
```
# A tibble: 1 x 2
  lower    upper
  <dbl>    <dbl>
1 0.233227 0.486773
```

This new confidence interval also contains the value of p . Let's further investigate by repeating this process 100 times to get 100 different confidence intervals derived from 100 different samples of the population `bowl`. Each sample will have a size of 50 just as the original sample. We will plot each of these confidence intervals as horizontal lines. At the center of each confidence interval is the point estimate denoted by a dot. We will also show a red line corresponding to the known population value of 0.375 red balls.

Of the 100 confidence intervals based on samples of size $n = 50$, 95 of them captured the population mean $p = 0.375$, whereas 5 of them did not include it. If we repeated this process of building confidence intervals more times with more samples, we'd expect 95% of them to contain the population parameter p . In other words, the procedure we have used to generate confidence intervals is “95% reliable” in that we can expect it to include the true population parameter 95% of the time if the process is repeated.

To further accentuate this point, let's perform a similar procedure using 90% confidence intervals instead. This time we will use the percentile method instead of the standard error method for computing the confidence intervals.

Of the 100 confidence intervals based on samples of size $n = 50$, 91 of them captured the population proportion $p = 0.375$, whereas 9 of them did not include it. Repeating this process for more samples would result in us getting closer and closer to 90% of the confidence intervals including the true value. It is common to say while interpreting a confidence interval to be “95% confident” or “90% confident” that the true value falls within the range of the specified

**FIGURE 9.8:** Reliability of 95 percent confidence intervals**FIGURE 9.9:** Reliability of 90 percent confidence intervals

confidence interval. We will use this “confident” language throughout the rest of this chapter, but remember that it has more to do with a measure of the reliability of the building process.

Back to our pennies example

After this elaboration on what the level corresponds to in a confidence interval, let’s conclude by providing an interpretation of the original confidence interval result we found in Subsection 9.4.3.

Interpretation: We are 95% confident that the true mean year of pennies in circulation in 2019 is between 1991.159 to 1999.46. This level of confidence is based on the percentile-based method including the true mean 95% of the time if many different samples (not just the one we used) were collected and confidence intervals were created.

The width of confidence intervals

The impact of confidence levels

When looking at the relative sizes of the orange horizontal lines in Figure 9.8 with 95% confidence intervals and Figure 9.9 with 90% confidence intervals, does anything stand out in terms of the width of the intervals to you? The statement of confidence in terms of the level should match with what you expect of the word “confident.” If someone says they are 99% confident about the high temperature between one value and another for the next day, we’d expect that range to be higher than if they said they were only 80% confident, right?

To elaborate on this a bit, if we wanted to make a guess as to what the forecasted summertime high temperature in Brussels, Belgium would be for a day in July, we could say pretty confidently that the high temperature wouldn’t be below 55° F (approximately 13° C) and that it wouldn’t be above 72° F (approximately 22° C). Let’s say we are 90% confident for a given day about this claim. What would we need to do to this range to increase our level of confidence?

We’d need to increase it! To be more confident about the range of plausible values for our high temperature, we need to add in more possible temperatures since we might have a cold streak or an unseasonably warm day.

What if we wanted to be a little less confident and say have a 50-50 chance of guessing at what the high temperature would be. Well, if we are OK being wrong 50% of the time, we could guess something in a much narrower range

for plausible values of the high temperature. Something like [61° F, 66° F] (approximately [16° C, 19° C]) might be a 50% confident guess, say. By narrowing our range, we've decreased our level of confidence. This analogy relates well (maybe not exactly) to confidence intervals in statistics.

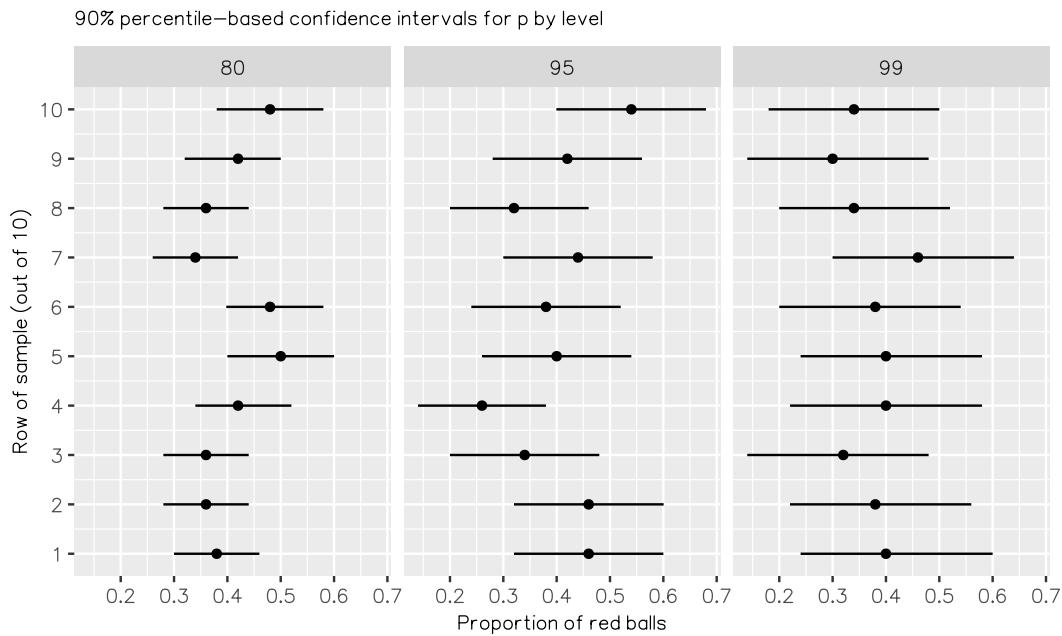
Higher confidence levels tend to produce wider confidence intervals.

Let's play with do a little more analysis using the `bowl` data to construct 80%, 95%, and 99% confidence intervals here to drill this idea home. We'll focus on the percentile-based method though a similar analysis could be done for the standard error-based method. Let's calculate 100 confidence intervals of each of these three different levels and then look at the median and mean length of these intervals. These will be stored in the `perc_cis_by_level` data frame.

Let's take a look into what the `perc_cis_by_level` data frame looks like and how a sample of 10 different confidence intervals each from the 80%, 95%, and 99% levels compare visually in terms of length. Then, we'll start computing some widths of the confidence intervals. Then we'll head into calculating the mean and median widths across the three different levels.

TABLE 9.5: 10 randomly sampled confidence intervals for p for varying confidence levels

replicate	point_estimate	lower	upper	confidence_level
93	0.40	0.26	0.540	95
85	0.40	0.22	0.580	99
63	0.40	0.32	0.482	80
32	0.30	0.22	0.380	80
70	0.46	0.30	0.640	99
8	0.34	0.22	0.480	95
13	0.36	0.28	0.440	80
72	0.34	0.26	0.420	80
77	0.42	0.28	0.560	95
59	0.40	0.32	0.500	80



We see that the sample proportion of reds varies in the `point_estimate` column with varying `lower` and `upper` bounds as well depending on the variability of the bootstrap distribution. The width of the confidence intervals appears to increase from left to right going from 80% confidence levels to 95% and then to 99%. Let's now compute the confidence interval (CI) width for each of these intervals and then get the median and mean length.

```
percentile_cis_by_level %>%
  mutate(width = upper - lower) %>%
  group_by(confidence_level) %>%
  summarize(median_width = median(width),
            mean_width = mean(width))
```

	confidence_level	median_width	mean_width
	<dbl>	<dbl>	<dbl>
1	80	0.16	0.16906
2	95	0.280000	0.266255
3	99	0.340100	0.341841

As expected, as the confidence level increases, the width of the corresponding confidence interval also increases. To be more confident, we need to increase the range of plausible values.

The impact of sample size

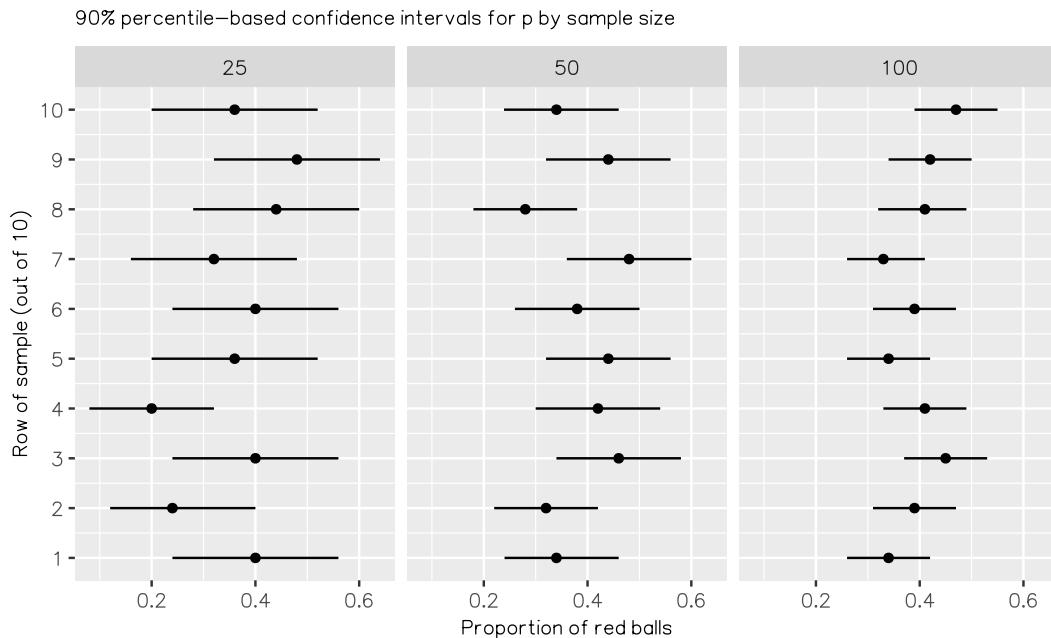
We can also observe the impact of sample size on these calculations and make some generalizations. You'll see in Subsection 9.8.2 some reasons via mathematical formulas for the behavior of confidence interval width and changing sample sizes too.

Let's hold the confidence level fixed at 90% using the percentile-based method, but take samples of size 25, 50, and 100 corresponding to the different sizes of the shovels available to us in Chapter 8. Do you expect smaller sample sizes to produce wider confidence intervals? Or should larger sample sizes produce wider ones? Let's investigate.

Recall the `virtual_samples_25`, `virtual_samples_50`, and `virtual_samples_100` data frames that were calculated in Chapter 8. As a reminder, here's the code needed to compute them.

```
virtual_samples_25 <- bowl %>%
  rep_sample_n(size = 25, reps = 1000, replace = FALSE)
balls_samples_50 <- bowl %>%
  rep_sample_n(size = 50, reps = 1000, replace = FALSE)
balls_samples_100 <- bowl %>%
  rep_sample_n(size = 100, reps = 1000, replace = FALSE)
```

As we did when investigating the role of confidence level, confidence intervals for each of the different 1000 samples of each of these three sample sizes has been saved into the `percentile_cis_by_n` data frame. Let's investigate width visually first and then look at the median and mean length of these intervals.



```
percentile_cis_by_n %>%
  mutate(width = upper - lower) %>%
  group_by(sample_size) %>%
  summarize(median_width = median(width),
            mean_width = mean(width))
```

```
# A tibble: 3 x 3
  sample_size median_width mean_width
  <dbl>        <dbl>      <dbl>
1      25       0.32       0.310986
2      50       0.22       0.222539
3     100       0.16       0.157986
```

So as the sample size increases the width of our confidence intervals decreases. This intuitively makes sense since as we have larger samples we are getting closer and closer to the actual size of the population. As we get closer to the actual size of the population, we will have less and less variability in the sample proportion red since there will be less and less variability in the samples pulled from the population.

9.7 Case study: Comparing two proportions

Let's now look into another example where the `infer` pipeline really shows off its power by looking at two variables. We'll have a response and an explanatory variable instead of just the one variable we've seen so far in the bowl of balls and pennies examples.

If you see someone else yawn, are you more likely to yawn? In an episode³ of the show *Mythbusters*, they tested the myth that yawning is contagious. The snippet from the show is available to view in the United States on the Discovery Network website here⁴. More information about the episode is also available on IMDb here⁵.

Fifty adults who thought they were being considered for an appearance on the show were interviewed by a show recruiter ("confederate") 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 unaware participants yawned. The data frame containing the results is available at `mythbusters_yawn` in the `moderndive` package. Let's check it out.

```
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 participant ID is stored in the `subj` variable with values of 1 to 50.

³<http://www.discovery.com/tv-shows/mythbusters/mythbusters-database/yawning-contagious/>

⁴<https://www.discovery.com/tv-shows/mythbusters/videos/is-yawning-contagious>

⁵<https://www.imdb.com/title/tt0768479/>

- The `group` variable is either "seed" for when a confederate was trying to influence the participant or "control" if a confederate did not interact with the participant.
- The `yawn` variable is either "yes" if the participant yawned or "no" if the participant did not yawn.

We can use the `janitor` package to get a glimpse into this data in a table format:

```
mythbusters_yawn %>%
  tabyl(group, yawn) %>%
  adorn_percentages() %>%
  adorn_pct_formatting() %>%
  # To show original counts
  adorn_ns()
```

group	no	yes
control	75.0% (12)	25.0% (4)
seed	70.6% (24)	29.4% (10)

We are interested in comparing the proportion of those that yawned after seeing a seed versus those that yawned with no seed interaction. We'd like to see if the difference between these two proportions is significantly larger than 0. If so, we'd have evidence to support the claim that yawning is contagious based on this study.

In looking over this problem, we can make note of some important details to include in our `infer` pipeline:

- We are calling a `success` having a `yawn` value of "yes".
- Our response variable will always correspond to the variable used in the `success` so the response variable is `yawn`.
- The explanatory variable is the other variable of interest here: `group`.

To summarize, we are looking to see the examine the relationship between yawning and whether or not the participant saw a seed yawn or not.

9.7.1 Compute the point estimate

```
mythbusters_yawn %>%
  specify(formula = yawn ~ group)
```

```
Error: A level of the response variable `yawn` needs to be specified  
for the `success` argument in `specify()`.
```

Note that the `success` argument must be specified in situations such as this where the response variable has only two levels.

```
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
```

We next want to calculate the statistic of interest for our sample. This corresponds to the difference in the proportion of successes.

```
mythbusters_yawn %>%  
  specify(formula = yawn ~ group, success = "yes") %>%  
  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. To further check to make sure that R knows exactly what we are after, we need to provide the `order` in which R should subtract these proportions of successes. As the error message states, we'll want to put "seed" first after `c()` and then "control": `order = c("seed", "control")`. Our point estimate is thus calculated:

```
obs_diff <- mythbusters_yawn %>%
  specify(formula = yawn ~ group, success = "yes") %>%
  calculate(stat = "diff in props", order = c("seed", "control"))
obs_diff
```

```
# A tibble: 1 × 1
  stat
  <dbl>
1 0.0441176
```

This value represents the proportion of those that yawned after seeing a seed yawn (0.2941) minus the proportion of those that yawned with not seeing a seed (0.25).

9.7.2 Bootstrap distribution

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 × 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.

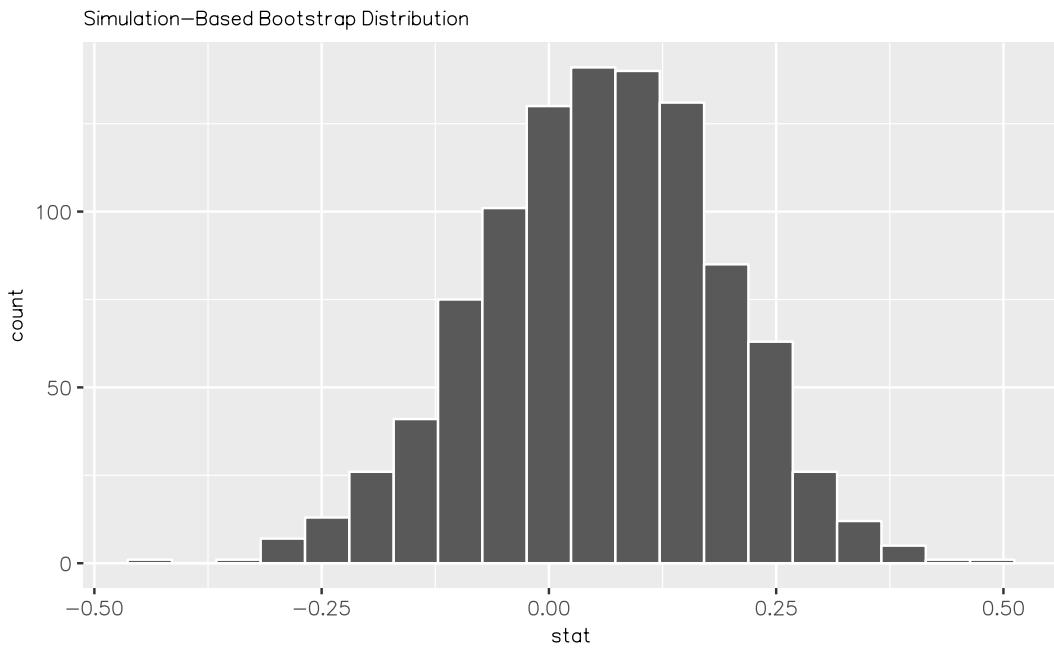
```
head(mythbusters_yawn) %>%  
  sample_n(size = 6, replace = TRUE)
```

```
# A tibble: 6 x 3  
  subj group  yawn  
  <int> <chr>  <chr>  
1     4 seed    yes  
2     6 control no  
3     3 seed    no  
4     5 seed    no  
5     3 seed    no  
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.

```
bootstrap_distribution <- mythbusters_yawn %>%  
  specify(formula = yawn ~ group, success = "yes") %>%  
  generate(reps = 1000, type = "bootstrap") %>%  
  calculate(stat = "diff in props", order = c("seed", "control"))
```

```
bootstrap_distribution %>%  
  visualize(bins = 20)
```



This distribution is roughly symmetric and bell-shaped but isn't quite there. Let's use the percentile-based method to compute a 95% confidence interval for the true difference in the proportion of those that yawn with and without a seed presented. The arguments are explicitly listed here but remember they are the defaults and simply `get_ci()` can be used.

```
bootstrap_distribution %>%
  get_ci(type = "percentile", level = 0.95)
```

```
# A tibble: 1 × 2
  `2.5%`  `97.5%`
  <dbl>    <dbl>
1 -0.202309 0.304763
```

The confidence interval shown here includes the value of 0. We'll see in Chapter 10 further what this means in terms of this difference being statistically significant or not, but let's examine a bit here first. The range of plausible values for the difference in the proportion of those that yawned with and without a seed is between -0.202 and 0.305.

Therefore, we are not sure which proportion is larger. Some of the bootstrap statistics showed the proportion without a seed to be higher and others showed the proportion with a seed to be higher. If the confidence interval was entirely above zero, we would be relatively sure (about “95% confident”) that the seed group had a higher proportion of yawning than the control group.

Note that this all relates to the importance of denoting the `order` argument in the `calculate()` function. Since we specified "seed" and then "control" positive values for the statistic correspond to the "seed" proportion being higher, whereas negative values correspond to the "control" group being higher.

We, therefore, have evidence via this confidence interval suggesting that the conclusion from the Mythbusters show that "yawning is contagious" being "confirmed" is not statistically appropriate.

9.8 Conclusion

9.8.1 Comparing bootstrap and sampling distributions

Earlier in this chapter, we mentioned that the variability of the sampling distribution is often well-approximated by the variability of the bootstrap distribution. Since we've computed both of these distributions for the `bowl` example, let's dig into both of them further to make comparisons.

Sampling distribution

Let's assume that `bowl` represents our population of interest. We'll next go over again how to create a sampling distribution for the population proportion of red balls, denoted by p , using the `rep_sample_n()` function seen in Chapter 8. Let's use a mega-virtual shovel of size 200 here. First, we will create 1000 samples from the `bowl` data frame.

```
thousand_samples <- bowl %>%
  rep_sample_n(size = 200, reps = 1000, replace = FALSE)
```

When creating a sampling distribution, we do not replace the items when we create each sample. This is in contrast to the bootstrap distribution. It's important to remember that the sampling distribution is sampling **without** replacement from the population to better understand sample-to-sample variability, whereas the bootstrap distribution is sampling **with** replacement from our original sample to better understand potential sample-to-sample variability. For the sampling distribution, we have access to the population whereas with the bootstrap distribution we are only going to pull ourselves up from our bootstraps using the single sample.

After sampling from `bowl` 1000 times, we next want to compute the proportion of red balls for each of the 1000 samples:

```
sampling_distribution <- thousand_samples %>%
  group_by(replicate) %>%
  summarize(stat = mean(color == "red"))

ggplot(sampling_distribution, aes(x = stat)) +
  geom_histogram(bins = 10, fill = "salmon", color = "white")
```

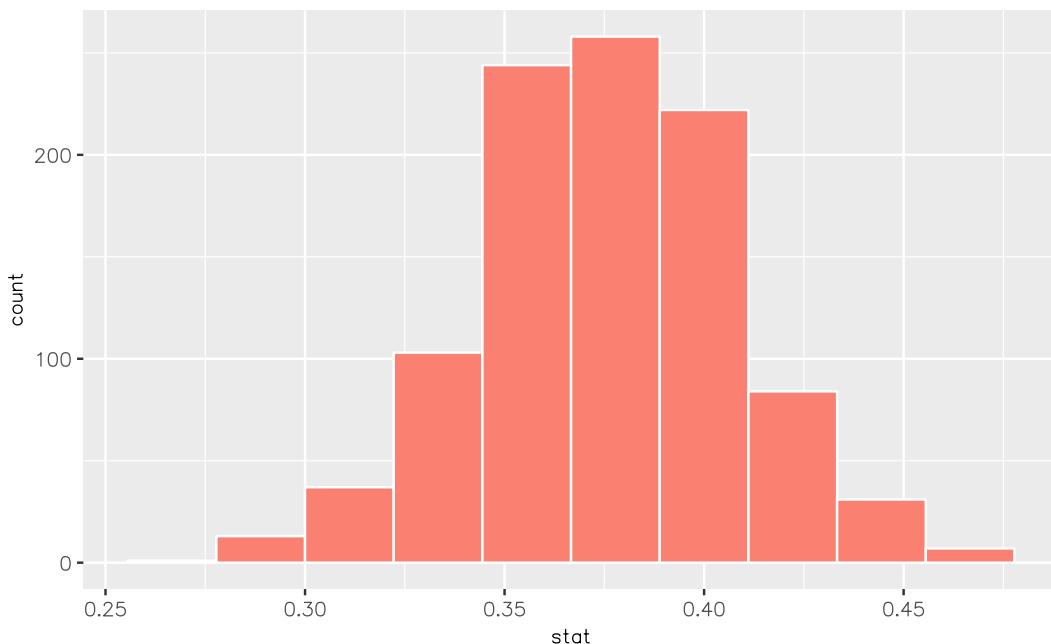


FIGURE 9.10: Sampling distribution for proportion red for n=200 samples of balls

We can also examine the variability in this sampling distribution by calculating the standard deviation of the `stat` column. Remember that the standard deviation of the sampling distribution is the **standard error**, frequently denoted as `se`.

```
sampling_distribution %>%
  summarize(se = sd(stat))
```

```
# A tibble: 1 × 1
  se
```

```
<dbl>
1 0.0323101
```

Bootstrap distribution

Let's now see how the shape of the bootstrap distribution compares to that of the sampling distribution. We'll shade the bootstrap distribution blue to further assist with remembering which is which, with the sampling distribution shaded salmon color. Let's walk through the steps needed with the `infer` pipeline to create the bootstrap distribution. We first need a sample of size 200 pulled from the `bowl` to give us a starting sample:

```
sample_200 <- bowl %>%
  sample_n(200, replace = FALSE)
```

1. specify variables

We first identify which variable(s) we are interested in for our inferential analysis.

```
sample_200 %>%
  specify(formula = color ~ NULL)
```

```
Error: A level of the response variable `color` needs to be specified for the
`success` argument in `specify()``.
```

The `infer` package sends an error here that we need to tell it which of the possible `color` options we'd like to call a `success`.

```
sample_200 %>%
  specify(formula = color ~ NULL, success = "red")
```

```
Response: color (factor)
# A tibble: 200 x 1
  color
  <fct>
  1 white
  2 white
  3 white
  4 white
  5 white
```

```
6 white
7 white
8 white
9 white
10 red
# ... with 190 more rows
```

2. generate bootstrap replicates

```
sample_200 %>%
  specify(formula = color ~ NULL, success = "red") %>%
  generate(reps = 1000, type = "bootstrap")
```

```
Response: color (factor)
# A tibble: 200,000 x 2
# Groups:   replicate [1,000]
  replicate color
  <int> <fct>
1       1 white
2       1 white
3       1 red
4       1 red
5       1 white
6       1 white
7       1 white
8       1 white
9       1 red
10      1 white
# ... with 199,990 more rows
```

3. calculate statistics

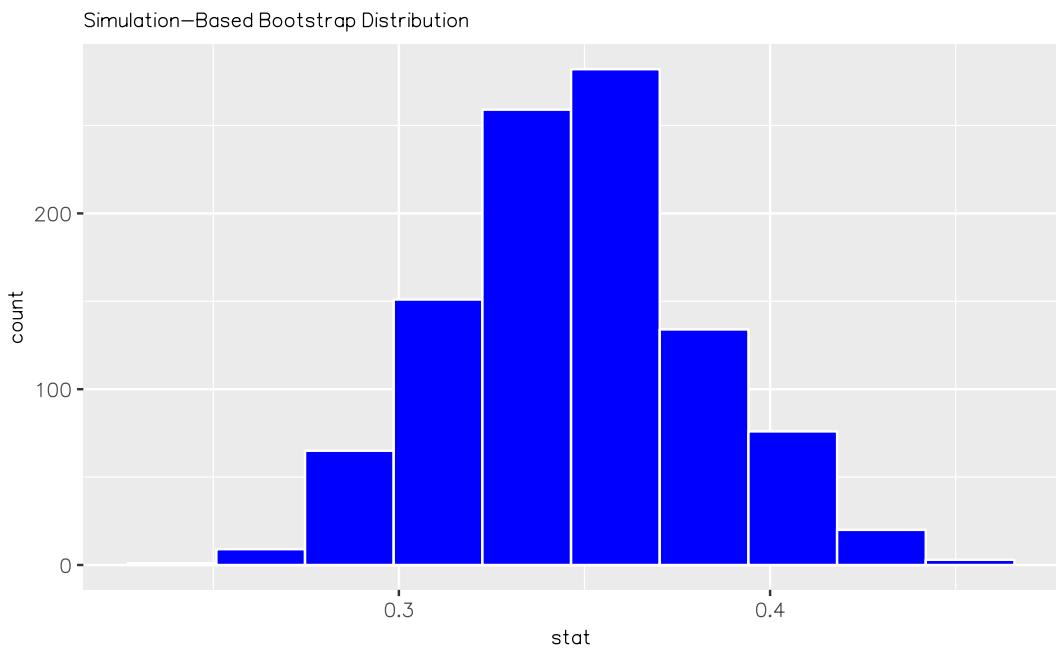
```
bootstrap_distribution_n_200 <- sample_200 %>%
  specify(formula = color ~ NULL, success = "red") %>%
  generate(reps = 1000, type = "bootstrap") %>%
  calculate(stat = "prop")
bootstrap_distribution_n_200

# A tibble: 1,000 x 2
  replicate stat
  <int> <dbl>
```

```
1      1 0.36
2      2 0.365
3      3 0.355
4      4 0.38
5      5 0.31
6      6 0.305
7      7 0.34
8      8 0.31
9      9 0.37
10     10 0.335
# ... with 990 more rows
```

4. visualize distribution of statistics

```
visualize(bootstrap_distribution_n_200, bins = 10, fill = "blue")
```



Side-by-side

Now that we have both the sampling distribution and the bootstrap distribution, let's put them on the same scales and examine their variability both visually and also by computing relevant standard deviations.

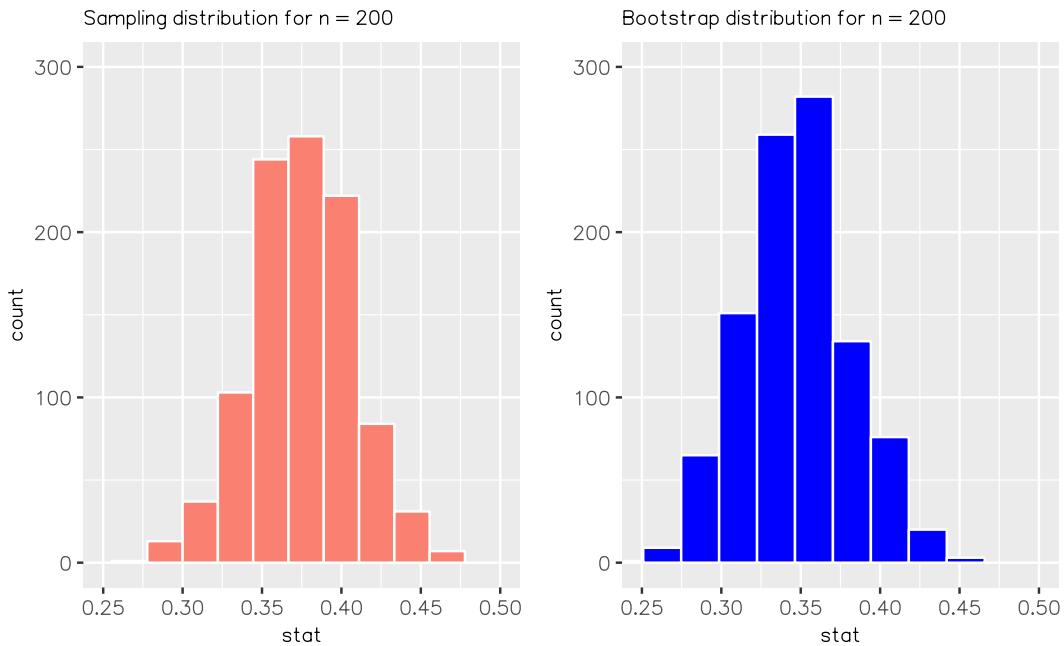


FIGURE 9.11: Comparing sampling and bootstrap distributions

```
sampling_distribution %>%
  summarize(se = sd(stat))
```

```
# A tibble: 1 × 1
  se
  <dbl>
1 0.0323101
```

```
bootstrap_distribution_n_200 %>%
  summarize(se = sd(stat))
```

```
# A tibble: 1 × 1
  se
  <dbl>
1 0.0344325
```

Notice that the bootstrap distribution's standard deviation is a good approximation for the standard error, the standard deviation of the sampling distribution. Note that while the standard deviations are similar, the center of the sampling distribution and the bootstrap distribution differ:

```
sampling_distribution %>%
  summarize(mean_of_sampling_means = mean(stat))
```

```
# A tibble: 1 × 1
  mean_of_sampling_means
  <dbl>
1 0.37501
```

```
bootstrap_distribution_n_200 %>%
  summarize(mean_of_bootstrap_means = mean(stat))
```

```
# A tibble: 1 × 1
  mean_of_bootstrap_means
  <dbl>
1 0.34816
```

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. Notice that this value matches up well with the mean of the sampling distribution. This is actually an artifact of the Central Limit Theorem introduced in Chapter 8. The mean of the sampling distribution is expected to be the mean of the overall population.

The unfortunate fact though is that we don't know the population mean in nearly all circumstances. The motivation of presenting it here was to show that the theory behind the Central Limit Theorem works using the tools you've worked with so far using the `ggplot2`, `dplyr`, `moderndive`, and `infer` packages.

If we aren't able to use the sample mean as a good guess for the population mean, how should we best go about estimating what the population mean may be if we can only select samples from the population. We've now come full circle and can discuss the underpinnings of the confidence interval and ways to interpret it.

9.8.2 Theory-based confidence intervals

When the bootstrap distribution has the nice symmetric, bell shape that we saw in the red balls example above, we can also use a formula to quantify the standard error. This provides another way to compute a confidence interval but is a little more tedious and mathematical. The steps are outlined below.

We've also shown how we can use the confidence interval (CI) interpretation in this case as well to support your understanding of this tricky concept.

Procedure for building a theory-based CI for p

To construct a theory-based confidence interval for p , the unknown true population proportion we

1. Collect a sample of size n
2. Compute \hat{p}
3. Compute the standard error

$$\text{SE} = \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$

4. Compute the margin of error

$$\text{MoE} = 1.96 \cdot \text{SE} = 1.96 \cdot \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$

5. Compute both end points of the confidence interval:

- The lower end point `lower_ci`:

$$\hat{p} - \text{MoE} = \hat{p} - 1.96 \cdot \text{SE} = \hat{p} - 1.96 \cdot \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$

- The upper end point `upper_ci`:

$$\hat{p} + \text{MoE} = \hat{p} + 1.96 \cdot \text{SE} = \hat{p} + 1.96 \cdot \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$

6. Alternatively, you can succinctly summarize a 95% confidence interval for p using the \pm symbol:

$$\hat{p} \pm \text{MoE} = \hat{p} \pm 1.96 \cdot \text{SE} = \hat{p} \pm 1.96 \cdot \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$

Confidence intervals based on 33 tactile samples

Let's load the tactile sampling data for the 33 groups from Chapter 8. Recall this data was saved in the `tactile_prop_red` data frame included in the `moderndive` package.

```
tactile_prop_red
```

Let's now apply the above procedure for constructing confidence intervals for p using the data saved in `tactile_prop_red` by adding/modifying new columns using the `dplyr` package data wrangling tools seen in Chapter 4:

1. Rename `prop_red` to `p_hat`, the official name of the sample proportion
2. Make explicit the sample size `n` of $n = 50$
3. the standard error `SE`
4. the margin of error `MoE`
5. the left endpoint of the confidence interval `lower_ci`
6. the 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
```

TABLE 9.6: 33 confidence intervals from 33 tactile samples of size $n=50$

group	red_balls	p_hat	n	SE	MoE	lower_ci	upper_ci
Ilyas, Yohan	21	0.42	50	0.070	0.137	0.283	0.557
Morgan, Terrance	17	0.34	50	0.067	0.131	0.209	0.471
Martin, Thomas	21	0.42	50	0.070	0.137	0.283	0.557
Clark, Frank	21	0.42	50	0.070	0.137	0.283	0.557
Riddhi, Karina	18	0.36	50	0.068	0.133	0.227	0.493
Andrew, Tyler	19	0.38	50	0.069	0.135	0.245	0.515
Julia	19	0.38	50	0.069	0.135	0.245	0.515
Rachel, Lauren	11	0.22	50	0.059	0.115	0.105	0.335
Daniel, Caroline	15	0.30	50	0.065	0.127	0.173	0.427
Josh, Maeve	17	0.34	50	0.067	0.131	0.209	0.471
Emily, Emily	16	0.32	50	0.066	0.129	0.191	0.449
Conrad, Emily	18	0.36	50	0.068	0.133	0.227	0.493
Oliver, Erik	17	0.34	50	0.067	0.131	0.209	0.471
Isabel, Nam	21	0.42	50	0.070	0.137	0.283	0.557
X, Claire	15	0.30	50	0.065	0.127	0.173	0.427
Cindy, Kimberly	20	0.40	50	0.069	0.136	0.264	0.536
Kevin, James	11	0.22	50	0.059	0.115	0.105	0.335
Nam, Isabelle	21	0.42	50	0.070	0.137	0.283	0.557
Harry, Yuko	15	0.30	50	0.065	0.127	0.173	0.427
Yuki, Eileen	16	0.32	50	0.066	0.129	0.191	0.449
Ramses	23	0.46	50	0.070	0.138	0.322	0.598
Joshua, Elizabeth, Stanley	15	0.30	50	0.065	0.127	0.173	0.427

TABLE 9.6: 33 confidence intervals from 33 tactile samples of size $n=50$
(continued)

group	red_balls	p_hat	n	SE	MoE	lower_ci	upper_ci
Siobhan, Jane	18	0.36	50	0.068	0.133	0.227	0.493
Jack, Will	16	0.32	50	0.066	0.129	0.191	0.449
Caroline, Katie	21	0.42	50	0.070	0.137	0.283	0.557
Griffin, Y	18	0.36	50	0.068	0.133	0.227	0.493
Kaitlin, Jordan	17	0.34	50	0.067	0.131	0.209	0.471
Ella, Garrett	18	0.36	50	0.068	0.133	0.227	0.493
Julie, Hailin	15	0.30	50	0.065	0.127	0.173	0.427
Katie, Caroline	21	0.42	50	0.070	0.137	0.283	0.557
Mallory, Damani, Melissa	21	0.42	50	0.070	0.137	0.283	0.557
Katie	16	0.32	50	0.066	0.129	0.191	0.449
Francis, Vignesh	19	0.38	50	0.069	0.135	0.245	0.515

Let's plot:

1. These 33 confidence intervals for p : from `lower_ci` to `upper_ci`
2. The true population proportion $p = 900/2400 = 0.375$ with a red vertical line

We see that:

- In 31 cases, the confidence intervals “capture” the true $p = 900/2400 = 0.375$
- In 2 cases, the confidence intervals do not “capture” the true $p = 900/2400 = 0.375$

Thus, the confidence intervals capture the true proportion $31/33 = 93.939\%$ of the time using this theory-based methodology.

Confidence intervals based on 100 virtual samples

Let's say however, we repeated the above 100 times, not tactiley, but virtually. Let's do this only 100 times instead of 1000 like we did before so that the results can fit on the screen. Again, the steps for compute a 95% confidence interval for p are:

1. Collect a sample of size $n = 50$ as we did in Chapter 8
2. Compute \hat{p} : the sample proportion red of these $n = 50$ balls
3. Compute the standard error $SE = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$
4. Compute the margin of error $MoE = 1.96 \cdot SE = 1.96 \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$
5. Compute both end points of the confidence interval:

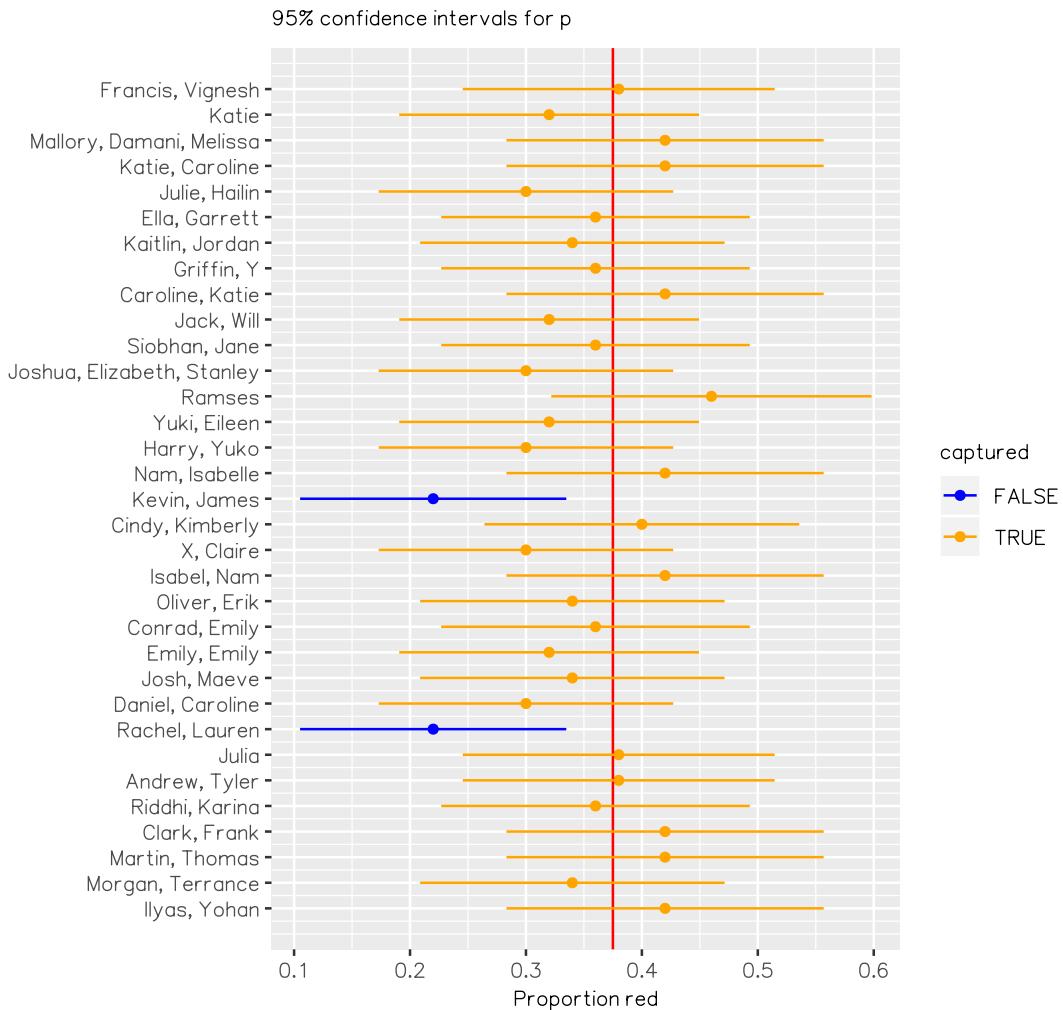


FIGURE 9.12: 33 confidence intervals based on 33 tactile samples of size $n=50$

- $\text{lower_ci}:$ $\hat{p} - \text{MoE} = \hat{p} - 1.96 \cdot \text{SE} = \hat{p} - 1.96 \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$
- $\text{upper_ci}:$ $\hat{p} + \text{MoE} = \hat{p} + 1.96 \cdot \text{SE} = \hat{p} + 1.96 \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$

Run the following three steps, being sure to `view()` the resulting data frame after each step so you can convince yourself of what's going on:

```
# First: Take 100 virtual samples of n=50 balls
virtual_samples <- bowl %>%
  rep_sample_n(size = 50, reps = 100)

# Second: For each virtual sample compute the proportion red
```

```

virtual_prop_red <- virtual_samples %>%
  group_by(replicate) %>%
  summarize(red = sum(color == "red")) %>%
  mutate(prop_red = red / 50)

# Third: Compute the 95% confidence interval as above
virtual_prop_red <- virtual_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
  )

```

Here are the results:

We see that of our 100 confidence intervals based on samples of size $n = 50$, 96 of them captured the true $p = 900/2400$, whereas 4 of them missed. As we create more and more confidence intervals based on more and more samples, about 95% of these intervals will capture. In other words our procedure is “95% reliable.”

Theoretical methods like this have largely been used in the past since we didn’t have the computing power to perform simulation-based methods such as bootstrapping. They are still commonly used though and if the normality assumptions are met, they can provide a nice option for finding confidence intervals and performing hypothesis tests as we will see in Chapter 10.

Where does the 1.96 come from?

We’ve been mentioning quite a bit throughout this chapter that if the distributions are bell-shaped and symmetric that things will likely work nicely for us. This bell-shaped distribution is commonly called the Gaussian or normal distribution. It has that characteristic shape of a bell that we’ve discussed.

The 1.96 in our formula for a 95% theory-based confidence interval is directly related to the normal distribution. The normal distribution is actually a family of distributions with each characterized by their mean and their standard deviation. The *standard normal distribution* is one of the most common since it acts as a standardization for all of the other normal distributions. In other

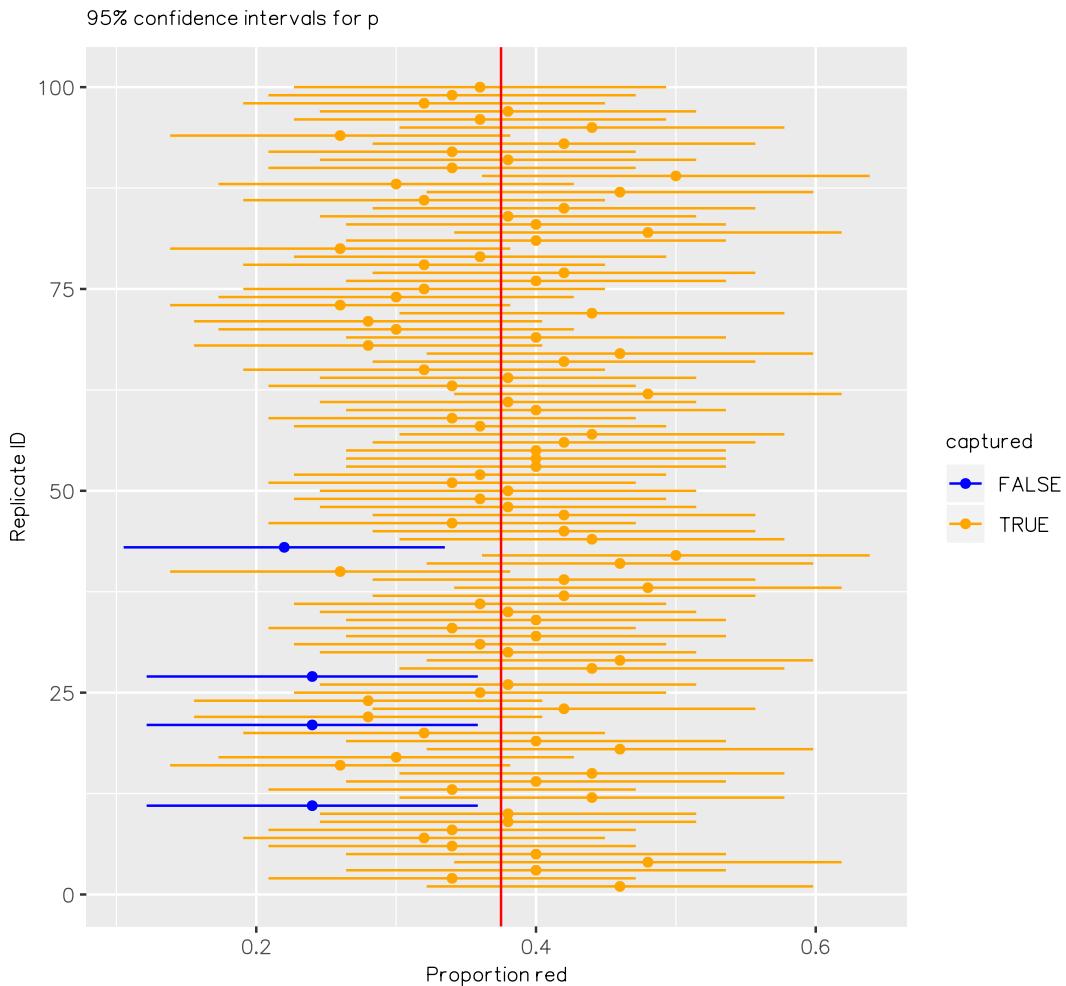
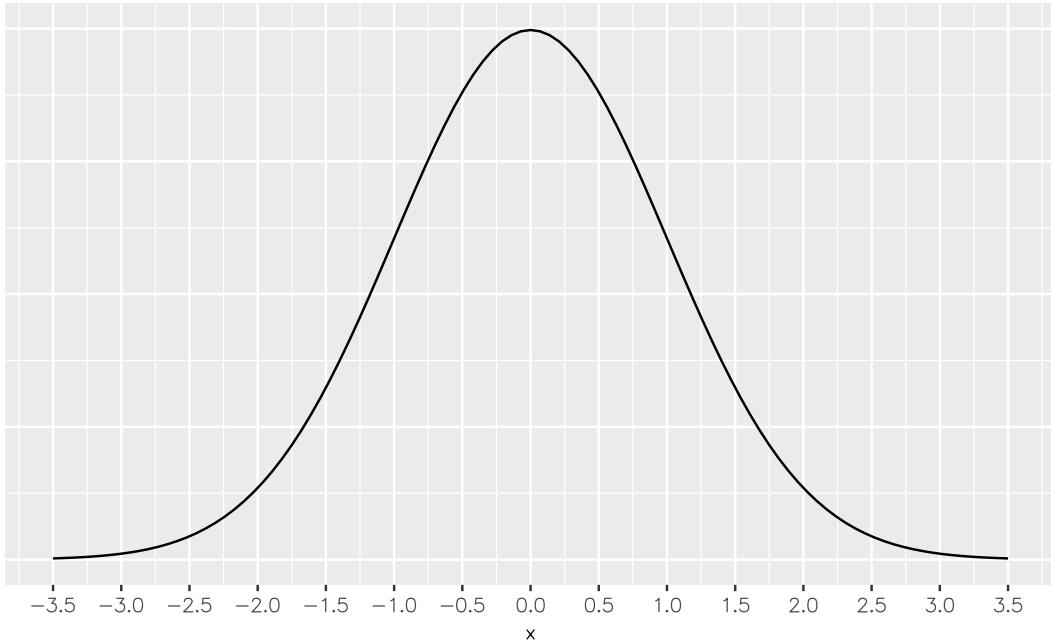


FIGURE 9.13: 100 confidence intervals based on 100 virtual samples of size $n=50$

words, via some formulas, any value of a normal distribution can be converted to its corresponding value on the standard normal distribution. Let's take a look visually at this standard normal distribution and the range of different values it can take on.



Let's draw a vertical line at both 1.96 and -1.96 on this plot.



Any guesses to how much area is under the black curve and to the right 1.96? The correct answer is very close to 2.5%. Since the normal distribution is symmetric there is also 2.5% of the area to the left of -1.96. Therefore, if we wanted to encapsulate the middle 95% of the values on the standard normal

distribution we'd be pretty close to between -1.96 and 1.96. That's the reason why we choose 1.96 as our multiplier in the formula above.

What if we wanted to get the multiplier for say a 90% theory-based confidence interval? R has a built-in function to help us with that:

```
qnorm(p = 0.95)
```

```
[1] 1.64
```

Here `q` stands for “quantile” and `norm` stands for normal. So the 95th percentile of the standard normal distribution falls at around 1.65. Let's check to see where the 2.5th percentile falls:

```
qnorm(p = 0.025)
```

```
[1] -1.96
```

This is what we expected above. Close to -1.96 corresponds to the spot that is 2.5% of the way into the values of the standard normal distribution. We'll elaborate more on these theory-based methods in Chapter 10, but this should give you a good start!

Learning check

Practice problems to come soon!

9.8.3 Summary table

In this chapter, we performed both tactile and virtual simulations of resampling/bootstrapping to infer about unknown parameters. We also presented a case study of bootstrapping in a real-life situation: the suggested contagiousness of yawning. We used the sample proportion \hat{p} to estimate the population proportion p and the sample mean $\bar{x} = \hat{\mu}$ to estimate the population mean. We also explored a two variable problem in our yawning case study. Let's review these and others again in Table 10.4.

TABLE 9.7: 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	$\hat{\mu}$ or \bar{x}
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	Sample regression slope	$\hat{\beta}_1$ or b_1
6	Population regression intercept	β_0	Sample regression intercept	$\hat{\beta}_0$ or b_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 10, we'll see 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.

9.8.4 Additional resources

An R script file of all R code used in this chapter is available here⁶.

⁶[scripts/09-confidence-intervals.R](#)

9.8.5 What's to come?

This chapter introduced the notions of bootstrapping and confidence intervals as ways to build intuition about population parameters using only the original sample information. We also concluded with a glimpse into statistical significance and we'll dig much further into this in Chapter 10 up next!

10

Hypothesis Testing

In preparation for our first print edition to be published by CRC Press in Fall 2019, we're remodeling this chapter a bit. Don't expect major changes in content, but rather only minor changes in presentation. Our remodeling will be complete and available online at [ModernDive.com¹](https://ModernDive.com) by early Summer 2019!

We saw some of the main concepts of hypothesis testing introduced in Chapters 8 and 9. We will expand further on these ideas here and also provide a framework for understanding hypothesis tests in general. Instead of presenting you with lots of different formulas and scenarios, we hope to build a way to think about all hypothesis tests. You can then adapt to different scenarios as needed down the road when you encounter different statistical situations.

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. If you'd like more practice or to see how this framework applies to different scenarios, you can find fully-worked out examples for many common hypothesis tests and their corresponding confidence intervals in Appendix B.

We recommend that you carefully review these examples as they also cover how the general frameworks apply to traditional normal-based methodologies like the *t*-test and normal-theory confidence intervals. You'll see there that these traditional methods are just approximations for the general computational frameworks, but require conditions to be met for their results to be valid. The general frameworks using randomization, simulation, and bootstrapping

do not hold the same sorts of restrictions and further advance computational thinking, which is one big reason for their emphasis throughout this textbook.

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(tidyverse)
library(janitor)
library(infer)
library(moderndive)
library(ggplot2movies)
library(nycflights13)
```

10.1 Hypothesis testing activity

Let's build on the ideas shown in Chapters 8 and 9 via an activity. From this activity we'll discuss much of the terminology used in hypothesis testing via an example based on gender differences in promotions from a 1970's study.

10.1.1 Question of interest

We will be looking to analyze "Are men and women rated for promotions differently?" We note again here as we did in Section 7.1 that this study from 1974 only focused on the gender binary of "`male`" and "`female`". We proceed with the example here to help to motivate the concepts of hypothesis testing via an interesting social question, but we again understand that a segment of our readers will not be included in this binary gender classification.

This data from this study in the "Journal of Applied Psychology" has been loaded into the `moderndive` package as `gender_promotions`. The study looks into different personnel decisions including promotion, professional development, and whether or not a grant a leave of absence based on gender. For our pur-

poses, we will focus on the promotion portion based on the decisions from 48 male bank supervisors in 1972.

To begin the study, the 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. Each of these resumes was the same in terms of content and style with the only difference being that of the name at the top of the resume.

It was hypothesized based on other studies at the time and commonly held assumptions that male supervisors were more likely to grant male candidates promotions than female candidates. Does this study also back up this claim? Or does it not provide evidence that there is a difference in the proportion of males and females promoted based on reviews by male supervisors?

10.1.2 What did we actually observe?

The `moderndive` package contains data from this 1974 study on the role of binary gender on promotions at banks in the `gender_promotions` data frame. Let's look at what the data shows in this sample as a table using the `janitor` package we saw in Section 9.7.

```
gender_promotions

# A tibble: 48 x 3
  id decision gender
  <int> <fct>   <fct>
1     1 promoted female
2     2 not      female
3     3 not      female
4     4 promoted male
5     5 promoted male
6     6 promoted male
7     7 promoted male
8     8 promoted male
9     9 promoted female
10    10 promoted male
# ... with 38 more rows
```

```
glimpse(gender_promotions)
```

```
Observations: 48
Variables: 3
$ id      <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ...
$ decision <fct> promoted, not, not, promoted, promoted...
$ gender   <fct> female, female, female, male, male, ma...
```

Let's lay out what this sample information looks like using decks of index cards. Let's take 48 note cards and assign them the `id` values of 1 to 48 and also whether or not that supervisor decided to promote from the second column of `gender_promotions`. Set those in, say, 8 rows each with 6 entries from left to right and top to bottom. If you'd like you can denote `promoted` as green in color and `not` as red in color corresponding to common stoplight colors. Feel free to tweak this to different colors to assist with color-blind individuals as you like.

Then take a second deck of 24 index cards of a third color corresponding to "male" and lastly a third deck of 24 index cards for "female". Now, setup this original sample by correctly matching male or female to the `id` of the supervisor by looking at the third column of `gender_promotions`. Place the card of appropriate gender next to each of the 48 supervisor cards matching up with what we saw in our original sample. Now let's aggregate the proportion of successes (promotions) based on the two genders using the `janitor` package.

```
gender_promotions %>%
  tabyl(gender, decision) %>%
  adorn_percentages() %>%
  adorn_pct_formatting() %>%
  # To show original counts
  adorn_ns()
```

gender	promoted	not
female	58.3% (14)	41.7% (10)
male	87.5% (21)	12.5% (3)

We see that males were chosen for promotion at a rate of $21/24 = 87.5\%$ whereas females were chosen at a rate of $14/24 \approx 58.3\%$. This leads to an *observed test statistic* of around 29.2% (0.292) when taking the proportion of promotions for males minus the proportion of promotions for females. Does this provide evidence that there is some discrimination in terms of gender in

promotions at banks? While males were suggested for promotion at a higher rate, sampling variability could be playing a role here.

If males and females were equally likely of being selected for promotion (that there was no gender discrimination in promotion), how likely would we be to see a difference as large (or larger) than this difference of ≈ 0.292 (29.2%)? Is a result like this uncommon in samples, assuming there is no gender discrimination? Or are there lots of other potential samples that could lead to a result like this?

In order to get to answer this question, we again will rely on the process of simulation to generate results. First let's build up a tactile simulation using notecards to understand what some instances assuming equal percentages for promoting males and females would look like. This process is called **permuting** and you'll see why next.

10.1.3 Using permuting once

We assume the two population proportions are equal ($H_0 : p_{male} - p_{female} = 0$). In other words, we assume that the proportion of males being selected for promotion (p_{male}) is the same as the proportion of females being selected for promotion (p_{female}). So what would our sample look like if we had made this assumption?

Going back to our index cards, pick up each of the 24 cards corresponding to males and females that you placed on top of the manager cards. The next step is to put the two stacks of index cards together, creating a new set of 48 cards. If we assume that the two population means are equal, we are saying that there is no association between promotion and gender (male vs female). If there really is no association between these two variables than for each of the 48 managers, it wouldn't matter whether they saw the name of a male or female candidate on the resume they were given. They'd each be equally likely of granting a promotion for each of the two binary genders. So how do we do this with the cards?

Now that we have the our 48 cards corresponding to gender in a single pile, shuffle them. Feel free to do this a couple times. Now take each of the cards off the top of the pile and assign them to the 48 different supervisors. Keep the supervisor cards in the same place they were before. We are, thus, randomly assigning the different values of the **explanatory** variable to each of the entries of the **response** variable. To reiterate, we hold the response variable of **promotion** fixed by not shuffling those cards but we shuffle the values of **gender**.

TABLE 10.1: First 10 rows of original (left) and permuted (right) data

id	decision	gender
1	promoted	female
2	not	female
3	not	female
4	promoted	male
5	promoted	male
6	promoted	male
7	promoted	male
8	promoted	male
9	promoted	female
10	promoted	male

id	decision	gender
1	promoted	female
2	not	male
3	not	female
4	promoted	female
5	promoted	female
6	promoted	female
7	promoted	female
8	promoted	female
9	promoted	male
10	promoted	male

*

as the explanatory variable. Let's check out what the first few rows of this permutation of the gender cards onto the supervisors might look like as data.

We see that from the permuting of the gender cards we have the following new assignments to supervisors with `ids` of 2, 4, 5, 6, 7, 8, and 9. In these rows, the gender has switched on the presented resumes from what was in the original sample of resumes. The other rows (supervisors) have remained the same in terms of gender of their candidate. What does the difference in the promotion rates of males and females look like now our permuted data?

gender	promoted	not
--------	----------	-----

```
female 66.7% (16) 33.3% (8)
male 79.2% (19) 20.8% (5)
```

We can calculate the *test statistic* for this permutation now as well to get a value of $79.2\% - 66.7\% = 12.5\%$. So this is one potential sample statistic that could come about based on chance alone if there was in fact no gender discrimination in promotions. We could have each of many friends help us out here to generate more permutations and resulting differences in the proportions of successes. Let's do that 33 times as we did in the previous chapters.

10.1.4 Using permuting 33 times

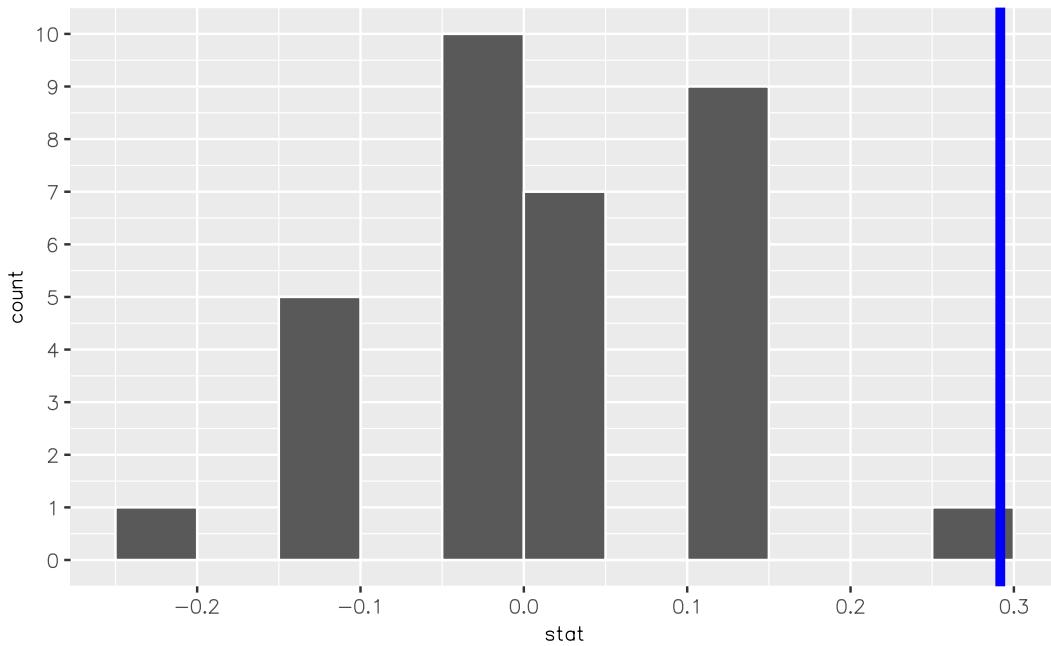
Now each of our 33 friends does the following:

1. Takes the two decks of cards.
2. Shuffles the cards corresponding to gender.
3. Assigns the shuffled cards to the original deck of supervisors' decisions.
4. Count how many cards fall into each of the four categories:
 - Promoted males
 - Non-promoted males
 - Promoted females
 - Non-promoted females
5. Determines the proportion of promoted males out of 24.
6. Determines the proportion of promoted females out of 24.
7. Subtracts those two differences to get a new value of the test statistic, assuming the null hypothesis is true.

Let's see what this leads to for our friends in terms of results and label where the observed test statistic falls in relation to our friends' statistics:

```
obs_diff_prop <- gender_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
```



We see that of the 33 samples we selected only one is close to as extreme as what we observed. Thus, we might guess that we are starting to see some data suggesting that gender discrimination might be at play. Many the statistics calculated appear close to 0 with the vast remainder appearing around values of a difference of -0.1 and 0.1. So what further evidence would we need to make this suggestion a little clearer? More simulations! As we've done before in Chapters 8 and 9, we'll use the computer to simulate these permutations and calculations many times. Let's do just that with the `infer` package in the next section.

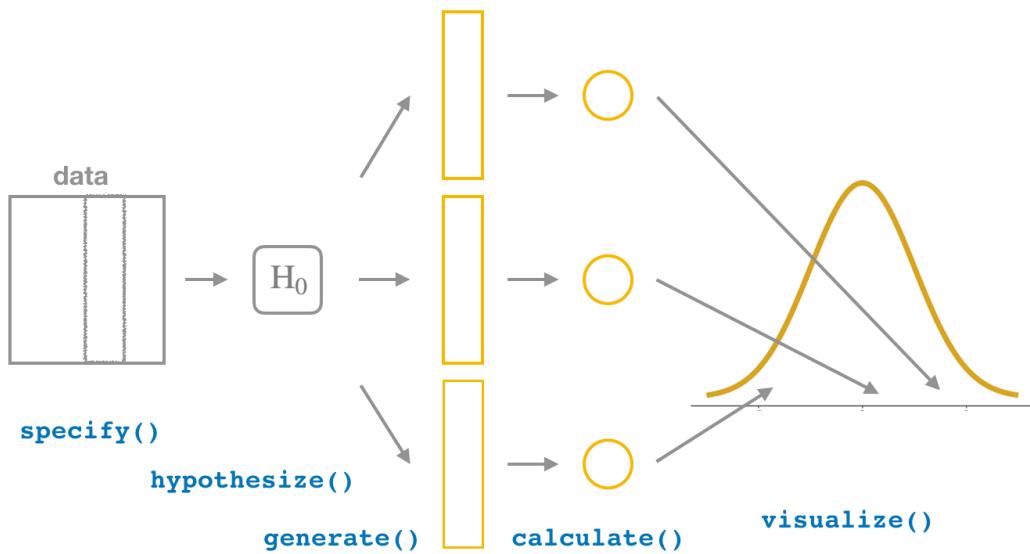
10.2 Hypothesis testing with `infer`

10.2.1 Revisiting the `infer` verb framework

In Chapter 9, you were introduced to the framework for inference including the following verbs: `specify()`, `generate()`, and `calculate()`. This was useful when calculating bootstrap distributions in order to develop confidence intervals in both the one-sample and two-sample cases. One of the great powers of the `infer` package is in extending confidence intervals to hypothesis testing by including one more verb: `hypothesize()`. Its main argument is `null` which is either

- "point" for point hypotheses involving a single sample or
- "independence" for testing for independence between two variables.

Hypothesis test



Let's see how this is done with the gender discrimination study from Section 10.1.

10.2.2 The `infer` pipeline for the activity

Remember that our goal here is to generate many different samples, assuming the null hypothesis is true. In doing so we will create a **null distribution**. This null distribution is similar to what we have seen so far with the *sampling distribution* in Chapter 8 and the *bootstrap distribution* in Chapter 9. Here though we have one more condition to apply in that we assume the null hypothesis is true, thus where the name of the *null* distribution comes from. The null distribution is still used to look at the variability from one sample to the next, but now we are interested in seeing where what we actually saw would fall on the “chance distribution.” In doing so, we’ll have a good sense for whether random chance is a good explanation for seeing the results in our observed sample or if there is something else going on which better aligns with H_a , the alternative hypothesis.

Let's explore the `infer` pipeline one more time here:

Choose the variables of interest

We use the `specify()` verb to denote the response and, if needed, explanatory variables for our study. In this case, we are investigating `decision` as the response variable and `gender` as the explanatory variable. Recall that for the `formula` argument we use 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. Note also the importance of including which of the two levels here "promoted" or "not" is what we are defining as a success from the `decision` (response) variable.

```
gender_promotions %>%
  specify(formula = decision ~ gender, success = "promoted")
```

```
Response: decision (factor)
Explanatory: gender (factor)
# A tibble: 48 x 2
  decision gender
  <fct>    <fct>
  1 promoted female
  2 not      female
  3 not      female
  4 promoted male
  5 promoted male
  6 promoted male
  7 promoted male
  8 promoted male
  9 promoted female
  10 promoted male
# ... with 38 more rows
```

Set the model for the null hypothesis

The next step after `specify()` for hypothesis tests is a new one: `hypothesize()`. Here the argument `null` gives which type of null hypothesis we are working with. In this case, we are testing for the independence of the two variables so we set `null = "independence"`.

```
gender_promotions %>%
  specify(formula = decision ~ gender, success = "promoted")
  hypothesize(null = "independence")
```

```
# A tibble: 48 x 2
  decision gender
  <fct>    <fct>
  1 promoted female
  2 not      female
  3 not      female
  4 promoted male
  5 promoted male
  6 promoted male
  7 promoted male
  8 promoted male
  9 promoted female
 10 promoted male
# ... with 38 more rows
```

Replicate samples assuming the null hypothesis is true

After we have set the model for the null hypothesis, we simulate the null hypothesis being true by permuting our original sample in much the same way as we did with index cards in Section 10.1. We'll now let the computer do this shuffling many times. Let's use the 1000 `reps` argument we've used before in the `generate()` verb, but this time generating permutations with `type = "permute"` instead of `type = "bootstrap"` that we used previously for confidence intervals in Chapter 9.

```
gender_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 female      1
  2 promoted female      1
  3 not      female      1
  4 promoted male       1
  5 promoted male       1
# ... with 47,995 more rows
```

```

6 promoted male      1
7 promoted male      1
8 promoted male      1
9 promoted female    1
10 promoted male     1
# ... with 47,990 more rows

```

Compute the statistic for each replicate

Now that we have 1000 replicated samples assuming the null hypothesis of independence of the two variables `decision` and `gender`, we `calculate()` the difference in proportions ("diff in props") for each permutation. We finish this step by including the `order` in which we'd like to take the difference. In this case, we've chosen '`male`' – '`female`'.

```

null_distribution_two_props <- gender_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_two_props

```

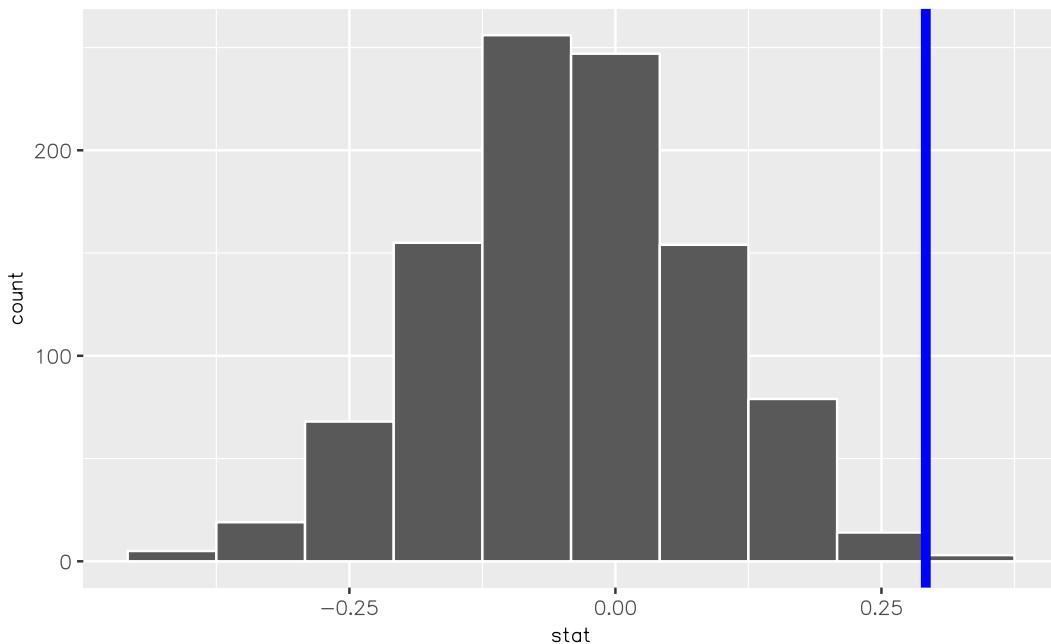
```

# A tibble: 1,000 x 2
  replicate   stat
  <int>   <dbl>
1       1  0.0416667
2       2  0.0416667
3       3 -0.125
4       4  0.0416667
5       5  0.0416667
6       6 -0.125
7       7  0.125
8       8  0.0416667
9       9  0.208333
10      10  0.0416667
# ... with 990 more rows

```

We conclude by showing where our observed statistic falls on this null distribution. In the sections that follow, we'll delve further into how to make a decision about whether or not the observed statistic is “extreme enough” to be considered statistically significant. In other words, we'll investigate what level of statistical evidence is needed for us to go against the original assumption of

random chance as the explanation for differences in the observed test statistic in favor of the alternative hypothesis.



We now see even greater evidence that gender discrimination was at play in how males and females were assigned promotion. We now have almost all of the 1000 statistics of the null distribution not as extreme as our observed test statistic. We'll formalize this in the image below as well as in Section 10.3 where we define the term **p-value** in relation to this set-up.

10.2.3 The “There Is Only One Test” framework

In a hypothesis test, we will use data from a sample to help us decide between two competing *hypotheses* about a population. We make these hypotheses more concrete by specifying them in terms of at least one *population parameter* of interest. We refer to the competing claims about the population as the **null hypothesis**, denoted by H_0 , and the **alternative (or research) hypothesis**, denoted by H_a . The roles of these two hypotheses are NOT interchangeable.

- The claim for which we seek significant evidence is assigned to the alternative hypothesis. The alternative is usually what the experimenter or researcher wants to establish or find evidence for.
- Usually, 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.

- We assess the strength of evidence by assuming the null hypothesis is true and determining how unlikely it would be to see sample results/statistics as extreme (or more extreme) as those in the original sample.

Hypothesis testing brings about many weird and incorrect notions in the scientific community and society at large. One reason for this is that statistics has traditionally been thought of as this magic box of algorithms and procedures to get to results and this has been readily apparent if you do a Google search of “flowchart statistics hypothesis tests.” There are so many different complex ways to determine which test is appropriate.

You’ll see that we don’t need to rely on these complicated series of assumptions and procedures to conduct a hypothesis test any longer. These methods were introduced in a time when computers weren’t powerful. Most cellphones (in 2019) have more power than the computers that sent NASA astronauts to the moon after all. We’ll see that ALL hypothesis tests can be broken down into the following framework given by Allen Downey here²:

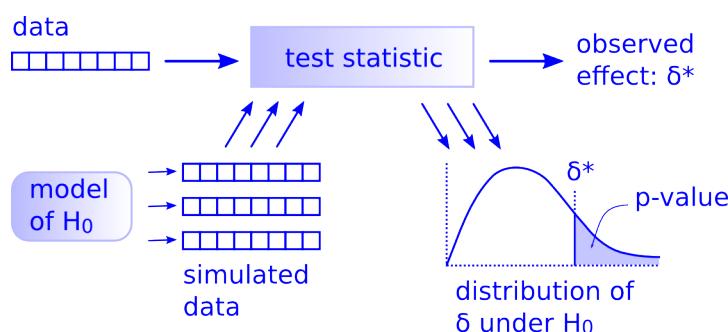


FIGURE 10.1: Hypothesis Testing Framework

In the examples that follow, we’ll phrase things in terms of this framework by Downey, which is one of the inspirations for the `infer` package. Downey finishes off the process we have been working through so far with the shading of the **p-value** in Figure 10.1. But what does this shading mean?

²<http://allendowney.blogspot.com/2016/06/there-is-still-only-one-test.html>

10.3 The *p*-value

Remember that we are interested in seeing where our observed sample difference in proportions of 0.292 falls on this null/randomization distribution. We are interested in seeing if males have a statistically higher rate of promotion so “more extreme” corresponds to values equal to or greater than what we saw. Thus, we’ll be looking for values as extreme or more extreme than what we saw in the right tail. Let’s shade our null distribution to show a visual representation of our *p*-value:

```
visualize(null_distribution_two_props, bins = 10) +  
  shade_p_value(obs_stat = obs_diff_prop, direction = "right")
```

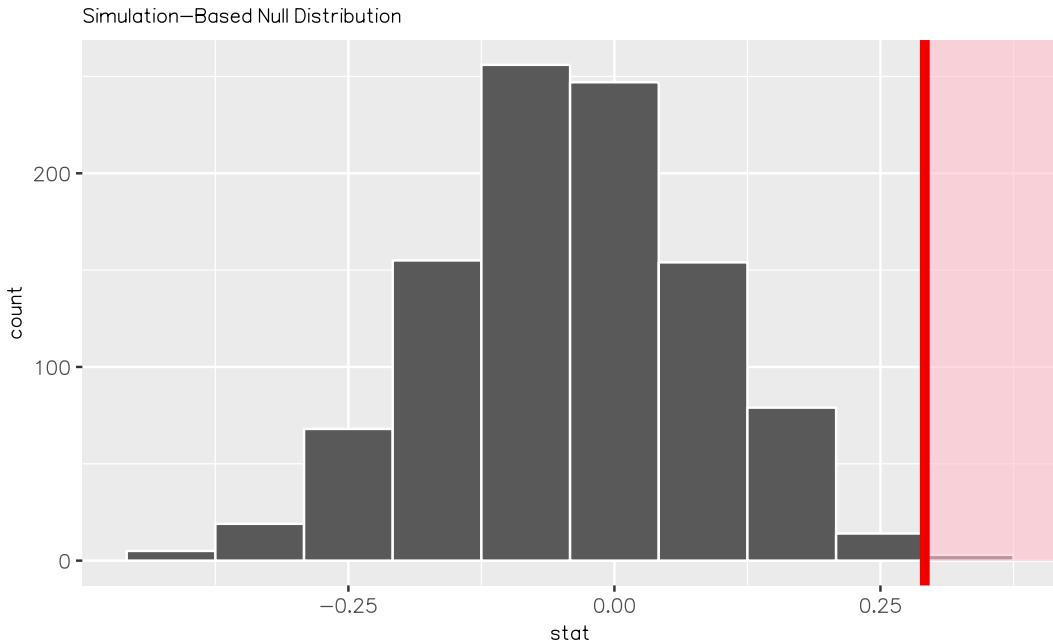


FIGURE 10.2: Shaded histogram to show *p*-value

Remember that the observed difference in means was 0.292. We have shaded red all values at or above that value. At this point, it is important to take a guess as to what the *p*-value may be. We can see that there are only a few permuted differences as extreme or more extreme than our observed effect (in both directions). Maybe we guess that this *p*-value is somewhere around 2%, or maybe 3%, but certainly not 30% or more. Lastly, we calculate the *p*-value directly using `infer`:

```
p_value <- null_distribution_two_props %>%
  get_p_value(obs_stat = obs_diff_prop, direction = "both")
p_value

# A tibble: 1 × 1
p_value
<dbl>
1 0.034
```

We have around 3.4% of values as extreme or more extreme than our observed statistic in both directions. With hypothesis tests, it is common and recommended to first set the **significance level** of the test. It is denoted as the Greek letter α . Common values for α are 0.1, 0.01, and 0.05 with 0.05 being the most common choice. This threshold is the point at which we decide that our observed results are “beyond a reasonable doubt.” Assuming we are using a 5% significance level for α , we have evidence supporting the conclusion that the proportion of males being chosen for promotion is greater than the proportion of females being chosen for promotion. The next important idea is to better understand just how much higher of a proportion for promotion can we expect the males to have compared to that of the females.

10.3.1 Corresponding confidence interval

One of the great things about the `infer` pipeline is that going between hypothesis tests and confidence intervals is incredibly simple. To create a null distribution, we ran

```
null_distribution_two_props <- gender_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 get the corresponding bootstrap distribution with which we can compute a confidence interval, we can just remove or comment out the `hypothesize()` step since we are no longer assuming the null hypothesis is true when we bootstrap. We also switch the `type` in `generate()` to be `"bootstrap"` to denote the confidence interval calculation instead of permutations for hypothesis tests.

```
percentile_ci_two_props <- gender_promotions %>%
  specify(formula = decision ~ gender, success = "promoted") %>%
  # hypothesize(null = "independence") %>%
  generate(reps = 1000, type = "bootstrap") %>%
  calculate(stat = "diff in props", order = c("male", "female")) %>%
  get_ci()
```

Thus, we can expect the true proportion of men promoted in the relevant population to be 0.042 to 0.532 higher than that of females. Remember that this is based on bootstrapping using `gender_promotions` as our original sample and the confidence interval process being 95% reliable.

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?

10.3.2 Summary

To review, these are the steps one would take whenever you'd like to do a hypothesis test comparing values from the distributions of two groups:

- Simulate many samples using a random process that matches the way the original data were collected and that *assumes the null hypothesis is true*.
 - Collect the values of a sample statistic for each sample created using this random process to build a *null distribution*.
 - Assess the significance of the *original* sample by determining where its sample statistic lies in the null distribution.
 - If the proportion of values as extreme or more extreme than the observed statistic in the randomization distribution is smaller than the pre-determined significance level α , we reject H_0 . Otherwise, we fail to reject H_0 . (If no significance level is given, one can assume $\alpha = 0.05$.)
-

10.4 Interpretation of hypothesis testing results

Hypothesis tests are often challenging to understand at first. In this section, we'll focus on ways to help with deciphering of the process in general.

10.4.1 Criminal trial analogy

We can think of hypothesis testing in the same context as a criminal trial in the United States. A criminal trial in the United States is a familiar situation in which a choice between two contradictory claims must be made.

1. The accuser of the crime must be judged either guilty or not guilty.
2. Under the U.S. system of justice, the individual on trial is initially presumed not guilty.
3. Only STRONG EVIDENCE to the contrary causes the not guilty claim to be rejected in favor of a guilty verdict.
4. The phrase “beyond a reasonable doubt” is often used to set the cutoff value for when enough evidence has been given to convict.

Theoretically, we should never say “The person is innocent.” but instead “There is not sufficient evidence to show that the person is guilty.”

Now let's compare that to how we look at a hypothesis test.

1. The decision about the population parameter(s) must be judged to follow one of two hypotheses.
2. We initially assume that H_0 is true.
3. The null hypothesis H_0 will be rejected (in favor of H_a) only if the sample evidence strongly suggests that H_0 is false. If the sample does not provide such evidence, H_0 will not be rejected.
4. The analogy to “beyond a reasonable doubt” in hypothesis testing is what is known as the **significance level**. This will be set before conducting the hypothesis test and is denoted as α . Common values for α are 0.1, 0.01, and 0.05.

Two possible conclusions

Therefore, we have two possible conclusions with hypothesis testing:

- Reject H_0
- Fail to reject H_0

Gut instinct says that “Fail to reject H_0 ” should say “Accept H_0 ” but this technically is not correct. Accepting H_0 is the same as saying that a person is 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.

When you run a hypothesis test, you are the jury of the trial. You decide whether there is enough evidence to convince yourself that H_a is true (“the person is guilty”) or that there was not enough evidence to convince yourself H_a is true (“the person is not guilty”). You must convince yourself (using statistical arguments) which hypothesis is the correct one given the sample information.

Important note: Therefore, DO NOT WRITE “Accept H_0 ” any time you conduct a hypothesis test. Instead write “Fail to reject H_0 .”

10.4.2 Types of errors in hypothesis testing

Unfortunately, just as a jury or a judge can make an incorrect decision in regards to a criminal trial by reaching the wrong verdict, there is some chance we will reach the wrong conclusion via a hypothesis test about a population parameter. As with criminal trials, this comes from the fact that we don’t have

complete information, but rather a sample from which to try to infer about a population.

The possible erroneous conclusions in a criminal trial are

- an innocent person is convicted (found guilty) or
- a guilty person is set free (found not guilty).

The possible errors in a hypothesis test are

- rejecting H_0 when in fact H_0 is true (Type I Error) or
- failing to reject H_0 when in fact H_0 is false (Type II Error).

The risk of error is the price researchers pay for basing an inference about a population on a sample. With any reasonable sample-based procedure, there is some 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 error, observe the following table based on a criminal trial:

		Actual result	
		Guilty	Not guilty
Verdict	Guilty		
	Guilty verdict	True Positive (Correct result)	False Positive (Type I Error)
Not guilty verdict		False Negative (Type II Error)	True Negative (Correct result)

FIGURE 10.3: Type I and Type II errors

If we are using sample data to make inferences about a parameter, we run the risk of making a mistake. Obviously, we want to minimize our chance of 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 called the **significance level** of a hypothesis test
- The probability of a Type II Error is denoted by β .

Formally, we can define α and β in regards to the table above, but for hypothesis tests instead of a criminal trial.

- α corresponds to the probability of rejecting H_0 when, in fact, H_0 is true.
- β corresponds to the probability of 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 an error does not exist. When we have to use incomplete information (sample data), it is not possible to have both $\alpha = 0$ and $\beta = 0$. We will always have the possibility of at least one error existing when we use sample data.

Usually, what is done is that α is set before the hypothesis test is conducted and then the evidence is judged against that significance level. Common values for α are 0.05, 0.01, and 0.10. If $\alpha = 0.05$, we are using a testing procedure that, used over and over with different samples, rejects a TRUE null hypothesis five percent of the time.

So if we can set α to be whatever we want, why choose 0.05 instead of 0.01 or even better 0.0000000000000001? Well, a small α means the test procedure requires the evidence against H_0 to be **very strong** before we can reject H_0 . This means we will almost never reject H_0 if α is very small. If we almost never reject H_0 , the probability of a Type II Error – failing to reject H_0 when we should – will *increase!* Thus, as α decreases, β increases and as α increases, β decreases. We, therefore, need to strike a balance in α and β and the common values for α of 0.05, 0.01, and 0.10 usually lead to a nice balance.

Learning check

(LC10.6) Reproduce the table above about errors, but for a hypothesis test, instead of the one provided for a criminal trial.

Logic of hypothesis testing

- Take a random sample (or samples) from a population (or multiple populations)
- If the sample data are consistent with the null hypothesis, do not reject the null hypothesis.
- If the sample data are inconsistent with the null hypothesis (in the direction of the alternative hypothesis), reject the null hypothesis and conclude that

there is evidence the alternative hypothesis is true (based on the particular sample collected).

10.4.3 Statistical significance

The idea that sample results are more extreme than we would reasonably expect to see by random chance if the null hypothesis were true is the fundamental idea behind statistical hypothesis tests. If data at least as extreme would be very unlikely if the null hypothesis were true, we say the data are **statistically significant**. Statistically significant data provide convincing evidence against the null hypothesis in favor of the alternative, and allow us to generalize our sample results to the claim about the population.

Learning check

(LC10.7) What is wrong about saying “The defendant is innocent.” based on the US system of criminal trials?

(LC10.8) What is the purpose of hypothesis testing?

(LC10.9) What are some flaws with hypothesis testing? How could we alleviate them?

10.5 Case study: comparing two means

10.5.1 Randomization/permutation

We will now focus on building hypotheses looking at the difference between two population means in an example. We will denote population means using the Greek symbol μ (pronounced “mu”). Thus, we will be looking to see if one group “out-performs” another group. This is quite possibly the most common type of statistical inference and serves as a basis for many other types of analyses when comparing the relationship between two variables.


```
TRUE ~ "Neither")) %>%
filter(genre != "Neither") %>%
select(-Action, -Romance)
```

The `case_when()` function in the `dplyr` package is useful for assigning values in a new variable based on the values of one or more other variables. The last step of `TRUE ~ "Neither"` is used when a particular movie is not set to either Action or Romance.

We are left with 8878 movies in our *population* dataset that focuses on only "Action" and "Romance" movies. Note that we call this a population dataset since it includes all of the information we have available to us about movies and ratings.

Learning check

(LC10.10) Why are the different genre variables stored as binary variables (1s and 0s) instead of just listing the `genre` as a column of values like "Action", "Comedy", etc.?

(LC10.11) What complications could come above with us excluding action romance movies? Should we question the results of our hypothesis test? Explain.

Let's now visualize the distributions of `rating` across both levels of `genre`. Think about what type(s) of plot is/are appropriate here before you proceed:

```
ggplot(data = movies_trimmed, aes(x = genre, y = rating)) +
geom_boxplot()
```

We can see that the middle 50% of ratings for "Action" movies is more spread out than that of "Romance" movies in the population. "Romance" has outliers at both the top and bottoms of the scale though. We are initially interested in comparing the mean `rating` across these two groups so a faceted histogram may also be useful:

```
ggplot(data = movies_trimmed, mapping = aes(x = rating)) +
geom_histogram(binwidth = 1, color = "white") +
facet_grid(genre ~ .)
```

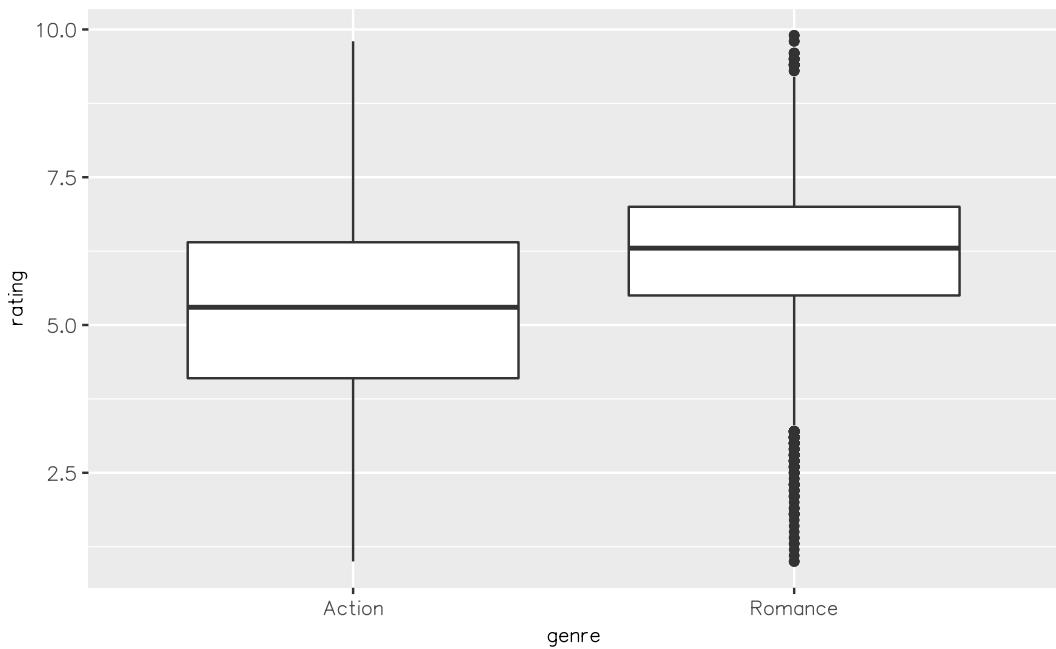


FIGURE 10.4: Rating vs genre in the population

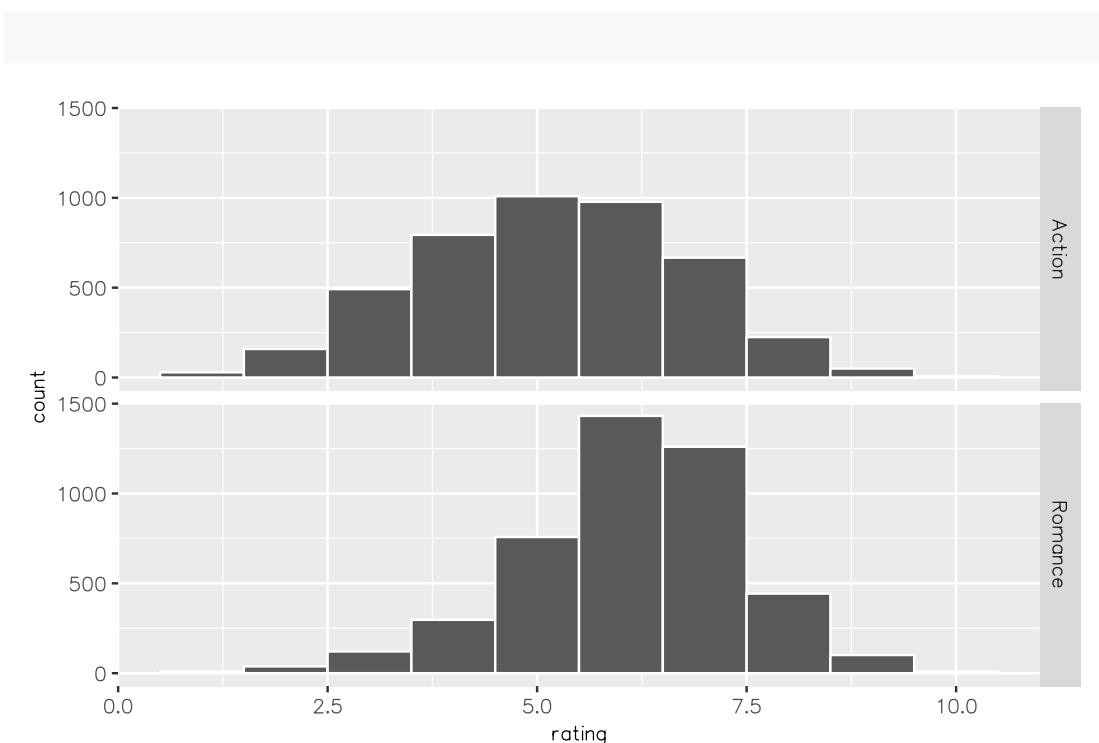


FIGURE 10.5: Faceted histogram of genre vs rating

Important note: Remember that we hardly ever have access to the population values as we do here. This example was used to show how well hypothesis testing procedures using methods like permutation can do at testing hypotheses about population parameters. In nearly all circumstances, we'll be needing to use only a sample of the population to try to infer conclusions about the unknown population parameter values. This example does show a nice relationship between statistics (where data is usually small and more focused on experimental settings) and data science (where data is frequently large and collected without experimental conditions).

10.5.3 Sampling → randomization

We can use hypothesis testing to investigate ways to determine, for example, whether a **treatment** has an effect over a **control** and other ways to statistically analyze if one group performs better than, worse than, or different than another. We are interested here in seeing how we can use a random sample of action movies and a random sample of romance movies from `movies` to determine if a statistical difference exists in the mean ratings of each group.

Learning check

(LC10.12) Define the relevant parameters here in terms of the populations of movies.

In what follows, we'll use the terminology from the “There is Only One Test” diagram discussed in Subsection 10.2.3. Carefully review it with the diagram handy to start to put the pieces together.

10.5.4 Data

Let's select a random sample of 34 action movies and a random sample of 34 romance movies. (The number 34 was chosen somewhat arbitrarily here.)

```
set.seed(2017)
movies_genre_sample <- movies_trimmed %>%
  group_by(genre) %>%
  sample_n(34) %>%
  ungroup()
```

Note the addition of the `ungroup()` function here. This will be useful shortly in allowing us to permute the values of `rating` across `genre`. Our analysis does not work without this `ungroup()` function since the data stays grouped by the levels of `genre` without it. We can now observe the distributions of our two sample ratings for both groups. Remember that these plots should be rough approximations of our population distributions of movie ratings for "Action" and "Romance" in our population of all movies in the `movies` data frame.

```
ggplot(data = movies_genre_sample, aes(x = genre, y = rating)) +
  geom_boxplot()
```

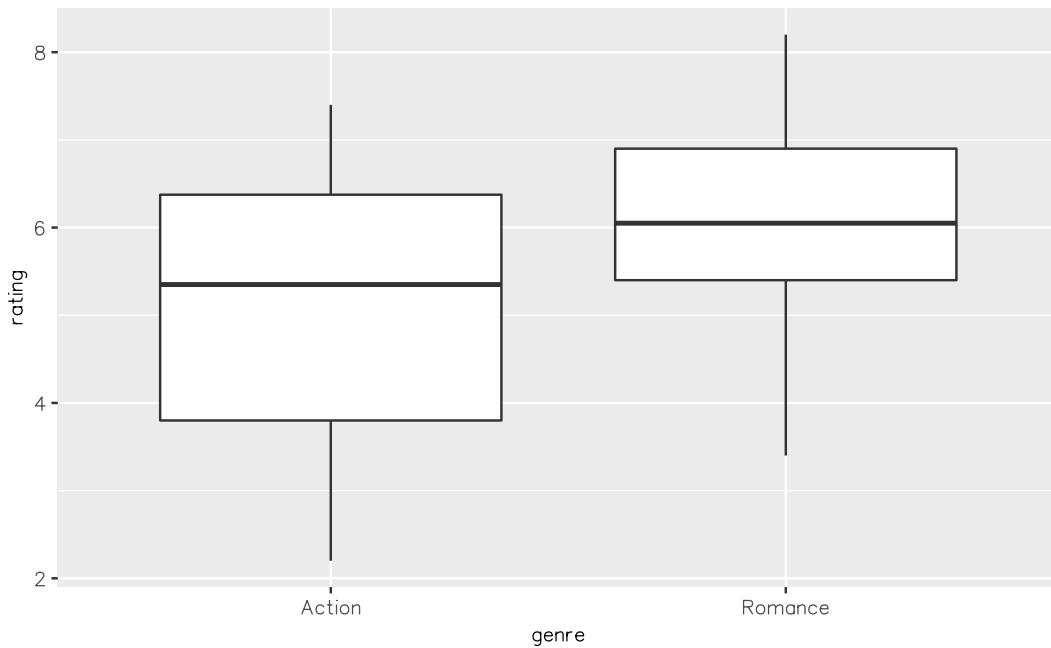


FIGURE 10.6: Genre vs rating for our sample

```
ggplot(data = movies_genre_sample, mapping = aes(x = rating)) +
  geom_histogram(binwidth = 1, color = "white") +
  facet_grid(genre ~ .)
```

Learning check

(LC10.13) What single value could we change to improve the approximation using the sample distribution on the population distribution?

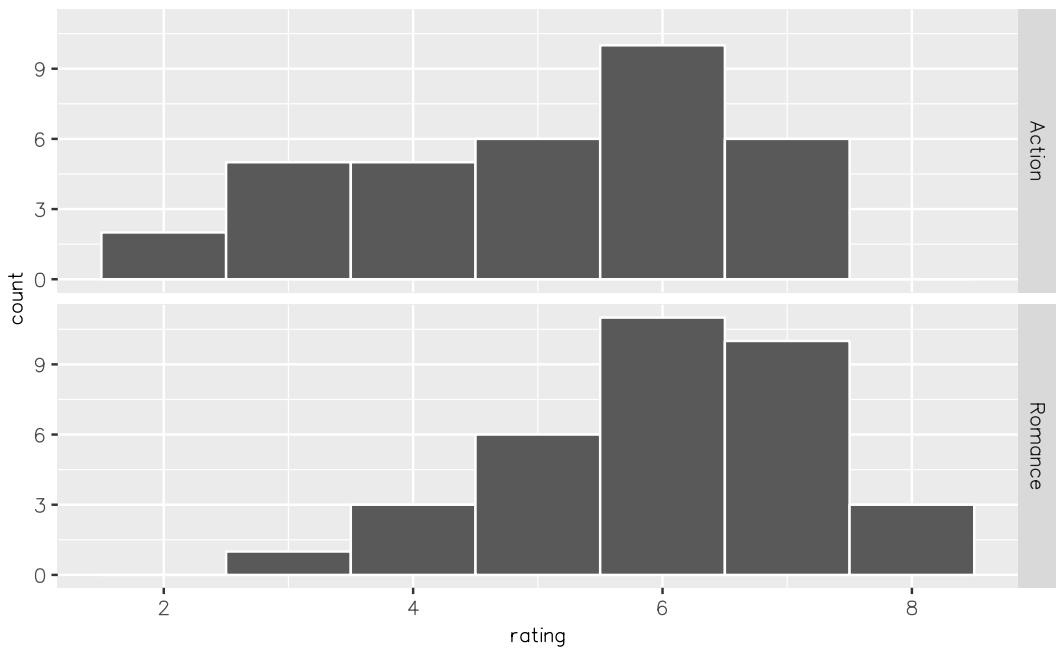


FIGURE 10.7: Genre vs rating for our sample as faceted histogram

Do we have reason to believe, based on the sample distributions of `rating` over the two groups of `genre`, 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, but the histogram isn't as clear. The two groups have somewhat differently shaped distributions but they are both over similar values of `rating`. It's often useful to calculate the mean and standard deviation as well, conditioned on the two levels of the explanatory variable. We do so here to get the mean rating across the "Action" and "Romance" levels.

```
summary_ratings <- movies_genre_sample %>%
  group_by(genre) %>%
  summarize(mean = mean(rating),
            std_dev = sd(rating),
            n = n())
summary_ratings
```

```
# A tibble: 2 x 4
  genre      mean  std Dev    n
  <fct>     <dbl> <dbl> <int>
1 Action     5.8   1.5    10
2 Romance    6.2   1.6    10
```

```

<chr>     <dbl>   <dbl> <int>
1 Action  5.11176 1.48870    34
2 Romance 6.06176 1.14944    34

```

Learning check

(LC10.14) Why did we not specify `na.rm = TRUE` here as we did in Chapter 4?

We see that the sample mean rating for romance movies, \bar{x}_r , is greater than the similar measure for action movies, \bar{x}_a . But is it statistically significantly greater (thus, leading us to conclude that the means are statistically different)? The standard deviation can provide some insight here but with these standard deviations being so similar it's still hard to say for sure.

Learning check

(LC10.15) Why might the standard deviation provide some insight about the means being statistically different or not?

10.5.5 Model of H_0

The hypotheses we specified can also be written in another form to better give us an idea of what we will be simulating to create our null distribution.

- $H_0 : \mu_r - \mu_a = 0$
- $H_a : \mu_r - \mu_a \neq 0$

10.5.6 Test statistic δ

We are, therefore, interested in seeing whether the difference in the sample means, $\bar{x}_r - \bar{x}_a$, is statistically different than 0. We can now come back to our `infer` pipeline for computing our observed statistic. Note the `order` argument that shows the mean value for "Action" being subtracted from the mean value of "Romance".

10.5.7 Observed effect δ^*

```
obs_diff <- movies_genre_sample %>%
  specify(formula = rating ~ genre) %>%
  calculate(stat = "diff in means", order = c("Romance", "Action"))
obs_diff
```

```
# A tibble: 1 × 1
  stat
  <dbl>
1 0.95
```

Our goal next is to figure out a random process with which to simulate the null hypothesis being true. Recall that $H_0 : \mu_r - \mu_a = 0$ corresponds to us assuming that the population means are the same. We would like to assume this is true and perform a random process to `generate()` data in the model of the null hypothesis.

10.5.8 Simulated data

Tactile simulation

Here, with us assuming the two population means are equal ($H_0 : \mu_r - \mu_a = 0$), we can look at this from a tactile point of view by using index cards. There are $n_r = 34$ data elements corresponding to romance movies and $n_a = 34$ for action movies. We can write the 34 ratings from our sample for romance movies on one set of 34 index cards and the 34 ratings for action movies on another set of 34 index cards. (Note that the sample sizes need not be the same.)

The next step is to put the two stacks of index cards together, creating a new set of 68 cards. If we assume that the two population means are equal, we are saying that there is no association between ratings and genre (romance vs action). We can use the index cards to create two **new** stacks for romance and action movies. In creating these two new stacks, we are assigning potentially new values to each of the movies in our sample via the process of permutation. Note that the **new** “romance movie stack” will likely have some of the original action movies in it and likewise for the “action movie stack” including some romance movies from our original set. Since we are assuming that each card is equally likely to have appeared in either one of the stacks this makes sense. First, we must shuffle all the cards thoroughly. After doing so, in this case with equal values of sample sizes, we split the deck in half.

We then calculate the new sample mean rating of the romance deck, and also the new sample mean rating of the action deck. This creates one simulation of the samples that were collected originally. We next want to calculate a statistic from these two samples. Instead of actually doing the calculation using index cards, we can use R as we have before to simulate this process. Let's do this just once and compare the results to what we see in `movies_genre_sample`.

```
movies_genre_sample %>%
  specify(formula = rating ~ genre) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 1) %>%
  calculate(stat = "diff in means", order = c("Romance", "Action"))
```

```
# A tibble: 1 x 1
  stat
  <dbl>
1 0.514706
```

Learning check

(LC10.16) How would the tactile shuffling of index cards change if we had different samples of say 20 action movies and 60 romance movies? Describe each step that would change.

(LC10.17) Why are we taking the difference in the means of the cards in the new shuffled decks?

10.5.9 Distribution of δ under H_0

The `generate()` step completes a permutation sending values of ratings to potentially different values of `genre` from which they originally came. It simulates a shuffling of the ratings between the two levels of `genre` just as we could have done with index cards. We can now proceed in a similar way to what we have done previously with bootstrapping by repeating this process many times to create simulated samples, assuming the null hypothesis is true.

```
generated_samples <- movies_genre_sample %>%
  specify(formula = rating ~ genre) %>%
```

```
hypothesize(null = "independence") %>%
  generate(reps = 5000)
```

A **null distribution** of simulated differences in sample means is created with the specification of `stat = "diff in means"` for the `calculate()` step. Recall that the **null distribution** is similar to the bootstrap distribution we saw in Chapter 9, but remember that it consists of statistics generated assuming the null hypothesis is true, whereas a bootstrap distribution does not make this assumption.

```
null_distribution_two_means <- movies_genre_sample %>%
  specify(formula = rating ~ genre) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 5000) %>%
  calculate(stat = "diff in means", order = c("Romance", "Action"))
```

We can now plot the distribution of these simulated differences in means:

```
null_distribution_two_means %>% visualize()
```

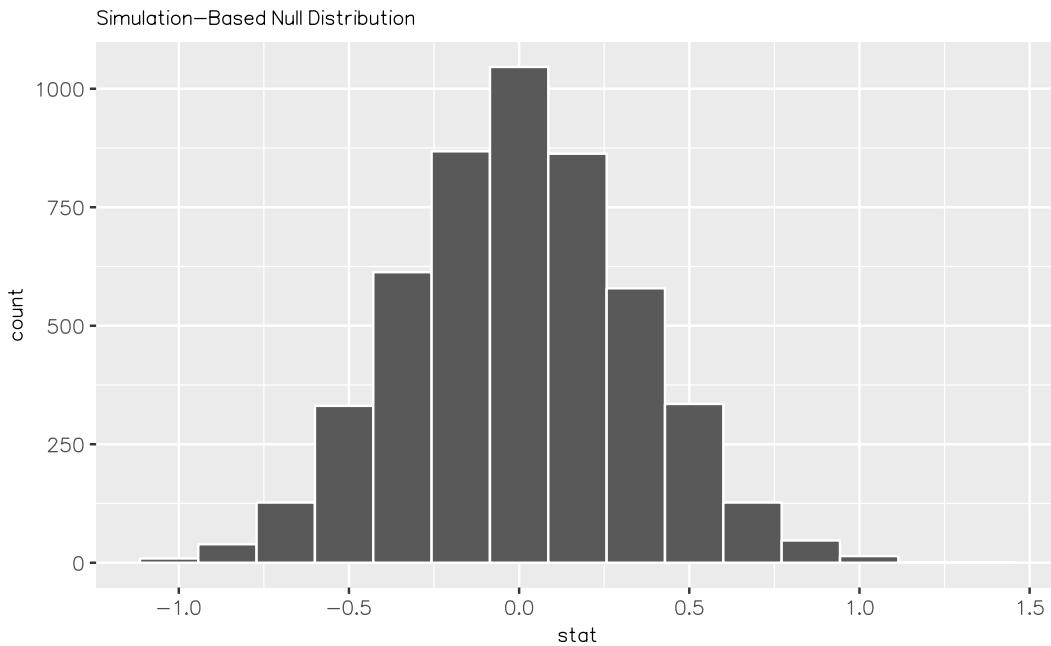


FIGURE 10.8: Simulated differences in means histogram

10.5.10 The p-value

Remember that we are interested in seeing where our observed sample mean difference of 0.95 falls on this null/randomization distribution. We are interested in simply a difference here so “more extreme” corresponds to values in both tails on the distribution. Let’s shade our null distribution to show a visual representation of our p -value:

```
visualize(null_distribution_two_means) +
  shade_p_value(obs_stat = obs_diff, direction = "both")
```

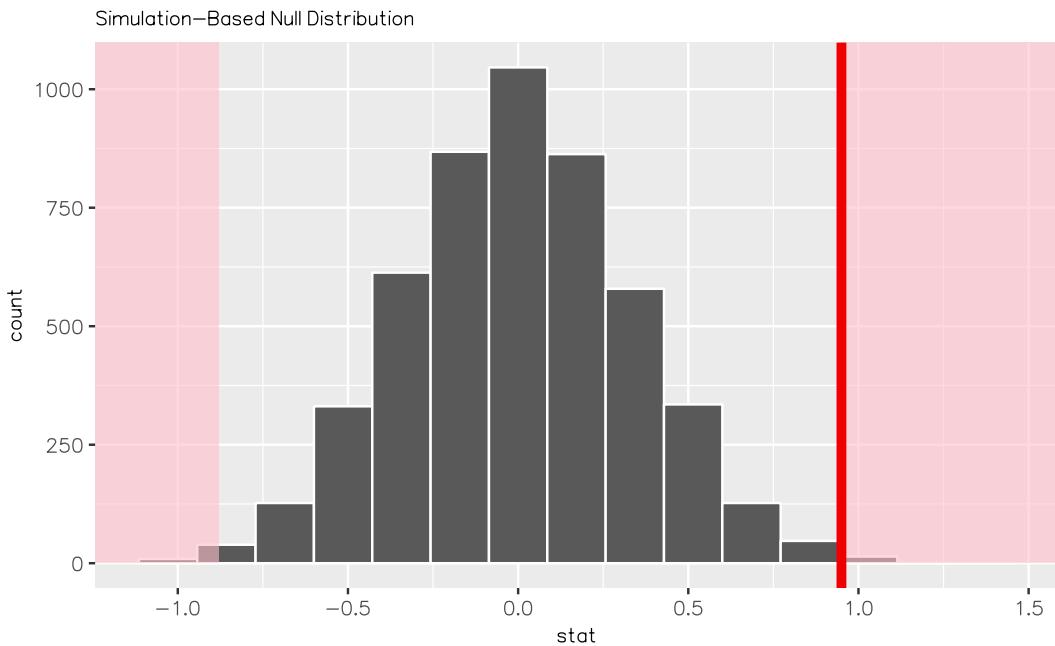


FIGURE 10.9: Shaded histogram to show p-value

Remember that the observed difference in means was 0.95. We have shaded red all values at or above that value and also shaded red those values at or below its negative value (since this is a two-tailed test). By giving `obs_stat = obs_diff` a vertical darker line is also shown at 0.95. To better estimate how large the p -value will be, we also increase the number of bins to 100 here from 20:

```
visualize(null_distribution_two_means, bins = 100) +
  shade_p_value(bins = 100, obs_stat = obs_diff, direction = "both")
```

At this point, it is important to take a guess as to what the p -value may be.

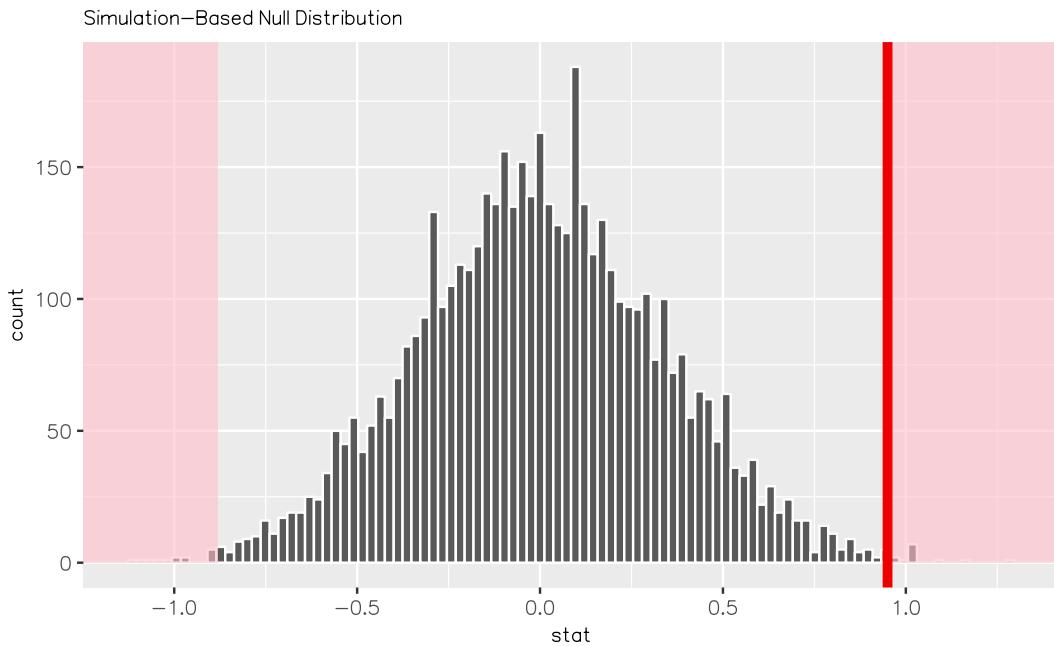


FIGURE 10.10: Histogram with vertical lines corresponding to observed statistic

We can see that there are only a few permuted differences as extreme or more extreme than our observed effect (in both directions). Maybe we guess that this p -value is somewhere around 2%, or maybe 3%, but certainly not 30% or more. Lastly, we calculate the p -value directly using `infer`:

```
pvalue <- null_distribution_two_means %>%
  get_p_value(obs_stat = obs_diff, direction = "both")
pvalue

# A tibble: 1 × 1
  p_value
  <dbl>
1 0.006
```

We have around 0.6% of values as extreme or more extreme than our observed statistic in both directions. Assuming we are using a 5% significance level for α , we have evidence supporting the conclusion that the mean rating for romance movies is different from that of action movies. The next important idea is to better understand just how much higher of a mean rating can we expect the romance movies to have compared to that of action movies.

10.5.11 Corresponding confidence interval

One of the great things about the `infer` pipeline is that going between hypothesis tests and confidence intervals is incredibly simple. To create a null distribution, we ran

```
null_distribution_two_means <- movies_genre_sample %>%
  specify(formula = rating ~ genre) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 5000) %>%
  calculate(stat = "diff in means", order = c("Romance", "Action"))
```

To get the corresponding bootstrap distribution with which we can compute a confidence interval, we can just remove or comment out the `hypothesize()` step since we are no longer assuming the null hypothesis is true when we bootstrap:

```
percentile_ci_two_means <- movies_genre_sample %>%
  specify(formula = rating ~ genre) %>%
#  hypothesize(null = "independence") %>%
  generate(reps = 5000) %>%
  calculate(stat = "diff in means", order = c("Romance", "Action")) %>%
  get_ci()
```

Setting `type = "bootstrap"` in `generate()`.

Note that we didn't originally set `type` in `generate()` but it was automatically set as `type = "permute"` based on the set-up of the problem. When we switch to build a confidence interval instead we change to `type = "bootstrap"` in `generate()` and we are giving a message here to confirm that decision.

```
percentile_ci_two_means
```

```
# A tibble: 1 x 2
`2.5%` `97.5%` 
<dbl>    <dbl>
1 0.332718 1.59288
```

Thus, we can expect the true mean of Romance movies on IMDB to have a rating 0.333 to 1.593 points higher than that of Action movies. Remember that this is based on bootstrapping using `movies_genre_sample` as our original sample and the confidence interval process being 95% reliable.

Learning check

(LC10.18) 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?

(LC10.19) 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?

(LC10.20) 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.

(LC10.21) 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?

(LC10.22) 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.

(LC10.23) What is the value of the *p*-value for the hypothesis test comparing the mean rating of romance to action movies?

(LC10.24) 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 When inference is not needed

We've now walked through a couple of different examples of how to use the `infer` package to conduct hypothesis tests. Whenever possible we always started with some exploratory data analysis. It's good to remember that there are cases where you need not perform a rigorous statistical inference. An important and time-saving skill is to **ALWAYS** do exploratory data analysis using `dplyr` and `ggplot2` before thinking about running a hypothesis test. As a

beginner to statistical inference, this helps you to get an intuition as to when statistical significance may be found. As a seasoned practitioner, this helps to make sure you can make a sophisticated guess as to statistical significance before conducting the test.

Let's look at such an example selecting a sample of flights traveling to Boston and to San Francisco from New York City in the `flights` data frame in the `nycflights13` package. (We will remove flights with missing data first using `na.omit` and then sample 100 flights going to each of the two airports.)

```
bos_sfo <- flights %>%
  na.omit() %>%
  filter(dest %in% c("BOS", "SFO")) %>%
  group_by(dest) %>%
  sample_n(100)
```

Suppose we were interested in seeing if the `air_time` to SFO in San Francisco was statistically greater than the `air_time` to BOS in Boston. As suggested, let's begin with some exploratory data analysis to get a sense for how the two variables of `air_time` and `dest` relate for these two destination airports:

```
bos_sfo_summary <- bos_sfo %>% group_by(dest) %>%
  summarize(mean_time = mean(air_time),
            sd_time = sd(air_time))
bos_sfo_summary
```

```
# A tibble: 2 x 3
  dest   mean_time   sd_time
  <chr>     <dbl>     <dbl>
1 BOS       39.14    4.78639
2 SFO      347.43   17.2646
```

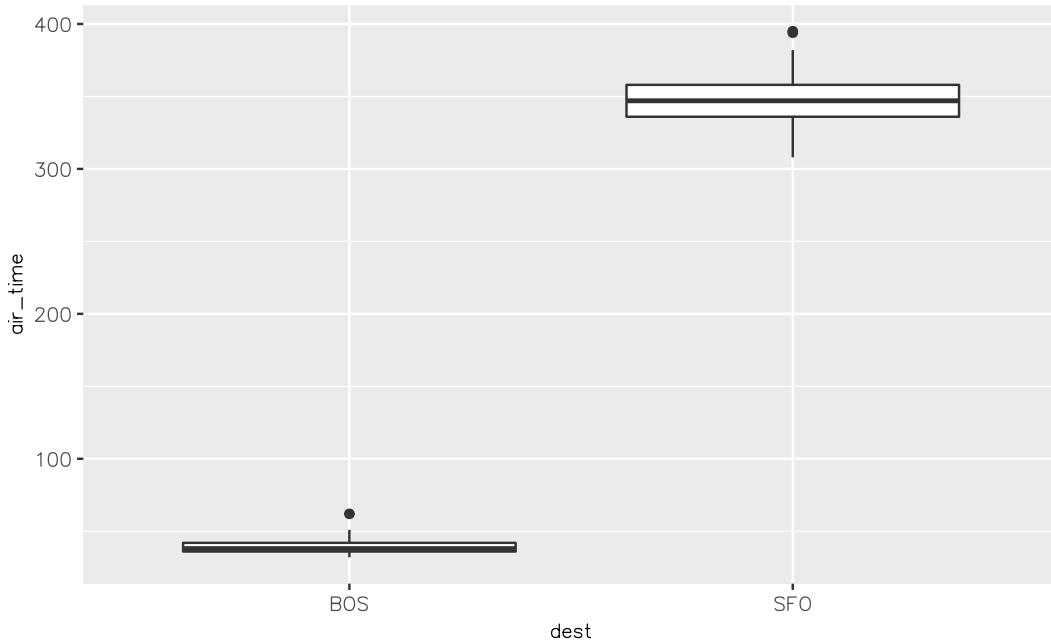
Looking at these results, we can clearly see that SFO `air_time` is much larger than BOS `air_time`. The standard deviation is also extremely informative here.

Learning check

(LC10.25) 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.

To further understand just how different the `air_time` variable is for BOS and SFO, let's look at a boxplot:

```
ggplot(data = bos_sfo, mapping = aes(x = dest, y = air_time)) +  
  geom_boxplot()
```



Since there is no overlap at all in terms of the `air_time` values for the two destination airports, we can conclude that the `air_time` for San Francisco flights is statistically greater (at any level of significance) than the `air_time` for Boston flights. This is a clear example of not needing to do anything more than some simple exploratory data analysis with descriptive statistics and data visualization to get an appropriate inferential conclusion. This is one reason why you should **ALWAYS** investigate the sample data first using `dplyr` and `ggplot2` via exploratory data analysis (EDA).

As you get more and more practice with hypothesis testing, you'll be better able to determine in many cases whether or not the results will be statistically significant. There are circumstances where it is difficult to tell, but you should always try to make a guess FIRST about significance after you have completed your data exploration and before you actually begin the inferential techniques.

10.6.2 Problems with p-values

One of the inherent problems that comes from this “sneaking a peak” process using EDA is that the user must remain vigilant at conducting the test itself as needed. The hunt for statistically significant values is one that science has recently tried to combat. Done incorrectly and tirelessly, this is called “p-hacking” and is incredibly dangerous. Instead of reporting results that were not significant, p-hackers look to find significant results and then only report those. This sometimes involves changing the research question of interest to better meet the goals of statistically significant results that hopefully lead to publication.

Note: We personally as authors are in favor of reporting hypothesis tests but only with their corresponding confidence intervals as needed. And we also encourage our readers to take the pledge to not p-hack their results, but rather to be transparent about tests that did not bring statistical significance and discuss potential reasons for these failings as they can. This helps others to further test these results with new experiments.

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 -values³
2. What a nerdy debate about p-values shows about science - and how to fix it⁴
3. Statisticians issue warning over misuse of P values⁵
4. You Can't Trust What You Read About Nutrition⁶
5. A Litany of Problems with p-values⁷

Point 5 advocates for using Bayesian inference instead of the “frequentist” methods we’ve shown throughout this book. We highly recommend that you do dig further into Bayesian methods if you’d like to expand your knowledge beyond this book, but they are beyond the scope of this book.

³https://en.wikipedia.org/wiki/Misunderstandings_of_p-values

⁴<https://www.vox.com/science-and-health/2017/7/31/16021654/p-values-statistical-significance-redefine-0005>

⁵<https://www.nature.com/news/statisticians-issue-warning-over-misuse-of-p-values-1.19503>

⁶<https://fivethirtyeight.com/features/you-can-t-trust-what-you-read-about-nutrition/>

⁷<http://www.fharrell.com/post/pval-litany/>

10.6.3 Comparing confidence intervals and hypothesis tests

To follow-up on the previous sections and this and the previous chapter, it's important to understand what information hypothesis tests and confidence intervals provide. In short, hypothesis tests are aligned with statistical significance whereas confidence intervals are more aligned with practical significance. Like we saw in the movie ratings example, the hypothesis test told us that we had evidence that action and romance movies had a different mean rating. This is statistical significance.

We also found a range of plausible values for what the mean difference in ratings for action versus romance movies we'd expect to see. This gave us some practical evidence that we could use in later analyses. Thus, confidence intervals are the recommended go-to method since they can be both used to check for a statistical difference between a quantitative calculation on two variables (by checking for the inclusion of 0 in the interval) and also for providing some practicality as well.

10.6.4 Summary table

In this chapter, we performed both tactile and virtual simulations of permutation to infer about hypotheses on unknown parameters. We also presented a case study of permuting in a real-life situation: movie ratings of action and romance movies. We used the difference in sample proportions $\hat{p}_1 - \hat{p}_2$ to estimate the difference in population proportions $p_1 - p_2$. We similarly used the difference in sample means $\bar{x}_1 - \bar{x}_2$ to estimate the difference in population mean $\mu_1 - \mu_2$. Here we use 1 and 2 to represent two different groups, but one could use other subscripts as needed to denote the groups. Let's review these and others one more time 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	$\hat{\mu}$ or \bar{x}
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	Sample regression slope	$\hat{\beta}_1$ or b_1
6	Population regression intercept	β_0	Sample regression intercept	$\hat{\beta}_0$ or b_0

We'll cover the remaining scenario as follows, using the terminology, notation, and definitions related to sampling you saw in Section 8.3:

- 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.

10.6.5 Building theory-based methods using computation

As a point of reference, we will now discuss the traditional theory-based way to conduct the hypothesis test for determining if there is a statistically significant difference in the sample mean rating of Action movies versus Romance movies. This method and ones like it work very well when the assumptions are met in order to run the test. They are based on probability models and distributions such as the normal and t -distributions. Recall that we briefly discussed the normal distribution in Subsection 9.8.2. We will extend that to look at the t -distribution here.

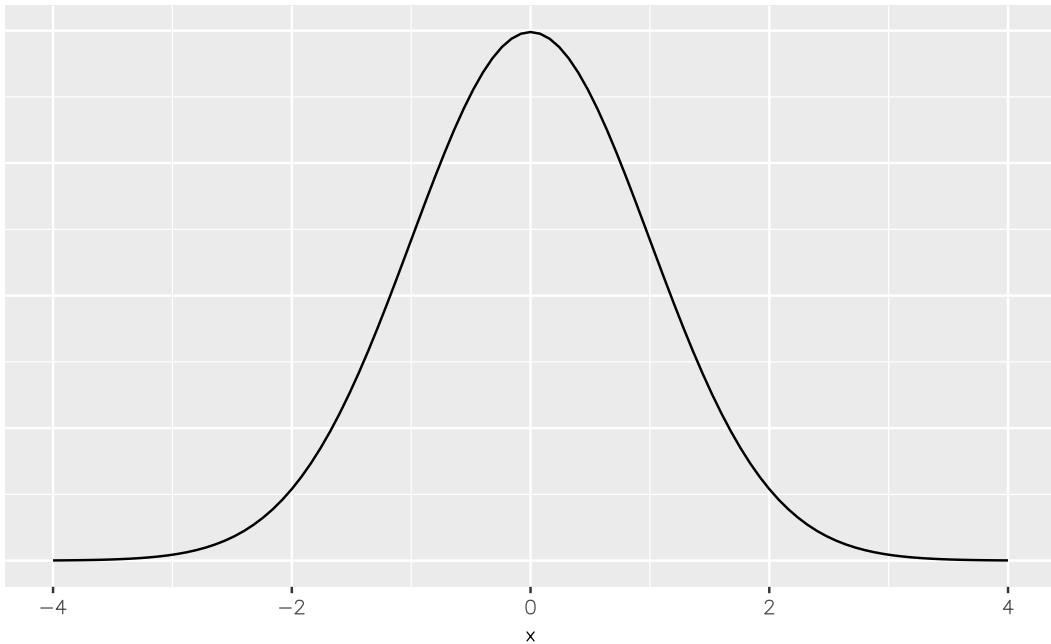
These traditional methods have been used for many decades back to the time when researchers didn't have access to computers that could run 5000 simulations in a few seconds. They had to base their methods on probability theory instead. Many fields and researchers continue to use these methods and that is the biggest reason for their inclusion here. It's important to remember that a hypothesis test based on the t distribution (commonly called a t -test) or a test based on the normal distribution (commonly called a z -test) is really just an approximation of what you have seen in this chapter already using simulation and randomization. The focus here is on understanding how the shape of the t -curve comes about without digging much into the mathematical underpinnings.

Example: t -test for two independent samples

What is commonly done in statistics is the process of standardization. What this entails is calculating the mean and standard deviation of a variable. Then you subtract the mean from each value of your variable and divide by the standard deviation. The most common standardization is known as the z -score. The formula for a z -score is

$$Z = \frac{x - \mu}{\sigma},$$

where x represent the value of a variable, μ represents the mean of the variable, and σ represents the standard deviation of the variable. 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. They have the common, bell-shaped pattern seen below.



Recall, that we hardly ever know the mean and standard deviation of the population of interest. This is almost always the case when considering the means of two independent groups. To help account for us not knowing the population parameter values, we can use the sample statistics instead, but this comes with a bit of a price in terms of complexity.

Another form of standardization occurs when we need to use the sample standard deviations as estimates for the unknown population standard deviations. This standardization is often called the *t*-score. For the two independent samples case like what we have for comparing action movies to romance movies, the formula is

$$T = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

There is a lot to try to unpack here.

- \bar{x}_1 is the sample mean response of the first group
- \bar{x}_2 is the sample mean response of the second group
- μ_1 is the population mean response of the first group
- μ_2 is the population mean response of the second group
- s_1 is the sample standard deviation of the response of the first group
- s_2 is the sample standard deviation of the response of the second group
- n_1 is the sample size of the first group
- n_2 is the sample size of the second group

Assuming that the null hypothesis is true ($H_0 : \mu_1 - \mu_2 = 0$), T is said to be distributed following a t distribution with degrees of freedom “roughly equal to” the smaller value of $n_1 + n_2 - 2$. The “degrees of freedom” can be thought of measuring how different the t distribution will be as compared to a normal distribution. The “roughly equal to” here corresponds to the formula for degrees of freedom being 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. Small sample sizes lead to small degrees of freedom and, thus, t distributions that have more values in the tails of their distributions. Large sample sizes lead to large degrees of freedom and, thus, t distributions that closely align with the standard normal, bell-shaped curve.

So, assuming H_0 is true, our formula simplifies a bit:

$$T = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}.$$

We have already built an approximation for what we think the distribution of $\delta = \bar{x}_1 - \bar{x}_2$ looks like using randomization above. Recall this distribution:

```
ggplot(data = null_distribution_two_means, aes(x = stat)) +
  geom_histogram(color = "white", bins = 20)
```

The `infer` package also includes some built-in theory-based statistics as well, so instead of going through the process of trying to transform the difference into a standardized form, we can just provide a different value for `stat` in `calculate()`. Recall the `generated_samples` data frame created via:

```
generated_samples <- movies_genre_sample %>%
  specify(formula = rating ~ genre) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 5000)
```

We can now created a null distribution of t statistics:

```
null_distribution_t <- generated_samples %>%
  calculate(stat = "t", order = c("Romance", "Action"))
null_distribution_t %>% visualize()
```

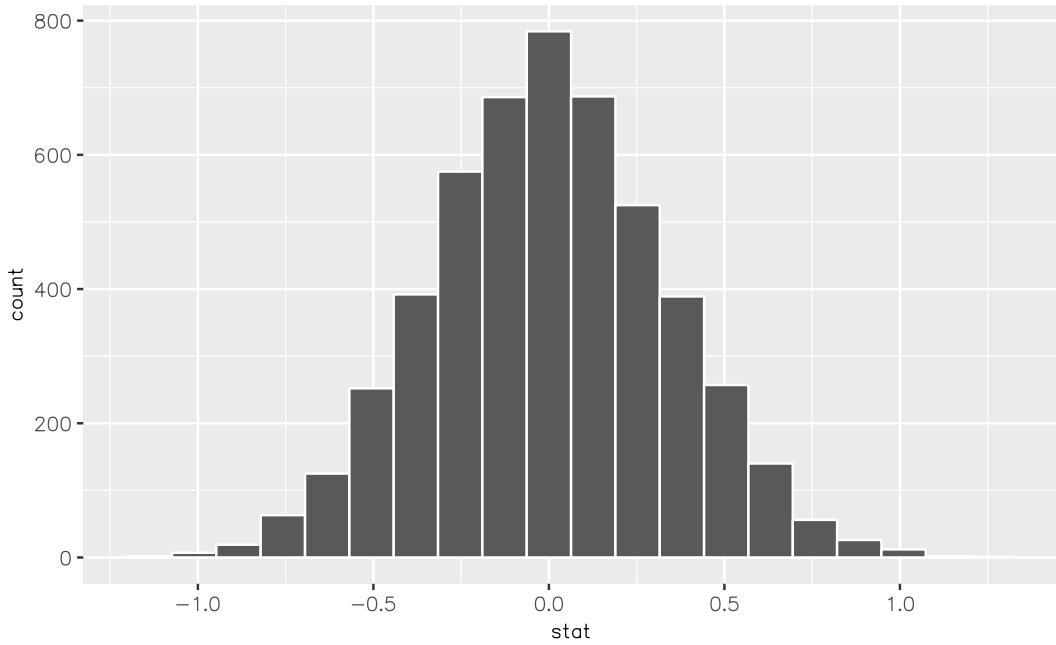
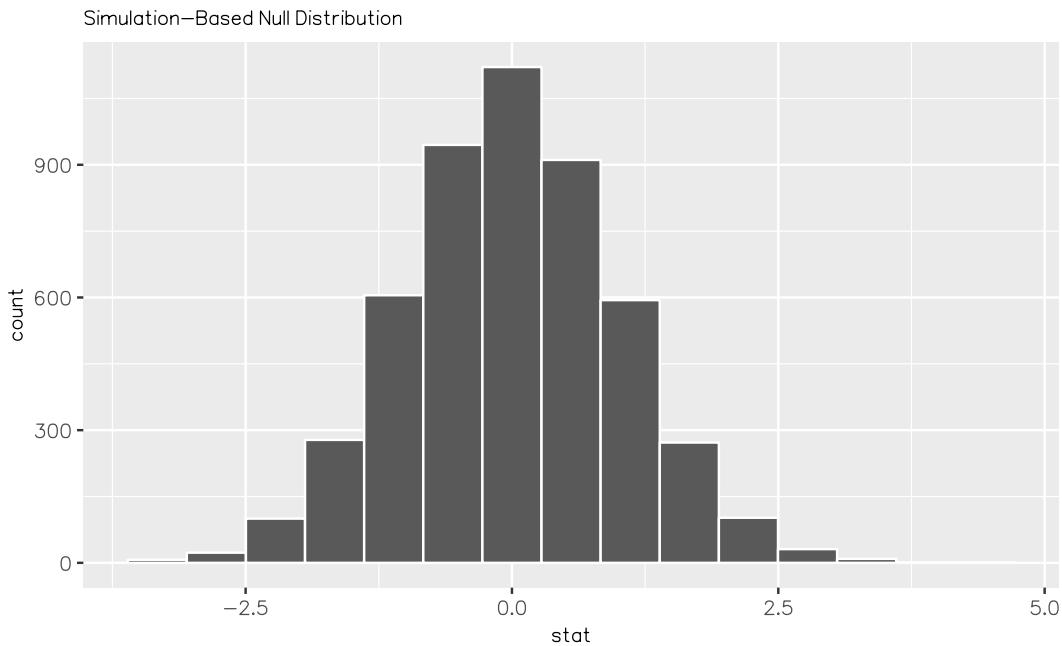


FIGURE 10.11: Simulated differences in means histogram

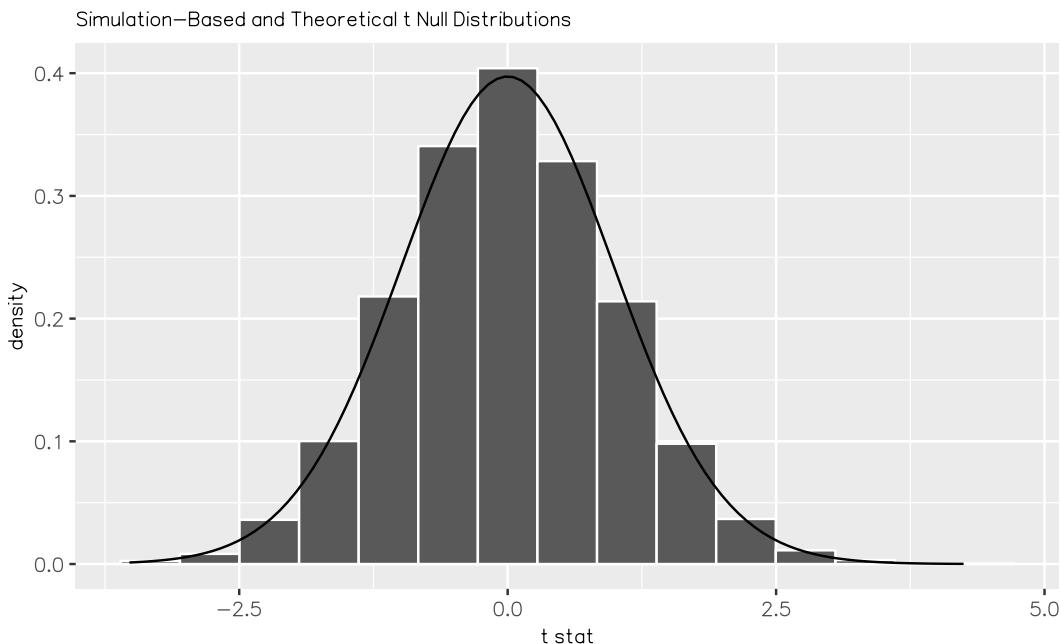


We see that the shape of this `stat = "t"` distribution is the same as that of `stat = "diff in means"`. The scale has changed though with the t values having less spread than the difference in means.

A traditional t -test doesn't look at this simulated distribution, but instead it

looks at the t -curve with degrees of freedom equal to 62.029. This calculation is based on the complicated formula referenced above. (Note that this value is pretty close to the “roughly equal to” number of $34 + 34 - 2 = 66$.) We can overlay this distribution over the top of our permuted t statistics using the `method = "both"` setting in `visualize()`.

```
null_distribution_t %>%
  visualize(method = "both")
```



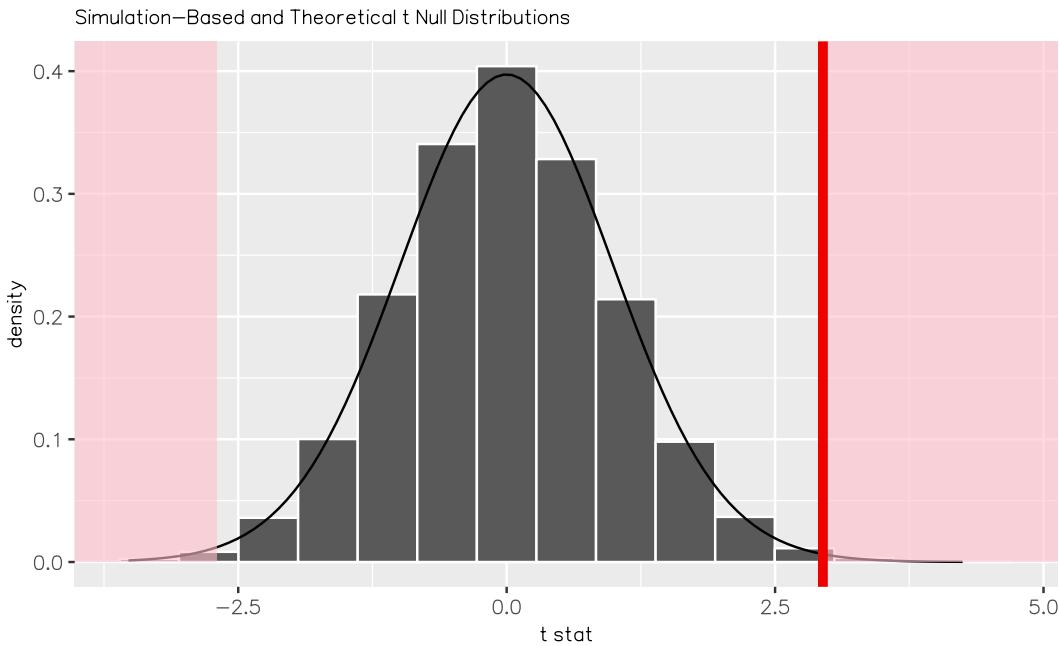
We can see that the curve does a good job of approximating the randomization distribution here. (More on when to expect for this to be the case when we discuss conditions for the t -test in a bit.) To calculate the p -value in this case, we need to figure out how much of the total area under the t -curve is at our observed T -statistic or more, plus also adding the area under the curve at the negative value of the observed T -statistic or below. (Remember this is a two-tailed test so we are looking for a difference—values in the tails of either direction.) Just as we converted all of the simulated values to T -statistics, we must also do so for our observed effect δ^* :

```
obs_t <- movies_genre_sample %>%
  specify(formula = rating ~ genre) %>%
  calculate(stat = "t", order = c("Romance", "Action"))
```

So graphically we are interested in finding the percentage of values that are at or above $\text{obs_t} = 2.945$ or at or below $-\text{obs_t} = -2.945$.

```
visualize(null_distribution_t, method = "both") +
  shade_p_value(obs_stat = obs_t, direction = "both")
```

Warning: Check to make sure the conditions have been met
for the theoretical method. `{infer}` currently does not check
these for you.



As we might have expected with this just being a standardization of the difference in means statistic that produced a small p -value, we also have a very small one here.

Conditions for t-test

The `infer` package does not automatically check conditions for the theoretical methods to work and this warning was given when we used `method = "both"`. In order for the results of the t -test to be valid, three conditions must be met:

1. Independent observations in both samples
2. Nearly normal populations OR large sample sizes ($n \geq 30$)
3. Independently selected samples

Condition 1: This is met since we sampled at random using R from our population.

Condition 2: Recall from Figure 10.5, that we know how the populations are distributed. Both of them are close to normally distributed. If we are a little concerned about this assumption, we also do have samples of size larger than 30 ($n_1 = n_2 = 34$).

Condition 3: This is met since there is no natural pairing of a movie in the Action group to a movie in the Romance group.

Since all three conditions are met, we can be reasonably certain that the theory-based test will match the results of the randomization-based test using shuffling. Remember that theory-based tests can produce some incorrect results in these assumptions are not carefully checked. The only assumption for randomization and computational-based methods is that the sample is selected at random. They are our preference and we strongly believe they should be yours as well, but it's important to also see how the theory-based tests can be done and used as an approximation for the computational techniques until at least more researchers are using these techniques that utilize the power of computers.

10.6.6 Additional resources

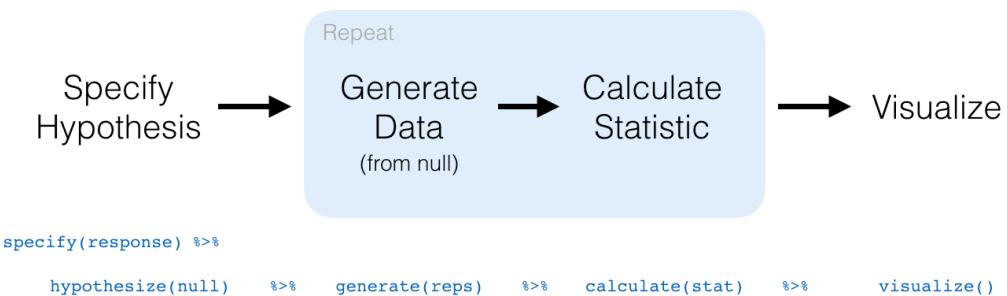
An R script file of all R code used in this chapter is available here⁸.

10.6.7 What's to come

We conclude by showing the `infer` pipeline diagram. In Chapter 11, we'll come back to regression and see how the ideas covered in Chapter 9 and this chapter can help in understanding the significance of predictors in modeling.

⁸[scripts/10-hypothesis-testing.R](#)

Hypothesis Test in `infer`



11

Inference for Regression

In preparation for our first print edition to be published by CRC Press in Fall 2019, we're remodeling this chapter a bit. Don't expect major changes in content, but rather only minor changes in presentation. Our remodeling will be complete and available online at ModernDive.com¹ by early Summer 2019!

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(ggplot2)
library(dplyr)
library(moderndive)
library(infer)
library(gapminder)
library(ISLR)
```

11.1 Simulation-based Inference for Regression

We can also use the concept of permuting to determine the standard error of our null distribution and conduct a hypothesis test for a population slope. Let's

go back to our example on teacher evaluations Chapters 6 and 7. We'll begin in the basic regression setting to test to see if we have evidence that a statistically significant *positive* relationship exists between teaching and beauty scores for the University of Texas professors. As we did in Chapter 6, teaching `score` will act as our outcome variable and `bty_avg` will be our explanatory variable. We will set up this hypothesis testing process as we have each before via the “There is Only One Test” diagram in Figure 10.1 using the `infer` package.

11.1.1 Data

Our data is stored in `evals` and we are focused on the measurements of the `score` and `bty_avg` variables there. Note that we don't choose a subset of variables here since we will `specify()` the variables of interest using `infer`.

```
evals %>%  
  specify(score ~ bty_avg)
```

```
Response: score (numeric)  
Explanatory: bty_avg (numeric)  
# A tibble: 463 x 2  
  score bty_avg  
  <dbl>    <dbl>  
1 4.7     5  
2 4.100   5  
3 3.9     5  
4 4.8     5  
5 4.600   3  
6 4.3     3  
7 2.8     3  
8 4.100   3.333  
9 3.4     3.333  
10 4.5    3.16700  
# ... with 453 more rows
```

11.1.2 Test statistic δ

Our test statistic here is the sample slope coefficient that we denote with b_1 .

11.1.3 Observed effect δ^*

We can use the `specify() %>% calculate()` shortcut here to determine the slope value seen in our observed data:

```
slope_obs <- evals %>%
  specify(score ~ bty_avg) %>%
  calculate(stat = "slope")
```

The calculated slope value from our observed sample is $b_1 = 0.067$.

11.1.4 Model of H_0

We are looking to see if a positive relationship exists so $H_A : \beta_1 > 0$. Our null hypothesis is always in terms of equality so we have $H_0 : \beta_1 = 0$. In other words, when we assume the null hypothesis is true, we are assuming there is NOT a linear relationship between teaching and beauty scores for University of Texas professors.

11.1.5 Simulated data

Now to simulate the null hypothesis being true and recreating how our sample was created, we need to think about what it means for β_1 to be zero. If $\beta_1 = 0$, we said above that there is no relationship between the teaching and beauty scores. If there is no relationship, then any one of the teaching score values could have just as likely occurred with any of the other beauty score values instead of the one that it actually did fall with. We, therefore, have another example of permuting in our simulating of data under the null hypothesis.

Tactile simulation

We could use a deck of 926 note cards to create a tactile simulation of this permuting process. We would write the 463 different values of beauty scores on each of the 463 cards, one per card. We would then do the same thing for the 463 teaching scores putting them on one per card.

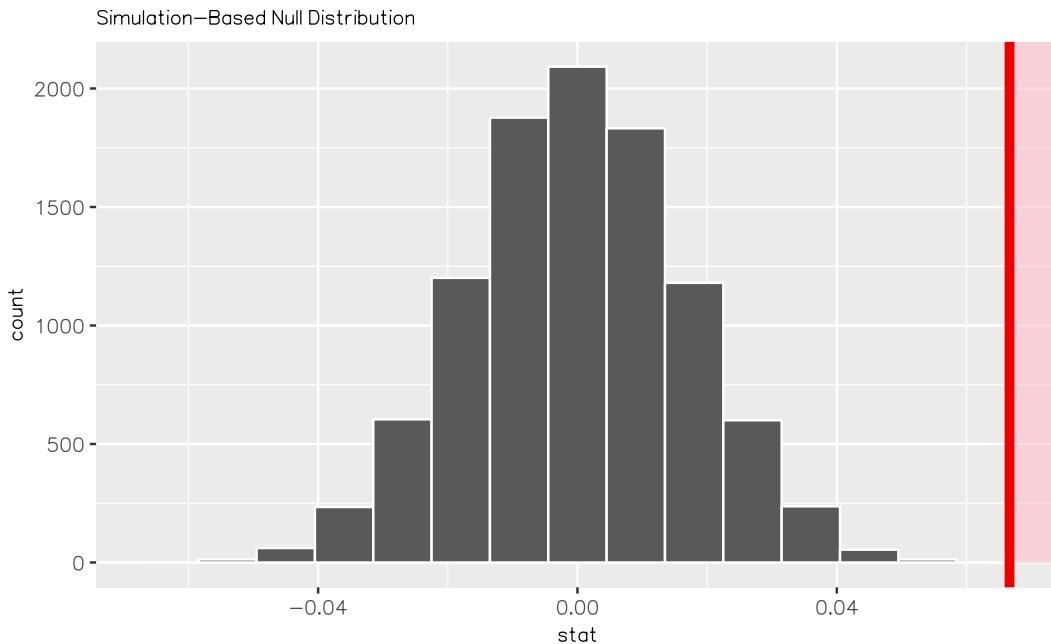
Next, we would lay out each of the 463 beauty score cards and we would shuffle the teaching score deck. Then, after shuffling the deck well, we would disperse the cards one per each one of the beauty score cards. We would then enter these new values in for teaching score and compute a sample slope based on this permuting. We could repeat this process many times, keeping track of our sample slope after each shuffle.

11.1.6 Distribution of δ under H_0

We can build our null distribution in much the same way we did in Chapter 10 using the `generate()` and `calculate()` functions. Note also the addition of the `hypothesize()` function, which lets `generate()` know to perform the permuting instead of bootstrapping.

```
null_slope_distn <- evals %>%
  specify(score ~ bty_avg) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 10000) %>%
  calculate(stat = "slope")
```

```
null_slope_distn %>%
  visualize(obs_stat = slope_obs, direction = "greater")
```



In viewing the distribution above with shading to the right of our observed slope value of 0.067, we can see that we expect the p-value to be quite small. Let's calculate it next using a similar syntax to what was done with `visualize()`.

11.1.7 The p-value

```
null_slope_distrn %>%
  get_pvalue(obs_stat = slope_obs, direction = "greater")  
  
# A tibble: 1 × 1
  p_value
  <dbl>
1      0
```

Since 0.067 falls far to the right of this plot beyond where any of the histogram bins have data, we can say that we have a *p*-value of 0. We, thus, have evidence to reject the null hypothesis in support of there being a positive association between the beauty score and teaching score of University of Texas faculty members.

Learning check

(LC11.1) 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.

11.2 Bootstrapping for the regression slope

With the *p*-value calculated as 0 in the hypothesis test above, we can next determine just how strong of a positive slope value we might expect between the variables of teaching score and beauty score (`bty_avg`) for University of Texas faculty. Recall the `infer` pipeline above to compute the null distribution. Recall that this assumes the null hypothesis is true that there is no relationship between teaching score and beauty score using the `hypothesize()` function.

```
null_slope_distrn <- evals %>%
  specify(score ~ bty_avg) %>%
```

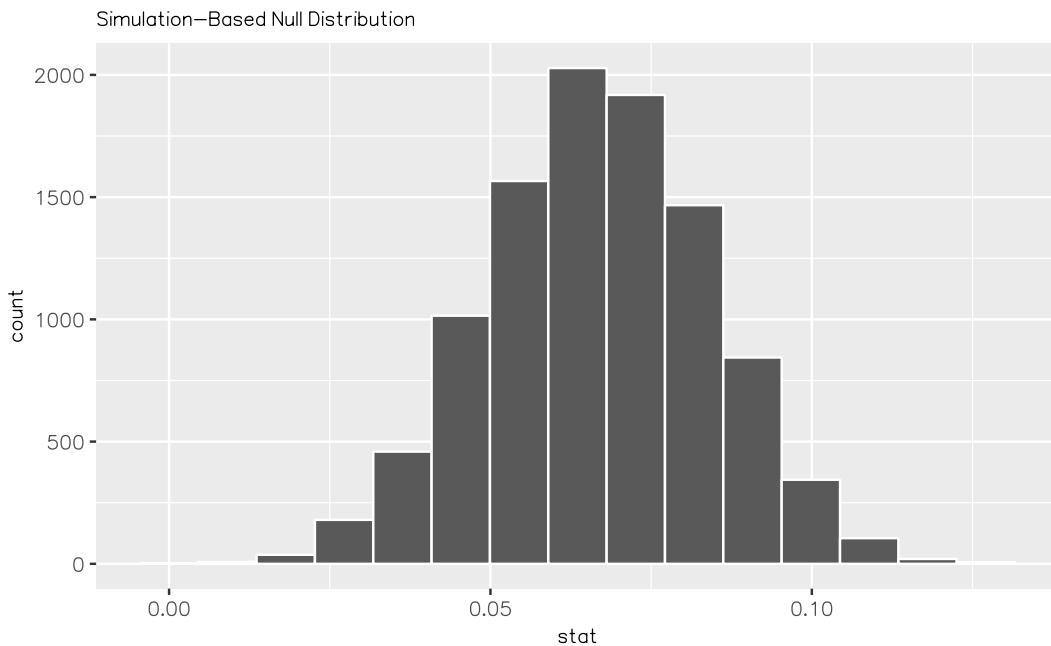
```
hypothesize(null = "independence") %>%
generate(reps = 10000, type = "permute") %>%
calculate(stat = "slope")
```

To further reinforce the process being done in the pipeline, we've added the `type` argument to `generate()`. This is automatically added based on the entries for `specify()` and `hypothesize()` but it provides a useful way to check to make sure `generate()` is creating the samples in the desired way. In this case, we permuted the values of one variable across the values of the other 10,000 times and calculated a "slope" coefficient for each of these 10,000 generated samples.

If instead we'd like to get a range of plausible values for the true slope value, we can use the process of bootstrapping:

```
bootstrap_slope_distn <- evals %>%
specify(score ~ bty_avg) %>%
generate(reps = 10000, type = "bootstrap") %>%
calculate(stat = "slope")
```

```
bootstrap_slope_distn %>% visualize()
```



Next we can use the `get_ci()` function to determine the confidence interval.

Let's do this in two different ways obtaining 99% confidence intervals. Remember that these denote a range of plausible values for an unknown true population slope parameter regressing teaching score on beauty score.

```
percentile_slope_ci <- bootstrap_slope_distr %>%
  get_ci(level = 0.99, type = "percentile")
percentile_slope_ci
```

```
# A tibble: 1 x 2
  `0.5%` `99.5%`
  <dbl>    <dbl>
1 0.0228716 0.110240
```

```
se_slope_ci <- bootstrap_slope_distr %>%
  get_ci(level = 0.99, type = "se", point_estimate = slope_obs)
se_slope_ci
```

```
# A tibble: 1 x 2
  lower     upper
  <dbl>    <dbl>
1 0.0219981 0.111276
```

With the bootstrap distribution being close to symmetric, it makes sense that the two resulting confidence intervals are similar.

11.3 Inference for multiple regression

11.3.1 Refresher: Professor evaluations data

Let's revisit the professor evaluations data that we analyzed using multiple regression with one numerical and one categorical predictor. In particular

- y : outcome variable of instructor evaluation `score`
- predictor variables
 - x_1 : numerical explanatory/predictor variable of `age`
 - x_2 : categorical explanatory/predictor variable of `gender`

```

library(ggplot2)
library(dplyr)
library(moderndive)

evals_multiple <- evals %>%
  select(score, ethnicity, gender, language, age, bty_avg, rank)

```

First, recall that we had two competing potential models to explain professors' teaching scores:

1. Model 1: No interaction term. i.e. both male and female profs have the same slope describing the associated effect of age on teaching score
2. Model 2: Includes an interaction term. i.e. we allow for male and female profs to have different slopes describing the associated effect of age on teaching score

11.3.2 Refresher: Visualizations

Recall the plots we made for both these models:

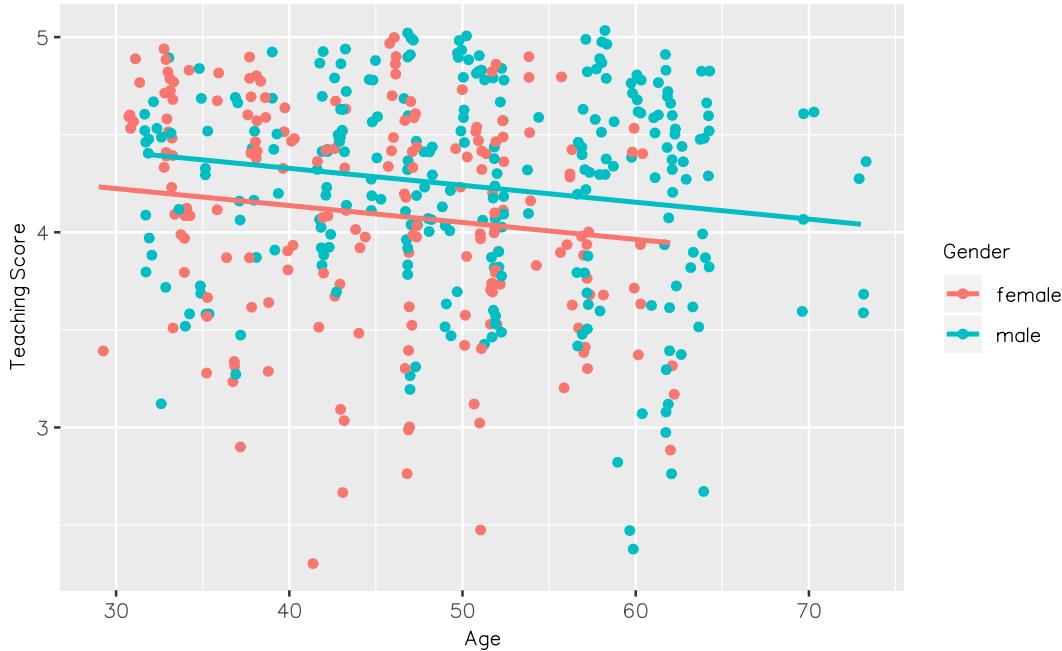
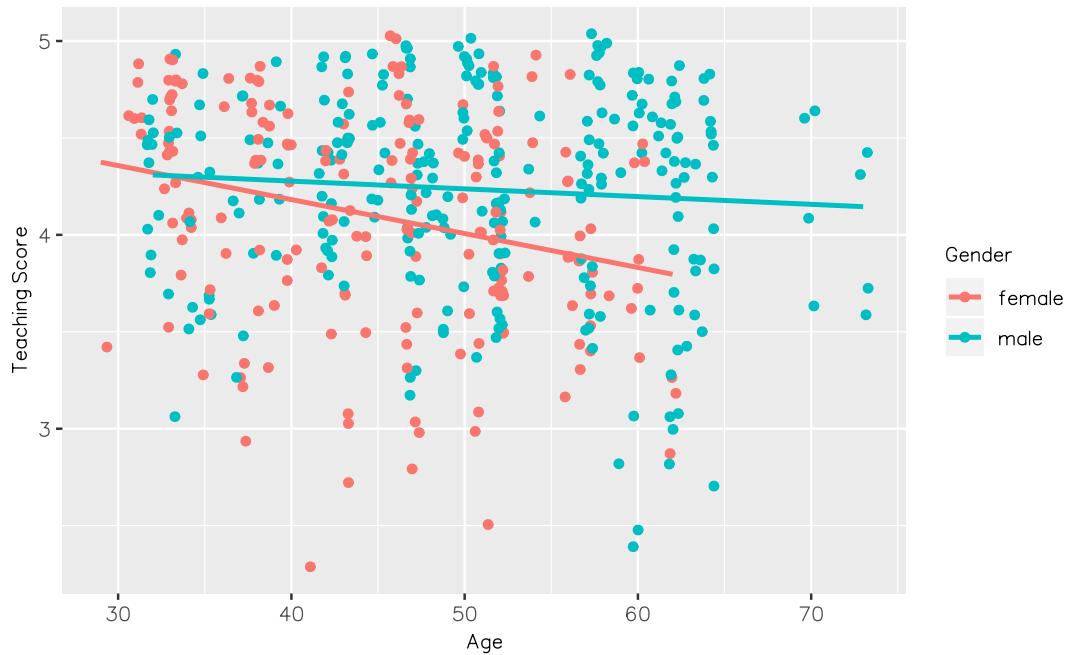


FIGURE 11.1: Model 1: no interaction effect included

**FIGURE 11.2:** Model 2: interaction effect included

11.3.3 Refresher: Regression tables

Last, let's recall the regressions we fit. First, the regression with no interaction effect: note the use of `+` in the formula in Table 11.1.

```
score_model_2 <- lm(score ~ age + gender, data = evals_multiple)
get_regression_table(score_model_2)
```

TABLE 11.1: Model 1: Regression table with no interaction effect included

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

Second, the regression with an interaction effect: note the use of `*` in the formula.

```
score_model_3 <- lm(score ~ age * gender, data = evals_multiple)
get_regression_table(score_model_3)
```

TABLE 11.2: Model 2: Regression table with interaction effect included

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

11.3.4 Script of R code

An R script file of all R code used in this chapter is available here².

11.4 Residual analysis

11.4.1 Residual analysis

Recall the 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 6.6. 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 11.4 below, but let’s keep this notion imprecise for now. Investigating any such patterns is known as *residual analysis* and is the theme of this section.

We’ll perform our residual analysis in two ways:

1. Creating a scatterplot with the residuals on the y -axis and the original explanatory variable x on the x -axis.
2. Creating a histogram of the residuals, thereby showing the *distribution* of the residuals.

First, recall in Figure ?? above we created a scatterplot where

- on the vertical axis we had the teaching score y ,
- on the horizontal axis we had the beauty score x , and
- the blue arrow represented the residual for one particular instructor.

²[scripts/11-inference-for-regression.R](#)

Instead, in Figure 11.3 below, let's create a scatterplot where

- On the vertical axis we have the residual $y - \hat{y}$ instead
- On the horizontal axis we have the beauty score x as before:

```
# Get data
evals_ch6 <- evals %>%
  select(score, bty_avg, age)
# Fit regression model:
score_model <- lm(score ~ bty_avg, data = evals_ch6)
# Get regression table:
get_regression_table(score_model)
```

```
# A tibble: 2 x 7
  term estimate std_error statistic p_value lower_ci
  <chr>   <dbl>     <dbl>      <dbl>    <dbl>    <dbl>
1 intercept 3.88     0.076     50.961     0       3.731
2 bty_avg   0.067    0.016      4.09      0       0.035
# ... with 1 more variable: upper_ci <dbl>
```

```
# Get regression points
regression_points <- get_regression_points(score_model)
```

```
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)
```

You can think of Figure 11.3 as Figure ?? but with the blue line flattened out to $y = 0$. Does it seem like there is *no systematic pattern* to the residuals? This question is rather qualitative and subjective in nature, thus different people may respond with different answers to the above question. However, it can be argued that there isn't a *drastic* pattern in the residuals.

Let's now get a little more precise in our definition of *no systematic pattern* in the residuals. Ideally, the residuals should behave *randomly*. In addition,

1. the residuals should be on average 0. In other words, sometimes the regression model will make a positive error in that $y - \hat{y} > 0$, sometimes the regression model will make a negative error in that $y - \hat{y} < 0$, but *on average* the error is 0.

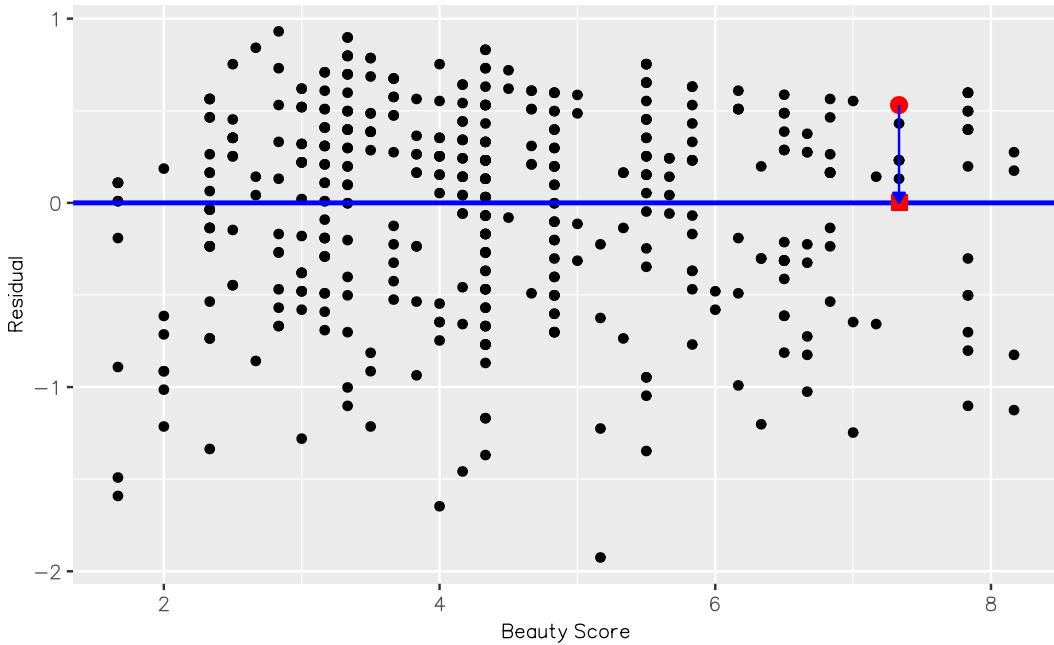


FIGURE 11.3: Plot of residuals over beauty score

2. Further, the value and spread of the residuals should not depend on the value of x .

In Figure 11.4 below, we display some hypothetical examples where there are *drastic* patterns to the residuals. In Example 1, the value of the residual seems to depend on x : the residuals tend to be positive for small and large values of x in this range, whereas values of x more in the middle tend to have negative residuals. In Example 2, while the residuals seem to be on average 0 for each value of x , the spread of the residuals varies for different values of x ; this situation is known as *heteroskedasticity*.

The second way to perform a residual analysis is to look at the histogram of the residuals:

```
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 fitted teaching score from the regression model tends to *underestimate* the true teaching score. This histogram has a slight *left-skew* in that there is a long tail on the left. Another way to say this is this data exhibits a *negative skew*. Is this

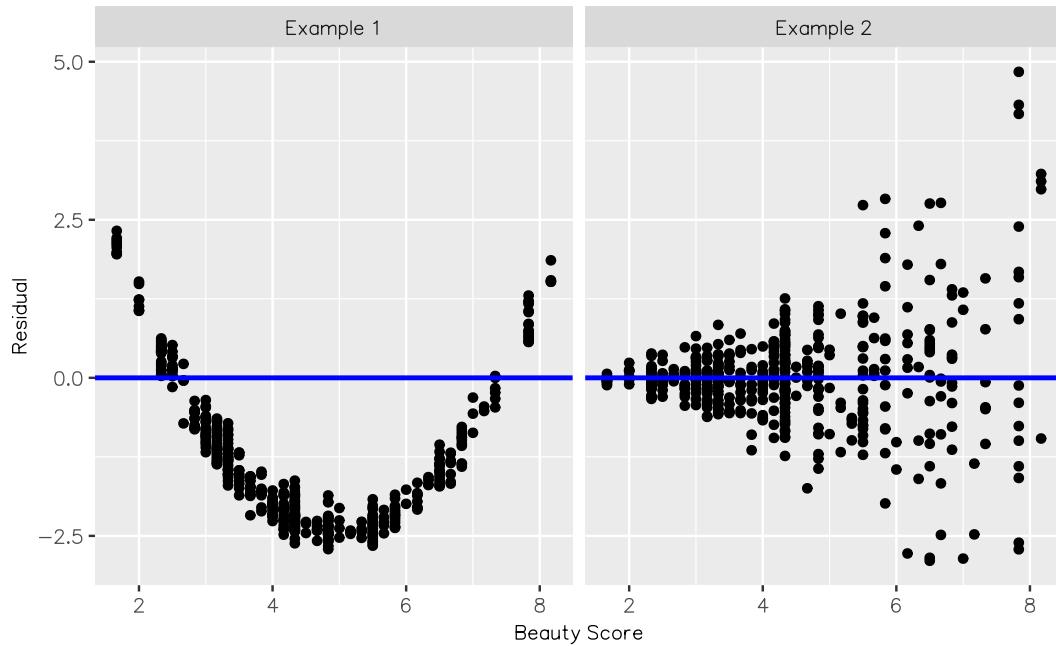


FIGURE 11.4: Examples of less than ideal residual patterns

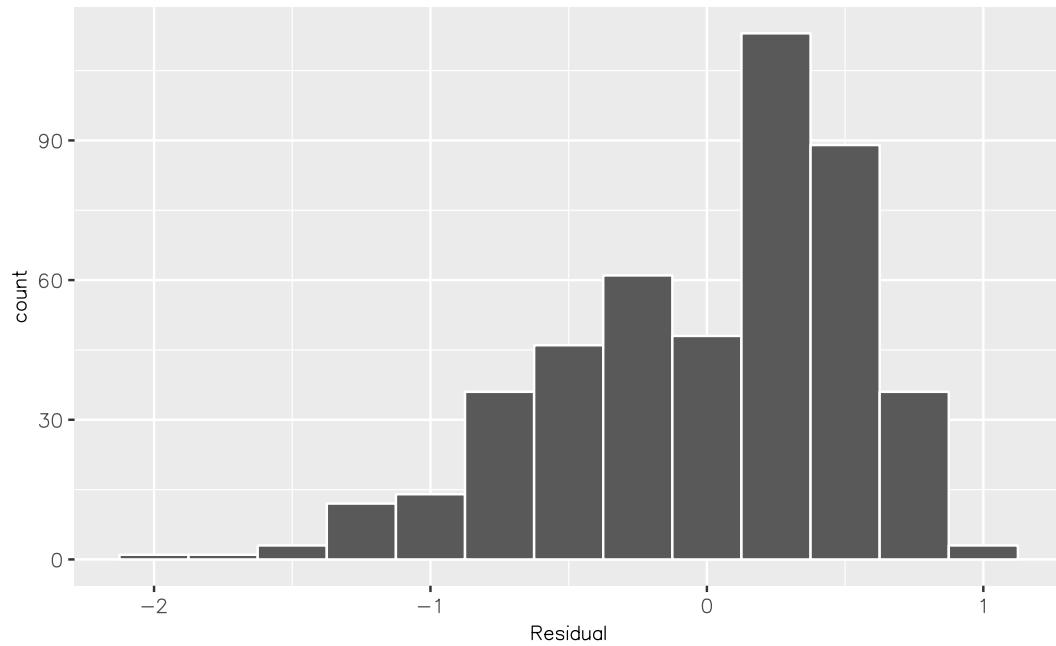


FIGURE 11.5: Histogram of residuals

a problem? Again, there is a certain amount of subjectivity in the response. In the authors' opinion, while there is a slight skew/pattern to the residuals, it isn't a large concern. On the other hand, others might disagree with our assessment. Here are examples of an ideal and less than ideal pattern to the residuals when viewed in a histogram:

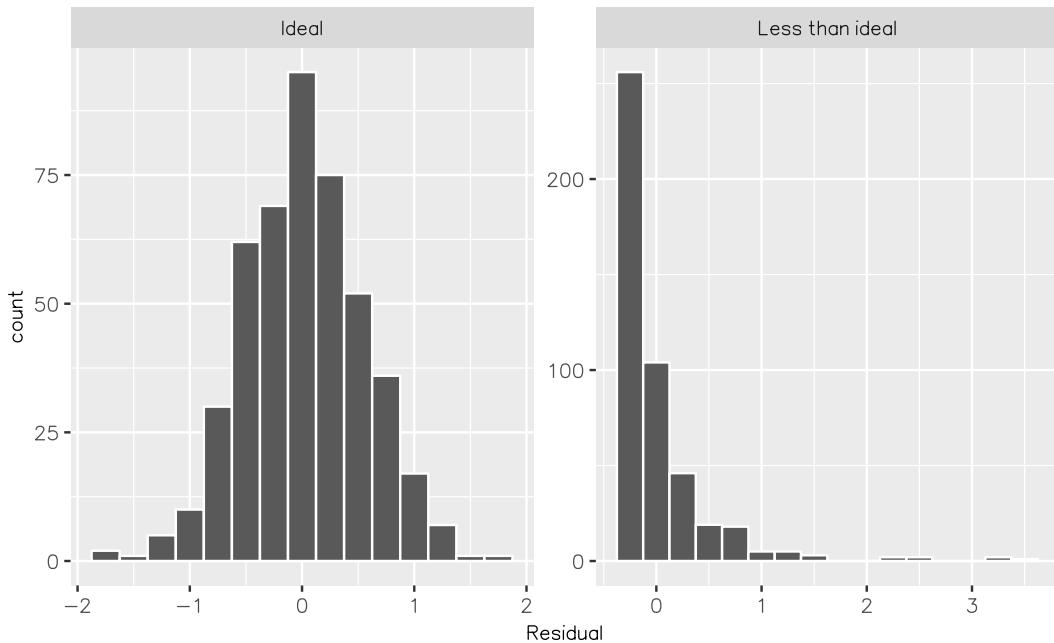


FIGURE 11.6: Examples of ideal and less than ideal residual patterns

In fact, we'll see later on that we would like the residuals to be *normally distributed* with mean 0. In other words, be bell-shaped and centered at 0! While this requirement and residual analysis in general may seem to some of you as not being overly critical at this point, we'll see later after when we cover *inference for regression* in Chapter 11 that for the last five columns of the regression table from earlier (`std_error`, `statistic`, `p_value`, `lower_ci`, and `upper_ci`) to have valid interpretations, the above three conditions should roughly hold.

Learning check

(LC11.2) 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.

11.4.2 Residual analysis

```
# Get data:
gapminder2007 <- gapminder %>%
  filter(year == 2007) %>%
  select(country, continent, lifeExp, gdpPercap)

# Fit regression model:
lifeExp_model <- lm(lifeExp ~ continent, data = gapminder2007)

# Get regression table:
get_regression_table(lifeExp_model)
```

```
# A tibble: 5 x 7
  term    estimate std_error statistic p_value lower_ci
  <chr>     <dbl>     <dbl>     <dbl>      <dbl>     <dbl>
1 inte~     54.806    1.025    53.446      0     52.778
2 cont~     18.802     1.8      10.448      0     15.243
3 cont~     15.922    1.646     9.675      0     12.668
4 cont~     22.843    1.695    13.474      0     19.490
5 cont~     25.913    5.328     4.863      0     15.377
# ... with 1 more variable: upper_ci <dbl>
```

```
# Get regression points
regression_points <- get_regression_points(lifeExp_model)
```

Recall our discussion on residuals from Section 11.4.1 where our goal was to investigate whether or not there was a *systematic pattern* to the residuals. Ideally since residuals can be thought of as error, there should be no such pattern. While there are many ways to do such residual analysis, we focused on two approaches based on visualizations.

1. A plot with residuals on the vertical axis and the predictor (in this case continent) on the horizontal axis
2. A histogram of all residuals

First, let's plot the residuals versus continent in Figure 11.7, but also let's plot all 142 points with a little horizontal random jitter by setting the `width = 0.1` parameter in `geom_jitter()`:

```
ggplot(regression_points, aes(x = continent, y = residual)) +
  geom_jitter(width = 0.1) +
  labs(x = "Continent", y = "Residual") +
  geom_hline(yintercept = 0, col = "blue")
```

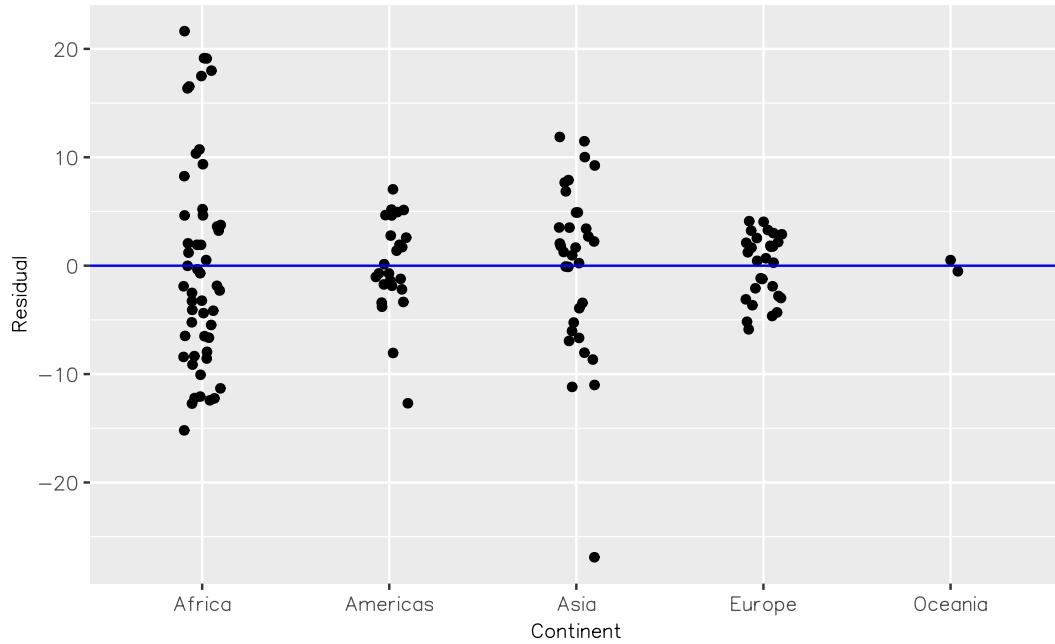


FIGURE 11.7: Plot of residuals over continent

We observe

1. There seems to be a rough balance of both positive and negative residuals for all 5 continents.
2. However, there is one clear outlier in Asia, which has a residual with the largest deviation away from 0.

Let's investigate the 5 countries in Asia with the shortest life expectancy:

```
gapminder2007 %>%
  filter(continent == "Asia") %>%
  arrange(lifeExp)
```

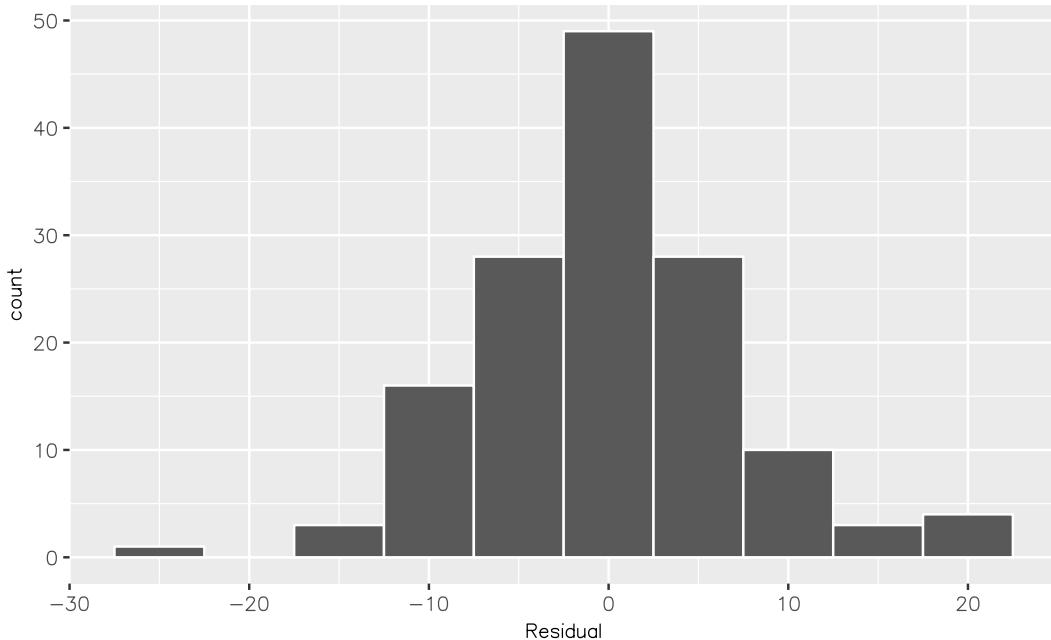
TABLE 11.3: Countries in Asia with shortest life expectancy

country	continent	lifeExp	gdpPerCap
Afghanistan	Asia	43.8	975
Iraq	Asia	59.5	4471
Cambodia	Asia	59.7	1714
Myanmar	Asia	62.1	944
Yemen, Rep.	Asia	62.7	2281

This was the earlier identified residual for Afghanistan of -26.9. Unfortunately given recent geopolitical turmoil, individuals who live in Afghanistan and, in particular in 2007, have a drastically lower life expectancy.

Second, let's look at a histogram of all 142 values of residuals in Figure 11.8. In this case, the residuals form a rather nice bell-shape, although there are a couple of very low and very high values at the tails. As we said previously, searching for patterns in residuals can be somewhat subjective, but ideally we hope there are no "drastic" patterns.

```
ggplot(regression_points, aes(x = residual)) +
  geom_histogram(binwidth = 5, color = "white") +
  labs(x = "Residual")
```

**FIGURE 11.8:** Histogram of residuals

Learning check

(LC11.3) Continuing with our regression using `gdpPercap` as the outcome variable and `continent` as the explanatory variable, use the `get_regression_points()` function to get the observed values, fitted values, and residuals for all 142 countries in 2007 and perform a residual analysis to look for any systematic patterns in the residuals. Is there a pattern? Please keep in mind that these types of questions are somewhat subjective and different people will most likely have different answers. The focus should be on being able to justify the conclusions made.

11.4.3 Residual analysis

Recall in Section 11.4.1, our first residual analysis plot investigated the presence of any systematic pattern in the residuals when we had a single numerical predictor: `bty_age`. For the `Credit` card dataset, since we have two numerical predictors, `Limit` and `Income`, we must perform this twice:

```
# Get data:
Credit <- Credit %>%
  select(Balance, Limit, Income, Rating, Age)
# Fit regression model:
Balance_model <- lm(Balance ~ Limit + Income, data = Credit)
# Get regression table:
get_regression_table(Balance_model)
```

```
# A tibble: 3 x 7
  term    estimate std_error statistic p_value lower_ci
  <chr>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1 intercept -385.179   19.465   -19.789      0 -423.446
2 Limit      0.264     0.006     44.955      0     0.253
3 Income     -7.663    0.385    -19.901      0     -8.42
# ... with 1 more variable: upper_ci <dbl>
```

```
# Get regression points
regression_points <- get_regression_points(Balance_model)
```

```
ggplot(regression_points, aes(x = Limit, y = residual)) +
  geom_point() +
  labs(x = "Credit limit (in $)",
       y = "Residual",
       title = "Residuals vs credit limit")

ggplot(regression_points, aes(x = Income, y = residual)) +
  geom_point() +
  labs(x = "Income (in $1000)",
       y = "Residual",
       title = "Residuals vs income")
```

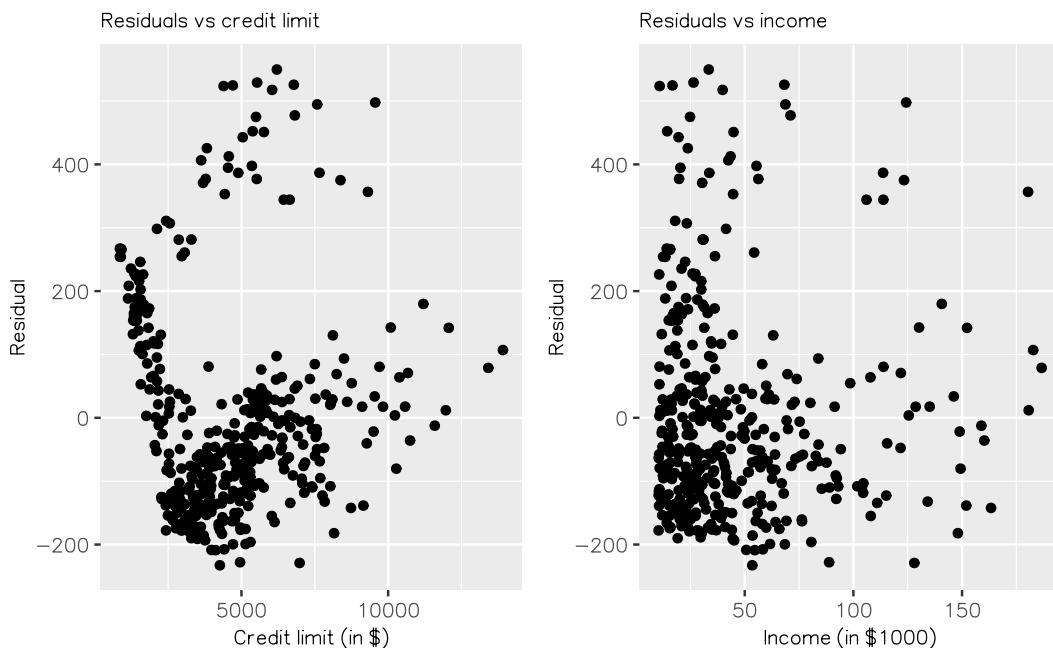


FIGURE 11.9: Residuals vs credit limit and income

In this case, there **does** appear to be a systematic pattern to the residuals. As the scatter of the residuals around the line $y = 0$ is definitely not consistent. This behavior of the residuals is further evidenced by the histogram of residuals in Figure 11.10. We observe that the residuals have a slight right-skew (recall we say that data is right-skewed, or positively-skewed, if there is a tail to the right). Ideally, these residuals should be bell-shaped around a residual value of 0.

```
ggplot(regression_points, aes(x = residual)) +
  geom_histogram(color = "white") +
  labs(x = "Residual")
```

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

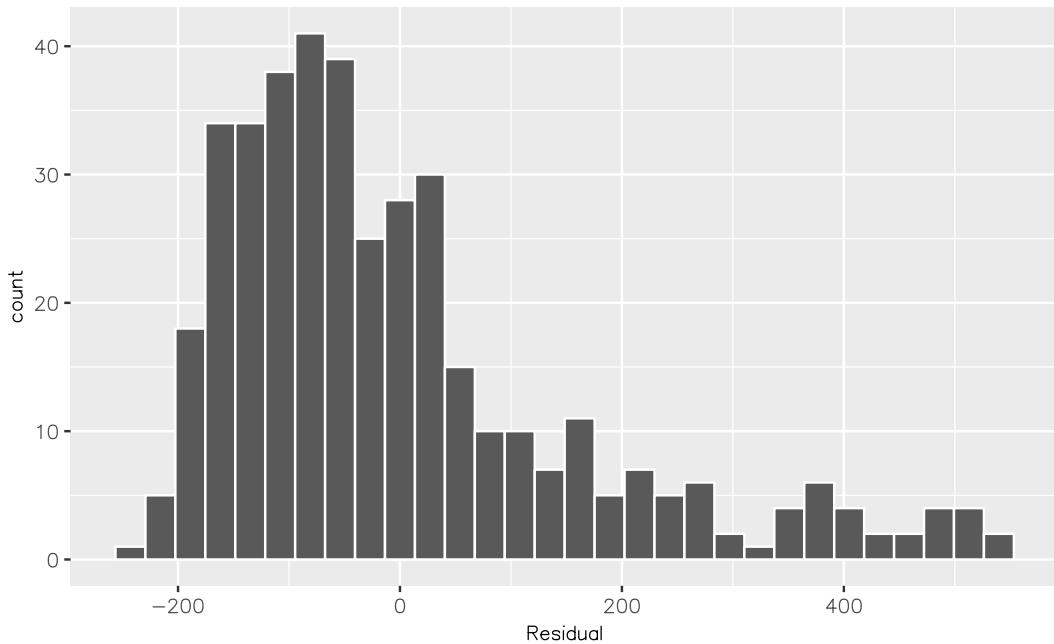


FIGURE 11.10: Relationship between credit card balance and credit limit/income

Another way to interpret this histogram is that since the residual is computed as $y - \hat{y} = \text{balance} - \text{balance_hat}$, we have some values where the fitted value \hat{y} is very much lower than the observed value y . In other words, we are underestimating certain credit card holders' balances by a very large amount.

Learning check

(LC11.4) Continuing with our regression using `Rating` and `Age` as the explanatory variables and credit card `Balance` as the outcome variable, use the `get_regression_points()` function to get the observed values, fitted values, and residuals for all 400 credit card holders. Perform a residual analysis and look for any systematic patterns in the residuals.

11.4.4 Residual analysis

```
# Get data:  
evals_ch7 <- evals %>%  
  select(score, age, gender)  
# Fit regression model:  
score_model_2 <- lm(score ~ age + gender, data = evals_ch7)  
# Get regression table:  
get_regression_table(score_model_2)  
  
# A tibble: 3 × 7  
  term    estimate std_error statistic p_value lower_ci  
  <chr>     <dbl>     <dbl>     <dbl>      <dbl>     <dbl>  
1 inte~     4.484     0.125    35.792      0       4.238  
2 age       -0.009000   0.003    -3.28     0.001  -0.014  
3 gend~     0.191     0.052     3.632      0       0.08700  
# ... with 1 more variable: upper_ci <dbl>  
  
# Get regression points  
regression_points <- get_regression_points(score_model_2)
```

As always, let's perform a residual analysis first with a histogram, which we can facet by gender:

```
ggplot(regression_points, aes(x = residual)) +  
  geom_histogram(binwidth = 0.25, color = "white") +  
  labs(x = "Residual") +  
  facet_wrap(~gender)
```

Second, the residuals as compared to the predictor variables:

- x_1 : numerical explanatory/predictor variable of `age`
- x_2 : categorical explanatory/predictor variable of `gender`

```
ggplot(regression_points, aes(x = age, y = residual)) +  
  geom_point() +
```

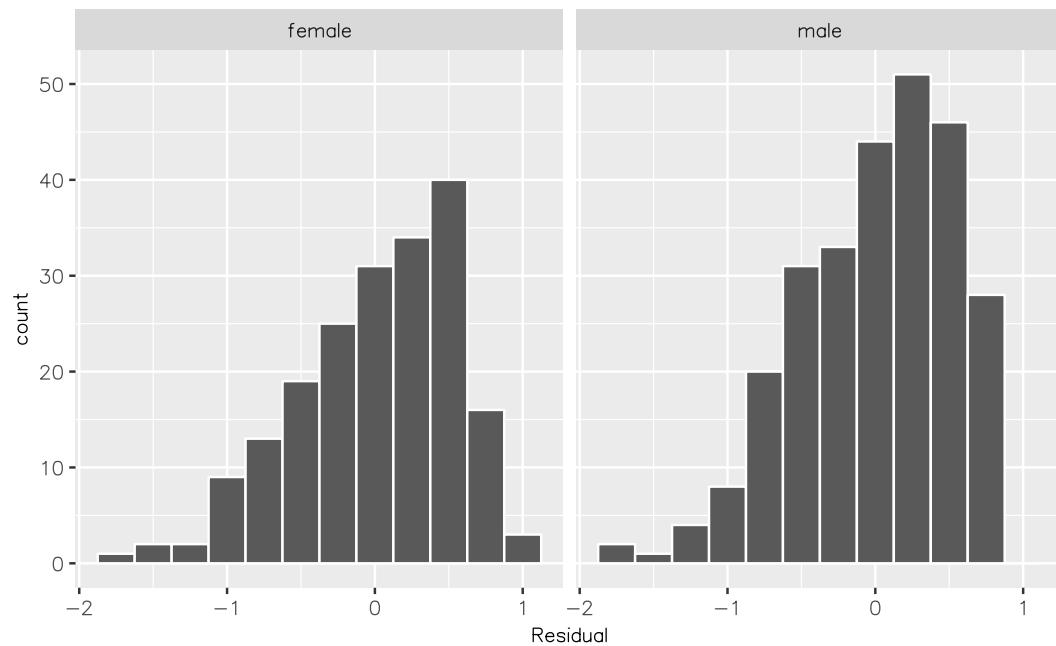


FIGURE 11.11: Interaction model histogram of residuals

```
labs(x = "age", y = "Residual") +  
  geom_hline(yintercept = 0, col = "blue", size = 1) +  
  facet_wrap(~ gender)
```

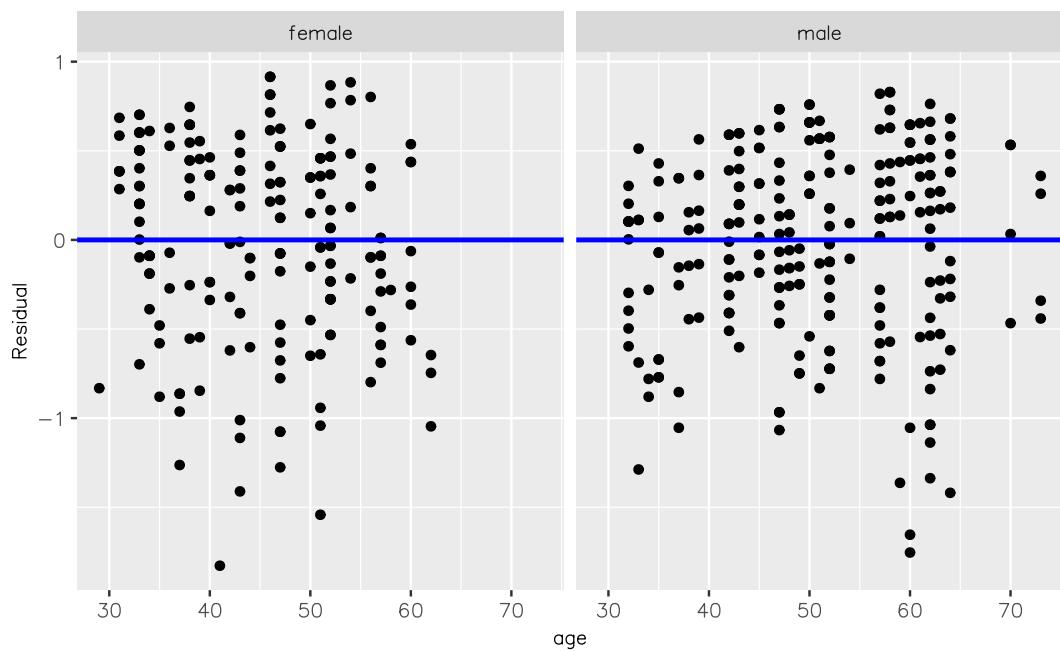


FIGURE 11.12: Interaction model residuals vs predictor

Part IV

Conclusion

12

Thinking with Data

In preparation for our first print edition to be published by CRC Press in Fall 2019, we're remodeling this chapter a bit. Don't expect major changes in content, but rather only minor changes in presentation. Our remodeling will be complete and available online at [ModrenDive.com¹](https://ModrenDive.com) by early Summer 2019!

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.

Let’s get a refresher of what you’ve covered so far. You first got started with 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.5: Understanding the concept of “tidy” data as a standardized data input format for all packages in the `tidyverse`
 - Ch.4: Wrangling data via the `dplyr` package.
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, we unpacked statistical inference using the `infer` package. In particular:

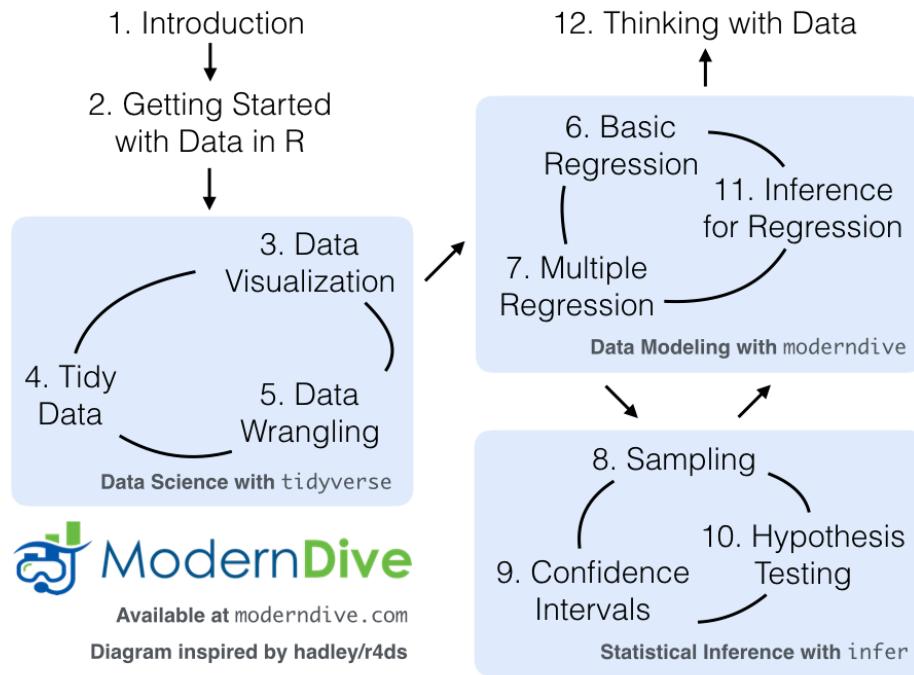


FIGURE 12.1: ModernDive Flowchart

- 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 practice significance of the results of the models.

All this was our approach of guiding you through your first experiences of “thinking with data”², 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”³ collection of preprints focusing on the practical side of

²<https://arxiv.org/pdf/1410.3127.pdf>

³<https://peerj.com/collections/50-practicaldatascistats/>

data science workflows and statistical analysis, curated by Jennifer Bryan⁴ and Hadley Wickham⁵:

There are many aspects of day-to-day analytical work that are almost absent from the conventional statistics literature and curriculum. And yet these activities 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”⁶ we also saw earlier and not just parts of it.

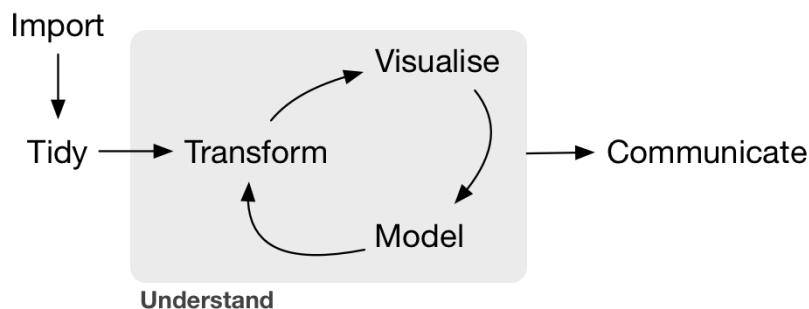


FIGURE 12.2: Data/Science Pipeline

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.com⁷, many of whose source datasets are accessible from the `fivethirtyeight` R package.

⁴<https://twitter.com/jennybryan?lang=en>

⁵<https://twitter.com/hadleywickham?lang=en>

⁶<http://r4ds.had.co.nz/explore-intro.html>

⁷<https://fivethirtyeight.com/>

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(ggplot2)
library(dplyr)
library(moderndive)
library(fivethirtyeight)
```

12.1 Case study: Seattle house prices

Kaggle.com⁸ 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, USA⁹ consisting of homes sold in between May 2014 and May 2015 in King County, Washington State, USA, which includes the greater Seattle metropolitan area. This CC0: Public Domain¹⁰ licensed dataset is included in the `moderndive` package in the `house_prices` data frame, which we'll refer to as the “Seattle house prices” dataset.

The dataset consists 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 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, where 1 square foot is about 0.09 square meters.
 2. x_2 : house `condition`, a categorical variable with 5 levels where 1 indicates “poor” and 5 indicates “excellent.”

⁸<https://www.kaggle.com/>

⁹<https://www.kaggle.com/harlfoxem/housesalesprediction>

¹⁰<https://creativecommons.org/publicdomain/zero/1.0/>

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)
```

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 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 i.e. 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, 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, 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 using a `geom_histogram()` as seen in Section 3.5. However, since `condition` is categorical, a barplot using a `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")
```

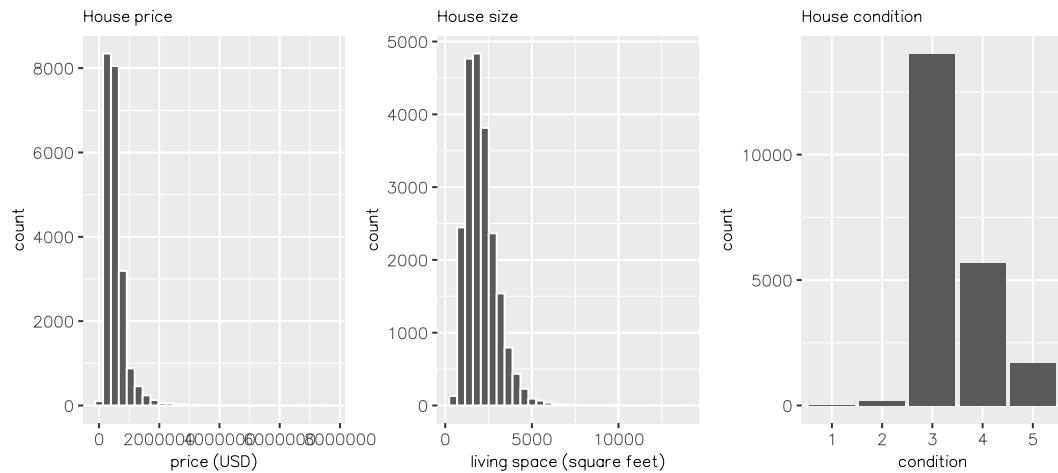


FIGURE 12.3: Exploratory visualizations of Seattle house prices data

We observe the following:

1. In the histogram for `price`:
 - Since `e+06` means 10^6 , or one million, we see that a majority of houses are less than 2 million dollars.
 - The x-axis stretches out far to the right to 8 million dollars, even though there appear to be no houses.
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 AKA 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 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 4 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 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 reports median house prices and not average house prices when describing the real estate market. We say here that the median 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 log10 transformations

At its simplest, `log10()` transformations returns base 10 *logarithms*. For example, since $1000 = 10^3$, `log10(1000)` returns 3. To undo a log10-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.” 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 e.g. \$676 or \$757, but tend to round to the nearest unit of hundreds of dollars e.g. \$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, `log10(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 \log_{10} -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 6th 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 6.
- 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")
```

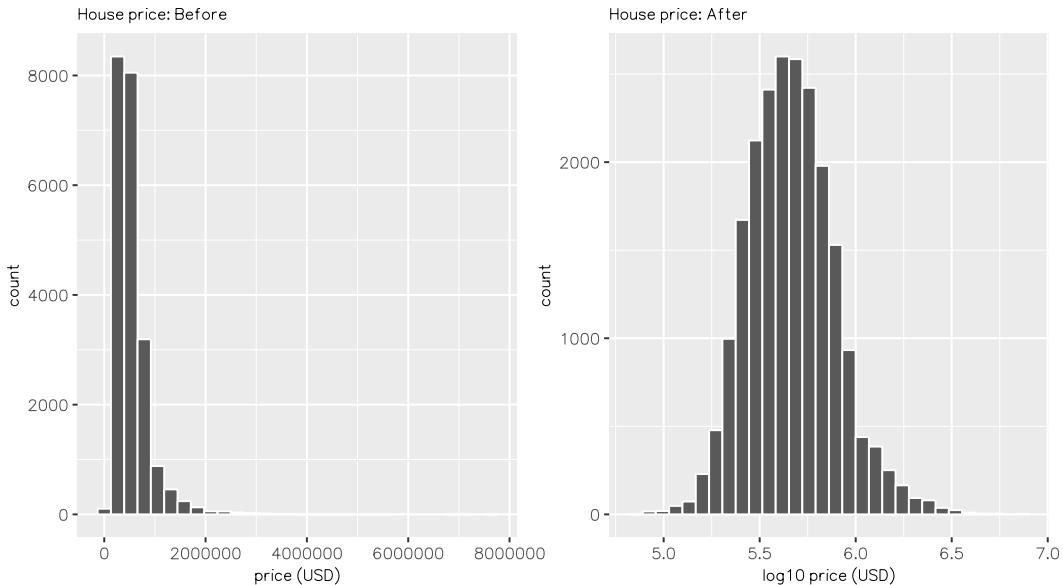


FIGURE 12.4: House price before and after log10-transformation

Observe that after the transformation, the distribution is much less skewed, and in this case, more symmetric and bell-shaped, although 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")
```

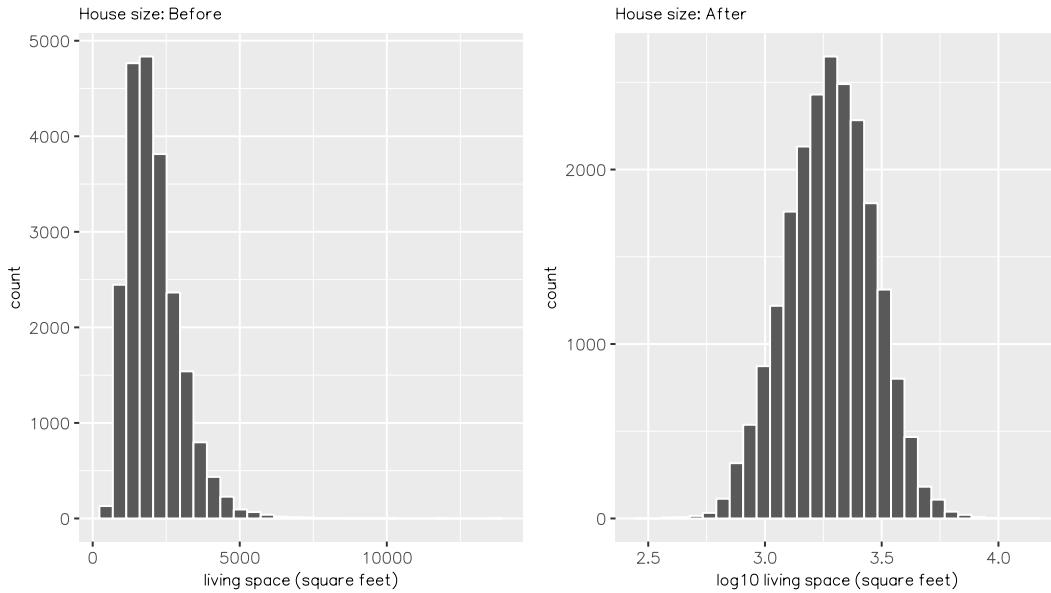


FIGURE 12.5: House size before and after log10-transformation

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 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:

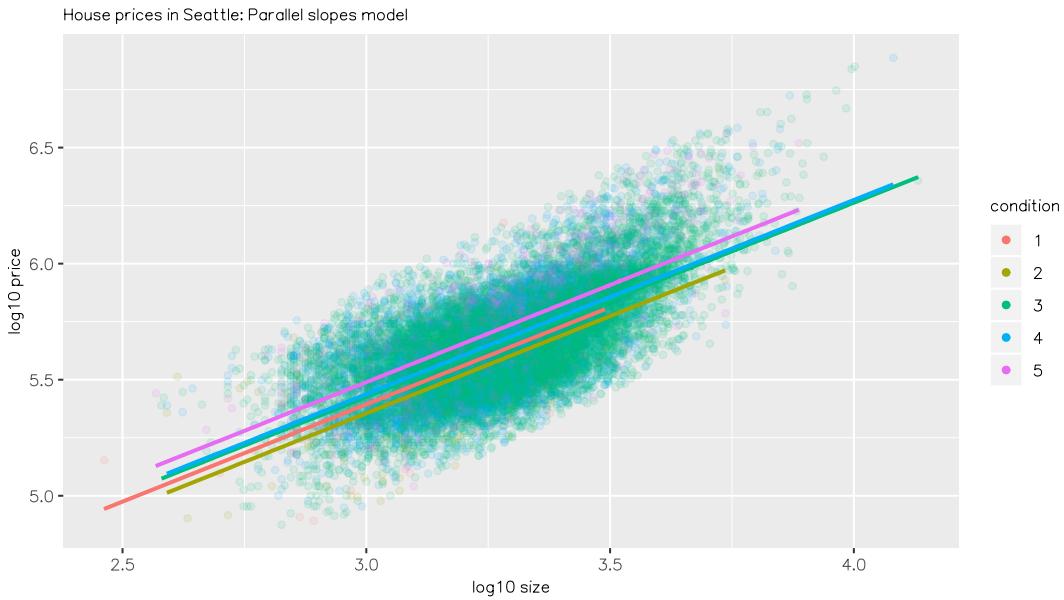


FIGURE 12.6: Parallel slopes model

Or 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 apparent in an alternative visualization to Figure 12.7 displayed in Figure 12.8 that uses facets instead:

```
ggplot(house_prices, aes(x = log10_size, y = log10_price, col = condition)) +
```

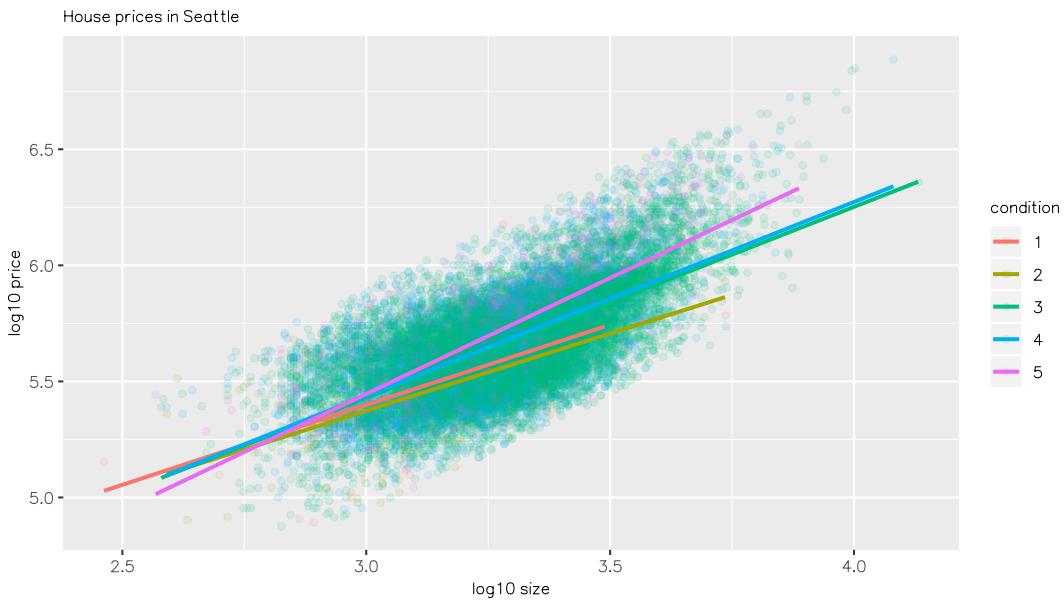


FIGURE 12.7: Interaction model

```
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)
```

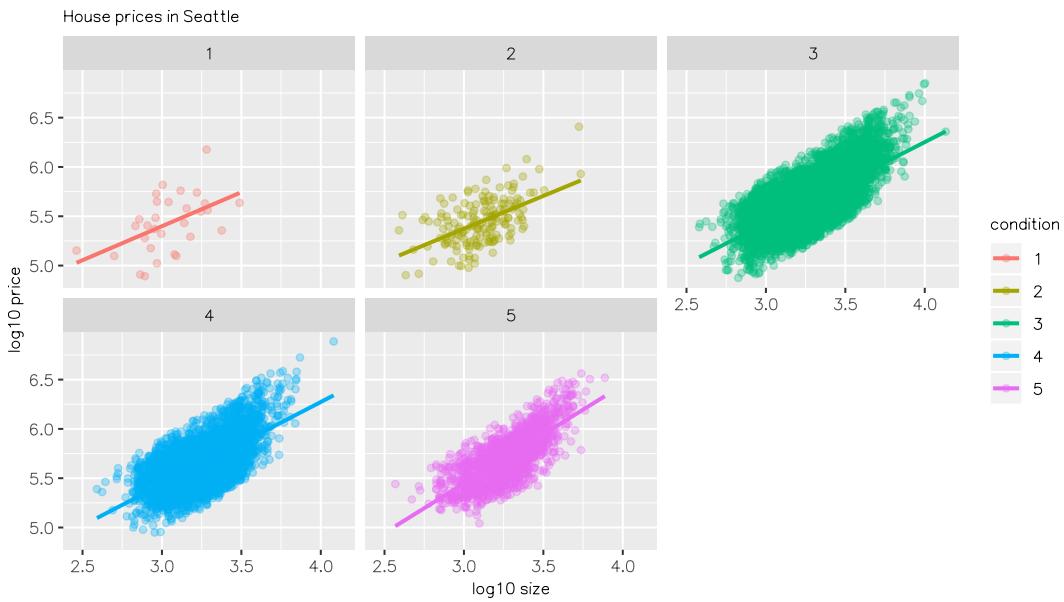


FIGURE 12.8: Interaction model with facets

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)
# Get regression table:
get_regression_table(price_interaction)
```

term	estimate	std_error	statistic	p_value	lower_ci	upper_ci
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 prices 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 3 most largest slopes out of the 5.

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: it 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 an 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.

```
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!

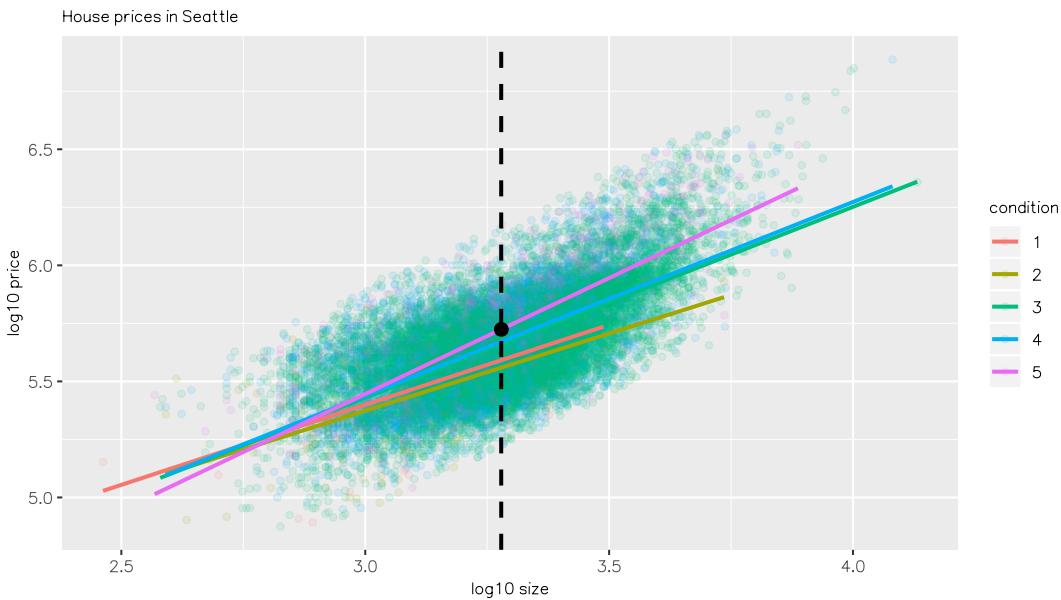


FIGURE 12.9: Interaction model with prediction

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 our predicted price for this home of condition 5 and size 1900 square feet is \$535,493.

Learning check

(LC12.1) 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

Note: This section is still under construction. If you would like to contribute, please check us out on GitHub at https://github.com/moderndive/moderndive_book.



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 Women¹¹ on the Bechdel test, an informal test of gender representation in a movie. 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

¹¹<http://fivethirtyeight.com/features/the-dollar-and-cents-case-against-hollywoods-exclusion-of-women/>

shared the data and R code¹² that they used to create for this story and many more of their articles on GitHub¹³.

ModernDive co-authors Chester Ismay¹⁴ and Albert Y. Kim¹⁵ along with Jennifer Chunnn¹⁶ went one step further by creating the `fivethirtyeight` R package¹⁷. The `fivethirtyeight` package takes FiveThirtyEight's article data from GitHub, “tames”¹⁸ 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 sets¹⁹ 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 here²⁰. 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 here²¹.

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:

¹²<https://github.com/fivethirtyeight/data/tree/master/bechdel>

¹³<https://github.com/fivethirtyeight/data>

¹⁴https://twitter.com/old_man_chester?lang=en

¹⁵<https://twitter.com/rudeboybert>

¹⁶<https://twitter.com/jchunn206>

¹⁷<https://fivethirtyeight-r.netlify.com/>

¹⁸http://rpubs.com/rudeboybert/fivethirtyeight_tamedata

¹⁹<https://fivethirtyeight-r.netlify.com/articles/fivethirtyeight.html#data-sets>

20 <https://fivethirtyeight-r.netlify.com/articles/>21 <https://fivethirtyeight-r.netlify.com/articles/bechdel.html>

```
# Preview data
glimpse(US_births_1994_2003)
```

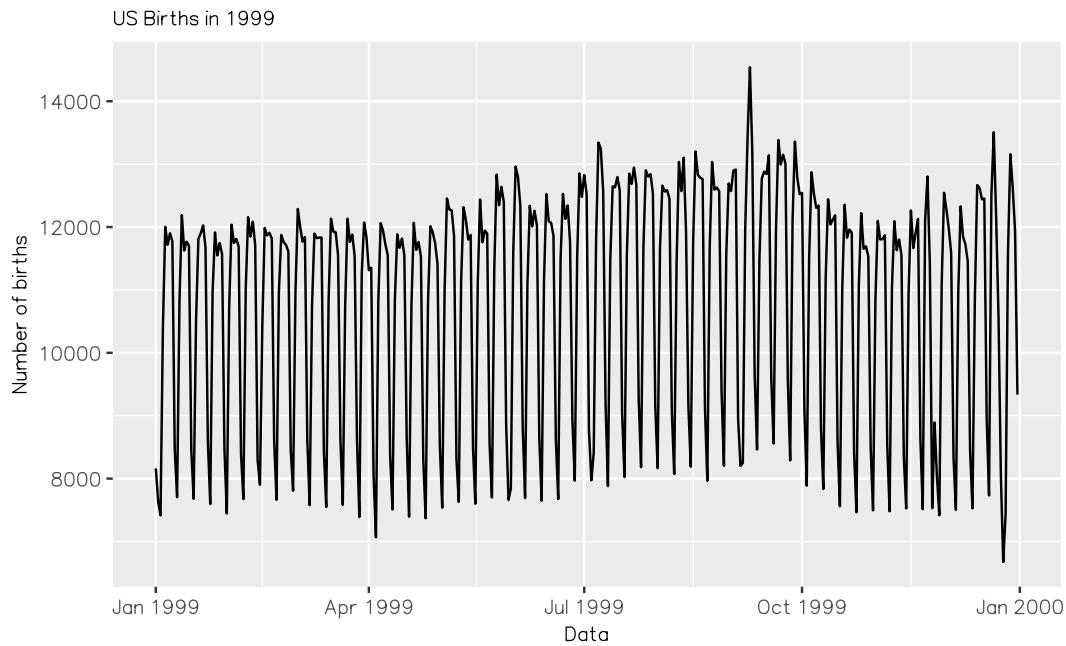
```
Observations: 3,652
Variables: 6
$ year      <int> 1994, 1994, 1994, 1994, 1994, 199...
$ month     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
$ date_of_month <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11...
$ date      <date> 1994-01-01, 1994-01-02, 1994-01-...
$ day_of_week <ord> Sat, Sun, Mon, Tues, Wed, 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 linegraph AKA 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")
```



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 in ? Time to think with data!

12.2.3 Other examples

Stand by!

12.2.4 Script of R code

An R script file of all R code used in this chapter is available here²².

Concluding remarks

If you've come to this point in the book, I'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

²²[scripts/12-thinking-with-data.R](#)

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!

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:

$$\text{Mean} = \frac{x_1 + x_2 + \dots + x_n}{n}$$

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:

$$\text{Standard deviation} = \sqrt{\frac{(x_1 - \text{Mean})^2 + (x_2 - \text{Mean})^2 + \dots + (x_n - \text{Mean})^2}{n - 1}}$$

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 discussion

B

Inference Examples

This appendix is designed to provide you with examples of the five basic hypothesis tests and their corresponding confidence intervals. Traditional theory-based methods as well as computational-based methods are presented.

Note: This appendix is still under construction. If you would like to contribute, please check us out on GitHub at https://github.com/moderndive/moderndive_book.

Please check out our sneak peak of `infer` below in the meanwhile. For more details on `infer` visit <https://infer.netlify.com/>.

```
include_image(path = "images/sign-2408065_1920.png",
              html_opts="height=100px",
              latex_opts = "width=20%")
```

```
{ width=20% }
```

Needed packages

```
library(dplyr)
library(ggplot2)
library(infer)
library(knitr)
library(kableExtra)
library(readr)
library(janitor)
```

B.1 Inference mind map

To help you better navigate and choose the appropriate analysis, we've created a mind map on <http://coggle.it> available here¹ and below.

B.2 One mean

B.2.1 Problem statement

The National Survey of Family Growth conducted by the Centers for Disease Control gathers information on family life, marriage and divorce, pregnancy, infertility, use of contraception, and men's and women's health. One of the variables collected on this survey is the age at first marriage. 5,534 randomly sampled US women between 2006 and 2010 completed the survey. The women sampled here had been married at least once. Do we have evidence that the mean age of first marriage for all US women from 2006 to 2010 is greater than 23 years? (Tweaked a bit from [Diez et al., 2014](#), [Chapter 4])

B.2.2 Competing hypotheses

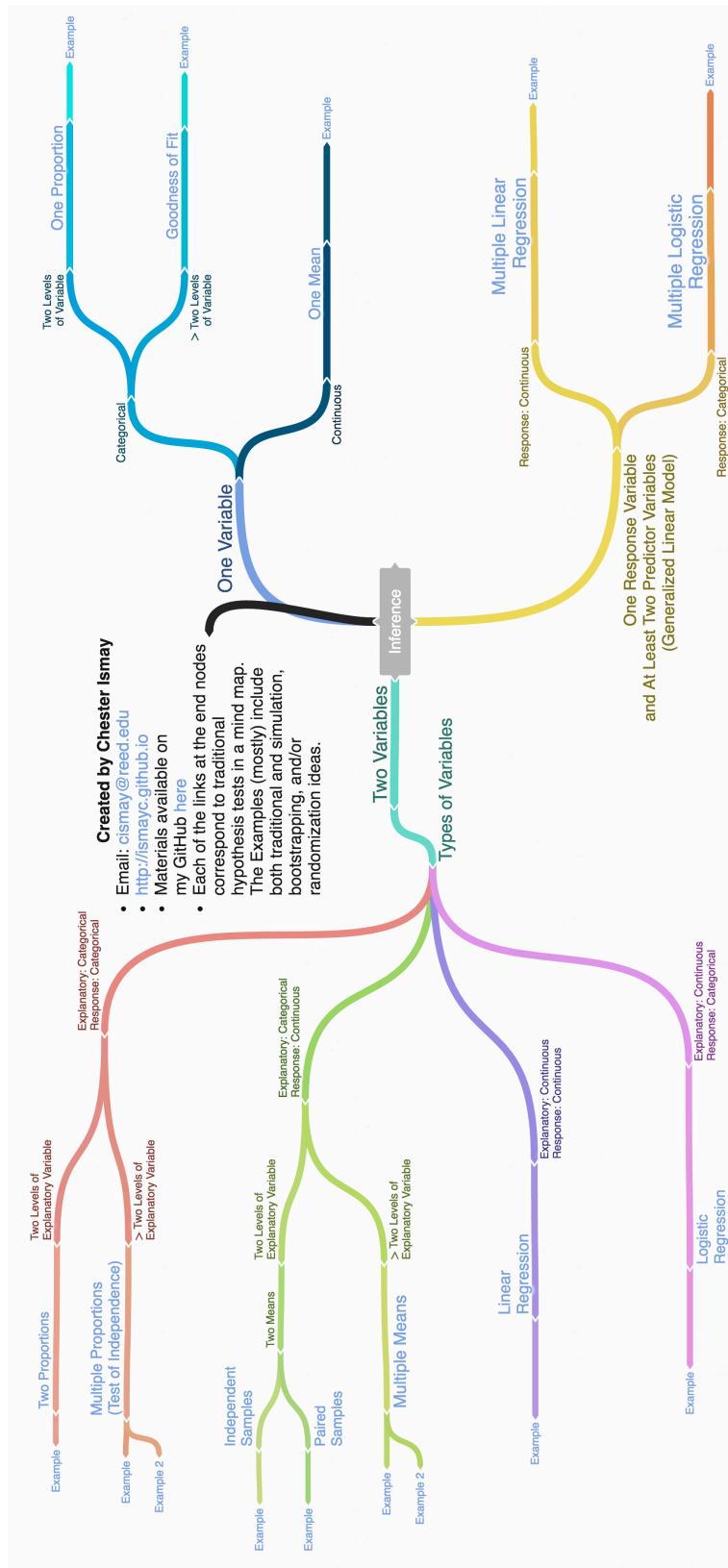
In words

- Null hypothesis: The mean age of first marriage for all US women from 2006 to 2010 is equal to 23 years.
- Alternative hypothesis: The mean age of first marriage for all US women from 2006 to 2010 is greater than 23 years.

In symbols (with annotations)

- $H_0 : \mu = \mu_0$, where μ represents the mean age of first marriage for all US women from 2006 to 2010 and μ_0 is 23.
- $H_A : \mu > 23$

¹<https://coggle.it/diagram/Vxlydu1akQFeqo6->

**FIGURE B.1:** Mind map for Inference

Set α

It's important to set the significance level before starting the testing using the data. Let's set the significance level at 5% here.

B.2.3 Exploring the sample data

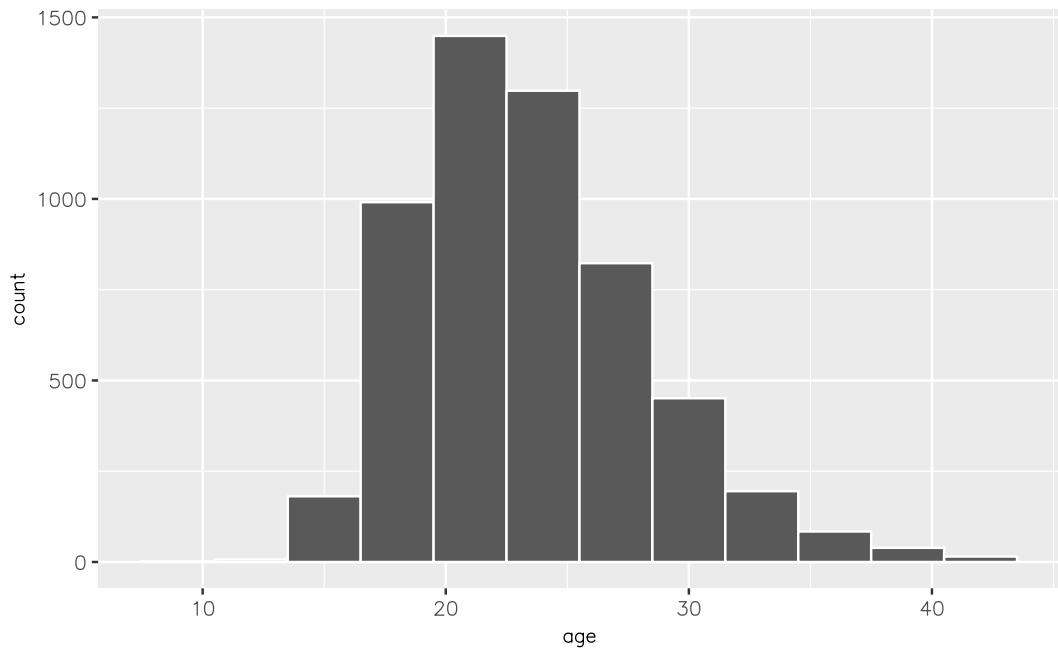
```
age_at_marriage <- read_csv("https://moderndive.com/data/ageAtMar.csv")
```

```
age_summ <- age_at_marriage %>%
  summarize(sample_size = n(),
            mean = mean(age),
            sd = sd(age),
            minimum = min(age),
            lower_quartile = quantile(age, 0.25),
            median = median(age),
            upper_quartile = quantile(age, 0.75),
            max = max(age))
kable(age_summ) %>%
  kable_styling(font_size = ifelse(knitr:::is_latex_output(), 10, 16),
                latex_options = c("HOLD_position"))
```

sample_size	mean	sd	minimum	lower_quartile	median	upper_quartile	max
5534	23.4	4.72	10	20	23	26	43

The histogram below also shows the distribution of `age`.

```
ggplot(data = age_at_marriage, mapping = aes(x = age)) +
  geom_histogram(binwidth = 3, color = "white")
```



The observed statistic of interest here is the sample mean:

```
x_bar <- age_at_marriage %>%
  specify(response = age) %>%
  calculate(stat = "mean")
x_bar
```

```
# A tibble: 1 × 1
  stat
  <dbl>
1 23.4402
```

Guess about statistical significance

We are looking to see if the observed sample mean of 23.44 is statistically greater than $\mu_0 = 23$. They seem to be quite close, but we have a large sample size here. Let's guess that the large sample size will lead us to reject this practically small difference.

B.2.4 Non-traditional methods

Bootstrapping for hypothesis test

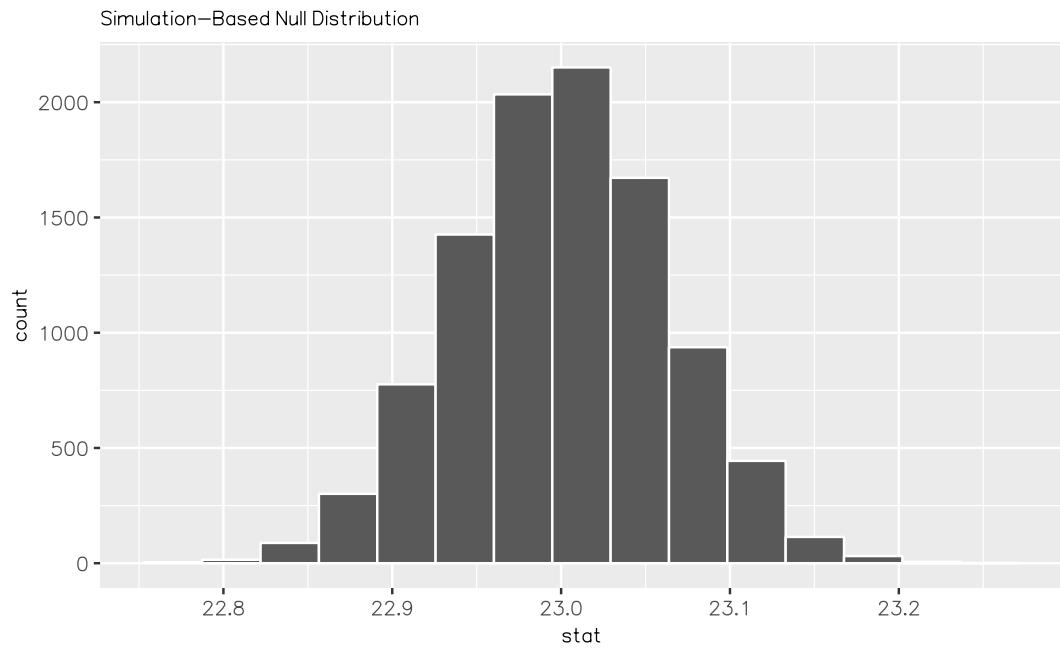
In order to look to see if the observed sample mean of 23.44 is statistically greater than $\mu_0 = 23$, we need to account for the sample size. We also need to determine a process that replicates how the original sample of size 5534 was selected.

We can use the idea of *bootstrapping* to simulate the population from which the sample came and then generate samples from that simulated population to account for sampling variability. Recall how bootstrapping would apply in this context:

1. Sample with replacement from our original sample of 5534 women and repeat this process 10,000 times,
2. calculate the mean for each of the 10,000 bootstrap samples created in Step 1.,
3. combine all of these bootstrap statistics calculated in Step 2 into a `boot_distn` object, and
4. shift the center of this distribution over to the null value of 23. (This is needed since it will be centered at 23.44 via the process of bootstrapping.)

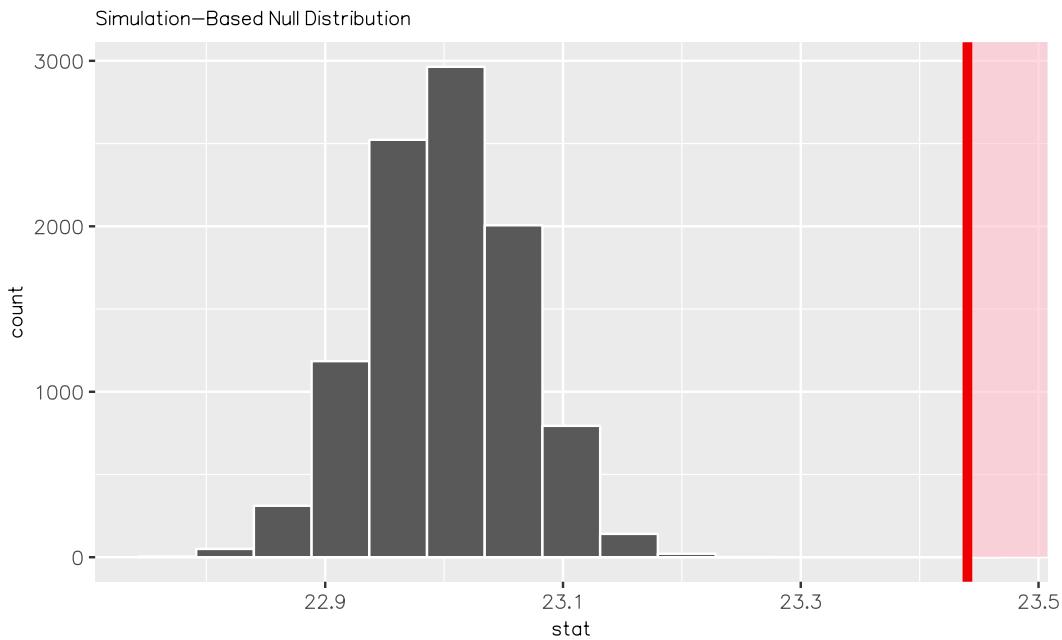
```
set.seed(2018)
null_distn_one_mean <- age_at_marriage %>%
  specify(response = age) %>%
  hypothesize(null = "point", mu = 23) %>%
  generate(reps = 10000) %>%
  calculate(stat = "mean")
```

```
null_distn_one_mean %>% visualize()
```



We can next use this distribution to observe our p -value. Recall this is a right-tailed test so we will be looking for values that are greater than or equal to 23.44 for our p -value.

```
null_distn_one_mean %>%
  visualize(obs_stat = x_bar, direction = "greater")
```



B.2.4.0.1 Calculate p -value

```
pvalue <- null_distn_one_mean %>%
  get_pvalue(obs_stat = x_bar, direction = "greater")
pvalue
```

```
# A tibble: 1 × 1
  p_value
  <dbl>
1      0
```

So our p -value is 0 and we reject the null hypothesis at the 5% level. You can also see this from the histogram above that we are far into the tail of the null distribution.

Bootstrapping for confidence interval

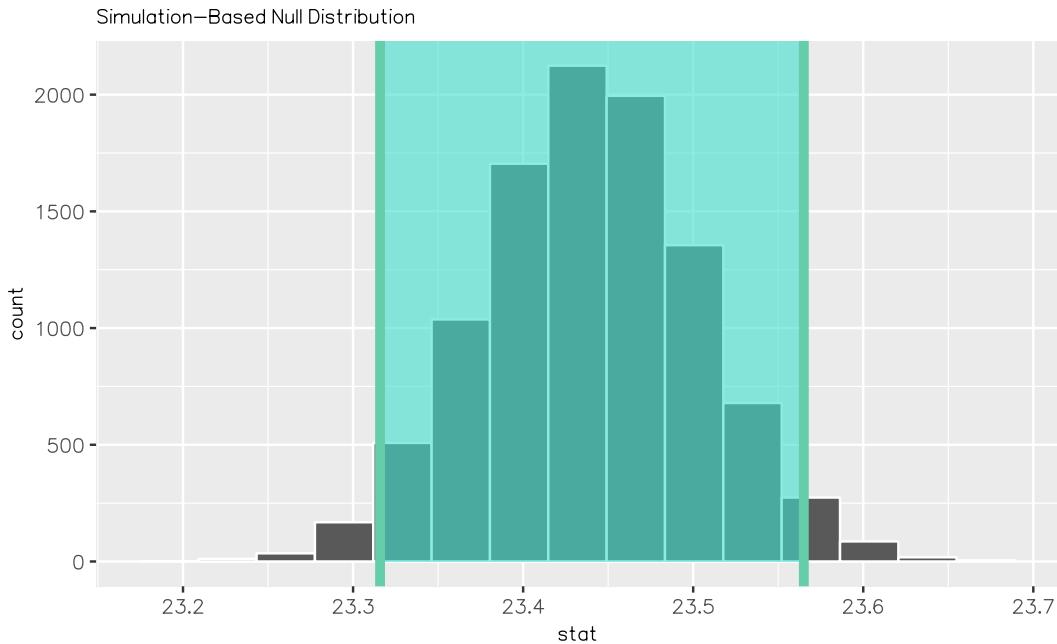
We can also create a confidence interval for the unknown population parameter μ using our sample data using *bootstrapping*. Note that we don't need to shift this distribution since we want the center of our confidence interval to be our point estimate $\bar{x}_{obs} = 23.44$.

```
boot_distn_one_mean <- age_at_marriage %>%
  specify(response = age) %>%
  generate(reps = 10000) %>%
  calculate(stat = "mean")
```

```
ci <- boot_distn_one_mean %>%
  get_ci()
ci
```

```
# A tibble: 1 x 2
`2.5%` `97.5%
<dbl>   <dbl>
1 23.3159 23.5651
```

```
boot_distn_one_mean %>%
  visualize(endpoints = ci, direction = "between")
```



We see that 23 is not contained in this confidence interval as a plausible value of μ (the unknown population mean) and the entire interval is larger than 23. This matches with our hypothesis test results of rejecting the null hypothesis in favor of the alternative ($\mu > 23$).

Interpretation: We are 95% confident the true mean age of first marriage for all US women from 2006 to 2010 is between 23.316 and 23.565.

B.2.5 Traditional methods

Check conditions

Remember that in order to use the shortcut (formula-based, theoretical) approach, we need to check that some conditions are met.

1. *Independent observations:* The observations are collected independently.

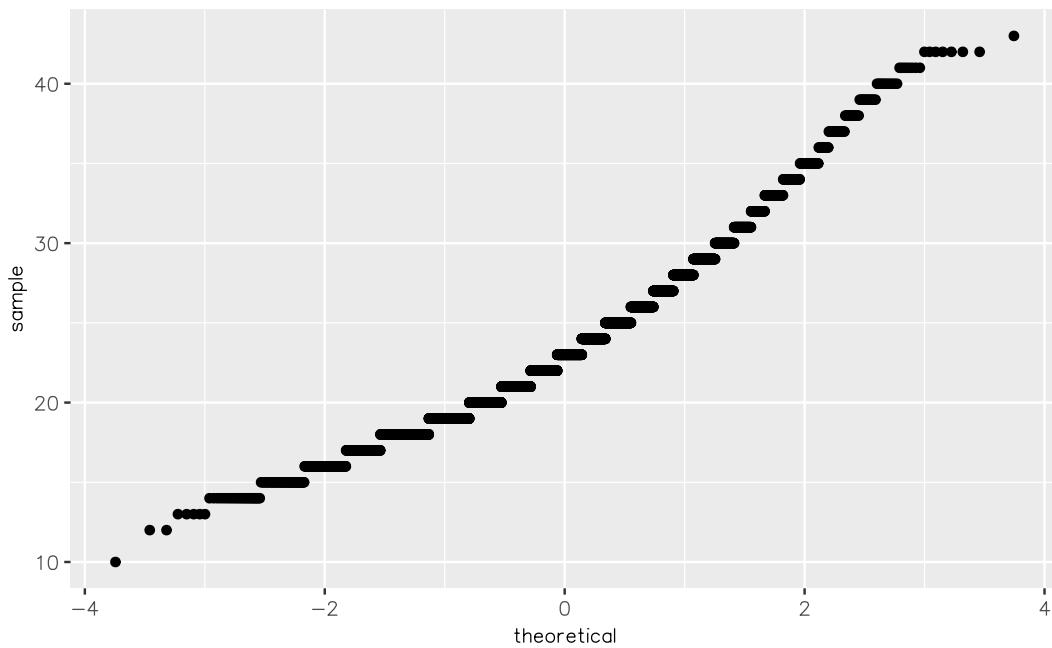
The cases are selected independently through random sampling so this condition is met.

2. *Approximately normal:* The distribution of the response variable should be normal or the sample size should be at least 30.

The histogram for the sample above does show some skew.

The Q-Q plot below also shows some skew.

```
ggplot(data = age_at_marriage, mapping = aes(sample = age)) +  
  stat_qq()
```



The sample size here is quite large though ($n = 5534$) so both conditions are met.

Test statistic

The test statistic is a random variable based on the sample data. Here, we want to look at a way to estimate the population mean μ . A good guess is the sample mean \bar{X} . Recall that this sample mean is actually a random variable that will vary as different samples are (theoretically, would be) collected. We are looking to see how likely is it for us to have observed a sample mean of $\bar{x}_{obs} = 23.44$ or larger assuming that the population mean is 23 (assuming the null hypothesis is true). If the conditions are met and assuming H_0 is true, we can “standardize” this original test statistic of \bar{X} into a T statistic that follows a t distribution with degrees of freedom equal to $df = n - 1$:

$$T = \frac{\bar{X} - \mu_0}{S/\sqrt{n}} \sim t(df = n - 1)$$

where S represents the standard deviation of the sample and n is the sample size.

B.2.5.0.1 Observed test statistic

While one could compute this observed test statistic by “hand”, the focus here is on the set-up of the problem and in understanding which formula for the test statistic applies. We can use the `t_test()` function to perform this analysis for us.

```
t_test_results <- age_at_marriage %>%
  infer::t_test(formula = age ~ NULL,
    alternative = "greater",
    mu = 23)
t_test_results

# A tibble: 1 x 6
  statistic   t_df     p_value alternative lower_ci upper_ci
    <dbl>   <dbl>      <dbl>     <chr>       <dbl>     <dbl>
1   6.93570  5533  2.25216e-12 greater     23.3358     Inf
```

We see here that the t_{obs} value is 6.936.

Compute p -value

The p -value—the probability of observing an t_{obs} value of 6.936 or more in our null distribution of a t with 5533 degrees of freedom—is essentially 0.

State conclusion

We, therefore, have sufficient evidence to reject the null hypothesis. Our initial guess that our observed sample mean was statistically greater than the hypothesized mean has supporting evidence here. Based on this sample, we have evidence that the mean age of first marriage for all US women from 2006 to 2010 is greater than 23 years.

Confidence interval

```
t.test(x = age_at_marriage$age,
  alternative = "two.sided",
  mu = 23)$conf
```

```
[1] 23.3 23.6
attr("conf.level")
[1] 0.95
```

B.2.6 Comparing results

Observing the bootstrap distribution that were created, it makes quite a bit of sense that the results are so similar for traditional and non-traditional methods in terms of the p -value and the confidence interval since these distributions look very similar to normal distributions. The conditions also being met (the large sample size was the driver here) leads us to better guess that using any of the methods whether they are traditional (formula-based) or non-traditional (computational-based) will lead to similar results.

B.3 One proportion

B.3.1 Problem statement

The CEO of a large electric utility claims that 80 percent of his 1,000,000 customers are satisfied with the service they receive. To test this claim, the local newspaper surveyed 100 customers, using simple random sampling. 73 were satisfied and the remaining were unsatisfied. Based on these findings from the sample, can we reject the CEO's hypothesis that 80% of the customers are satisfied? [Tweaked a bit from <http://stattrek.com/hypothesis-test/proportion.aspx?Tutorial=AP>]

B.3.2 Competing hypotheses

In words

- Null hypothesis: The proportion of all customers of the large electric utility satisfied with service they receive is equal 0.80.
- Alternative hypothesis: The proportion of all customers of the large electric utility satisfied with service they receive is different from 0.80.

In symbols (with annotations)

- $H_0 : \pi = p_0$, where π represents the proportion of all customers of the large electric utility satisfied with service they receive and p_0 is 0.8.
- $H_A : \pi \neq 0.8$

Set α

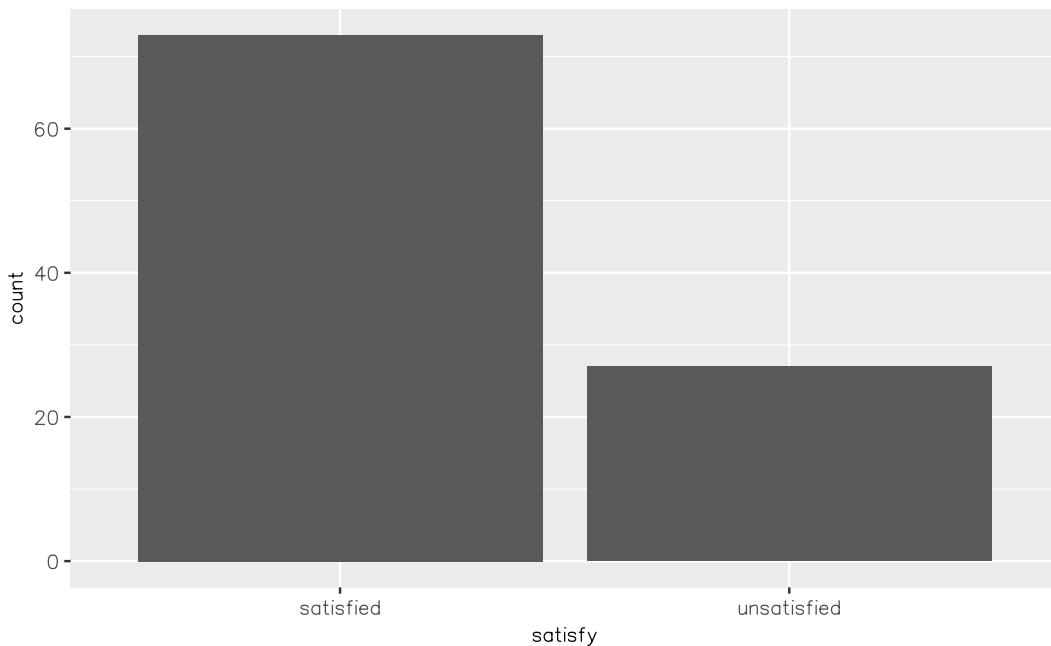
It's important to set the significance level before starting the testing using the data. Let's set the significance level at 5% here.

B.3.3 Exploring the sample data

```
elec <- c(rep("satisfied", 73), rep("unsatisfied", 27)) %>%
  as_data_frame() %>%
  rename(satisfy = value)
```

The bar graph below also shows the distribution of `satisfy`.

```
ggplot(data = elec, aes(x = satisfy)) +
  geom_bar()
```



The observed statistic is computed as

```
p_hat <- elec %>%
  specify(response = satisfy, success = "satisfied") %>%
  calculate(stat = "prop")
p_hat

# A tibble: 1 × 1
  stat
  <dbl>
1 0.73
```

Guess about statistical significance

We are looking to see if the sample proportion of 0.73 is statistically different from $p_0 = 0.8$ based on this sample. They seem to be quite close, and our sample size is not huge here ($n = 100$). Let's guess that we do not have evidence to reject the null hypothesis.

B.3.4 Non-traditional methods

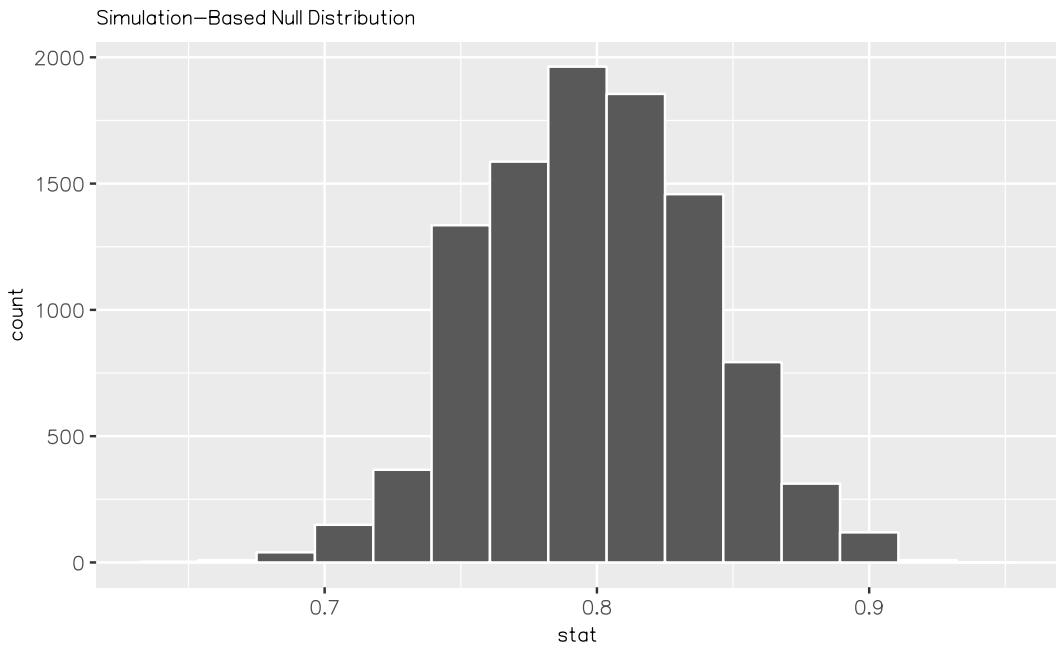
Simulation for hypothesis test

In order to look to see if 0.73 is statistically different from 0.8, we need to account for the sample size. We also need to determine a process that replicates how the original sample of size 100 was selected. We can use the idea of an unfair coin to *simulate* this process. We will simulate flipping an unfair coin (with probability of success 0.8 matching the null hypothesis) 100 times. Then we will keep track of how many heads come up in those 100 flips. Our simulated statistic matches with how we calculated the original statistic \hat{p} : the number of heads (satisfied) out of our total sample of 100. We then repeat this process many times (say 10,000) to create the null distribution looking at the simulated proportions of successes:

```
set.seed(2018)
null_distn_one_prop <- elec %>%
  specify(response = satisfy, success = "satisfied") %>%
  hypothesize(null = "point", p = 0.8) %>%
```

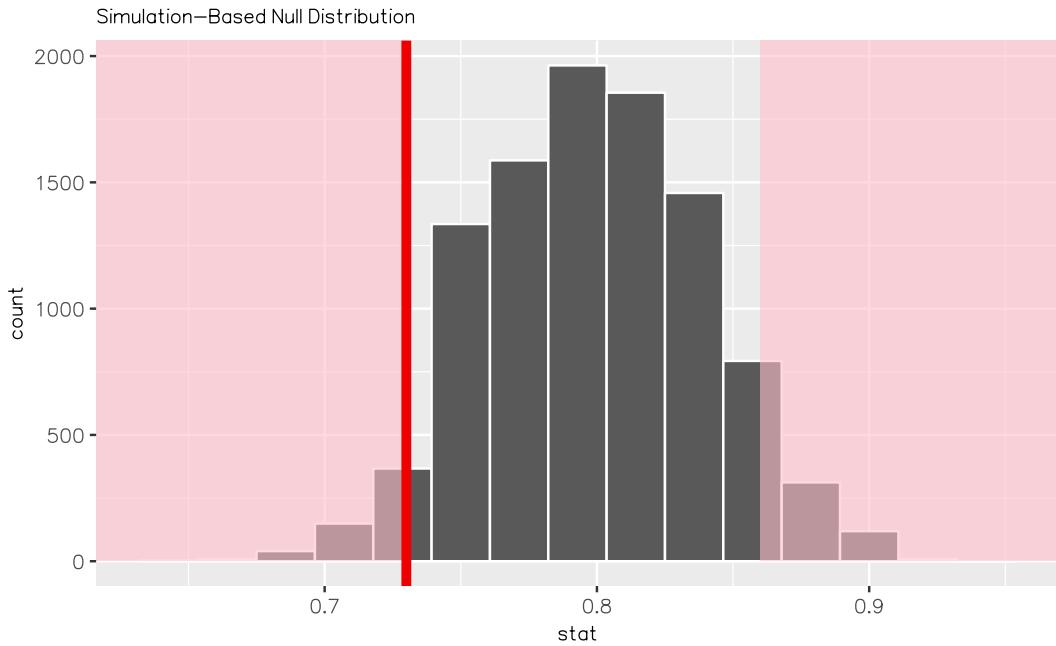
```
generate(reps = 10000) %>%  
calculate(stat = "prop")
```

```
null_distn_one_prop %>% visualize()
```



We can next use this distribution to observe our p -value. Recall this is a two-tailed test so we will be looking for values that are $0.8 - 0.73 = 0.07$ away from 0.8 in BOTH directions for our p -value:

```
null_distn_one_prop %>%  
visualize(obs_stat = p_hat, direction = "both")
```



B.3.4.0.1 Calculate p-value

```
pvalue <- null_distn_one_prop %>%
  get_pvalue(obs_stat = p_hat, direction = "both")
pvalue
```

```
# A tibble: 1 × 1
  p_value
  <dbl>
1 0.1136
```

So our p -value is 0.114 and we fail to reject the null hypothesis at the 5% level.

Bootstrapping for confidence interval

We can also create a confidence interval for the unknown population parameter π using our sample data. To do so, we use *bootstrapping*, which involves

1. sampling with replacement from our original sample of 100 survey respondents and repeating this process 10,000 times,
2. calculating the proportion of successes for each of the 10,000 bootstrap samples created in Step 1.,

3. combining all of these bootstrap statistics calculated in Step 2 into a `boot_distn` object,
4. identifying the 2.5th and 97.5th percentiles of this distribution (corresponding to the 5% significance level chosen) to find a 95% confidence interval for π , and
5. interpret this confidence interval in the context of the problem.

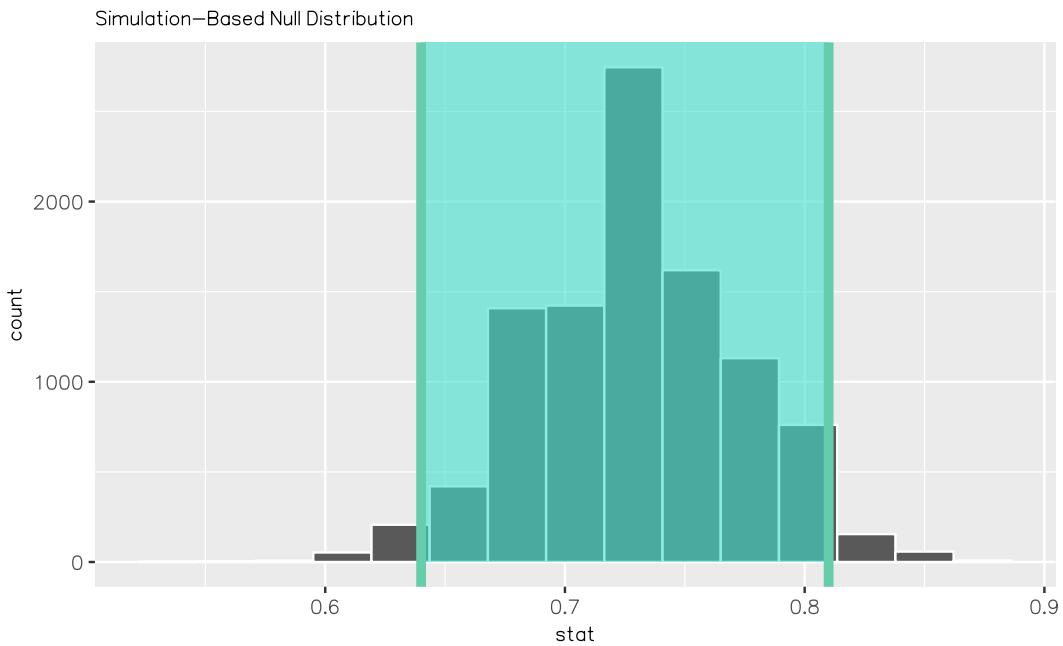
```
boot_distn_one_prop <- elec %>%
  specify(response = satisfy, success = "satisfied") %>%
  generate(reps = 10000) %>%
  calculate(stat = "prop")
```

Just as we use the `mean` function for calculating the mean over a numerical variable, we can also use it to compute the proportion of successes for a categorical variable where we specify what we are calling a “success” after the `==`. (Think about the formula for calculating a mean and how R handles logical statements such as `satisfy == "satisfied"` for why this must be true.)

```
ci <- boot_distn_one_prop %>%
  get_ci()
ci
```

```
# A tibble: 1 x 2
`2.5%` `97.5%
<dbl>    <dbl>
1     0.64     0.81
```

```
boot_distn_one_prop %>%
  visualize(endpoints = ci, direction = "between")
```



We see that 0.80 is contained in this confidence interval as a plausible value of π (the unknown population proportion). This matches with our hypothesis test results of failing to reject the null hypothesis.

Interpretation: We are 95% confident the true proportion of customers who are satisfied with the service they receive is between 0.64 and 0.81.

B.3.5 Traditional methods

Check conditions

Remember that in order to use the shortcut (formula-based, theoretical) approach, we need to check that some conditions are met.

1. *Independent observations:* The observations are collected independently.

The cases are selected independently through random sampling so this condition is met.

2. *Approximately normal:* The number of expected successes and expected failures is at least 10.

This condition is met since 73 and 27 are both greater than 10.

Test statistic

The test statistic is a random variable based on the sample data. Here, we want to look at a way to estimate the population proportion π . A good guess is the sample proportion \hat{P} . Recall that this sample proportion is actually a random variable that will vary as different samples are (theoretically, would be) collected. We are looking to see how likely is it for us to have observed a sample proportion of $\hat{p}_{obs} = 0.73$ or larger assuming that the population proportion is 0.80 (assuming the null hypothesis is true). If the conditions are met and assuming H_0 is true, we can standardize this original test statistic of \hat{P} into a Z statistic that follows a $N(0, 1)$ distribution.

$$Z = \frac{\hat{P} - p_0}{\sqrt{\frac{p_0(1 - p_0)}{n}}} \sim N(0, 1)$$

B.3.5.0.1 Observed test statistic

While one could compute this observed test statistic by “hand” by plugging the observed values into the formula, the focus here is on the set-up of the problem and in understanding which formula for the test statistic applies. The calculation has been done in R below for completeness though:

```
p_hat <- 0.73
p0 <- 0.8
n <- 100
(z_obs <- (p_hat - p0) / sqrt((p0 * (1 - p0)) / n))
```

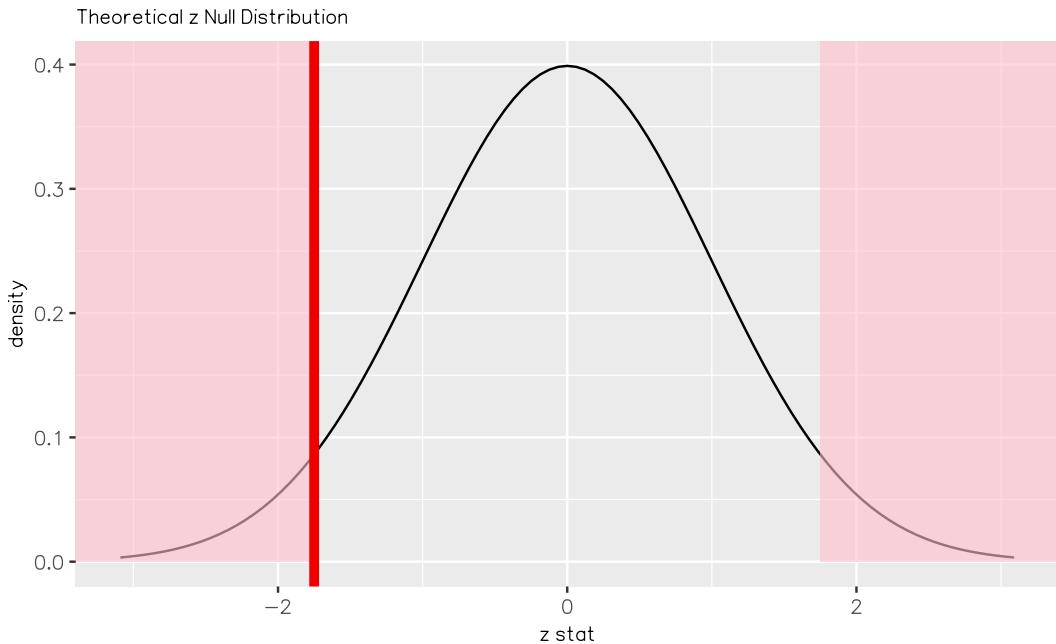
```
[1] -1.75
```

We see here that the z_{obs} value is around -1.75. Our observed sample proportion of 0.73 is 1.75 standard errors below the hypothesized parameter value of 0.8.

Visualize and compute p -value

```
elec %>%
  specify(response = satisfy, success = "satisfied") %>%
  hypothesize(null = "point", p = 0.8) %>%
```

```
calculate(stat = "z") %>%
visualize(method = "theoretical", obs_stat = z_obs, direction = "both")
```



```
2 * pnorm(z_obs)
```

```
[1] 0.0801
```

The p -value—the probability of observing an z_{obs} value of -1.75 or more extreme (in both directions) in our null distribution—is around 8%.

Note that we could also do this test directly using the `prop.test` function.

```
stats::prop.test(x = table(elec$satisfy),
n = length(elec$satisfy),
alternative = "two.sided",
p = 0.8,
correct = FALSE)
```

```
1-sample proportions test without continuity
correction

data: table(elec$satisfy), null probability 0.8
```

```
X-squared = 3, df = 1, p-value = 0.08
alternative hypothesis: true p is not equal to 0.8
95 percent confidence interval:
0.636 0.807
sample estimates:
p
0.73
```

prop.test does a χ^2 test here but this matches up exactly with what we would expect: $x_{obs}^2 = 3.06 = (-1.75)^2 = (z_{obs})^2$ and the p -values are the same because we are focusing on a two-tailed test.

Note that the 95 percent confidence interval given above matches well with the one calculated using bootstrapping.

State conclusion

We, therefore, do not have sufficient evidence to reject the null hypothesis. Our initial guess that our observed sample proportion was not statistically greater than the hypothesized proportion has not been invalidated. Based on this sample, we have do not evidence that the proportion of all customers of the large electric utility satisfied with service they receive is different from 0.80, at the 5% level.

B.3.6 Comparing results

Observing the bootstrap distribution and the null distribution that were created, it makes quite a bit of sense that the results are so similar for traditional and non-traditional methods in terms of the p -value and the confidence interval since these distributions look very similar to normal distributions. The conditions also being met leads us to better guess that using any of the methods whether they are traditional (formula-based) or non-traditional (computational-based) will lead to similar results.

B.4 Two proportions

B.4.1 Problem statement

A 2010 survey asked 827 randomly sampled registered voters in California “Do you support? Or do you oppose? Drilling for oil and natural gas off the Coast of California? Or do you not know enough to say?” Conduct a hypothesis test to determine if the data provide strong evidence that the proportion of college graduates who do not have an opinion on this issue is different than that of non-college graduates. (Tweaked a bit from Diez et al., 2014, [Chapter 6])

B.4.2 Competing hypotheses

In words

- Null hypothesis: There is no association between having an opinion on drilling and having a college degree for all registered California voters in 2010.
- Alternative hypothesis: There is an association between having an opinion on drilling and having a college degree for all registered California voters in 2010.

Another way in words

- Null hypothesis: The probability that a Californian voter in 2010 having no opinion on drilling and is a college graduate is the **same** as that of a non-college graduate.
- Alternative hypothesis: These parameter probabilities are different.

In symbols (with annotations)

- $H_0 : \pi_{college} = \pi_{no_college}$ or $H_0 : \pi_{college} - \pi_{no_college} = 0$, where π represents the probability of not having an opinion on drilling.
- $H_A : \pi_{college} - \pi_{no_college} \neq 0$

Set α

It's important to set the significance level before starting the testing using the data. Let's set the significance level at 5% here.

B.4.3 Exploring the sample data

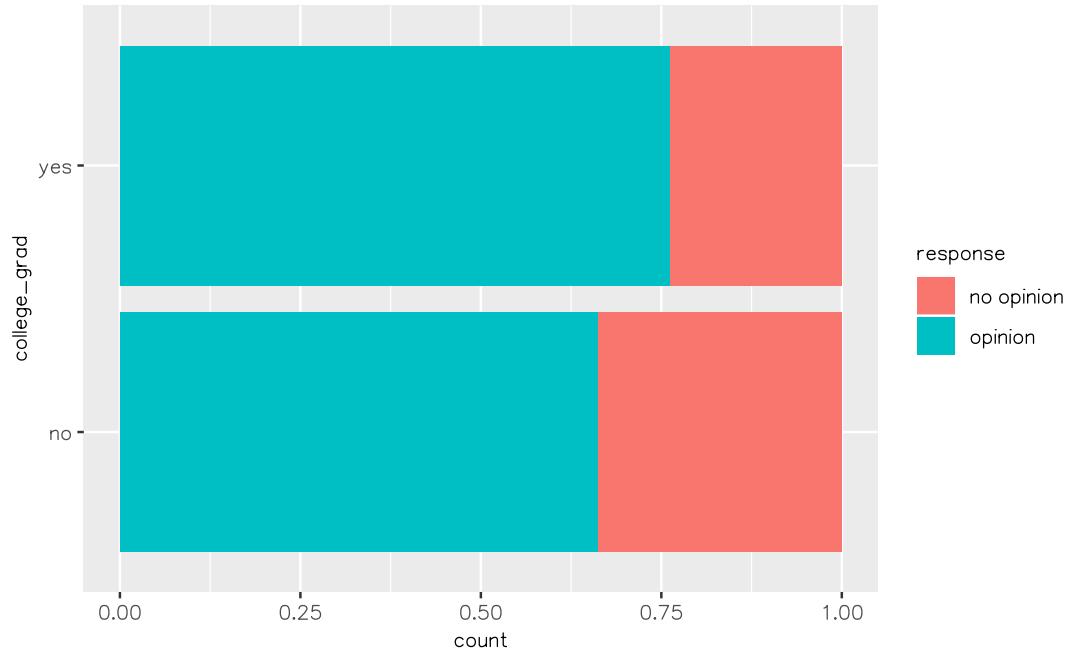
```
offshore <- read_csv("https://moderndive.com/data/offshore.csv")
```

```
offshore %>% tabyl(college_grad, response)
```

college_grad	no	opinion	opinion
no	131	258	
yes	104	334	

```
off_summ <- offshore %>%
  group_by(college_grad) %>%
  summarize(prop_no_opinion = mean(response == "no opinion"),
  sample_size = n())
```

```
ggplot(offshore, aes(x = college_grad, fill = response)) +
  geom_bar(position = "fill") +
  coord_flip()
```



Guess about statistical significance

We are looking to see if a difference exists in the size of the bars corresponding to `no opinion` for the plot. Based solely on the plot, we have little reason to believe that a difference exists since the bars seem to be about the same size, BUT...it's important to use statistics to see if that difference is actually statistically significant!

B.4.4 Non-traditional methods

Collecting summary info

The observed statistic is

```
d_hat <- offshore %>%
  specify(response ~ college_grad, success = "no opinion") %>%
  calculate(stat = "diff in props", order = c("yes", "no"))
d_hat

# A tibble: 1 × 1
  stat
  <dbl>
1 -0.0993180
```

Randomization for hypothesis test

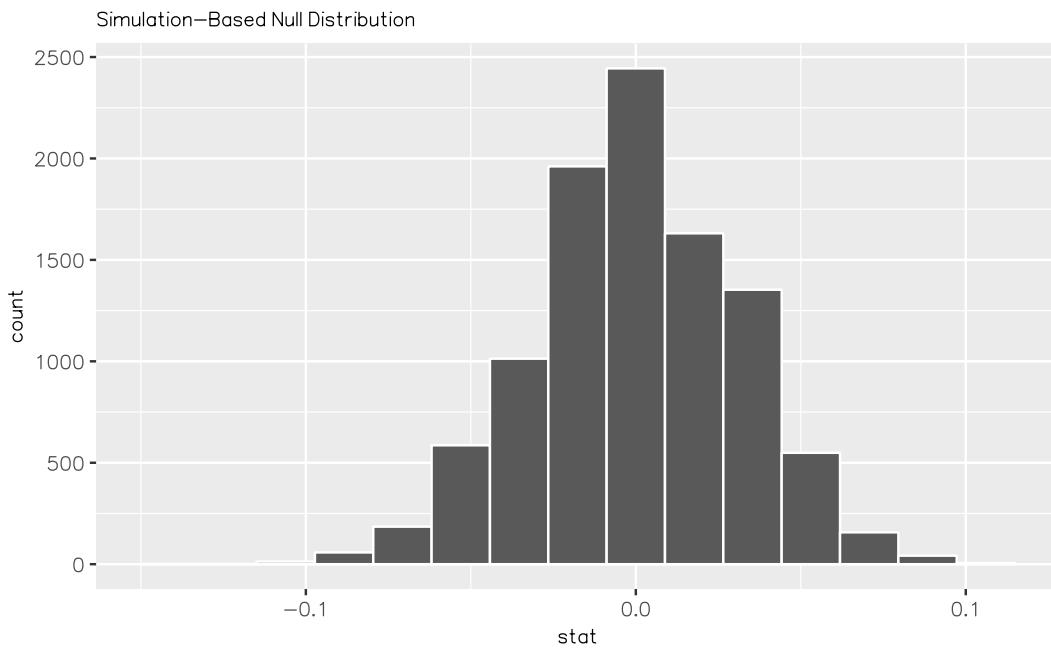
In order to look to see if the observed sample proportion of no opinion for college graduates of 0.337 is statistically different than that for graduates of 0.237, we need to account for the sample sizes. Note that this is the same as looking to see if $\hat{p}_{grad} - \hat{p}_{nograd}$ is statistically different than 0. We also need to determine a process that replicates how the original group sizes of 389 and 438 were selected.

We can use the idea of *randomization testing* (also known as *permutation testing*) to simulate the population from which the sample came (with two groups of different sizes) and then generate samples using *shuffling* from that simulated population to account for sampling variability.

```
set.seed(2018)
null_distn_two_props <- offshore %>%
```

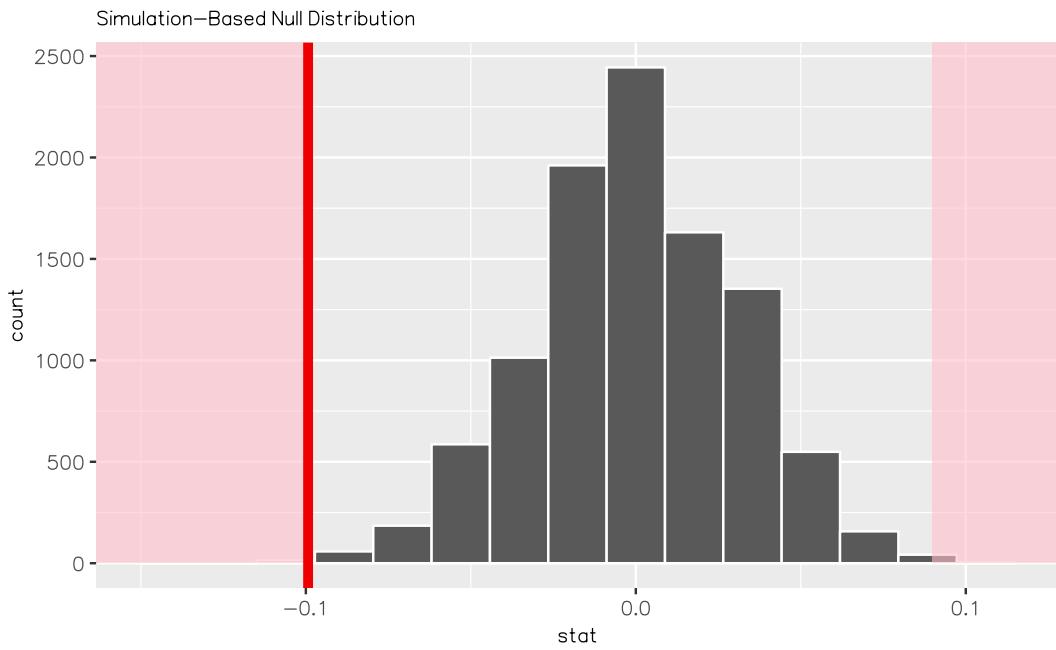
```
specify(response ~ college_grad, success = "no opinion") %>%
hypothesize(null = "independence") %>%
generate(reps = 10000) %>%
calculate(stat = "diff in props", order = c("yes", "no"))
```

```
null_distn_two_props %>% visualize()
```



We can next use this distribution to observe our p -value. Recall this is a two-tailed test so we will be looking for values that are greater than or equal to -0.099 or less than or equal to 0.099 for our p -value.

```
null_distn_two_props %>%
visualize(obs_stat = d_hat, direction = "two_sided")
```



B.4.4.0.1 Calculate p-value

```
pvalue <- null_distn_two_props %>%
  get_pvalue(obs_stat = d_hat, direction = "two_sided")
pvalue

# A tibble: 1 × 1
  p_value
  <dbl>
1 0.003
```

So our p -value is 0.003 and we reject the null hypothesis at the 5% level. You can also see this from the histogram above that we are far into the tails of the null distribution.

Bootstrapping for confidence interval

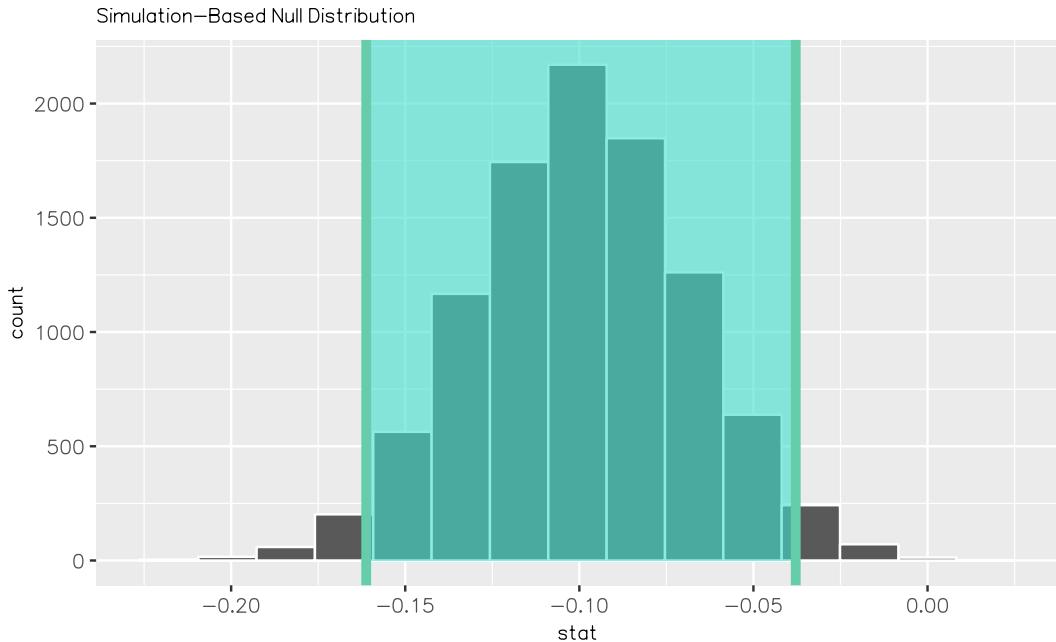
We can also create a confidence interval for the unknown population parameter $\pi_{college} - \pi_{no_college}$ using our sample data with *bootstrapping*.

```
boot_distn_two_props <- offshore %>%
  specify(response ~ college_grad, success = "no opinion") %>%
  generate(reps = 10000) %>%
  calculate(stat = "diff in props", order = c("yes", "no"))
```

```
ci <- boot_distn_two_props %>%
  get_ci()
ci
```

```
# A tibble: 1 x 2
`2.5%`    `97.5%
<dbl>      <dbl>
1 -0.161207 -0.0378500
```

```
boot_distn_two_props %>%
  visualize(endpoints = ci, direction = "between")
```



We see that 0 is not contained in this confidence interval as a plausible value of $\pi_{college} - \pi_{no_college}$ (the unknown population parameter). This matches with our hypothesis test results of rejecting the null hypothesis. Since zero is not a plausible value of the population parameter, we have evidence that the

proportion of college graduates in California with no opinion on drilling is different than that of non-college graduates.

Interpretation: We are 95% confident the true proportion of non-college graduates with no opinion on offshore drilling in California is between 0.16 dollars smaller to 0.04 dollars smaller than for college graduates.

B.4.5 Traditional methods

B.4.6 Check conditions

Remember that in order to use the short-cut (formula-based, theoretical) approach, we need to check that some conditions are met.

1. *Independent observations:* Each case that was selected must be independent of all the other cases selected.

This condition is met since cases were selected at random to observe.

2. *Sample size:* The number of pooled successes and pooled failures must be at least 10 for each group.

We need to first figure out the pooled success rate:

$$\hat{p}_{obs} = \frac{131 + 104}{827} = 0.28.$$

We now determine expected (pooled) success and failure counts:

$$0.28 \cdot (131 + 258) = 108.92, 0.72 \cdot (131 + 258) = 280.08$$

$$0.28 \cdot (104 + 334) = 122.64, 0.72 \cdot (104 + 334) = 315.36$$

3. *Independent selection of samples:* The cases are not paired in any meaningful way.

We have no reason to suspect that a college graduate selected would have any relationship to a non-college graduate selected.

B.4.7 Test statistic

The test statistic is a random variable based on the sample data. Here, we are interested in seeing if our observed difference in sample proportions corresponding to no opinion on drilling ($\hat{p}_{college,obs} - \hat{p}_{no_college,obs} = 0.033$) is statistically different than 0. Assuming that conditions are met and the null hypothesis is true, we can use the standard normal distribution to standardize

the difference in sample proportions ($\hat{P}_{college} - \hat{P}_{no_college}$) using the standard error of $\hat{P}_{college} - \hat{P}_{no_college}$ and the pooled estimate:

$$Z = \frac{(\hat{P}_1 - \hat{P}_2) - 0}{\sqrt{\frac{\hat{P}(1 - \hat{P})}{n_1} + \frac{\hat{P}(1 - \hat{P})}{n_2}}} \sim N(0, 1)$$

where $\hat{P} = \frac{\text{total number of successes}}{\text{total number of cases}}$.

Observed test statistic

While one could compute this observed test statistic by “hand”, the focus here is on the set-up of the problem and in understanding which formula for the test statistic applies. We can use the `prop.test` function to perform this analysis for us.

```
z_hat <- offshore %>%
  specify(response ~ college_grad, success = "no opinion") %>%
  calculate(stat = "z", order = c("yes", "no"))

z_hat
```

```
# A tibble: 1 × 1
  stat
  <dbl>
1 -3.16081
```

The observed difference in sample proportions is 3.16 standard deviations smaller than 0.

The p -value—the probability of observing a Z value of -3.16 or more extreme in our null distribution—is 0.0016. This can also be calculated in R directly:

```
2 * pnorm(-3.16, lower.tail = TRUE)
```

```
[1] 0.00158
```

B.4.8 State conclusion

We, therefore, have sufficient evidence to reject the null hypothesis. Our initial guess that a statistically significant difference did not exist in the proportions of no opinion on offshore drilling between college educated and non-college

educated Californians was not validated. We do have evidence to suggest that there is a dependency between college graduation and position on offshore drilling for Californians.

B.4.9 Comparing results

Observing the bootstrap distribution and the null distribution that were created, it makes quite a bit of sense that the results are so similar for traditional and non-traditional methods in terms of the p -value and the confidence interval since these distributions look very similar to normal distributions. The conditions were not met since the number of pairs was small, but the sample data was not highly skewed. Using any of the methods whether they are traditional (formula-based) or non-traditional (computational-based) lead to similar results.

B.5 Two means (independent samples)

B.5.1 Problem statement

Average income varies from one region of the country to another, and it often reflects both lifestyles and regional living expenses. Suppose a new graduate is considering a job in two locations, Cleveland, OH and Sacramento, CA, and he wants to see whether the average income in one of these cities is higher than the other. He would like to conduct a hypothesis test based on two randomly selected samples from the 2000 Census. (Tweaked a bit from Diez et al., 2014, [Chapter 5])

B.5.2 Competing hypotheses

In words

- Null hypothesis: There is no association between income and location (Cleveland, OH and Sacramento, CA).

- Alternative hypothesis: There is an association between income and location (Cleveland, OH and Sacramento, CA).

Another way in words

- Null hypothesis: The mean income is the **same** for both cities.
- Alternative hypothesis: The mean income is different for the two cities.

In symbols (with annotations)

- $H_0 : \mu_{sac} = \mu_{cle}$ or $H_0 : \mu_{sac} - \mu_{cle} = 0$, where μ represents the average income.
- $H_A : \mu_{sac} - \mu_{cle} \neq 0$

Set α

It's important to set the significance level before starting the testing using the data. Let's set the significance level at 5% here.

B.5.3 Exploring the sample data

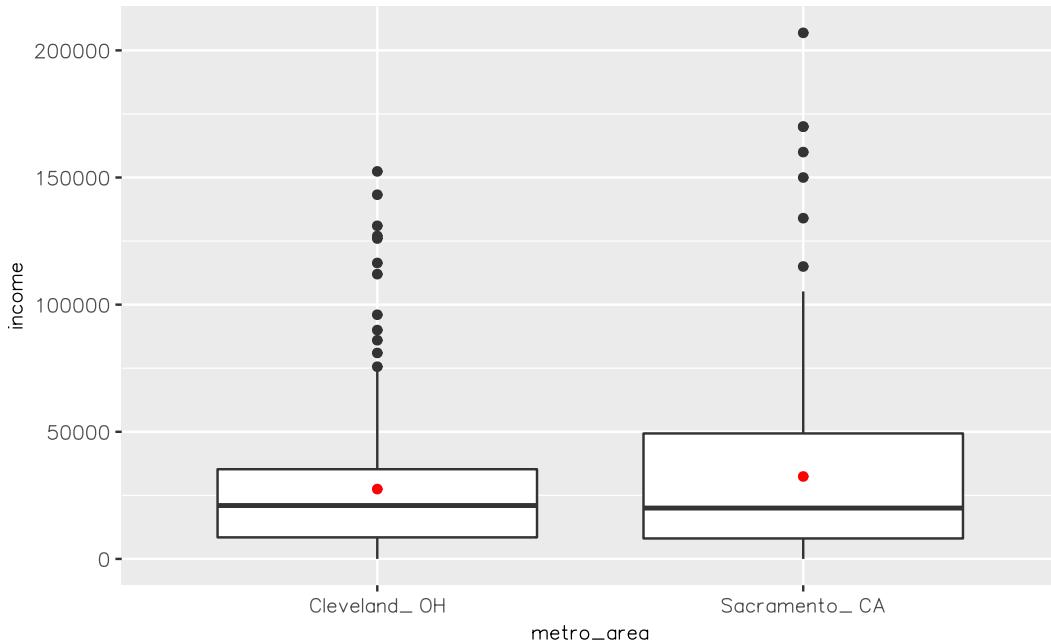
```
cle_sac <- read.delim("https://moderndive.com/data/cleSac.txt") %>%
  rename(metro_area = Metropolitan_area_Detailed,
         income = Total_personal_income) %>%
  na.omit()
```

```
inc_summ <- cle_sac %>% group_by(metro_area) %>%
  summarize(sample_size = n(),
            mean = mean(income),
            sd = sd(income),
            minimum = min(income),
            lower_quartile = quantile(income, 0.25),
            median = median(income),
            upper_quartile = quantile(income, 0.75),
            max = max(income))
kable(inc_summ) %>%
  kable_styling(font_size = ifelse(knitr:::is_latex_output(), 10, 16),
                latex_options = c("HOLD_position"))
```

metro_area	sample_size	mean	sd	minimum	lower_quartile	median	upper_quartile	max
Cleveland_ OH	212	27467	27681	0	8475	21000	35275	152400
Sacramento_ CA	175	32428	35774	0	8050	20000	49350	206900

The boxplot below also shows the mean for each group highlighted by the red dots.

```
ggplot(cle_sac, aes(x = metro_area, y = income)) +
  geom_boxplot() +
  stat_summary(fun.y = "mean", geom = "point", color = "red")
```



Guess about statistical significance

We are looking to see if a difference exists in the mean income of the two levels of the explanatory variable. Based solely on the boxplot, we have reason to believe that no difference exists. The distributions of income seem similar and the means fall in roughly the same place.

B.5.4 Non-traditional methods

Collecting summary info

We now compute the observed statistic:

```
d_hat <- cle_sac %>%
  specify(income ~ metro_area) %>%
  calculate(stat = "diff in means",
             order = c("Sacramento_ CA", "Cleveland_ OH"))
d_hat

# A tibble: 1 × 1
  stat
  <dbl>
1 4960.48
```

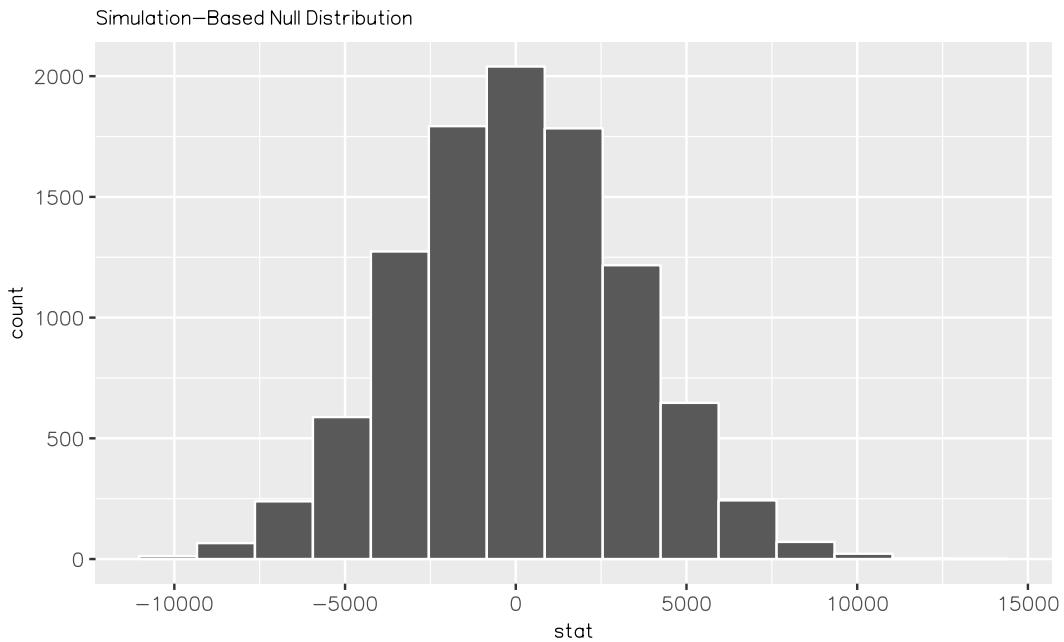
Randomization for hypothesis test

In order to look to see if the observed sample mean for Sacramento of 27467.066 is statistically different than that for Cleveland of 32427.543, we need to account for the sample sizes. Note that this is the same as looking to see if $\bar{x}_{sac} - \bar{x}_{cle}$ is statistically different than 0. We also need to determine a process that replicates how the original group sizes of 212 and 175 were selected.

We can use the idea of *randomization testing* (also known as *permutation testing*) to simulate the population from which the sample came (with two groups of different sizes) and then generate samples using *shuffling* from that simulated population to account for sampling variability.

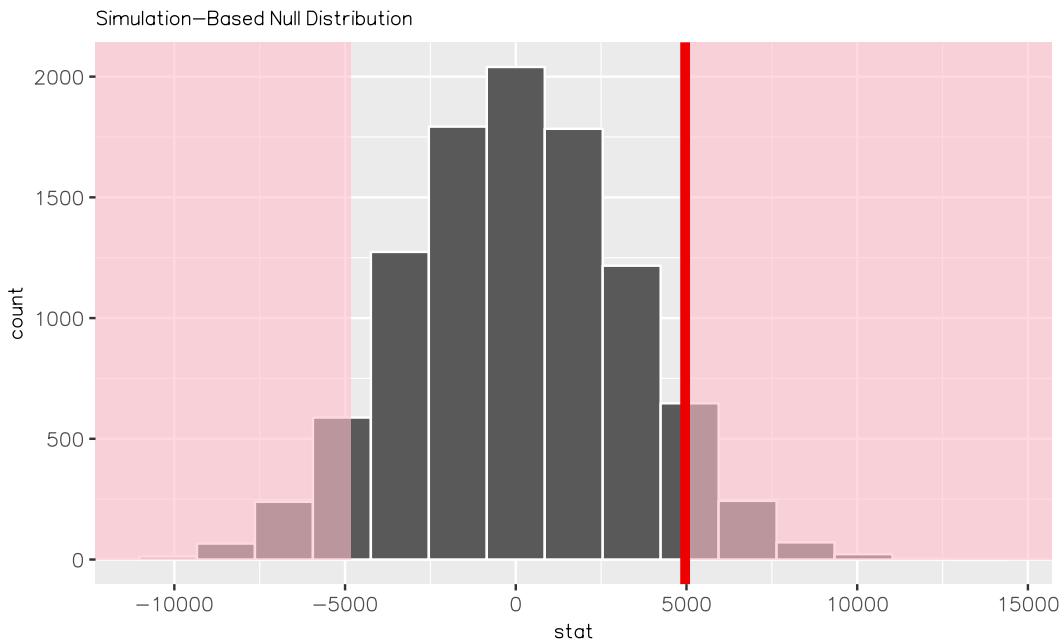
```
set.seed(2018)
null_distn_two_means <- cle_sac %>%
  specify(income ~ metro_area) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 10000) %>%
  calculate(stat = "diff in means",
            order = c("Sacramento_ CA", "Cleveland_ OH"))

null_distn_two_means %>% visualize()
```



We can next use this distribution to observe our p -value. Recall this is a two-tailed test so we will be looking for values that are greater than or equal to 4960.477 or less than or equal to -4960.477 for our p -value.

```
null_distn_two_means %>%
  visualize(obs_stat = d_hat, direction = "both")
```



B.5.4.0.1 Calculate p-value

```
pvalue <- null_distn_two_means %>%
  get_pvalue(obs_stat = d_hat, direction = "both")
pvalue
```

```
# A tibble: 1 × 1
  p_value
  <dbl>
1 0.1298
```

So our p -value is 0.13 and we fail to reject the null hypothesis at the 5% level. You can also see this from the histogram above that we are not very far into the tail of the null distribution.

Bootstrapping for confidence interval

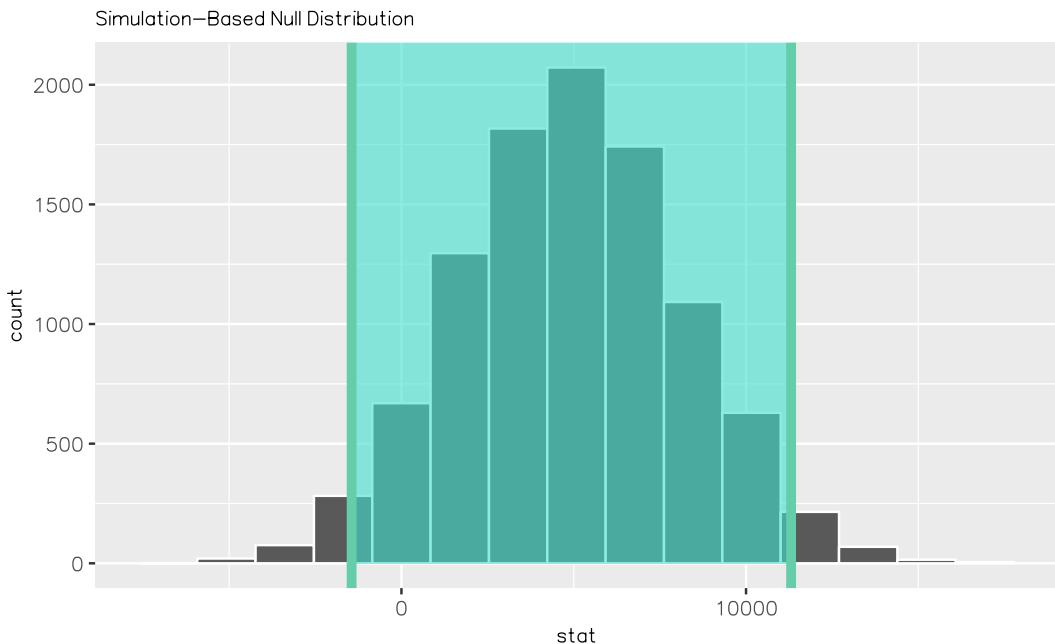
We can also create a confidence interval for the unknown population parameter $\mu_{sac} - \mu_{cle}$ using our sample data with *bootstrapping*. Here we will bootstrap each of the groups with replacement instead of shuffling. This is done using the `groups` argument in the `resample` function to fix the size of each group to be the same as the original group sizes of 175 for Sacramento and 212 for Cleveland.

```
boot_distn_two_means <- cle_sac %>%
  specify(income ~ metro_area) %>%
  generate(reps = 10000) %>%
  calculate(stat = "diff in means",
             order = c("Sacramento_ CA", "Cleveland_ OH"))
```

```
ci <- boot_distn_two_means %>%
  get_ci()
ci
```

```
# A tibble: 1 × 2
  `2.5%` `97.5%`
  <dbl>    <dbl>
1 -1445.53 11307.8
```

```
boot_distn_two_means %>%
  visualize(endpoints = ci, direction = "between")
```



We see that 0 is contained in this confidence interval as a plausible value of $\mu_{sac} - \mu_{cle}$ (the unknown population parameter). This matches with our hypothesis test results of failing to reject the null hypothesis. Since zero is a plausible value of the population parameter, we do not have evidence that Sacramento incomes are different than Cleveland incomes.

Interpretation: We are 95% confident the true mean yearly income for those living in Sacramento is between 1445.53 dollars smaller to 11307.82 dollars higher than for Cleveland.

Note: You could also use the null distribution based on randomization with a shift to have its center at $\bar{x}_{sac} - \bar{x}_{cle} = \4960.48 instead of at 0 and calculate its percentiles. The confidence interval produced via this method should be comparable to the one done using bootstrapping above.

B.5.5 Traditional methods

B.5.5.0.1 Check conditions

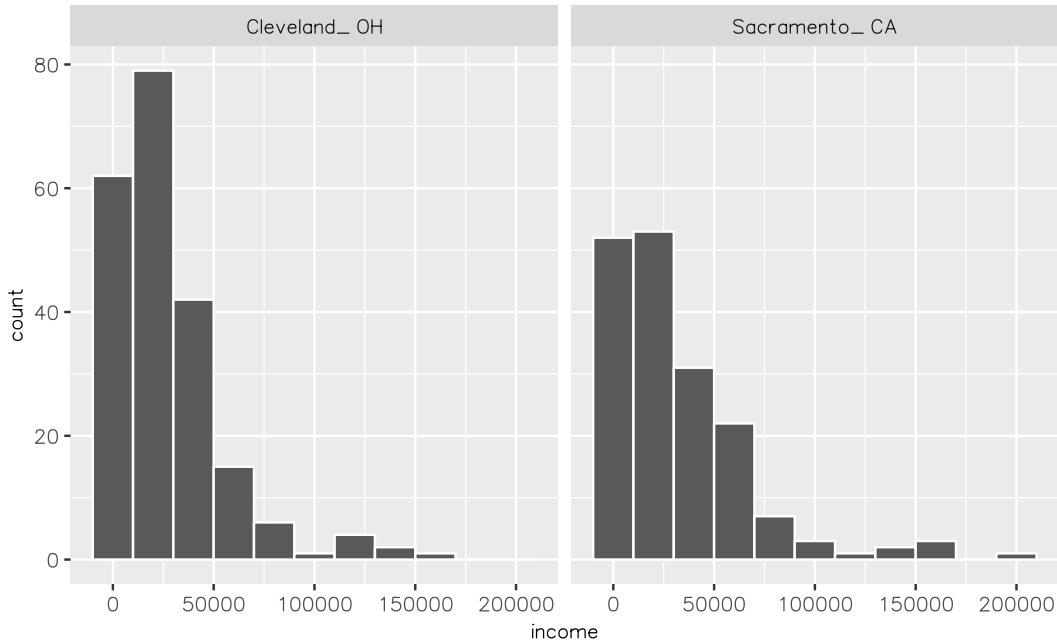
Remember that in order to use the short-cut (formula-based, theoretical) approach, we need to check that some conditions are met.

1. *Independent observations:* The observations are independent in both groups.

This `metro_area` variable is met since the cases are randomly selected from each city.

2. *Approximately normal:* The distribution of the response for each group should be normal or the sample sizes should be at least 30.

```
ggplot(cle_sac, aes(x = income)) +
  geom_histogram(color = "white", binwidth = 20000) +
  facet_wrap(~ metro_area)
```



We have some reason to doubt the normality assumption here since both the histograms show deviation from a normal model fitting the data well for each

group. The sample sizes for each group are greater than 100 though so the assumptions should still apply.

3. *Independent samples:* The samples should be collected without any natural pairing.

There is no mention of there being a relationship between those selected in Cleveland and in Sacramento.

B.5.6 Test statistic

The test statistic is a random variable based on the sample data. Here, we are interested in seeing if our observed difference in sample means ($\bar{x}_{sac,obs} - \bar{x}_{cle,obs} = 4960.477$) is statistically different than 0. Assuming that conditions are met and the null hypothesis is true, we can use the t distribution to standardize the difference in sample means ($\bar{X}_{sac} - \bar{X}_{cle}$) using the approximate standard error of $\bar{X}_{sac} - \bar{X}_{cle}$ (invoking S_{sac} and S_{cle} as estimates of unknown σ_{sac} and σ_{cle}).

$$T = \frac{(\bar{X}_1 - \bar{X}_2) - 0}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}} \sim t(df = \min(n_1 - 1, n_2 - 1))$$

where 1 = Sacramento and 2 = Cleveland with S_1^2 and S_2^2 the sample variance of the incomes of both cities, respectively, and $n_1 = 175$ for Sacramento and $n_2 = 212$ for Cleveland.

Observed test statistic

Note that we could also do (ALMOST) this test directly using the `t.test` function. The `x` and `y` arguments are expected to both be numeric vectors here so we'll need to appropriately filter our datasets.

```
cle_sac %>%
  specify(income ~ metro_area) %>%
  calculate(stat = "t",
             order = c("Cleveland_ OH", "Sacramento_ CA"))

# A tibble: 1 x 1
  stat
  <dbl>
1 -1.50062
```

We see here that the observed test statistic value is around -1.5.

While one could compute this observed test statistic by “hand”, the focus here is on the set-up of the problem and in understanding which formula for the test statistic applies.

B.5.7 Compute p -value

The p -value—the probability of observing an t_{174} value of -1.501 or more extreme (in both directions) in our null distribution—is 0.13. This can also be calculated in R directly:

```
2 * pt(-1.501, df = min(212 - 1, 175 - 1), lower.tail = TRUE)
```

```
[1] 0.135
```

We can also approximate by using the standard normal curve:

```
2 * pnorm(-1.501)
```

```
[1] 0.133
```

Note that the 95 percent confidence interval given above matches well with the one calculated using bootstrapping.

B.5.8 State conclusion

We, therefore, do not have sufficient evidence to reject the null hypothesis. Our initial guess that a statistically significant difference not existing in the means was backed by this statistical analysis. We do not have evidence to suggest that the true mean income differs between Cleveland, OH and Sacramento, CA based on this data.

B.5.9 Comparing results

Observing the bootstrap distribution and the null distribution that were created, it makes quite a bit of sense that the results are so similar for traditional and non-traditional methods in terms of the p -value and the confidence interval since these distributions look very similar to normal distributions. The conditions also being met leads us to better guess that using any of

the methods whether they are traditional (formula-based) or non-traditional (computational-based) will lead to similar results.

B.6 Two means (paired samples)

Problem statement

Trace metals in drinking water affect the flavor and an unusually high concentration can pose a health hazard. Ten pairs of data were taken measuring zinc concentration in bottom water and surface water at 10 randomly selected locations on a stretch of river. Do the data suggest that the true average concentration in the surface water is smaller than that of bottom water? (Note that units are not given.) [Tweaked a bit from <https://onlinecourses.science.psu.edu/stat500/node/51>]

B.6.1 Competing hypotheses

In words

- Null hypothesis: The mean concentration in the bottom water is the same as that of the surface water at different paired locations.
- Alternative hypothesis: The mean concentration in the surface water is smaller than that of the bottom water at different paired locations.

In symbols (with annotations)

- $H_0 : \mu_{diff} = 0$, where μ_{diff} represents the mean difference in concentration for surface water minus bottom water.
- $H_A : \mu_{diff} < 0$

Set α

It's important to set the significance level before starting the testing using the data. Let's set the significance level at 5% here.

B.6.2 Exploring the sample data

```
zinc_tidy <- read_csv("https://moderndive.com/data/zinc_tidy.csv")
```

We want to look at the differences in `surface - bottom` for each location:

```
zinc_diff <- zinc_tidy %>%
  group_by(loc_id) %>%
  summarize(pair_diff = diff(concentration)) %>%
  ungroup()
```

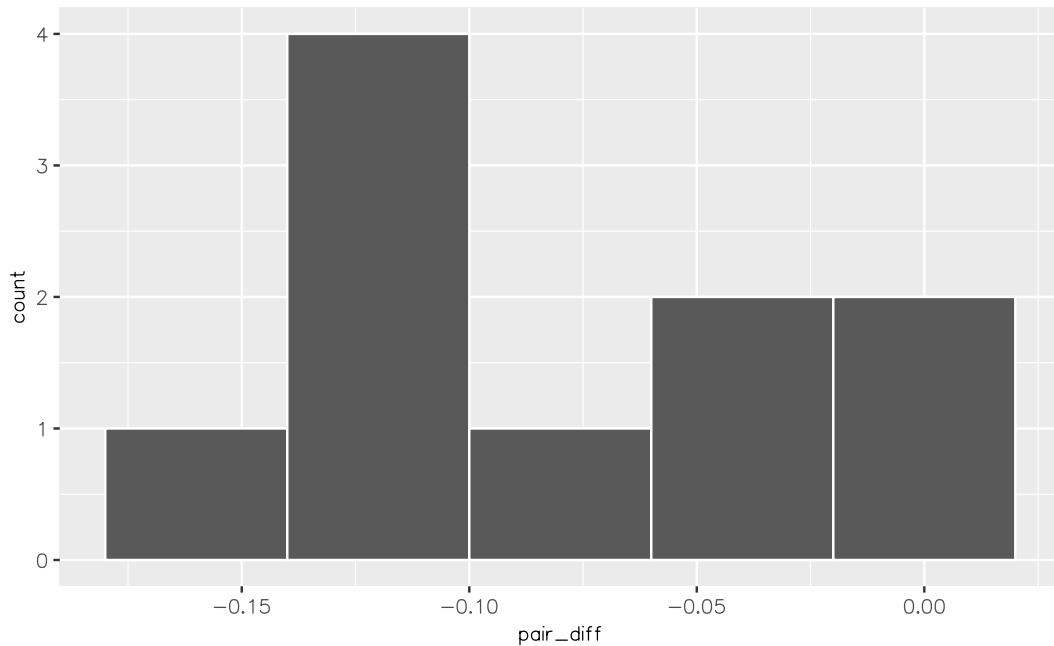
Next we calculate the mean difference as our observed statistic:

```
d_hat <- zinc_diff %>%
  specify(response = pair_diff) %>%
  calculate(stat = "mean")
d_hat
```

```
# A tibble: 1 x 1
  stat
  <dbl>
1 -0.0804
```

The histogram below also shows the distribution of `pair_diff`.

```
ggplot(zinc_diff, aes(x = pair_diff)) +
  geom_histogram(binwidth = 0.04, color = "white")
```



Guess about statistical significance

We are looking to see if the sample paired mean difference of -0.08 is statistically less than 0. They seem to be quite close, but we have a small number of pairs here. Let's guess that we will fail to reject the null hypothesis.

B.6.3 Non-traditional methods

Bootstrapping for hypothesis test

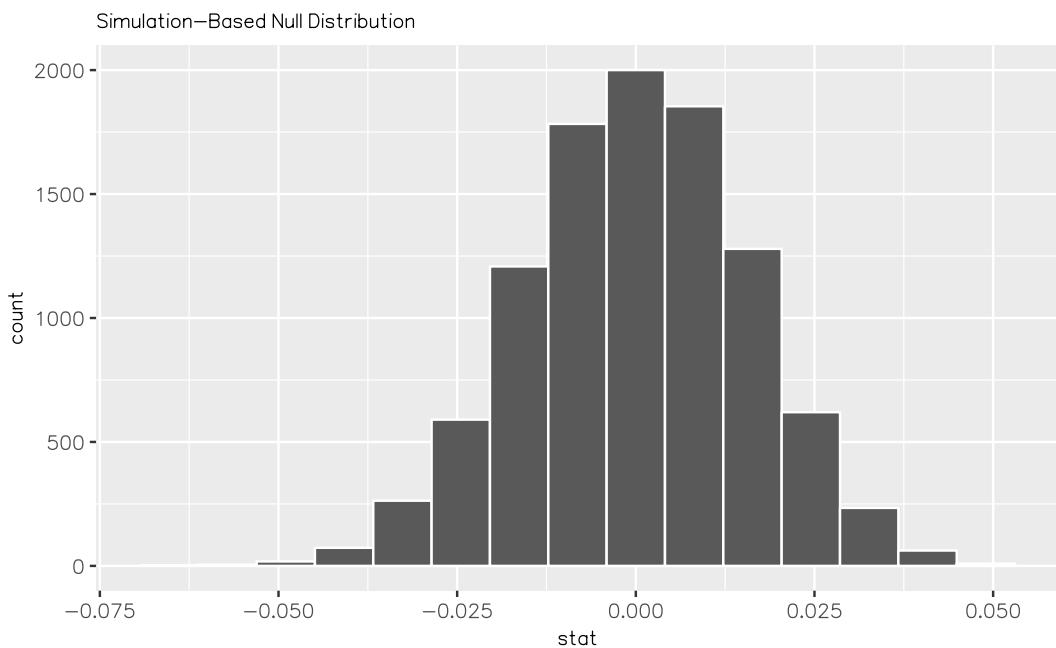
In order to look to see if the observed sample mean difference $\bar{x}_{diff} = 4960.477$ is statistically less than 0, we need to account for the number of pairs. We also need to determine a process that replicates how the paired data was selected in a way similar to how we calculated our original difference in sample means.

Treating the differences as our data of interest, we next use the process of **bootstrapping** to build other simulated samples and then calculate the mean of the bootstrap samples. We hypothesize that the mean difference is zero.

This process is similar to comparing the One Mean example seen above, but using the differences between the two groups as a single sample with a hypothesized mean difference of 0.

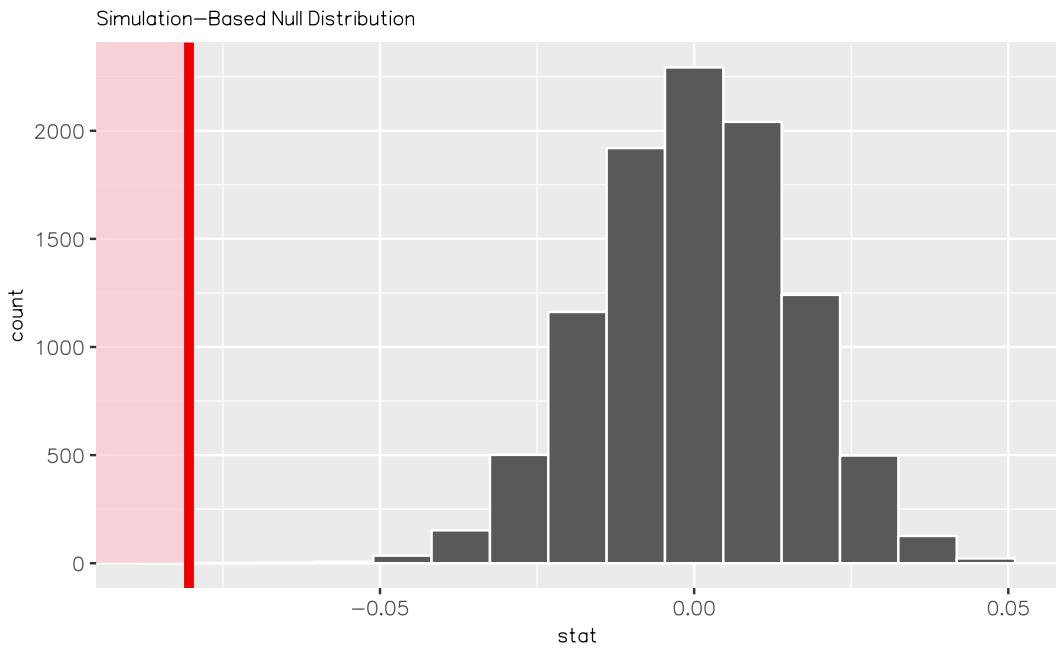
```
set.seed(2018)
null_distn_paired_means <- zinc_diff %>%
  specify(response = pair_diff) %>%
  hypothesize(null = "point", mu = 0) %>%
  generate(reps = 10000) %>%
  calculate(stat = "mean")
```

```
null_distn_paired_means %>% visualize()
```



We can next use this distribution to observe our p -value. Recall this is a left-tailed test so we will be looking for values that are less than or equal to 4960.477 for our p -value.

```
null_distn_paired_means %>%
  visualize(obs_stat = d_hat, direction = "less")
```



B.6.3.0.1 Calculate p-value

```
pvalue <- null_distn_paired_means %>%
  get_pvalue(obs_stat = d_hat, direction = "less")
pvalue

# A tibble: 1 × 1
  p_value
  <dbl>
1      0
```

So our p -value is essentially 0 and we reject the null hypothesis at the 5% level. You can also see this from the histogram above that we are far into the left tail of the null distribution.

Bootstrapping for confidence interval

We can also create a confidence interval for the unknown population parameter μ_{diff} using our sample data (the calculated differences) with *bootstrapping*. This is similar to the bootstrapping done in a one sample mean case, except now our data is differences instead of raw numerical data. Note that this code

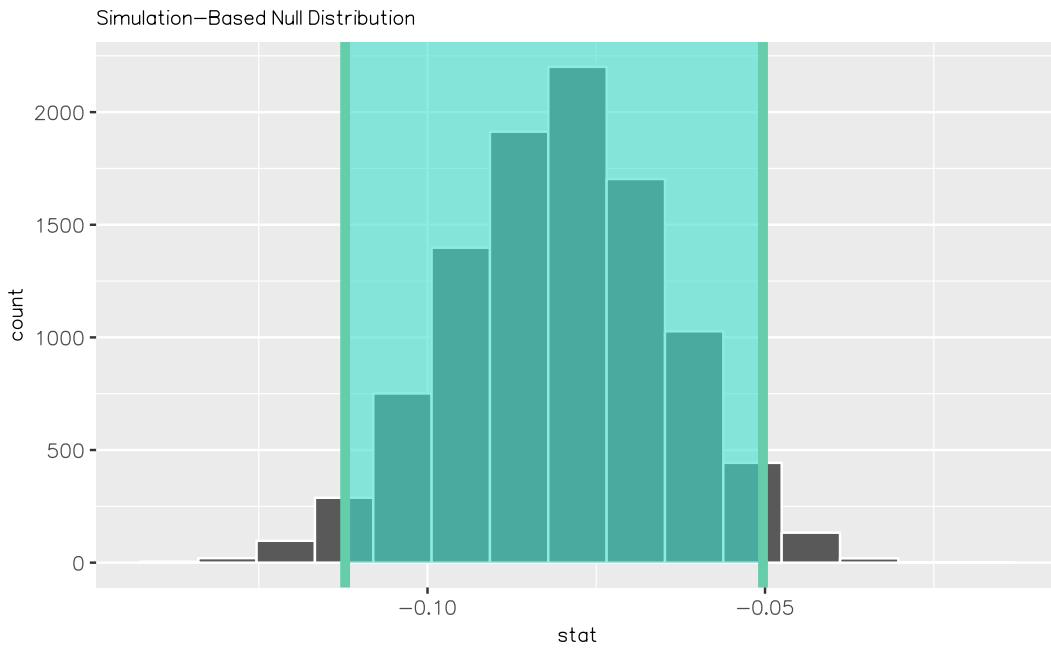
is identical to the pipeline shown in the hypothesis test above except the `hypothesize()` function is not called.

```
boot_distn_paired_means <- zinc_diff %>%
  specify(response = pair_diff) %>%
  generate(reps = 10000) %>%
  calculate(stat = "mean")
```

```
ci <- boot_distn_paired_means %>%
  get_ci()
ci
```

```
# A tibble: 1 × 2
  `2.5%` `97.5%`
  <dbl>    <dbl>
1 -0.112200 -0.0503
```

```
boot_distn_paired_means %>%
  visualize(endpoints = ci, direction = "between")
```



We see that 0 is not contained in this confidence interval as a plausible value of μ_{diff} (the unknown population parameter). This matches with our hypothesis test results of rejecting the null hypothesis. Since zero is not a plausible value of

the population parameter and since the entire confidence interval falls below zero, we have evidence that surface zinc concentration levels are lower, on average, than bottom level zinc concentrations.

Interpretation: We are 95% confident the true mean zinc concentration on the surface is between 0.11 units smaller to 0.05 units smaller than on the bottom.

B.6.4 Traditional methods

Check conditions

Remember that in order to use the shortcut (formula-based, theoretical) approach, we need to check that some conditions are met.

1. *Independent observations:* The observations among pairs are independent.

The locations are selected independently through random sampling so this condition is met.

2. *Approximately normal:* The distribution of population of differences is normal or the number of pairs is at least 30.

The histogram above does show some skew so we have reason to doubt the population being normal based on this sample. We also only have 10 pairs which is fewer than the 30 needed. A theory-based test may not be valid here.

Test statistic

The test statistic is a random variable based on the sample data. Here, we want to look at a way to estimate the population mean difference μ_{diff} . A good guess is the sample mean difference \bar{X}_{diff} . Recall that this sample mean is actually a random variable that will vary as different samples are (theoretically, would be) collected. We are looking to see how likely is it for us to have observed a sample mean of $\bar{x}_{diff,obs} = 0.0804$ or larger assuming that the population mean difference is 0 (assuming the null hypothesis is true). If the conditions are met and assuming H_0 is true, we can “standardize” this original test statistic of \bar{X}_{diff} into a T statistic that follows a t distribution with degrees of freedom equal to $df = n - 1$:

$$T = \frac{\bar{X}_{diff} - 0}{S_{diff}/\sqrt{n}} \sim t(df = n - 1)$$

where S represents the standard deviation of the sample differences and n is the number of pairs.

B.6.4.0.1 Observed test statistic

While one could compute this observed test statistic by “hand”, the focus here is on the set-up of the problem and in understanding which formula for the test statistic applies. We can use the `t_test` function on the differences to perform this analysis for us.

```
t_test_results <- zinc_diff %>%
  infer::t_test(formula = pair_diff ~ NULL,
                alternative = "less",
                mu = 0)
t_test_results

# A tibble: 1 x 6
  statistic  t_df   p_value alternative lower_ci   upper_ci
  <dbl>     <dbl>    <dbl>     <chr>      <dbl>     <dbl>
1 -4.86381     9 4.45558e-4 less        -Inf -0.0500982
```

We see here that the t_{obs} value is -4.864.

Compute p -value

The p -value—the probability of observing a t_{obs} value of -4.864 or less in our null distribution of a t with 9 degrees of freedom—is 0. This can also be calculated in R directly:

```
pt(-4.8638, df = nrow(zinc_diff) - 1, lower.tail = TRUE)
```

```
[1] 0.000446
```

State conclusion

We, therefore, have sufficient evidence to reject the null hypothesis. Our initial guess that our observed sample mean difference was not statistically less than

the hypothesized mean of 0 has been invalidated here. Based on this sample, we have evidence that the mean concentration in the bottom water is greater than that of the surface water at different paired locations.

B.6.5 Comparing results

Observing the bootstrap distribution and the null distribution that were created, it makes quite a bit of sense that the results are so similar for traditional and non-traditional methods in terms of the p -value and the confidence interval since these distributions look very similar to normal distributions. The conditions were not met since the number of pairs was small, but the sample data was not highly skewed. Using any of the methods whether they are traditional (formula-based) or non-traditional (computational-based) lead to similar results here.

C

Reach for the Stars

Needed packages

```
library(dplyr)
library(ggplot2)
library(knitr)
library(dygraphs)
library(nycflights13)
```

C.1 Sorted barplots

Building upon the example in Section 3.8:

```
flights_table <- table(flights$carrier)
flights_table
```

9E	AA	AS	B6	DL	EV	F9	FL	HA	MQ
18460	32729	714	54635	48110	54173	685	3260	342	26397
00	UA	US	VX	WN	YV				
32	58665	20536	5162	12275	601				

We can sort this table from highest to lowest counts by using the `sort` function:

```
sorted_flights <- sort(flights_table, decreasing = TRUE)
names(sorted_flights)
```

```
[1] "UA" "B6" "EV" "DL" "AA" "MQ" "US" "9E" "WN" "VX" "FL"
```

```
[12] "AS" "F9" "YV" "HA" "OO"
```

It is often preferred for barplots to be ordered corresponding to the heights of the bars. This allows the reader to more easily compare the ordering of different airlines in terms of departed flights (Robbins, 2013). We can also much more easily answer questions like “How many airlines have more departing flights than Southwest Airlines?”.

We can use the sorted table giving the number of flights defined as `sorted_flights` to **reorder** the `carrier`.

```
ggplot(data = flights, mapping = aes(x = carrier)) +
  geom_bar() +
  scale_x_discrete(limits = names(sorted_flights))
```

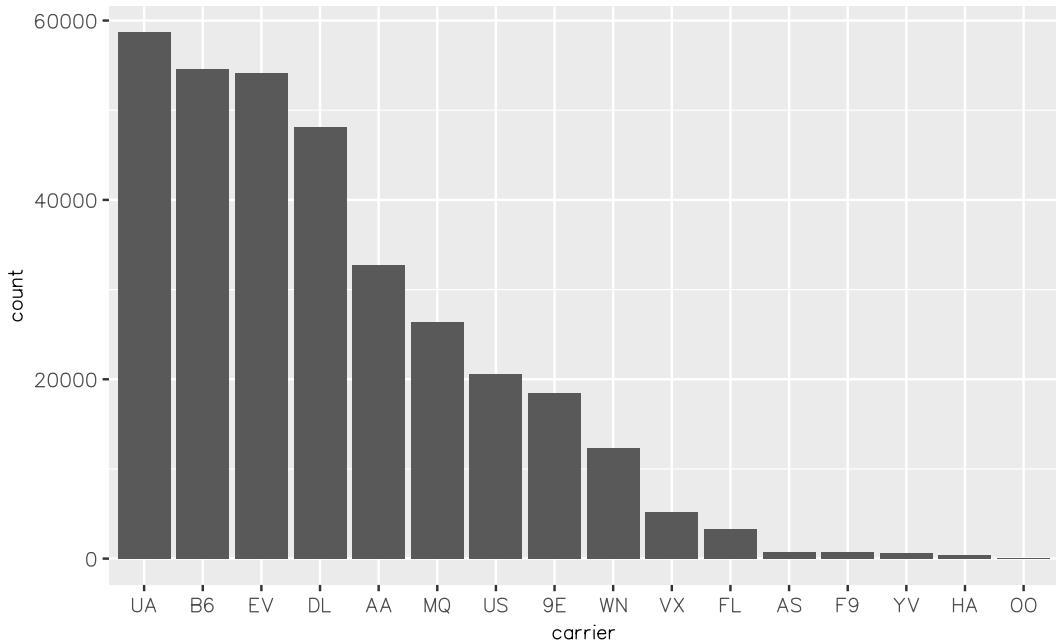


FIGURE C.1: Number of flights departing NYC in 2013 by airline - Descending numbers

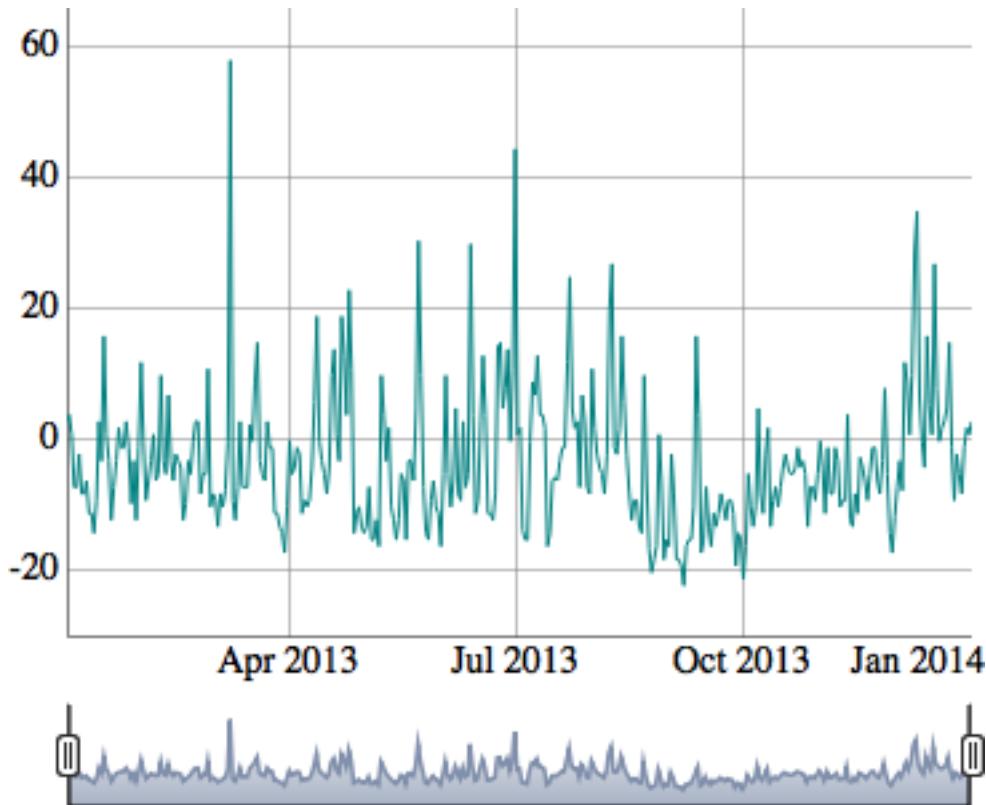
The last addition here specifies the values of the horizontal x axis on a discrete scale to correspond to those given by the entries of `sorted_flights`.

C.2 Interactive graphics

C.2.1 Interactive linegraphs

Another useful tool for viewing linegraphs such as this is the `dygraph` function in the `dygraphs` package in combination with the `dyRangeSelector` function. This allows us to zoom in on a selected range and get an interactive plot for us to work with:

```
library(dygraphs)
flights_day <- mutate(flights, date = as.Date(time_hour))
flights_summarized <- flights_day %>%
  group_by(date) %>%
  summarize(median_arr_delay = median(arr_delay, na.rm = TRUE))
rownames(flights_summarized) <- flights_summarized$date
flights_summarized <- select(flights_summarized, -date)
dyRangeSelector(dygraph(flights_summarized))
```



The syntax here is a little different than what we have covered so far. The `dygraph` function is expecting for the dates to be given as the `rownames` of the object. We then remove the `date` variable from the `flights_summarized` data frame since it is accounted for in the `rownames`. Lastly, we run the `dygraph` function on the new data frame that only contains the median arrival delay as a column and then provide the ability to have a selector to zoom in on the interactive plot via `dyRangeSelector`. (Note that this plot will only be interactive in the HTML version of this book.)

D

Learning Check Solutions

D.1 Chapter 2 Solutions

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

(LC2.1) Repeat the above 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.

(LC2.2) “Load” the `dplyr`, `nycflights13`, and `knitr` packages as well by repeating the above steps.

Solution: If the following code runs with no errors, you’ve succeeded!

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

(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

Solution: This is data on a flight. Not a flight path! Example:

- a flight path would be United 1545 to Houston
- a flight would be United 1545 to Houston at a specific date/time. For example: 2013/1/1 at 5:15am.

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

Solution: Hint: Type `?flights` in the console to see what all the variables mean!

- Categorical:
 - `carrier` the company
 - `dest` the destination
 - `flight` the flight number. Even though this is a number, its simply a label. Example United 1545 is not less than United 1714
- Quantitative:
 - `distance` the distance in miles
 - `time_hour` time

(LC2.5) What does `int`, `dbl`, and `chr` mean in the output above?

Solution:

- `int`: integer. Used to count things i.e. a discrete value. Ex: the # of cars parked in a lot
- `dbl`: double. Used to measure things. i.e. a continuous value. Ex: your height in inches
- `chr`: character. i.e. text

(LC2.6) 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.

Solution: `lat` `long` represent the airport geographic coordinates, `alt` is the altitude above sea level of the airport (Run `airports %>% filter(faa == "DEN")` to see the altitude of Denver International Airport), `tz` is the time zone difference with respect to GMT in London UK, `dst` is the daylight savings time zone, and `tzone` is the time zone label.

(LC2.7) 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 dataset that matches these conditions.

Solution:

- In the `weather` example in LC3.8, the combination of `origin`, `year`, `month`, `day`, `hour` are identification variables as they identify the observation in question.
 - Anything else pertains to observations: `temp`, `humid`, `wind_speed`, etc.
-

D.2 Chapter 3 Solutions

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

(LC3.1) Take a look at both the `flights` and `alaska_flights` data frames by running `View(flights)` and `View(alaska_flights)` in the console. In what respect do these data frames differ?

Solution: `flights` contains all flight data, while `alaska_flights` contains only data from Alaskan carrier “AS”. We can see that `flights` has 336776 rows while `alaska_flights` has only 714

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

Solution: The later a plane departs, typically the later it will arrive.

(LC3.3) What variables (not necessarily in the `flights` data frame) would you expect to have a negative correlation (i.e. a negative relationship) with `dep_delay`? Why? Remember that we are focusing on numerical variables here.

Solution: An example in the `weather` dataset is `visibility`, which measures visibility in miles. As visibility increases, we would expect departure delays to decrease.

(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?

Solution: The point (0,0) means no delay in departure nor arrival. From the point of view of Alaska airlines, this means the flight was on time. It seems most flights are at least close to being on time.

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

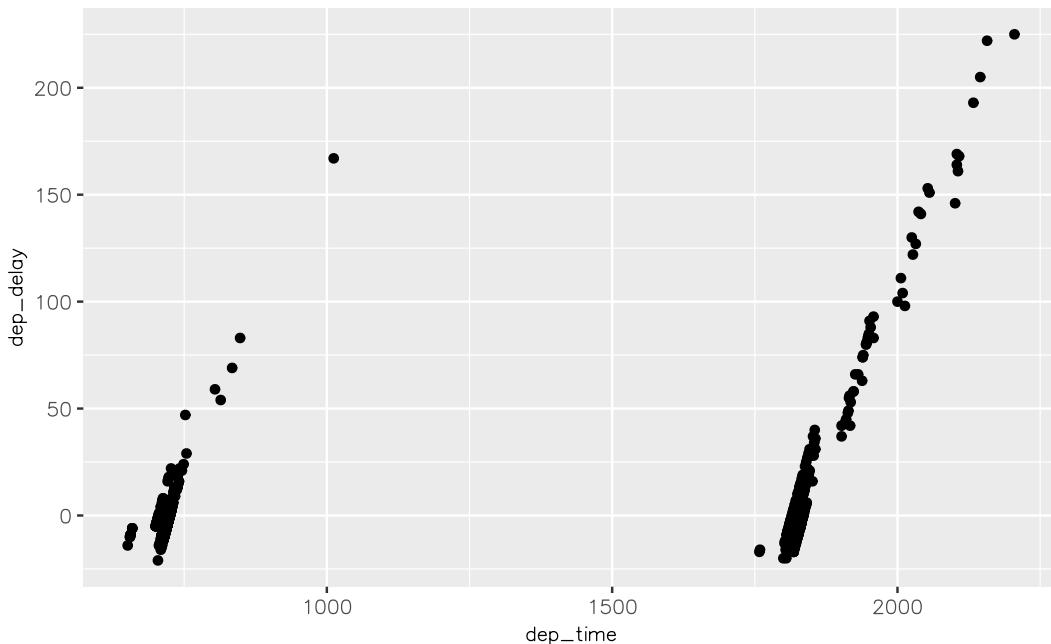
Solution: Different people will answer this one differently. One answer is most flights depart and arrive less than an hour late.

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

Solution: Many possibilities for this one, see the plot below. Is there a pat-

tern in departure delay depending on when the flight is scheduled to depart? Interestingly, there seems to be only two blocks of time where flights depart.

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



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

Solution: Why is setting the `alpha` argument value useful with scatterplots? What further information does it give you that a regular scatterplot cannot? *It thins out the points so we address overplotting. But more importantly it hints at the (statistical) **density** and **distribution** of the points: where are the points concentrated, where do they occur. We will see more about densities and distributions in Chapter 6 when we switch gears to statistical topics.*

(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?

Solution: After viewing the Figure 3.4 above, give a range of arrival delays and departure delays that occur 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? *The lower plot suggests that most Alaska flights from NYC depart*

between 12 minutes early and on time and arrive between 50 minutes early and on time.

(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 the console. In what respect do these data frames differ?

Solution: Take a look at both the `weather` and `early_january_weather` data frames by running `view(weather)` and `view(early_january_weather)` in the console. In what respect do these data frames differ? *The rows of `early_january_weather` are a subset of `weather`.*

(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?

Solution: `view()` the `flights` data frame again. Why does the `time_hour` variable correctly identify the hour of the measurement whereas the `hour` variable does not? *Because to uniquely identify an hour, we need the year/month/day/hour sequence, whereas there are only 24 possible hour's.*

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

Solution: Why should linegraphs be avoided when there is not a clear ordering of the horizontal axis? *Because lines suggest connectedness and ordering.*

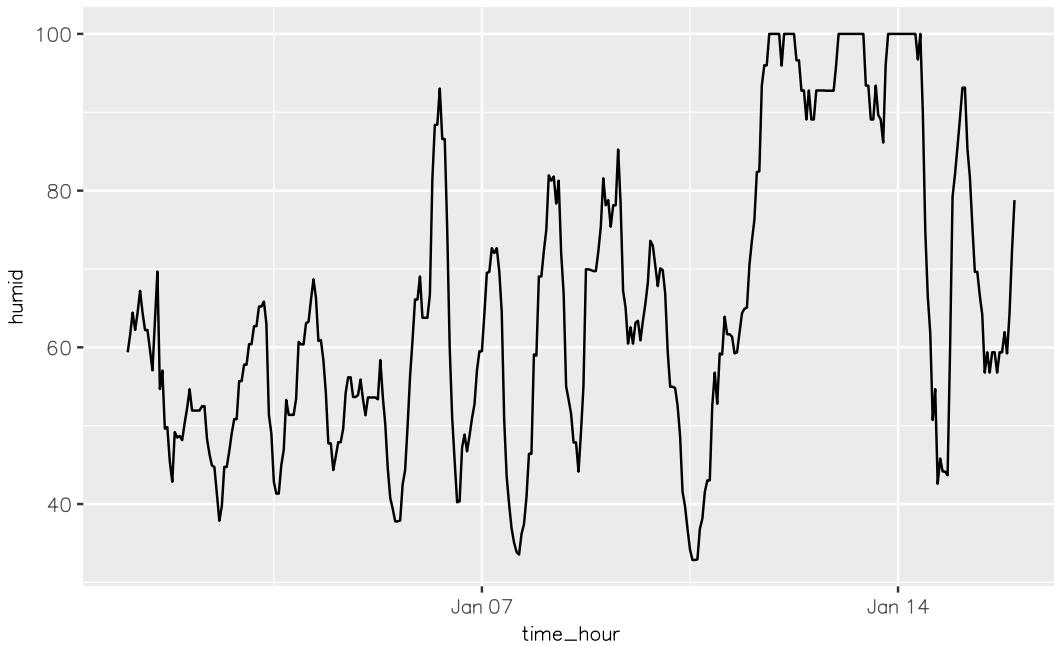
(LC3.12) Why are linegraphs frequently used when time is the explanatory variable?

Solution: Why are linegraphs frequently used when time is the explanatory variable? *Because time is sequential: subsequent observations are closely related to each other.*

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

Solution: Plot a time series of a variable other than `temp` for Newark Airport in the first 15 days of January 2013. *Humidity is a good one to look at, since this very closely related to the cycles of a day.*

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



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

Solution: The distribution doesn't change much. But by refining the bin width, we see that the temperature data has a high degree of accuracy. What do I mean by accuracy? Looking at the `temp` variable by `view(weather)`, we see that the precision of each temperature recording is 2 decimal places.

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

Solution: It is rather symmetric, i.e. there are no **long tails** on only one side of the distribution

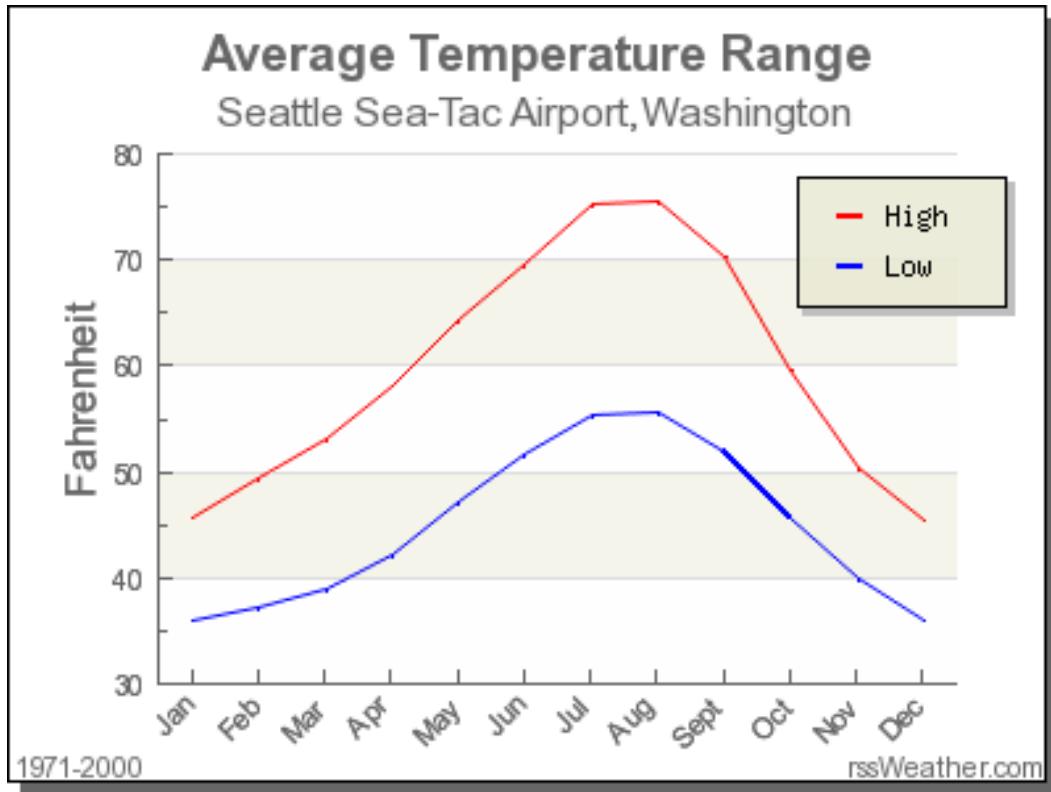
(LC3.16) What would you guess is the “center” value in this distribution? Why did you make that choice?

Solution: The center is around 55.26°F. By running the `summary()` command, we see that the mean and median are very similar. In fact, when the distribution is symmetric the mean equals the median.

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

Solution: This can only be answered relatively speaking! Let's pick things to be relative to Seattle, WA temperatures:

While, it appears that Seattle weather has a similar center of 55°F, its temperatures are almost entirely between 35°F and 75°F for a range of about 40°F.

**FIGURE D.1**

Seattle temperatures are much less spread out than New York i.e. much more consistent over the year. New York on the other hand has much colder days in the winter and much hotter days in the summer. Expressed differently, the middle 50% of values, as delineated by the interquartile range is 30°F:

(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?

Solution:

- Certain months have much more consistent weather (August in particular), while others have crazy variability like January and October, representing changes in the seasons.
- Because we see `temp` recordings split by `month`, we are considering the relationship between these two variables. For example, for example for summer months, temperatures tend to be higher.

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

Solution:

- While month is technically a number between 1-12, we're viewing it as a categorical variable here. Specifically an **ordinal categorical** variable since there is a ordering to the categories
- 25, 50, 75, 100 are temperatures

(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.

Solution:

- We'd have 365 facets to look at. Way too many.
- We don't really care about day-to-day fluctuation in weather so much, but maybe more week-to-week variation. We'd like to focus on seasonal trends.

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

Solution: Again, like in LC (LC3.17), this is a relative question. I would say yes, because in New York City, you have 4 clear seasons with different weather. Whereas in Seattle WA and Portland OR, you have two seasons: summer and rain!

(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.

Solution: It appears to be an outlier. Let's revisit the use of the `filter` command to hone in on it. We want all data points where the `month` is 5 and `temp < 25`

```
weather %>%
  filter(month==5 & temp < 25)

# A tibble: 1 x 16
  origin year month   day hour  temp  dewp humid wind_dir
  <chr>   <dbl> <dbl> <int> <int> <dbl> <dbl> <dbl>    <dbl>
1 JFK      2013     5     8    22  13.1 12.02 95.34      80
# ... with 7 more variables: wind_speed <dbl>,
#   wind_gust <dbl>, precip <dbl>, pressure <dbl>,
#   visib <dbl>, time_hour <dttm>, temp_in_C <dbl>
```

There appears to be only one hour and only at JFK that recorded 13.1 F (-10.5 C) in the month of May. This is probably a data entry mistake! Why wasn't the weather at least similar at EWR (Newark) and LGA (La Guardia)?

(LC3.23) Which months have the highest variability in temperature? What reasons do you think this is?

Solution: We are now interested in the **spread** of the data. One measure some of you may have seen previously is the standard deviation. But in this plot we can read off the Interquartile Range (IQR):

- The distance from the 1st to the 3rd quartiles i.e. the length of the boxes
- You can also think of this as the spread of the **middle 50%** of the data

Just from eyeballing it, it seems

- November has the biggest IQR, i.e. the widest box, so has the most variation in temperature
- August has the smallest IQR, i.e. the narrowest box, so is the most consistent temperature-wise

Here's how we compute the exact IQR values for each month (we'll see this more in depth Chapter 5 of the text):

1. group the observations by `month` then
2. for each group, i.e. `month`, summarize it by applying the summary statistic function `IQR()`, while making sure to skip over missing data via `na.rm=TRUE` then
3. arrange the table in descending order of `IQR`

```
weather %>%
  group_by(month) %>%
  summarize(IQR = IQR(temp, na.rm=TRUE)) %>%
  arrange(desc(IQR))
```

month	IQR
11	16.02
12	14.04
1	13.77
9	12.06
4	12.06
5	11.88
6	10.98
10	10.98
2	10.08
7	9.18
3	9.00
8	7.02

(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?

Solution: Because there are 12 unique values of `month` yielding only 12 boxes in our boxplot. There are many more unique values of `pressure` (469 unique values in fact), because values are to the first decimal place. This would lead to 469 boxes, which is too many for people to digest.

(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?

Solution: In a histogram, the bin corresponding to where an outlier lies may not be high enough for us to see. In a boxplot, they are explicitly labelled separately.

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

Solution: Histograms are for numerical variables i.e. the horizontal part of each histogram bar represents an interval, whereas for a categorical variable each bar represents only one level of the categorical variable.

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

Solution: See above.

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

Solution: Envoy Air is carrier code `MQ` and thus 26397 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?

Solution: What a pain! We'll see in Chapter 5 on Data Wrangling that applying `arrange(desc(n))` will sort this table in descending order of `n!`

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

Solution: In our **opinion**, comparisons using horizontal lines are easier than comparing angles and areas of circles.

(LC3.31) What is your opinion as to why pie charts continue to be used?

Solution: Legacy?

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

Solution: Because the red, green, and blue bars don't all start at 0 (only red does), it makes comparing counts hard.

(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?

Solution: The different airlines prefer different airports. For example, United is mostly a Newark carrier and JetBlue is a JFK carrier. If airlines didn't prefer airports, each color would be roughly one third of each bar.}

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

Solution: We can easily compare the different airports for a given carrier using a single comparison line i.e. things are lined up

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

Solution: It is hard to get totals for each airline.

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

Solution: Not that different than using side-by-side; depends on how you want to organize your presentation.

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

Solution: Now we can also compare the different carriers **within** a particular airport easily too. For example, we can read off who the top carrier for each airport is easily using a single horizontal line.

D.3 Chapter 4 Solutions

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

(LC4.1) What's another way 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.

Solution:

```
# Original in book
not_BTV_SEA <- flights %>%
  filter(!(dest == "BTV" | dest == "SEA"))

# Alternative way
not_BTV_SEA <- flights %>%
  filter(!dest == "BTV" & !dest == "SEA")

# Yet another way
not_BTV_SEA <- flights %>%
  filter(dest != "BTV" & dest != "SEA")
```

(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?

Solution: The missing patients may have died of lung cancer! So to ignore them might seriously **bias** your results! It is very important to think of what the consequences on your analysis are of ignoring missing data! Ask yourself:

- There is a systematic reasons why certain values are missing? If so, you might be biasing your results!
- If there isn’t, then it might be ok to “sweep missing values under the rug.”

(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?

Solution: It corresponds to a count of the number of observations/rows:

```
weather %>%  
  summarize(count = n())
```

```
# A tibble: 1 × 1  
  count  
  <int>  
1 26115
```

(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))
```

Solution: Consider the output of only running the first two lines:

```
weather %>%  
  summarize(mean = mean(temp, na.rm = TRUE))
```

```
# A tibble: 1 × 1  
  mean  
  <dbl>  
1 55.2604
```

Because after the first `summarize()`, the variable `temp` disappears as it has been collapsed to the value `mean`. So when we try to run the second `summarize()`, it can't find the variable `temp` to compute the standard deviation of.

(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?

Solution:

The standard deviation is a quantification of **spread** and **variability**. We see

that the period in November, December, and January has the most variation in weather, so you can expect very different temperatures on different days.

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

Solution:

Note: `group_by(day)` is not enough, because `day` is a value between 1-31. We need to `group_by(year, month, day)`

```
library(dplyr)
library(nycflights13)

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

(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?

Solution:

```
by_monthly_origin
```

In `by_monthly_origin` the `month` column is now first and the rows are sorted by `month` instead of `origin`. If you compare the values of `count` in `by_origin_monthly` and `by_monthly_origin` using the `view()` function, you'll see that the values are actually the same, just presented in a different order.

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

Solution: We could summarize the count from each airport using the `n()` function, which *counts rows*.

All remarkably similar! Note: the `n()` function counts rows, whereas the `sum(VARIABLE_NAME)` function sums all values of a certain numerical variable `VARIABLE_NAME`.

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

Solution:

- `filter` picks out rows from the original dataset without modifying them, whereas
- `group_by %>% summarize` computes summaries of numerical variables, and hence reports new values.

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

Solution:

- Say a flight departed 20 minutes late, i.e. `dep_delay = 20`
- Then arrived 10 minutes late, i.e. `arr_delay = 10`.
- Then `gain = dep_delay - arr_delay = 20 - 10 = 10` is positive, so it “made up/gained time in the air.”
- 0 means the departure and arrival time were the same, so no time was made up in the air. We see in most cases that the `gain` is near 0 minutes.
- I never understood this. If the pilot says “we’re going make up time in the air” because of delay by flying faster, why don’t you always just fly faster to begin with?

(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`.

Solution: No because you can’t do direct arithmetic on times. The difference in time between 12:03 and 11:59 is 4 minutes, but `1203-1159 = 44`

(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.

Solution: Most of the time the gain is a little under zero, most of the time the gain is between -50 and 50 minutes. There are some extreme cases however!

(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`?

Solution: Because `hour` is simply a value between 0 and 23; to identify a specific hour, we need to know which year, month, day and at which airport.

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

Solution: This question is subjective! What surprises me is the high number of flights to Boston. Wouldn't it be easier and quicker to take the train?

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

Solution: When datasets are in normal form, we can easily `_join` them with other datasets! For example, we can join the `flights` data with the `planes` data.

(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.

Solution:

(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`.

Solution:

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

Solution: To narrow down the data frame, to make it easier to look at. Using `view()` for example.

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

Solution:

(LC4.20) 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 coding right away, but 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 4.1 as your toolbox!

Solution: Here are some examples of student-written pseudocode¹. Based on our own pseudocode, let's first display the entire solution.

Let's now break this down step-by-step. To compute the available seat miles for a given flight, we need the `distance` variable from the `flights` data frame and the `seats` variable from the `planes` data frame, necessitating a join by the key variable `tailnum` as illustrated in Figure 4.7. To keep the resulting data frame easy to view, we'll `select()` only these two variables and `carrier`:

Now for each flight we can compute the available seat miles `ASM` by multiplying the number of seats by the distance via a `mutate()`:

Next we want to sum the `ASM` for each carrier. We achieve this by first grouping by `carrier` and then summarizing using the `sum()` function:

However, because for certain carriers certain flights have missing `NA` values, the resulting table also returns `NA`'s. We can eliminate these by adding a `na.rm = TRUE` argument to `sum()`, telling R that we want to remove the `NA`'s in the sum. We saw this in Section 4.3:

Finally, we `arrange()` the data in `desc()`ending order of `ASM`.

While the above data frame is correct, the IATA `carrier` code is not always useful. For example, what carrier is `WN`? We can address this by joining with the `airlines` dataset using `carrier` is the key variable. While this step is not absolutely required, it goes a long way to making the table easier to make sense of. It is important to be empathetic with the ultimate consumers of your presented data!

D.4 Chapter 5 Solutions

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

¹<https://twitter.com/rudeboybert/status/964181298691629056>

```
library(nycflights13)
library(tidyr)
library(readr)
```

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

Solution: Rows correspond to observations, while columns correspond to variables.

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

Solution: Tidy datasets are an organized way of viewing data. This format is required for the `ggplot2` and `dplyr` packages for data visualization and wrangling.

(LC5.3) Take a look the `airline_safety` data frame included in the `fivethirtyeight` data. 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?”². Let’s ignore the `incl_reg_subsidiaries` and `avail_seat_km_per_week` variables for simplicity:

```
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
```

²<https://fivethirtyeight.com/features/should-travelers-avoid-flying-airlines-that-have-had-crashes-in-the-past/>

```

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?

Solution: Using the `gather()` function from the `tidyverse` package:

```

airline_safety_smaller_tidy <- airline_safety_smaller %>%
  gather(key = incident_type_years, value = count, -airline)
airline_safety_smaller_tidy

```

```

# A tibble: 336 x 3
  airline      incident_type_years count
  <chr>        <chr>                  <int>
1 Aer Lingus   incidents_85_99       2
2 Aeroflot     incidents_85_99       76
3 Aerolineas Argentinas incidents_85_99  6
4 Aeromexico   incidents_85_99       3
5 Air Canada   incidents_85_99       2
6 Air France   incidents_85_99       14
7 Air India    incidents_85_99       2
8 Air New Zealand incidents_85_99  3
9 Alaska Airlines incidents_85_99  5
10 Alitalia    incidents_85_99       7
# ... with 326 more rows

```

If you look at the resulting `airline_safety_smaller_tidy` data frame in the spreadsheet viewer, you’ll see that the variable `incident_type_years` has 6 possible values: "incidents_85_99", "fatal_accidents_85_99", "fatalities_85_99", "incidents_00_14", "fatal_accidents_00_14", "fatalities_00_14" corresponding to the 6 columns of `airline_safety_smaller` we tidied.

(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.

Solution: Running the following in the console:

Let’s now compare the `dem_score` and `dem_score_tidy`. `dem_score` has democracy

score information for each year in columns, whereas in `dem_score_tidy` there are explicit variables `year` and `democracy_score`. While both representations of the data contain the same information, we can only use `ggplot()` to create plots using the `dem_score_tidy` data frame.

(LC5.5) Read in the life expectancy data stored at [https://moderndive.com/
data/le_mess.csv](https://moderndive.com/data/le_mess.csv) and convert it to a tidy data frame.

Solution: The code is similar

We observe the same construct structure with respect to `year` in `life_expectancy` vs `life_expectancy_tidy` as we did in `dem_score` vs `dem_score_tidy`:

D.5 Chapter 6 Solutions

To come!

```
library(ggplot2)
library(dplyr)
library(moderndive)
library(gapminder)
#library(skimr)
```

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. (2015). *ggplot2movies: Movies Data*. R package version 0.0.1.
- 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., and Woo, K. (2018). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.1.0.
- 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. (2018). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.9.

